UNIVERSITAT POLITÈCNICA DE CATALUNYA
Departament de Llenguatges i Sistems Informàtics
Programa de Doctorat en Intel.ligència Artificial

TESI DOCTORAL

Case-Based Sequence Analysis in Dynamic, Imprecise, and Adversarial Domains

Francisco J. Martin

2004

Memòria presentada per optar al títol de
Doctor en Informàtica

El treball contingut en aquesta memòria ha estat realitzat a l'Institut d'Investigació en Intel.ligència Artificial (IIIA) del Consell Superior d'Investigacions Científiques (CSIC) sota la direcció del Dr. Enric Plaza i Cervera.

To Juan Cervera, Antonia and Felix Martin, and Jose María Torregrosa
"How I wish, how I wish you were here" [WG75]

To Alba, Andreu, Angie, and Cris
"It was pain, sunny days and rain, I knew you'd feel the same things" [Fol01]

Abstract

The central question addressed in this thesis is: "How can the CBR problem solving paradigm be enhanced to support the analysis of unsegmented sequences of observational data stemming from multiple coincidental sources?". To gain a deeper understanding of this question, we investigate a new CBR model, called Ceaseless CBR, that considers problems that occur simultaneously and whose descriptions (interleaved and without well-defined boundaries) are partially obtained over time through a unique unsegmented sequence. We provide an instantiation of this model sustained by four main innovations: (i) sequential cases represented using actionable trees—a highly intuitive and machine learnable knowledge structure that models the hierarchical structure of sequences and represents master parallel and serial cases as whole-part structures; (ii) a dynamic sequence similarity—a new adaptable similarity based on a subsumption scoring scheme that makes elastic comparisons between sequences and adapts to data over time; (iii) a ceaseless CBR process that automates the discovery of sequential cases and transforms huge volumes of unsegmented sequential data into explanations that can take advantage of human cognitive processing to prioritize problems according to their urgency; and (iv) a formal framework for the evaluation of alert triage systems that facilitates the construction of robust systems in imprecise environments. The model proposed has been satisfactory evaluated in intrusion detection alert triage using real-world data and constitutes a contribution to problems that involve temporally-evolving sequences of observational data such those that arise in real-world domains (international event analysis, network-based sensor-driven monitoring, electric power-delivery supervision, intrusion detection, etc) where the on-line analysis of observational data, disturbed by dynamic, imprecise, and adversarial conditions, is a must in order to diagnose or predict undesired or exceptional situations (e.g. a collision, a conflict, a fault, an intrusion, an outage, etc).

Research Support

This research was carried out at the Artificial Intelligence Research Institute (IIIA) of the Spanish Council for Scientific Research (CSIC) under supervision of Dr. Enric Plaza. I also performed a three months stay at the University of Bath where I collaborated with Dr. Julian Padget. The final document was polished while I was a visiting researcher at the Oregon State University working with Dr. Thomas Dietterich. I was supported partially by the Catalan Autonomous Government (DGR-CIRIT scholarship FI-DT/96-8472) and iSOCO-Intelligent Software Components S.A.

Acknowledgments

My first and foremost thanks go to Enric Plaza, my advisor, who taught me to understand the difference between knowing the path and walking the path. Without his support along my several PhD undertakings and interruptions, this work would never have been finished. For more than three years, I was out running a start-up company. When I came back, Enric had the ability to restore his prodigal student to good standing.

Many thanks to Francesc Esteva and Ramon López de Màntaras for creating the marvelous research environment at the Artificial Intelligence Research Institute (IIIA) where I did most of my work. The IIIA perfectly accommodates good work, good people, and good times. I would also like to thank all the good people I met at the IIIA who, troughout the years, have shaped my way of thinking: Jaume Agustí, Edgar Altamirano, Eva Armengol, Richard Benjamins, Piero Bonissone, Lola Cañamero,Thomas Dietterich, Peyman Faratin, M. Carmen del Toro, Gonçal Escalada, Marc Esteva, Pere Garcia, Lluís Godo, David Gutierrez, Joan Ricard Ibañez, Javier Larrosa, Jordy Levy, Maite López, Felip Manyà, Miguel Mateos, Pedro Meseguer, Oscar Molina, Pablo Noriega, Jordi Puigsegur, Josep Puyol, Jose Antonio Reyes, Ricardo Oscar Rodríguez, Jordi Sabater, Marco Schorlemmer, Carles Sierra, Lluís Vila, and Adriana Zapico. I am specially indebted to Josep LLuís Arcos for all his advice and kindness. Along the way, I programmed NoosWeb, Noostoo, Plural, and Alba all of them based on Noos. Josep Lluís always supported and helped me with Noos. On a more personal note, I had the luck of finding three of my best friends ever at the IIIA: Xavi Canals (the best person in the world), Noyda Matos (the best friend in the world), and Juan Antonio Rodríguez (aka Jar, the best guy in the world). They

have always been there in the good and not so good moments inside and outside the institute.

Outside of the IIIA, I wish to express my sincere appreciation to Ulises Cortes for his advice and help throughout this PhD. I would like to express my gratitude to Julian Padget and all the good people at the University of Bath where I spent three wonderful months: Russell Bradford, Angela Cobban, Andreas Kind, Jeremy Leach, Andy Morgan, and Rob Simmonds. I owe a great debt to Ignacio Ruiz (Dr Nax aka Datils) for collecting data for one of our data-sets and nurturing me through the intricate path between black hats, white hats, and every shade of grey hacks. I wish to acknowledge both anonymous reviewers that provided me with valuable comments on a preliminary version of this document and anonymous reviewers who helped substantially improve various parts of this document when they were previously published. I owe thanks to Vicente Botti who sparked my interest in Artificial Intelligence and María Angeles Pastor who awoke my interest in research.

The completion of this thesis would have been impossible without the support of my wife, Cris, whose love encouraged me in my efforts. My special thanks to my mother-in-law Pilar Perales for taking care of our children wherever and whenever needed. My lovely thanks to my aunt Maria Feliciana who took care of me during my first days in Barcelona. Finally, I would like to thank my parents, Purificacion and Salvador, for the tremendous effort and sacrifices for their children. They gave us what they could never get: *estudios*.

Contents

List of Figures

List of Tables

“Your daddy he’s an outlaw
And a wanderer by trade
He’ll teach you how to pick and choose
And how to throw the blade.
He oversees his kingdom
So no stranger does intrude
His voice it trembles as he calls out
For another plate of food”
[Dyl76]

1

Introduction

This thesis investigates a new Case-Based Reasoning (CBR) model for the analysis of unsegmented sequences of observational data stemming from multiple coincidental sources in complex multi-agent settings. This Chapter provides the reader with a comprehensive overview of the thesis. We commence our exposition pointing out in Section 1.1 the fundamentals that motivated this research and why this new model is required. After a short introduction to Case-Based Reasoning, we expose the pro and cons that this problem solving paradigm poses when dealing with problems that require the on-line analysis of temporally-evolving sequences of observational data. We also establish in this Section the primordial objective of this thesis: To enhance the CBR paradigm to solve situations that are expressed by means of unsegmented, noisy sequences of complex events that arrive continuously over time. In Section 1.2 we describe the concrete application domain—alert triage in intrusion detection—where we have conducted an exploratory analysis of the techniques proposed and evaluated the performance of our contributions. We succinctly enumerate the problems that this domain poses and specify in detail the problems that the model tackles: triage, correlation, and segmentation. Section 1.3 briefly describes how we have approached those problems given the reader a compact overview of the new CBR model proposed that we have called Ceaseless CBR. The Chapter concludes with a formal overview of the organization of this document and signaling some typographic conventions.

1.1 MOTIVATION AND OBJECTIVE

There are a number of problems, whose ubiquity impulsed us to initiate this research, that can best be addressed making use of their underlying sequential structure. These problems arise in an ever-increasing diversity of domains such as discovery of black holes in large-scale astrophysical data, forecasting conflicts in international event analysis, fraud detection in cellular telephones, prevention of outages in electric power-delivery systems, etc. In these problems, an approximate system's state or behavior can be modeled through continuous observations that can be represented as sequences of symptom events over time. An observable symptom event[1] is a complex object made up of numeric, qualitative and structured features. By analyzing those sequences and using past experience one can make a diagnosis about what led to a current system's exceptional state and try to predict likely future events and therefore a future system's state or behavior. Often, those sequences of symptom events stem from automated real-time systems (e.g., IDSes, network sensors, BGP routers, newswire services, etc) that collect and interpret data from multiple coincidental sources. Each symptom event (alert, alarm, warning, announcement, etc) is triggered based on an automated judgment on the presence of some condition of interest in the present or the occurrence of an event in the future. However, a proper mapping between each possible event or combination of events and the root cause (or disorder) that triggered it is not always straightforward and commonly requires a human (expert) operator to disambiguate possible conflicts. Often the presence of a condition only can be indicated for sure after correlating several events (e.g., a burst, a syn-flood attack, a multi-stage attack, a cascading failure, etc). From all this it follows that such sequence analysis task requires continuous human oversight. Automating this task, and therefore reducing manpower requirements, demands to employ artificial intelligence techniques that go beyond simple rule-based systems—if-then statements that compile an expert's knowledge about the interdependence of different events. The main drawbacks of rule-based systems are: (i) in complex domains to list the complete set of rules beforehand is practically infeasible; (ii) they do not easily adapt to changing circumstances; (iii) they require constant maintenance what makes writing and keeping up to date rules a challenging, time-consuming and error-prone process; and (iv) significant manpower is required to subsequently verify their outcome [Lew95, HMP02]. In addition, the imprecise and adversarial conditions that often surround the aforementioned domains and the fact that they usually lack a strong theoretical foundation makes this endeavor particularly difficult.

Since its inception in the early 1980s and along the years, Case-Based Reasoning (CBR) has been often qualified as the perfect substitute of rule-based systems in complex domains. Nevertheless, as signaled below, most CBR systems commonly presuppose individualized problem descriptions with well-specified boundaries that

[1] Throughout this document, we refer to a symptom event simply as an event, an alert, or a manifestation indistinguishably to denote an observable event; and we refer to the origin of a problem as problem, root cause, attack, or disorder also indistinguishably to denote the non-directly observable cause of an event.

encompass all the information needed to solve the current problem in only "one shot". This assumption makes impracticable their direct deployment in these domains.

1.1.1 Case-Base Reasoning

Case-Based Reasoning (CBR) is a well-established automated reasoning paradigm within the Artificial Intelligence (AI) field that provides computers with the ability to continuously improve their performance over time reusing past experience—emulating one of the main hallmarks of human intelligence [SA77, RS89, Kol93, Lea96, Wat97, Sch99]. Nowadays, CBR systems are widely extended in a diversity of domains [Wat97]. In essence, a CBR system relies on storing problem-solving experiences used facing past situations to subsequently retrieve and reuse them when similar situations roll around. The ubiquitousness of CBR systems is, in part, due to two so pervasive assumptions that make them well-suited for a wide-spectrum of real world domains: (i) situations recur and (ii) similar situations require similar solutions. Based on these assumptions many CBR systems have been developed following the dominant mainstream CBR model that establishes four sequential processes for confronting a new situation: *retrieve*, *reuse*, *revise*, and *retain* [AP94] (see below for a succinct overview of each process). Furthermore, other great advantage of CBR over a significant number of automated problem solving paradigms is its appropriateness for dealing with weak-theory domains [Aam95]. That is, ill-structured problems where uncertainty, ambiguity and missing data abound [Aam95, RTT96]. There is no doubt that in domains where a strong theory doest not exist reusing past experience constitutes the most valuable heuristic for handling intractable problems.

Now, let us suppose a case-based reasoner based on the mainstream CBR model that maintains a repository of cases, case-base, or library where constantly stores solutions (problem-solving expertise) deployed to solve old problems (situations) paired with the corresponding problem descriptions that originated them. The process deployed to solve a new situation by such system could briefly be described as follows.

New case acquisition Initially, a new individual problem description with well-defined boundaries is received passively (i.e., one by one).

Retrieve Subsequently, an individual problem description (or new case) is used to retrieve one or more cases from the case base. The retrieved cases are similar to the new case along some user-defined dimensions that are also used to rank them.

Reuse Subsequently, the most similar case is reused, in some domain-specific way. For example, in transformational adaptation the process may add, delete, substitute, generalize, and specialize parts or aspects of the solution. In derivational analogy [Car86], the solution process is reused, while in constructive adaptation a new solution is built through a search-based technique guided by cases [PA96]. See Wilke and Bergmann's work for a continuum of adaptation models [WB98].

Revise Afterwards, the solution proposed is revised in one way or another commonly employing user's feedback that is usually concerned whith the correctness of the solution.

Retain Finally, the revised solution and the problem description form a new case (experience) that is retained by integrating it into the case-base according to predetermined indexing criteria (but without taking in consideration its sequential relationship with other solved problems).

The fact of the matter is that most CBR systems are built on that model and devised bearing in mind the following three interrelated assumptions.

Non-coincidental sources There is only a problem occurring at a time. Said differently, problems are solved successively one after another without considering problems that concur and whose origin could be related and could require a joint solution.

Full-fledged problem descriptions A problem description is provided in only one shot (instantaneous situations) with well-defined and clear limits—i.e., the boundaries of each case are perfectly delimited[2].

Individual cases independency Cases are manipulated (retrieved, reused, revised or retained) in isolation without contemplating their sequential (spatial or temporal) structure or (serial or parallel) relationship with other cases in the past. That is, they assume (snapshot) cases that are independent of each other, and therefore relations among cases (e.g., sequential relationships) are not taken into account.

These assumptions make the model unsuitable for a number of challenging problems—mainly those that involve temporally-evolving sequences of observational data. In other words, most CBR systems try to solve problems in one shot neglecting the sequential behavior of most real world domains and the simultaneous occurrence of interleaved problems proper to multi-agent settings. All the aforementioned assumptions may well be true enough in many application domains. Even so, the question arises of how far we can go with them. That is to say, those assumptions restrict the number of real world scenarios in which CBR can be applied discouraging its use in challenging problems such as those that arise in real-world domains (e.g., international event analysis, network management, intrusion detection, etc). In these domains, the on-line analysis of observational data is a must in order to diagnose or predict undesired situations. Thus the interest of investigating new techniques that

[2]Sometimes when only a partial problem description is provided, an interactive dialogue is engaged with the user to better delimit and clarify the descriptions provided [AMB98]. Through this conversation with the user, a complete and individual description is obtained in the end. In our approach notice that the complete description could never arrive due to, for example, noise, an attacker interruption of his plans, a malfunction of the own sensor, etc

allow one to alleviate such constraints and applying CBR in a variety of much more complex domains.

Let us consider in the next Section some issues of the problem we cope with that merit closer examination.

1.1.2 The Problem

As we have mentioned above, CBR practitioners are sometimes oblivious that there often situations in the real world where problems occur simultaneously and whose descriptions come interleaved or in continuous form—i.e., without well-defined boundaries between adjacent problem descriptions—and additionally require continuous response to changing circumstances—i.e., a timely action once a proper solution has been identified. We are often faced in daily life with multiple coincidental problems that even overlap one another since they may be occurring at the same time originated by different agents. For example, in automated monitoring a watchful agent constantly assesses the current situation in order to provide an early warning that facilitates a timely and convenient response, that in turn should impede an undesired or exceptional situation (e.g., a collision, a conflict, a fault, an intrusion, an outage, etc). Sometimes a unique problem (root cause or disorder) generates a series of events—i.e., its problem description is broken up into small pieces that arrive at intervals. Sometimes a combination of problems (one o more) may in turn cause other (one or more) problems what can be interpreted as a big problem composed of small subproblems (e.g., cascading failures). Thereabout the granularity with which problems are modeled, represented, composed, and analyzed is a primordial issue. Another telling aspect that deserves consideration is the presence of noise. Often events are transmitted through middling channels whose reliability cannot be guaranteed (e.g., SNMP, UDP, etc). This unreliability causes the presence of corrupted (lost and spurious) events that seriously interfere the proper analysis of the real events. The fact of the matter is that automated sensors in many different and important domains (in addition to being noisy and imperfect) lack the intelligence to disambiguate (differentiate and synthesize) the parts corresponding to distinct problems so they resort to piecing all the sensed parts together into only one sequence (e.g., the syslog event logging system). That is, several problem descriptions corresponding to problems occurring in parallel are serialized into a unique sequence without comprehensible criteria for its further understanding. That sequence exhibits tremendous variability what interferes its correct interpretation and requires continuous human oversight. We say that such sequence is (i) *sparse* (i.e., only few segments are really meaningful and appear disperse into countless irrelevant others [Esk02]); (ii) *unsegmented* (i.e., there is no clean mark that separates meaningful segments from other); and (iii) *hides* n problems stemming from s different (and possibly recurrent) unknown sources. To anticipate an undesired situation is crucial to quicken the recognition of the segments along the sequence that conform a problem description that resemble unsought past situations. Nonetheless, this is one of the most striking aspects of our undertaking, since each problem description is composed of an undetermined number of parts (one per event) that arrive at different intervals and are blurred together with other

problems' parts into a single on-line stream. Moreover, a problem cannot be solved successfully based on a single part but on a number of correlated parts. Unfortunately, determining how parts are correlated with each other and predicting or discovering the number of parts are two intrinsic particularities of the problem itself. These severe difficulties and the need of guaranteeing a rapid response make this problem extremely difficult.

A further complication arises when we consider the specific domains in which a situated monitoring agent might operate. Few will dispute the claim that in the real world we are likely to confront open and weak domain theories rather than complete and robust domain theories[3]. In this thesis we have considered highly dynamic environments (e.g., computer networks with constantly software, hardware, and configuration updates where in addition ad hoc networks are deployed, used, and then dismantled). In such environments the pace at which changes arise and the imperfection of the sensors force agents to reason in terms of incomplete and imprecise information (e.g., what today is dangerous may tomorrow become innocuous, what today is secure may tomorrow be vulnerable, who today is an ally may tomorrow be an enemy, etc). Furthermore, sometimes we have to count on the presence of hostile and skulking agents that hinder their actions in a sneaky manner trying to interfere and delay our assessment of the situation so that we cannot recognize their stealthy and malicious plans (intentions) promptly.

To put the whole matter in a nutshell, we cope here with problems that are much more challenging than most of those solved by CBR practitioners before, given that each problem description is composed of an undetermined number of parts that arrive continuously over time and are blurred together with other problems' parts into a single on-line stream. Therefore, the most remarkable driver and motivation of this thesis has been to answer the following question: how can a past situation be identified when the current situation is specified in terms of an unsegmented, sparse sequence of complex objects that arrive continuously over time? Said differently, how can the CBR paradigm be enriched to support the analysis of unsegmented sequences of observational data stemming from multiple coincidental sources?

1.1.3 Objective

Answering the latter question practically brings us directly to the objective of this work: *To enhance the CBR paradigm to support the analysis of unsegmented sequences of observational data stemming from multiple coincidental sources*. Namely, this thesis aims at establishing a first CBR model, that we have called Ceaseless CBR, to solve situations that are expressed by means of unsegmented, noisy sequences of complex events that arrive continuously over time. This endeavor entails to provide a model that enables reasoning in terms of problem descriptions that are broken up into small pieces that are mixed with other problems' pieces and the possibility of

[3] An open problem is characterized by incompleteness and frequent changes whereas a weak domain theory involves uncertain relationships between its concepts [Aam95].

- How can the sequential structure of past experiences be conveniently represented?
- How can we measure the similarity between a sequential case and the sequence of symptom events at hand in a noisy environment?
- How and when can we decide that a sequential case or set of sequential cases satisfactory explains a sequence of observable symptom events?
- How can the discovery of new sequential cases be automated?
- How can we measure the performance of Ceaseless CBR?

Fig. 1.1 Challenges of Ceaseless Case-Based Reasoning.

combining a number of cases that best match the sequential structure of the problem at hand. A number of key challenges arise from this objective, that we have decomposed into five distinct sub-objectives.

1. First of all, we need to conveniently represent a case so that it encompasses the sequential structure of a past problem and facilitates its recognition and reconstruction—we call sequential cases to case representations that accomplish those requirements.

2. Secondly, we should devise a similarity measure able to compare a sequence of events embodied by a sequential case with the noisy, sparse sequence of events at hand.

3. Thirdly, a mechanism has to be provided such that given a sequence of events corresponding to different intervening problems computes the best collection of sequential cases that can completely explain that sequence considering the risk and costs of different alternative combinations.

4. Fourthly, we should be able to automate the discovery of sequential cases so that continuous human intervention can be avoided as much as possible.

5. Finally, we need to be able to evaluate the performance of the model proposed.

We have summarized these challenging objectives in Figure 1.1. Next, we briefly overview an application domain with the above dynamic, imprecise, and adversarial characteristics.

1.2 THE APPLICATION DOMAIN

The concrete application domain that we have selected for our exploratory analysis is *intrusion detection* [ACF99] and concretely *alert triage*—the rapid and approximate prioritization for subsequent action of an Intrusion Detection System (IDS) alert stream [MP03b]. This is a challenging domain where many issues still remain

open and meets the features mentioned above [ACF99]. Next, we provide a brief introduction to intrusion detection alert triage.

Security managers[4] responsibilities include, among other perimeter defense tasks, prevention, detection and response to computer security incidents. Nowadays, Intrusion Detection Systems (IDSes) [ACF99] have become common tools (eTrust, eSafe, IntruShield, RealSecure, etc) deployed by security managers to combat unauthorized use of computer systems. First research on *computer-aided intrusion detection* (or simply *intrusion detection*) goes back to the 80's [And80]. The main task of IDSes can be generalized as the real-time monitoring of *computer activities* searching for deviations from normal behavior or *nonintrusive* behavior. Computer activities can be monitored at different level of detail: system calls traces, operating system logs, audit trail records, resources usage, network connections, etc. An intrusion is defined as any set of actions that attempt to compromise the integrity, confidentiality, or availability of a resource disregarding the success or failure of those actions [SZ00]. Whenever an intrusion (or suspicious activity) is detected an IDS notifies to the proper authority evoking *alerts*. Commonly, alerts take the form of emails, database or log entries, etc and their format and content vary according to the particular IDS. An IDS alert stream merges alerts stemming from an undetermined number of sources that, broadly speaking, may be classified into: network entities' malfunctions or misconfigurations (including the own IDS), innocuous attacks, and malicious attacks.

The fact of the matter is that current IDSes generate an unmanageable number of false positive alerts[5] which in turn increases the difficulties for the proper identification of real and malicious attacks. Security managers are so overwhelmed that they frequently disable the alert device due to the consistent assumption that nothing is wrong reinforced by the fact that the alert device "cried wolf" too often [Swe96]. There are those who even postulate that current IDSes not only have failed to provide an additional layer of security but have also added complexity to the security management task. Therefore, there is a compelling need for developing a new generation of tools that help to automate security management tasks such as alert triage. It should be stressed that as long as the number of networked organizations proliferates and the number of computer security threats increases this need accentuates. To make the aforementioned tasks more manageable we envisage a new generation of intrusion detection tools under the heading of *agent-aided intrusion detection*. The different techniques devised throughout this work have been embodied within a long-lived autonomous agent—called Alba (Alert Barrage)—that could be cataloged as a striking example of this tendency [MP03e]. Chapter 7 of the thesis describes how we have developed such first prototype.

[4] Also known as SSO-Site Security Officers.
[5] Alerts signaled when there is a manifest absence of intrusive behavior.

1.2.1 Alert Triage

The objective of alert triage is to minimize the number of recurrent false and innocuous alerts that security managers receive and improve the chances of anticipating or at least signaling malicious attacks[6]. Alert triage produces a rapid and approximate prioritization for subsequent action of an IDS alert stream [MP01][7]. Alert triage is the main task to be performed by IDS modules that have been named differently in the literature such as alert correlators [NC02], d-boxes [SS98], analyzers [WE02], ACCs [DW01], assistants [MP03e], etc. In our case alert triage can be seen as a classification task that takes as input the alerts generated by a number of intrusion detection probes or sensors (e.g., Snort [Roe99] see Chapter 7 of the thesis for further details) and classifies them producing a tag indicating its malignancy (degree of threat) for each alert. Such tags prioritize alerts according to their malignancy. Depending on its priorization each alert will require an automatic response, notification, or manual response from the site security officer (SSO). The tag assigned to alerts can be a (fuzzy) label (e.g. malicious, innocuous, falsepositive, etc) or it can be a continuous value. Alert triage can be explained in terms of the following priority formulation [PFV02] where *mission* is defined in terms of the critical and sensitive assets of the target network (i.e., the system under surveillance):

Problem 1 (Priority Formulation) *Given an alert stream where each alert is defined according to a pre-established alert model (see Section 3.2) return a subset of high-impact alerts such that the threat that represents each alert in such subset is above a pre-specified decision threshold.*

Examining such formulation it can be said that the decision threshold will discern between two (apparently) disjoint classes of alerts (i.e those alerts that require notification to the SSO and those alerts that do not). Thus, we suppose that once the alerts have been prioritized there will be only two possible responses (notification and ¬notification). Therefore, without loosing generality, we will proceed in our evaluation as if alert triage was a detection task. Chapter 6 of the thesis is devoted to throughly describe the framework that we have introduced to formally analyze the performance of alert triage systems in different environments.

Ceaseless CBR aims at enabling the triage of alert streams that are subject to the bothersome characteristics that we expose in the next Section.

[6] When limited resources come into play, determining how to most efficiently use them becomes paramount. This is the foundation for triage. Triage comes from French verb “trier” that means to sort. For example, prehospital triage in mass casualty incidents (MCI) aims at maximizing the number of patients who will survive an incident when there are more patients than available resources [BKS96, Str98].

[7] In information systems, triage is usually deployed by information retrieval systems that prioritize a number of documents before a human reader receive them and can be extended to domains where automated processes produce a huge amount of data that needs a preliminary pre-process [MP01].

1.2.2 Some Alert Triage Issues

This Section describes some particular issues, most of them introduced generally in Section 1.1.2, that poses alert triage in intrusion detection such as *partially-ordered plans*, *deception*, *multiple-goals*, etc The objective here is to pinpoint some of the differences that entail this domain with respect triage in other domains and to give the reader a collection of running examples that we use throughout this document.

Relative Environment-Dependency Alerts should have different priority depending on the specific environment where they are raised. That is to say, the importance of an alert varies according to the network under surveillance and its actual security policy and mission. Said differently, an alert can be classified as malicious or innocuous depending on whether the destination is really susceptible or not to that attack. Nonetheless, innocuous alerts cannot be complete ignored. A common example is the following.

Example 1 *Imagine a Code-Red Worm[8] attack against a Linux site. It makes no sense to directly notify the security manager for such attack since it has not impact in the system[9]. However neither makes it sense to completely ignore it, since it could come from a trusted machine that from now on we should have to consider that it is compromised.*

Thus, the solution often suggested to mitigate the elevate number of environment-dependent alerts (i.e., to instruct the IDS sensor to not evoke this kind of alerts) is debatable. Although this solution could reduce the number of false positives in the beginning, it could also increase the number of false negatives in the end[10].

Multi-stage Attacks There are intrusions that require only a single action (hit-and-run intrusions) to cause a malign effect (e.g. the Ping of Death attack [Nor99]). Other intrusions, on the contrary, are sophisticated multi-stage attacks where each intruder's action is a step intended to result in a change of state of the target computer system that prepares it to accept the next intruder's action (e.g., the Mitnick attack that we explain below [Nor99, SM96, MS02]). Normally, an isolated alert (per se) cannot be correctly classified as malicious or innocuous. Imagine the following example:

[8]This worm uses a known buffer overflow vulnerability contained in the file Idq.dll (ISAPI extension) and affects IIS 4.0 and 5.0 web servers runing on Microsoft Windows NT 4.0 and Windows 2000 (see CVE-2001-0500 and CVE-2001-0506 for further information). Notice that if a Windows machine with a listening IIS and root.exe exists then full-system level access is possible.
[9]Except for minor irritation that comes from the large (irritating) number of alerts that generates.
[10]Thus, usually, alerts are stored for subsequent post-mortem analyses independently of whether they were dismissed or not and correctly prioritized to be notified to the proper authority.

Example 2 *An attempted-recon alert*[11] *coming from an unknown IP has been received. This alert can correspond to an attacker performing a reconnaissance or on the contrary it could really correspond to one of our engineers checking at our customer's office that one of our web services is up*[12].

Therefore, alerts cannot be classified separately but considering their relationship with other alerts. Then, we need to provide a form of *collective classification* [TAK02]. That is, correlating several alerts and then deciding on the priority of all of the alerts together. Often we need to compile more evidence before going further and assigning the correct priority or initiating a considered response (e.g., creating a new rule in a firewall). Furthermore, some single actions cause multiple effects [GG01]. For example, an attacker could launch a massive and indiscriminate reconnaissance attack (using a port scanner) against a network causing a denial of services but in turn he or she is compiling information for further attacks. Thus, we also say that attackers have multiple goals.

Multiple Goals Properly determining the priority of an alert (or group of alerts in the case of multi-stage attacks) does not only require identifying the attack but also to infer the goals of the the attacker. Take the case of the following example [GG01].

Example 3 *We have just received an alert indicating a synflood attack*[13]. *The goal of this attack could be (i) denial of services—just to put down the flooded target—or (ii) access to a third system that trust the flooded target—i.e., to impede that the flooded target resets a forged connection during a Mitnick attack. The proper response in each case varies. In the first case, to prevent it we only need to modify the firewall to limit the number of connections from the attacking source. In the second case, the firewall needs to be updated to reject any connection that trusts the flooded computer.*

Executing one or other countermeasure causes different disruption in our system or could even not impede an attack. If we are wrong and respond thinking that the goal is a denial of services then the attack will surely be completed. On

[11] This sort of alerts corresponds to automated actions carried out by port scanners such as nmap (Network Mapper) [Fyo97, Hil01] that try to obtain relevant information (security loopholes) from the target computer for subsequent attacks.

[12] What is quite certain is that usually instructed attackers would use a stealthy nmap that often help us to differentiate between both.

[13] This a denial of service attack that explodes a flaw in many TCP/IP implementations. This attack never completes the three-way handshake needed to establish a TCP connection sending out an unmanageable number of forged SYN packets (the first hand of the 3-way handshake) that flood the target. Each SYN packet received causes a buffer to be allocated by the target until the internal limits of the listen() call are reached. This is one of the stage of the multi-stage attack originally attributed to Kevin Mitnick [SM96, MS02].

the other hand, if we are wrong again and respond as it was an access to the system then we could interrupt some external and perhaps mission-critical connections to our network. Thus, determining the goal of the attacker is crucial (i.e., recognizing plans). Although combining multiple reports and information to identify an attacker's goals have been proposed, current IDSes are far away from this functionality [GG01]. Moreover, an attacker could have multiple concurrent goals. A single, but striking, example is the following. After accessing to the system and stealing some confidential information the attacker could use the compromised computer to initiate subsequent attacks against other systems. A further complication is that the sequence of actions (plans) executed by attackers are very flexible (partially ordered) and varied in terms of time and actions. For example, the same sequence of actions executed in only five minutes or temporally distributed along several days could cause the same malign effect.

Multiple sources In addition to an attacker performing a multi-stage attack with multiple possible goals we also contemplate the possibility of multiple attackers (human, artificial, or a combination of both) at a time. As a matter of fact, while our networks are continuously random-chosen and bombarded with worm propagation attacks, new a more dangerous attacks constantly appear that could also randomly choose our network as a target. As a consequence, our alert streams present a continuous overlapping of different unpredictable sources with more predictable sources that consciously selected our network as a target (e.g., old or current annoyed employees).

Sparseness Only a few elements of the alert stream are meaningful. The alerts providing evidence that an attack is undergoing are dispersed into the alert stream among countless irrelevant (false positive) alerts [Esk02]. For this reason, we say that we look for situations that are *hidden* in the alert stream.

Deception We are facing an adversarial domain where the opponent is a hostile agent that not only tries to hinder his actions (i.e., masquerading his identity) but also tries to delude our analysis (i.e., denying the service of the sensors). See [Pta98] for a report on eluding network intrusion detection. Hence, to overcome this hindrance it is required to be able to infer hindered actions from other observed actions and from observation of state change [GG01].

Large Alphabets The alphabets that conform the sequences under consideration are very large compared to other domains (e.g., 4 in the case of DNA sequences {Adenine (A), Cytosine (C), Guanine (G) and Thymine (T)}). In our case the alphabet is composed by the identifiers corresponding to the IDS alerts (around 1800 using Snort as of writing this document [Roe99]).

Complex Objects Alerts are complex object made up of numeric, qualitative and structured features. When comparing two simple objects (such as numbers or symbols) it is easy to determine a perfect match (e.g., Cytosine is different from Guanine). However, when comparing complex objects, as in our case,

a similarity relationship has to be established to determine how similar they are or simply whether they are similar or not. As a consequence of this, most techniques for sequence analysis such as discrete sequence analysis or time-series prediction algorithms cannot be directly applied since they are based on sequences of numerical values, usually real numbers, equidistant over time.

On-line Analysis and Learning Several sequence analysis algorithms require the whole sequence to be available beforehand. In our case, only a prefix of the sequence is available over time. Although a number of intrusion detection techniques are devised for post-mortem analysis, we aim at developing techniques able to consume and analyze sequences incrementally in quasi-real-time to conveniently assist security managers. Our model should allow an agent to learn new experiences while interacting with the environment (on-line learning [MCM83, Pfl02]) as well as introspecting (off-line earning).

From all this it follows that this domain requires to efficiently and incrementally recognize hidden partially ordered plans from multiple concurrent attackers that are represented by large alphabets of complex objects overlapped into an on-line sparse stream [GG01]. Assigning the correct priority to alerts demands the ability to recognize subtle relationships between alerts so that they can be conveniently correlated and prioritized together. We will briefly discuss alert correlation in the next Section.

1.2.3 Alert Correlation

An effective triage implies to be able to predict the malignancy, the innocuousness or the falsehood of alerts. As we have seen above, in spite of the fact that a number of individual alerts can be classified univocally (e.g., Ping of Death), other alerts require to be pieced together before they can be correctly triaged. This task of piecing alerts together that are related in one sense or another is called correlation. Alert correlation aims at avoiding poor judgments that try to find an explanation for each alert in isolation without considering the bigger picture [Lew95]. Additionally, alert correlation helps to form a more concise vision of the alert stream providing security managers with a better understanding of current attacks. Correctly correlating alerts is a critical challenge for network security that often implies comparing network events that are sequentially distributed in time and space [JC]. Although simple and obvious methods are useful they are subject to deception (see above). For example, the most naive method to correlate alerts in intrusion detection is considering their source and destination IP addresses as well as TCP ports. This method is most times unreliable due to the ability of attackers to forge (spoof) their source addresses[14]. Therefore more sophisticated and efficient algorithms are required. For instance, capturing the logical steps or strategies behind attacks and correlating alerts using the necessary

[14]The only current valid method against forgery or spoofing is obligating computers to work constantly in paranoiac mode (i.e., doing both forward and reverse DNS lookups each time a new connection is established).

conditions for an attack to be successful (prerequisites) and its possible consequences [NC02]. That is to say, correlating the alert(s) of two attacks when the consequences of the first match the prerequisites of the second [NC02]. In Section 2.6.1 and 2.7.1 we examine further issues of alert correlation in network management and intrusion detection respectively.

An effective triage requires to correlate and find the best explanation for each particular alert or group of alerts received so far. Finding the best explanation for a sequence of events can be modeled as a segmentation problem as we see in next Section.

1.2.4 Alert Segmentation

We suppose that a sequence under observation, $\vec{S}$, is an interleaving of subsequences of alerts (segments or subdivisions) resulting from multiple sources (disorders or problems). The subsequences of alerts due to coincidental sources may overlap as shown in the following example. Imagine that there are three disorders that respectively cause the following sequences of events $[a_{\varphi_1}, b_{\varphi_1}, c_{\varphi_1}]$, $[c_{\varphi_2}, c_{\varphi_2}]$, and $[d_{\varphi_3}, c_{\varphi_3}]$. Then the sequence finally observed could be $[c_{\varphi_2}, d_{\varphi_3}, c_{\varphi_3}, a_{\varphi_1}, b_{\varphi_1}, c_{\varphi_2}, c_{\varphi_1}]$ or $[a_{\varphi_1}, c_{\varphi_2}, d_{\varphi_3}, b_{\varphi_1}, c_{\varphi_2}, c_{\varphi_3}, c_{\varphi_1}]$ or whatever other sequence permuting the order of each source but always maintaining the total ordering for each subsequence. Let us assume that a given point in time there is in the system a prefix of $\vec{S}$ composed of exactly p alerts. We suppose that there are $h+1$ hidden sources and that each segment was caused by only one source. More precisely, each alert was originated by one and only one disorder (malfunction, fault, attack, etc). For the sake of brevity we ignore the presence of noise in this discussion and postpone it for Chapter 5 of the thesis. If we consider the set Υ made up of all possible sources (where we denote all possible unknown sources by φ_0). A sequence $\vec{S}$, made up of $n \geq p$ alerts, can be seen as a concatenation of m segments $\vec{S}_1, \vec{S}_2, \cdots, \vec{S}_m$ stemming from a set of $h' \leq h+1$ possible sources. We define the best explanation of a sequence of alerts as follows[15]:

Problem 2 (Best Explanation) *Given a set of possible sources Υ, and let $\vec{S}$ be a sequence of n alerts where each alert is defined according to a pre-established alert model (see Section 3.2) such that only a prefix of length $p \leq n$ is available find: a subset $\Upsilon' \subseteq \Upsilon$ that maximizes the probability that all sources in Υ' intervene and each alert in $\vec{S}[1..p]$ is explained by a source from Υ'*

This problem has been shown to be NP-complete elsewhere, demonstrating that it is a generalization of the NP-complete set cover problem [KS95]. This problem appears with different guises in the literature. For example, Kauz and Allen proposed to recognize plans computing the minimum set cover of the plan graphs [KA86].

[15]This formulation is in effect very close to the one exposed by Gionis and Mannila [GM03]. But our problem differ clearly in three aspects. First, we do not have the complete sequence beforehand only a prefix. Second, the segments are made up of complex objects, and, third, we consider noisy sequences.

Their work laid the groundwork of most of the subsequent plan recognition systems that we will mention in Chapter 8 of the thesis. As we view respectively in Sections 2.6.1 and 2.6.2, Katzela and Schwartz designed a collection of fault localization algorithms to find the best explanation of a set of alerts [KS95]; and Peng and Reggia introduced a formal theory of diagnostic inference named *parsimonious covering theory* [PR90] both based on the same *covering* concept. Depending on the number of sources allowed, the computation and therefore complexity of the solution to the best explanation problem above varies. When the number of sources is restricted to be minor than the number of segments then problem is NP-hard as recently demonstrated [GM03]. This problem has received unbalanced attention in the literature with respect to when the number of sources is permitted to be equivalent to the number of segments [Chu89, GM03]. There are number of works that have discussed the last problem mainly for genome sequences [KCH04]. However, in these works the result does not considered the relationship between different segments within the sequence. Said differently, the sources assigned to each segment are independent each other and therefore the number of sources is equal to the number of segments. A further complication is that we deal with sequences of complex objects and not merely with sequences of points in $\mathbb{R}^d : d > 1$ [GM03]. As we will see in Chapter 5 of the thesis, the Ceaseless CBR model provides a heuristic approach to the intractability of these problems. These problems arise in a number of domains as succinctly we describe in the following Section.

1.2.5 Other Application Domains

Finally, in spite of the fact that we have placed particular emphasis on intrusion detection alert triage, there are many other domains that exhibit similar characteristics that constitute fertile fields for the application of our approach such as: forecasting conflicts in international event analysis [Sch95], fault identification in communication networks [KS95, RH95], BGP misconfigurations in network management [MWA02, ZPW02], telecommunications alert management [Kle99], and autonomous robotic navigation [RS93a], etc.

Next Section provides a first overview of our new CBR model that we develop in more detail in subsequent Chapters of the thesis.

1.3 CEASELESS CASE-BASED REASONING

The goal of this Section is twofold. Firstly, to give the reader a compact overview of our approach. Secondly, to provide a summary of our contributions. Ceaseless CBR, the novel CBR model proposed in this work is characterized by the following salient features:

Sequential cases A sequential case is a *compositional case* where additionally a temporal order has been established among all the parts that comprise it. A compositional case (the whole) is an assemblage of several cases (the parts).

The parts of a compositional case lay in a hierarchical structure. Sequential cases can be understood as *Predictive Compositional Hierarchies* (PCH) [Pfl02, Lam00]. Therefore when a sequential case is partially-matched it enables the prediction of the whole based on the matched parts (sub-cases). We introduce *actionable trees* as a knowledge structure that enables the representation of sequential cases. An actionable tree is a multi-rooted tree-like structure with the semantics that roots symbolize observable symptom events, intermediate nodes (in the trunk and crown) embody composite cases, and the arcs represent part-whole relationships as well as the likelihood of occurrence of the whole given its parts. Moreover, an intermediate node specifies constraints among the distinct sub-cases that compound it. An actionable tree has a direct map with a context-free grammar that allows one to compute the set of yields that the actionable tree represents. Each yield denotes a possible succession of symptom events that could be caused by the same problem or disorder that the sequential case represents. We use a semi-ring structure to define a general likelihood framework over actionable trees that facilitates the independence between their compositional structure and the model used to compute their evidence (deterministic, probabilistic, Dempster-Shafer, etc) [BMR97]. The composition of sequential cases is based on the operations provided by the semi-ring. Sequential cases provide two clear advantages. On the one hand, they enable part-whole reasoning within an object-centered language [Lam00]. On the other hand, they allow an object-centered language based on terminological knowledge to express uncertainty about the presence of individuals and compositions [KLP97]. Moreover, sequential cases can be automatically discovered by means of the Ceaseless CBR method that we explain below.

Dynamic Sequence Similarity A dynamic similarity between two sequences of complex objects based on the following components.

1. A dynamic subsumption scoring scheme that establishes the similarity between two individual alerts according to their probability of occurrence and their level of abstraction in a taxonomic hierarchy. This scoring scheme, based on the reciprocal of the *Odds*, assigns a high score to rare alerts that are subsumed by sequential cases whereas frequent alerts receive a low score. The subsumption scoring scheme is continuously updated upon the arrival of new alerts.
2. A semi-global alignment obtained by insertion of a number of fictitious elements such that both sequences have the same length and in the individual alignment of the elements at least one of the two elements is not fictitious.
3. Two operations, *abduction* and *neglection*, based on the *alpha-rarity* of each alert that alter a sequence so that corresponding elements in both sequences can be comparable. The abduction operation injects an alert in the alert stream at a given position whereas the neglection operation

ignores an alert in the alert stream. These operations are useful for dealing respectively with lost and spurious symptom events.

4. A dynamic programming algorithm that computes the score of the optimal alignment according to the following policy: the rarer the sequence of alerts the higher the score of the alignment. In Chapter 5 we see how this similarity has been normalized to provide an upper bound that allows us establish a decision threshold for retrieval purposes.

As we see in Chapter 4, the main advantage of our dynamic sequence similarity is the capability of retrieving sequential cases contemplating not only the present but also the past and the future to compute the perceived similarity of sequences in dynamic situations [KS01].

Ceaseless CBR process A constructive situation awareness process governed ceaselessly by: (i) observational data and (ii) the sequential case base. The sequence of alerts received so far pushes towards a situation (exceptional or not) whereas the sequential case base pulls towards the best explanation expressed in terms of partially-matched sequential cases (case activations). We have broken up this process into four sub-processes:

Ceaseless Retrieve Using the above dynamic sequence similarity and a pre-specified window model (see Section 3.5), this process continuously compares the sequence of alerts with the sequential cases in the case base. For each sequential case that is similar above a pre-established threshold a *case activation* is created. A case activation represents a hypothesis that could explain part of the current sequence of alerts. Case activations that fulfill the constraints specified by sequential cases can be pieced together over time. This reduces the number of elements to compound overall *explanations* and facilitates the discovery of new sequential cases. As a matter of fact, Ceaseless CBR operationalizes the discovery of new actual sequential cases in only one pass over the alert stream. This turns out to be an additional advantage of our approach since it does not require to be provided with an initial sequential case-base.

Ceaseless Reuse This process combines case activations to form *explanations*. Explanations are ensembles of case activations that explain the whole sequence of alerts at hand. A belief function determines when each explanation is susceptible of being used to prioritize the corresponding alerts. However, if this process prioritizes alerts too soon, that is, without being completely sure of the presence of a (possible) exceptional situation the number of false positives will be high and the ultimate objective (to triage the alert stream) will not be achieved. On the contrary, if it prioritizes too late and an exceptional situation is really occurring then the time to enable a prompt response is reduced. Thus we take a decision-theoretic approach that maximizes the overall utility of each complete explanation and define a measure of urgency that guides the decisions of this process over time.

That utility takes into account the distinct costs that could be incurred after each prioritization as well as the possible damage costs. The utility values assigned to each explanation under consideration are estimated based on the success of similar decisions in previous situations. As we will see in Chapter 5 of the thesis this process facilitates the segmentation of the sequence of alerts seen so far providing the best (sub-)explanation for each segment.

Ceaseless Revise This process continuously provides a human (expert) operator with the set of most likely explanations given the alerts received so far (instead of presenting a solution periodically). The operator can define a threshold such that individual explanations whose likelihood is above it produce an automatic triage of the corresponding alerts. The operator's feedback may create a completely new sequential case or update a past sequential case (adding, deleting or altering observable events or constraints among them), altering its risk (threat, exposure, or cost) or the corresponding prioritization. The operator's feedback produces a set of revised solutions that in turn produces the triage of the corresponding alerts and initiates a back-propagation process that automatically updates the current case activations. Operator's feedback can be substituted by a model of the system under surveillance what allows us to completely automate the whole process. Keeping up-to-date such model becomes paramount as we will see in Chapter 7.

Ceaseless Retain Once a solution has been revised by the user, the retain process updates the sequential case base, specifically: the probability of occurrence of each sequential case is updated as well as the probability of occurrence of each alert in the sequential cases that have been used in the solution. Those sequential cases whose probability of occurring together is above the probability of occurring separately are merged together in a new sequential case. Other features of intervening sequential cases such as risk or cost can also be updated in this process.

Moreover, we have introduced a formal framework for the evaluation of alert triage that allows us to adjust the different knobs and switches provided by our model and make practical choices when assessing different components of alert triage. This framework allows us to measure the performance of alert triage in ideal, cost-based, and imprecise environments. Ideal environments do not contemplate misclassification costs and are valid at design time when we want to check new triage techniques and fix some kind of requirement (such as the maximum number of false positives allowable). In cost-based environments correct decision outcomes have associated a benefit whereas incorrect decision outcomes have associated a cost. These scenarios are valid to test alert triage systems in simulated environments and to determine their optimal decision threshold. Finally, in imprecise environments, misdetection costs not only will be unknown a priori but also will vary over time. These scenarios are useful for the evaluation of systems for real-world deployment.

In Chapter 7 of the thesis we describe a research prototype where we have instantiated the Ceaseless CBR model. Concretely, a first version of an agent-aided intrusion detection tool called Alba (Alert Barrage) that assists a network administrator's decision making [MP03a]. As we will see in Chapter 8 of the thesis, part of our future work is to operationalize the techniques proposed into a agent capable of long-term cumulative learning and entrusted with mission-critical decision-making tasks in dynamic, imprecise, and adversarial environments [MPR00a, Pfl02].

1.4 THE THESIS

Conceptually, this thesis is divided into three parts: (i) context; (ii) fundamentals and contributions; and (iii) conclusion and future work. Chapters 1 and 2 respectively introduce our approach and situate the thesis within the state of the art pinpointing the novelty and contributions of the work developed. Chapters 3, 4, and 5 discuss our contribution and are devoted, in this order, to explain Sequential Cases, Dynamic Sequence Similarity, and Ceaseless CBR. Chapter 6 explains the evaluation framework developed to measure the performance of our techniques which have been coded into a first prototype presented in Chapter 7. Chapter 8 concludes the thesis not without first pointing out future lines of work.

The thesis contains the following Chapters and Appendices.

Chapter 1 provided the reader with a comprehensive overview of the thesis. We first described the fundamentals that motivated this research and why this new model is required after a short introduction to Case-Based Reasoning and its pro and cons to deal with problems that require the on-line analysis of temporally-evolving sequences of observational data. Then, we established the main objective of this thesis: To enhance the CBR paradigm to solve situations that are expressed by means of unsegmented, noisy sequences of complex events that arrive continuously over time. In Section 1.2 we described the concrete application domain—alert triage in intrusion detection—where we have conducted an exploratory analysis of the techniques proposed and evaluated our contributions. We succinctly enumerated the problems that this domain poses and specified in detail the problems that the thesis tackles: triage, correlation, and segmentation. We briefly described how we have approached those problems given an overview of our new CBR model that we have called Ceaseless CBR. The Chapter concludes with a roadmap of how this document is organized.

Chapter 2 puts the work covered by the thesis into perspective. Firstly, we overview four of the main and unusual CBR issues that we deal with in this thesis and that were partially opened, most of them early in the 90s, by separate seminal works [BM88, Sha90, Red90, RS93a, Jac97]: problem descriptions without well-defined boundaries [Sha90] in Section 2.1, continuous performance [RS93a] in Section 2.2, representing time-extended situations [Jac97] in Section 2.3, and combining relevant pieces of multiple cases [BM88, Red90] in Section 2.4.

Then we provide background on some relevant issues for the fully comprehension of this document and review in summary form some additional works. In Section 2.5 we examine issues related with different case-based sequence analysis tasks: sequence generation, sequence recognition, sequence prediction, and sequence learning. In Section 2.6 we see how traditional knowledge-based troubleshooting techniques such those used by rule-based systems or model-based systems cannot precisely capture the dynamic complexity of large systems, and how CBR emerges as a suitable paradigm to do so [Lew93, Gup98]. We also overview some previous case-based explanation models that lay the groundwork of Ceaseless CBR. In Section 2.7 we review much of the relevant alert triage approaches in intrusion detection, the two unique case-based reasoning models for intrusion detection in the literature that we are aware of, and finally, in Section 2.8, we point out the lack of literature on the evaluation of alert triage systems and briefly overview two ROC analysis alternatives.

Chapter 3 introduces the knowledge structures and techniques that sustain sequential cases. We commence our exposition in Section 3.1 pointing out the relevance of hierachical knowledge structures, the unbalanced attention that compositional hierarchies have received with respect to taxonomic hierarchies in the AI literature, and the importance of partonomic knowledge for properly reasoning on composite or aggregated objects. In Section 3.2 we introduce the alert model that makes our approach independent of the alert device and define most of the terms needed through the rest of the thesis. In Section 3.4 we throughly describe the so-called *actionable trees*. Actionable trees are predictive compositional hierarchies that are useful for dealing with sequences of complex objects and reasoning in terms of partial sequences that continuously arrive over time. Finally, in Section 3.5 we overview several window models to compute aggregates of interest over a sequence of alerts.

Chapter 4 introduces a similarity measure to compare the sequences of alerts yielded by actionable trees (yields or episodes) against the current window of alerts in the alert stream. Our simple but powerful similarity measure has two outstanding characteristics: continuously adaption to data seen so far and promotion of rareness. We first overview in Section 4.1 some concepts about similarity. Then, we introduce in Section 4.2 the notion of sequence similarity and overview two approaches for its computation—*edit distances* and *alignments*. Then, in Section 4.3 we formally define our dynamic sequence similarity as a semi-global alignment and describe several of its components: (i) a dynamic subsumption scoring scheme based on the reciprocal of *Odds* that uses a taxonomic hierarchy to properly compute the score given to each pair of alerts; (ii) an operation of abduction that automatically injects an alert in arbitrary positions of the alert stream; (iii) a operation of neglection that ignores an alert at an arbitrary position in the alert stream; and (iv) a system of dynamic programming recurrence equations that returns the score of the optimal alignment between a suffix and a prefix of the respective sequences being compared. These two operations allow us to deal respectively with lost and spurious alerts

in the alert stream. Section 4.4 illustrates how our dynamic sequence similarity works using a simple example. Finally, in Section 4.5 we show how those components behave over time using our real-world data-sets.

Chapter 5 throughly describes Ceaseless CBR. Ceaseless CBR can be seen as a constructive situation awareness process governed ceaselessly by *observational data*, a *sequential case base*, and *case activations*. Such concepts are first described in Section 5.1. Then, Section 5.2 explains how Ceaseless Retrieve uses dynamic sequence similarity to retrieve similar sequential cases that subsume the sequence of alerts at hand and create *case activations* that anticipate as much as possible all the situations that might likely occur. Section 5.3 describes how Ceaseless Reuse construct explanations—combinations of case activations that completely explain the sequence of alerts at hand—and provides a likelihood assessment for each explanation considering its likely consequences. We also see how to tradeoff risk versus efficiency estimating a degree of *urgency* that finally determines when the prioritization of each alert has to be done. Section 5.4 explains how Ceaseless Revise allows users to supervise the explanations provided by Ceaseless Reuse and guide its search towards the best explanations. Finally, Section 5.5 describes how Ceaseless Retain constantly updates the sequential case base with the revised sequential cases and with the frequencies of alerts seen so far.

Chapter 6 is devoted to evaluation of the techniques proposed throughout the thesis. We will describe our experimental setting and evaluate Ceaseless CBR along five different dimensions: (i) performance using a new formal framework for the evaluation of alert triage (ii) capability of discovering new sequential cases; (iii) CPU time requirements; (iv) capability of predicting future alerts (or preemptive ratio); and last but not least important, alert load reduction achieved. This Chapter is divided into three sections. Section 6.1 overviews ROC analysis and exposes some well-known concepts in the context of detection systems. Section 6.2 describes the construction of a decision-theoretic framework for the evaluation of alert triage that is based on a combination of ROC analysis and computational geometry. We show how this framework not only allows one to select the best alert triage system but also to make practical choices when assessing different components of alert triage. We introduce *t-area* a new measure for computing the performance of non-parametric systems in ROC spaces. The performance of alert triage depends on the environment where the evaluation is carried out (i.e., whether misdetection costs are considered or not and whether are known beforehand or not). We contemplate three possible environments (scenarios) in increased order of uncertainty and therefore of complexity: *ideal environments*, *cost-based environments*, and *imprecise environments*. Finally, Section 6.3 explains the set of experiments that we have conducted using the data-sets described in Appendix A and draws some interesting conclusions on the performance achieved. Our evaluations demonstrate how a Ceaseless CBR-enhaced IDS system provides improvement in both the number of alerts that could be managed by a SSO and the speed with which they could be triaged.

Chapter 7 brings the discussion down to earth, describing the architecture of a first prototype of an autonomous agent called Alba tasked with alert triage that employs the new methods proposed throughout the thesis to assist a network administrator's decision making. Initially, Section 7.1 provides a big picture of Alba architecture. The underlying network sensors as well as the rest of the IDS machinery needed to support Alba is explained in Section 7.2. Alba's architecture is sustained by two main components. First, a domain-specific language called SOID (Simple Ontology for Intrusion Detection) that facilitates the representation of the environment and contextual information in which the IDS operates and enables the use of the automated problem solving paradigms proposed. Indeed, SOID constitutes an instantiation of the formal alert model proposed in Chapter 3 on top of the Noos knowledge representation language [AP96]. SOID is described in Section 7.3. Second, an overlay management system that wraps a conventional IDS up with the knowledge and reasoning capabilities to properly evaluate the extent of threats. We explain the different knobs and switches of this system and the technology used to construct it in Section 7.4.

Chapter 8 concludes by summarizing the contributions of the thesis, benefits of this research, limitations of the proposed approach, and speculates about profitable directions for future research. In Section 8.1 we will review the five major directions along which our research has evolved: (i) development of a representational structure of sequential cases that support part-of and temporal dependencies; (ii) specification of a dynamic sequence similarity; (iii) an efficient CBR inference that exploits sequential cases and probabilities; (iv) a validation framework for alert triage; and (v) a novel agent-aided intrusion detection tool. In Section 8.2, we succinctly describe three of the main lines for future work. Firstly, the extension of the basic components offered by our model to facilitate more elaborate problem solving tasks such adversarial plan recognition. Secondly, the evolution of our model to contemplate automated monitoring agents able to communicate and coordinate with each other, enabling in turn higher-level collaboration. That is to say, a multi-agent approach where Ceaseless CBR agents cooperate each other to solve more complex problems (e.g., problems that are inherently distributed) or to improve its individual competence and performance. Ceaseless CBR agents could share information at different levels: ranging from observational data or case activations to explanations and sequential cases. We have previously investigated techniques and mechanisms that lay the groundwork of this future line. For example, DistCBR and ColCBR [PAM96, PAM97, MPA99], conversation protocols [MPR00b], inter-agents [MPR98, MPR00a], etc. Thirdly, we will investigate how completely delegate problem determination tasks to the own computer system. That is to say, how to evolve our techniques to become part of autonomic problem determination tools with self-healing capabilities.Specifically, Ceaseless CBR could be suitable to perform the analysis of sensed event data within an autonomic

manager control loop architecture as well as a symptom service able to compile cases that indicate problems and their possible causes and remedies.

Appendix A describes the Rustoord, Naxpot, and Huckleberry data-sets that we created to evaluate the performance of our techniques. In total we compiled almost 500000 alerts in more than 15 months of continuous observation in three different real scenarios. We also show the results of some interesting tests over the data-sets such as checking that they fit Zipf's Law.

Appendix B gives a listing of the most common acronyms used throughout the thesis.

1.5 A WORD ABOUT NOTATION

Throughout this document we use the following typographic conventions:

Italic	is used for terms that will be defined later on.
Sanserif	is used for domain application specific names or application names
$\vec{x}$	represents multi-attribute or complex objects.
$\vec{S}$	is used for representing sequences.
$[\cdots,\cdots]$	sequences are also represented using square-bracketed lists of comma-separated elements.
$\vec{S}_j$	denotes the jth of sequence $\vec{S}$.
$\vec{S}[j]$	also denotes the jth of sequence $\vec{S}$.
$\{\cdots,\cdots\}$	sets are denoted by a comma-separated list of elements between bracets.
$\langle,\rangle$	tuples are denoted by angle-bracketed lists of comma-separated elements. Specific elements of a tuple are denoted using dots "." (e.g., A.f denotes element f in tuple A).
$\|A\|$	denotes the length of A, if A is a sequence or its cardinality, if it is a set.

The purpose of this introductory Chapter was to provide the reader with an extended overview of the work developed. Next Chapter of the thesis will place our contribution within the context of related work.

2

State of the Art

This Chapter puts the work covered by the thesis into perspective. We divide the analysis of previous works around eight categories: *noisy case boundaries*, *continuous performance*, *time-extended situations*, *compositional cases*, *case-based sequence analysis*, *case-based troubleshooting*, and *intrusion detection*.

Firstly, we overview four of the main and unusual CBR issues that we deal with in this thesis and that were partially opened, most of them early in the 90s, by separate seminal works [BM88, Sha90, Red90, RS93a, Jac97]: problem descriptions without well-defined boundaries [Sha90] in Section 2.1, continuous performance [RS93a] in Section 2.2, representing time-extended situations [Jac97] in Section 2.3, and combining relevant pieces of multiple cases [BM88, Red90] in Section 2.4. Then we provide background on some relevant issues for the fully comprehension of this document and review in summary form some additional works. In Section 2.5 we examine issues related with different case-based sequence analysis tasks: sequence generation, sequence recognition, sequence prediction, and sequence learning. Our problem domain, intrusion detection, can be considered as a special case for troubleshooting systems where the cause of the problems are due to the confluence of two factors: the presence of a number of vulnerabilities or exposures in a computer network and the presence of an external agent (human, artificial or a combination of both) with malign intentions that exploit such vulnerabilities or exposures. Thus, in Section 2.6 we see how traditional knowledge-based troubleshooting techniques such those used by rule-based systems or model-based systems cannot precisely capture the dynamic complexity of large systems, and how CBR emerges as a suitable paradigm to do so [Lew93, Gup98]. We also overview some previous case-based explanation models and see how Ceaseless CBR can also be seen as a reminding-based explanation model that facilitates the generation of plausible explanations in dynamic, imprecise, and

adversarial environments when situations are given by unsegmented sequences of observational data stemming from multiple coincidental sources [Lea95]. In Section 2.7 we review much of the relevant alert triage approaches in intrusion detection and the two unique case-based reasoning models for intrusion detection in the literature that we are aware of. Finally, in Section 2.8 we point out the lack of literature on the evaluation of alert triage systems and briefly overview two ROC analysis alternatives. ROC analysis lays the groundwork of the formal framework that we introduce in Chapter 6 for the evaluation of these systems.

2.1 NOISY CASE BOUNDARIES

Shavlik was the first to notice that most CBR systems usually presuppose *well-defined current situations*—situations where the boundaries of the current case are cleanly defined [Sha90]. Shavlik proposed a case-based approach to gene finding that worked in the presence of noisy boundaries in DNA sequences which have no absolute START codon[1]. When the case at hand is not well-delimited, traditional partial matching algorithms for gene finding used to fail to recover previous cases. Thus, Shavlik introduced a robust algorithm, FIND-IT, based on the BLAST similarity-search program [MMA90] that was able to produce multiple, partial matches and then combining a selection of them into a consistent whole after detecting and correcting errors. This process coincides, generally speaking, with the process performed by Ceaseless CBR to find out the best explanation. FIND-IT was robust in the presence of errors in the input data as well as in the case library. Gene finding has four particularities that alleviate some of the complexities that we are dealing with in intrusion detection alert triage. First, the presence of end-markers known as STOP codons[2] facilitates, in some sense, the segmentation of whatever sequence—i.e., given a sequence of DNA all the open-reading frames (ORFs)[3] in the sequence can be collected. Second, a reduced reading frame—i.e., there are only three possible reading frames for each nucleotide in a sequence. Third, the alphabet to conform sequences is really small—i.e., four nucleotides and 20 amino acids. Forth, each element in the sequence is a simple symbol, and, fifth, the complete sequence to analyze is available beforehand. In intrusion detection there no exists an alert that indicates the end of an attack. The duration of an attack is undefined and therefore an attack can result in a number of alerts that are detected with minutes, hours, or even days of separation. Furthermore there are interfering alerts in between stemming from other attacks, malfunctions, etc. Moreover, in our case, the elements that compound a sequence are only partially available given that they are collected over time. For these reasons, we have to operate incrementally and choose a convenient window model—i.e., landmark, sliding, damped, etc—for aggregate monitoring. Finally, the alphabet we deal with is pretty

[1] Three-letter strings from the alphabet of nucleotids{A,G,T,C}.
[2] A STOP codon tells the cell to stop translating the DNA.
[3] An ORF is the segment of DNA between two consecutive STOP codons.

much larger (e.g., the SNORT IDS had around 1700 possible alerts as of our first prototype was done) and each element in the sequence is a complex object made up of qualitative, quantitative, and structured data.

Other CBR approaches are able to handle partial problem descriptions. Conversational CBR, which has been successfully applied to interactive help-desk, [AMB98, ABM01] also assumes partial rather than complete problem descriptions. A conversational CBR system engages an interactive dialogue with the user to complete the problem description. Problem descriptions are provided in natural language text and cases provide a text to be matched against problem descriptions and a collection of questions. Conversational CBR constantly elicits refinements of this description and suggests solutions based on a interactive dialogue with the user using the collection of questions of the best matched cases. In our approach, however, descriptions are provided by automate processes therefore instead of dialoguing with the user (or autonomously gathering information to continue inference [CYA99]) we have to wait until new events (evidence) occur.

2.2 CONTINUOUS REPRESENTATION, PERFORMANCE, ADAPTATION, AND LEARNING

Ram and Santamaría observed with much truth that CBR is mostly deployed as a high-level problem solving paradigm where situations are represented using discrete and static symbolic representations [RAM92, RS93a, RS93b]. They investigated the use of CBR in problem domains that required a continuous representation as well as continuous on-line performance such as autonomous robotic navigation systems. As we see below, our problem domain does not require a continuous representation but we are obligated to provide continuous performance, adaptation and learning. In robotic navigation systems, a navigation module is responsible for moving a robot through an intricate path among physical objects. In order to attain its objective the robot has to go from a starting location to a desired place avoiding such obstacles. The robot uses a schema-based reactive control module [Ark89] composed of a collection of parameterizable schemas that represent individual motor behaviors. The navigation module can instantiate and combine those schemas to provide more elaborate behaviors. The objective of CBR is to guide the reactive control module over time. This implies the on-line selection and modification of the schema parameters based on the current environment. They proposed a novel problem-solving and learning method that operates continuously (without stopping and thinking) and allowed the robot to adapt to novel environments and to learn from its experience. Basically, in these systems perception and action are used to explore the environment and detect regularities while learning generalizes those regularities into cases that are adapted to provide recommendations on the future consequences of actions. Ram and Santamaría proposed to perform both tasks concurrently and that case representations were updated incrementally through experiences The so-called Continuous

CBR method was characterized by combining three "continuous" aspects within an integrated framework:

Continuous Representations Both situations and cases are described using a time-varying series of real values. As noted above, in our approach we do not have a continuous representation instead the input is a temporally-evolving sequence of complex objects.

Continuous Performance The method constantly executes the best short-term actions available and continuously evaluate their outcome. This capability is essential to handle problem domains that demand continuous actions such as driving a car. In Ceaseless CBR we also require continuous performance however we only provide an explanation when a measure of urgency requires so.

Continuous Adaptation and Learning The method provides on-line adaptation and learning from experiences. This allows a robot to incrementally compile new knowledge to conveniently adapt its actions to new environments. This is required given the variety and difficulty of environments which a robot could cope with. Likewise, Ceaseless CBR constantly updates its sequential case base. Moreover, an additional off-line process allows Ceaseless CBR to compound and synthesize sequential cases that recur.

In Continuous CBR, cases are high-level representations of system-environment interactions from low-level sensory motor representations. A case is pair composed of a *sensory input vector* of analog values that represents the environment and *sensory output vector* of analog values that represents the actions taken. A sensory vector is a time history of real parametric values of perceptual or control parameters. Thus a case embodies a behavioral situation that reflects the interaction of the system with a particular environment. Said differently, cases represent observed regularities between particular environmental configurations and the outcomes of different actions. Cases are used to predict future consequences of actions and to consequently prescribe the values of the schema parameters within the reactive control module. In Ceaseless CBR a case is made up of the a set of sequences of alerts structured in a compositional hierarchy and the priority assigned to such alerts. Although we could easily have automated some basic responses, our first prototype (see Chapter 7 of the thesis) does not performs actions by itself, it only suggests them. The system administrator is who takes charge of performing the actions to solve the current situation. In Ceaseless CBR, sequential cases also aim at predicting future actions of an attacker and therefore to anticipate undesired situations.

Ram et al devised a self-improving navigation method that used reactive control augmented with Continuous CBR and developed two systems using that method [RAM92, RS93a, RS93b]. We briefly overview both methods in the following.

ACBARRS (A Case-Based Reactive Robotic System) The first system was endowed with a collection of hand-crafted cases and the ability to adapt them [RAM92]. Cases represented *behavior assemblages*. That is to say, collection of cooperating behaviors for standard complex environments that can be adapted to

guide the robot in novel situations. Moreover, a noticeable novelty of their approach was that cases were not only used to propose the behavior(s) but also a set of *behavior adaptations* (i.e., they not only propose a solution but also determine how the solution have to be adapted). The system was able to modify its behavior according to the most recent past experience (behavior adaptation) or select new assemblages of behaviors based on the current environment (behavior switching or selection). Behavior adaptation implied to perform global modifications on the reactive control module and radically adapt to new sudden changes in the environment whereas behavior adaptation only implied local modifications. In Ceaseless CBR, sequential cases also modify the behavior of the system since they are reused to compute the urgency with which the system has to provide an explanation. Their experiments shown that ACBARRS was very robust and with a good performance in unknown environments [RAM92]. However, using a fixed library of hand-coded cases impeded the system to improve its behavior through experience.

SINS (Self-Improving Navigation System) The second system in addition was able to autonomously construct representational structures using a hybrid CBR and reinforcement learning method [RS93b]. SINS has the same components and properties that ACBARRS except for that it relies on learnt cases rather than on hand-crafted cases. In other words, SINS was able to learn and modify its own cases over time using experience. In both systems, each new situation is constantly matched against cases to determine the most similar case that is subsequently adapted to be used to guide navigaton. Nevertheless, SINS improved its performance over time learning how and when to tune the reactive control discerning between different environments and the adaptations requires by each of them. In order to do that SINS combined CBR and reinforcement learning (RL) [SSR96, TD97]. CBR facilitated on-line parameter adaptation whilst RL enabled on-line case learning and adapation. The outcomes of the system were continuously monitored what allowed the learning sub-module to update incrementally case representations through experience. The learning module monitored the system and incrementally modified the case representations to accommodate the changes (see [RS93b] for further details).

Ceaseless CBR is closely-related to Continuous CBR. Both methods need to provide a timely response to a time-varying situation (i.e., continuous on-line performance). While Continuous CBR practically operates in real-time Ceaseless CBR only aspires to work on a quasi-real time basis. This is due to the fact that we have to evaluate time-evolving sequences of complex objects as opposed to only vectors of analog values as Continuous CBR does. The input of Ceaseless CBR are unsegmented sequences of events dispersed over time and the task is to segment the sequence to provide the best explanation of the current situation and suggest an action. In Continuous CBR the current situation is given by a series of equally time-spaced real values and the task is to directly execute the actions. This allows a robot to explore all possible actions given an situation and to learn the best action for using it in future similar situations. In Continuous CBR an action is executed without considering its associ-

ated cost. In our problem domain, we simply cannot afford to explore all possible actions[4] and have to select the best explanation considering the associated costs. A drawback of Continuous CBR is that continuous cases are neither easily-interpretable by a human nor easy-to-integrate with higher-level reasoning and learning methods. In Ceaseless CBR cases are more intuitive and explanations could be easily used by higher-level reasoning methods or shared among distributed Ceaseless CBR agents as we will point out in Chapter 8 of the thesis.

An important issue to address in continuous domains that provide a flux of cases is how store and manage them efficiently. We did not mention above but Continuous CBR stored *abstract cases* rather than the entire time history of the parameters of each schema for each situation that the robot solves. Abstract cases are like *sequential master cases* in Ceaseless CBR. Sequential master cases are able to efficiently summarize a set of similar situations. When a new experience is ready to be stored into the sequential case base, it is incorporated into a sequential master case (if it exists) [Lew93, Lew95, DV95]. We only store sequential master cases and continously update their probabilities since it would not be feasible to store the whole sequence of complex objects. The same occurs in Continuous CBR where it is almost impossible to store the complete time series of real values over time. As we see below, our approach additionally provides other level of abstraction to cope with unknown situations. Some other works have summarized similar solutions using scripts [SHP96, Jac97]. The problem of the inefficiencies (in case-base space and retrieval time) caused by the huge number of cases that arise in continuous domains was also studied by Sánchez-Marrè et al [SCR99] in the domain of waste-water treatment plants [CMC00]. They proposed a measure of relevance based on L'Eixample distance [SCR98] in terms of which only relevant cases (represented by tables of attributes) were learnt and a lazy learning algorithm that dictated how new cases were stored. See Bergmann's work for a study of the influence of abstraction in CBR [BW96].

A number of works have also used case-base reasoning for continuous performance as part of a autonomous robot navigation system. For example, Kopeikina et al also represented cases that evolve over time to deal with time-constrained processes [KBL88]. Likhachev and Arkin [LA01] used a similar approach to Continuous CBR for the selection and modification of behavioral assemblage parameters in real-time. Langley and Pfleger developed a case-based approach to recognize and learn places [LP95]. They used a case base library to retain place descriptions (spatial knowledge) for a physical agent. They assumed a case-base scheme that allowed a physical agent to incrementally learn new place knowledge as long as it was sensing its environment [LP95].

As a conclusion of this Section we could say that Ram and Santamaría placed a number of questions having crucial relevance to our work such as "When are two experiences different enough to warrant consideration as independent cases?" or "What

[4]Imagine the above robot carrying nitroglycerin, it could have some trouble to explore every possible action if it needs to crash against each possible obstacle. Surely, it will be unlikely to finish its task.

is the scope of a single case?". We try to answer these questions in Chapter 3. In next Section we consider a number of approaches in other problem domains that have represented situations using continuous cases or at least have used representations that go beyond single *snapshot cases* as most of current CBR systems [JAS02].

2.3 TIME-EXTENDED SITUATIONS

Jacynski observed that most CBR approaches only cope with *instantaneous situations*[5] [Jac97] and only few CBR systems deal with *time-extended situations*[6] [RS93a, Nak94, Rou94, BC96, SHP96]. An instantaneous situation is a finite set of data that represents the state of the world at a particular point in time whereas a time-extend situation reflects the evolution of the world either through a continuum of instantaneous situations along a specific time line or through a sequence of events like Ceaseless CBR. Jacynski investigated time-extended situtations in the context of knowledge-management and was the first proposing a general framework for the management (representation and retrieval) of time-extended situations instead of a doman-specific solution [RS93a, BC96, SHP96]. Jacynski modeled the process under control by a number of variables whose evolving-values were represented by a collection of time series with incomplete and noisy data. The use of a channel for each different source under observation lies a major difference with our approach since we only consider a unique stream of data and part of the problem is precisely to discern the values that correspond to separate sources. He considered both equally-time spaced observations (sampled data) as well as event-driven observations. As we will experimentally show in Chapter 6, Ceaseless CBR is also resilient to different window models. Jacynski proposed an object-centered representation model with two well-separated knowledge structures: *records* and *cases*. Records are composed of the time series of all variables between an interval of time whereas cases refers to time-extended situations that define when the case is relevant. A time-extended situation is composed of a behavioral part and an optional instantaneous part. Behavioral parts refer in turn to a record at a precise reference date, define a collection of elementary behaviors and a set of temporal constraints among behaviors. Elementary behaviors have a starting and ending date expressed in terms of Allen's theory of temporal intervals [All83]. Instantaneous parts store the record context.

Jacynski distinguished three types of cases: *abstract*, *potential*, and *concrete*. Abstract cases are crafted by a human expert or constructed through automatic generalization. Abstract cases correspond to *sequential master cases* in Ceaseless CBR [DV95]. Potential cases represent templates made up of temporal constraints that guide the search of new possible cases. Potential cases correspond to *abstract cases* in Ceaseless CBR. Concrete cases represent an elementary experience in a time-extended situation. Concrete cases correspond to *case activations* in Ceaseless CBR

[5] Also known as snapshot cases [JAS02].
[6] Also known as time-dependent situations [JAS02].

as we see below. Jacynski proposed a case retrieval strategy based on those types of cases and on the following heuristic. It is more preferable to retrieve abstract cases than concrete cases that in turn are more preferable than potential cases. The reasons for this are (i) that an abstract case stems from domain knowledge; and (ii) that a concrete case has been reused at least once. This strategy clearly differ from our retrieval strategy as we noted below. This reusable framework, developed on top of the CBR*Tools library [JT97a] was applied to plan nutrition control assitance [Jac97] and to the prediction of user behavior for web navigation reusing past navigations of a group of users [JT97b, TJK99, Tro00]. Likewise, we provide an implementation of the Ceaseless CBR model on top of an object-centered language—Noos in our approach [AP96]. However, we do not separate the observation data (records) from knowledge represented in cases instead we divide the knowledge into a taxonomic hierarchy and a compositional hierarchy. The taxonomic hierarchy, developed on top of Noos, represents all the concepts that conform the knowledge domain. The compositional hierarchy is made up of all sequential cases that reflect and constrain how different alerts can be compounded together indicating the temporal (parallel or serial) relationships among them. Moreover, a likelihood model expresses for each sequential case how likely is it to occur completely given some of its parts. As mentioned above, we distinguish two kind of cases: *sequential master cases* and *sequential abstract cases*. Sequential master cases group together cases that can be represented using the same *actionable tree* and therefore are stored only once but constantly updated with the statistics that reflect their different activations and how likely they are to occur. Sequential abstract cases are constructed based on the informational order provided by the taxonomic hierarchy used to represent the alerts (symptom events) at the roots of the actionable tree. Sequential abstract cases are used as a back-up for the explanation of a group of unknown alerts and allow Ceaseless CBR to discover new sequential cases. Sequential abstract cases offer an explanation to unknown situations. Thus, when Ceaseless CBR can retrieve a sequential abstract case means that the current case-base cannot properly define the current situation and new sequential case has to be created. The priority associated to this new sequential case requires the operator's oversight. Thus, in our approach is preferable to retrieve sequential master cases rather than sequential abstract cases. In Ceaseless CBR a case activation is a sequential case that has been partially matched and represents a plausible hypothesis on the occurrence of the complete sequential case given the sequence of events observed so far. Case activations that have been partially abstracted can be seen as Jaczynski's potential cases.

Only a few additional CBR works have dealt with time-extended situations [JT97b, SU98, TJK99, JAS02] or considered dynamic situations [KS01]. The most noticeable being the work due to Jaere et al [JAS02] who introduced *temporal cases* as a method for representing time-dependent situations within a knowledge-intensive CBR framework. They provided a theoretical foundation for this method based on Allen's theory of temporal intervals [All83]. They extended the CREEK CBR system [Aam91] (see Section 2.6.2) with temporal cases for the prediction of faulty situations in oil well drilling (e.g., a drill string getting stuck in the drill-hole interrupting the drilling process). The system was able to either alerting or alarming depending

respectively on whether a predicted undesired situation could be avoided or not. Previously, Bichindaritz had also proposed MNAOMIA [BC96], a CBR system that also integrated Allen's formalism and organized its memory around hierarchies of trends learnt by incremental concept learning from cases. MNAOMIA was applied to the control (diagnosis, treatment planning, etc) of disorders in psychiatry. A significant difference of our approach is that we do consider only the order of occurrence between events rather than the exact time of occurrence of the events. This allows us to avoid the complexity of a ever-increasing number of temporal restrictions [JAS02]. Serial cases represent situations where a case necessarily occurs before another (i.e., its occurrence enables the occurrence of the other). Serial cases subsume the Allen's *before* and *meets* relations (i.e C_i is before C_j or C_i meets C_j) [All83]. Parallel cases subsume all possible Allen's relations since they only stipulate that two cases occur within a pre-fixed period of time in whatever order. In our problem domain the separation in time between to consecutive actions usually does not influence the final impact on the system (e.g., once an attacker has performed a reconnoissance he could launch an immediate attack just instants after or on the contrary wait and launch it two or three days later, the consequences would surely be the same). However, this issue is crucial for recognition purposes. For example, if we use small size windows to analyze the sequence of alerts, then sufficiently-separate alerts could deceive our system, since we could not be able to correlate them. A similar issue has not been addressed in the above approaches.

A number of domains have been the scenario chosen by other approaches to address temporal issues in CBR. Take as an example the following references. Autonomous robot navigation as we saw above [RS93a, LA01, LP95]. In process supervision, Rougegrez proposed REBECAS for process forecast [Rou94], Cortes et al developed DAI-DEPUR for problem determination in waste-water treatement plants [CMC00]. Colomer et al provided a the dynamic time warping algorithm improved with a qualitative representation of process variables by means of episodes for diagnosis purposes in a laboratory plant [CMG02]. Meléndez et al proposed FUTURA for electric load forecasting [MV03]. Schmidt et al presented TeCoMed to provide early warnings against forthcoming waves of even epidemics in trend prognoses for medical problems [SHP96, SG03]. Hansen combined CBR and fuzzy sets for weather prediction [Han00]. Zehraoui developed CASEP to enable the prediction of sequence of user's navigation actions in a web store [Zeh03]. Fdez-Riverola and Corchado provided a hybrid neuro-symbolic model used to predict the apparition of oceanic red tides [FC03b].

As a concluding remark for this Section, we summarize the differences between time-series analysis in other CBR approaches and ours. Time-series analysis in CBR has been commonly based on the three following assumptions about observations. First, a separate stream of observations is used for each sensor attached to the system being modeled [Her99]. Second, an observation is a numerical value, usually a real number. Only a few works have addressed time-series built on categorical or ordinal data with a serial structure [WH98]. Third, observations are equidistant over time (i.e., taken at regular intervals of time). Thus many models only apply to equally spaced time series (i.e. daily exchange rate or weekly rainfall). In any case, each observation

is provided together with the time at which it occurred [Her99]. Nevertheless, as we noticed in Chapter 1, there are real-world domains such as network-based sensor-driven monitoring [MR97, DPV02], international event sequence analysis [Sch95] or intrusion detection alert triage [CM02, GG01, MP03e, Esk02] where those assumptions cannot be sustained. In these domains, firstly, we cope with problems that model not equally time-spaced observations whose serial structure comprise a number of features, with values that may be numeric, qualitative, or structured [DPV02]. Secondly, these observations stem from several sensors at different sources, usually, parallelized through a single data stream. This implies the necessity of correlation techniques to discern observations that belong to the same sequential structure. Thirdly, the exact time at which events occur is not so important as the order in which they occur.

A singular particularity of our approach with respect to many of the works presented above is that we do not need to match the beginning of a sequence (or episode [CMG02]) for retrieving a potential case that explain the current situation and consequently engage inference. Representing sequential cases as compositional hierarchies also allows us to make not only forward predictions but also backward predictions to dealt with missing information. This characteristic in our approach is highly valuable considering that we count on an noisy and imprecise environments where for example sensors could fail to notify an alert or attackers continuously exploit new unknown vulnerabilities.

2.4 COMPOSITIONAL CASES

Relatively little attention has been spent on CBR systems that are able to combine relevant pieces of several past cases when solving a new problem [BM88, Red90, Hin92, Vel92, DK93, GAD94, PLL95, BA95, RA96, MPG96]. The pioneering works in this aspect correspond to Barletta et al [BM88] and Redmond [Red90]. These approaches proposed to store cases as separate pieces, or *snippets* [Kol88]—i.e., sub-cases sharing a set of conjuctive goals [Red90]. The representation of cases by means of snippets has several advantages as illustrated in the CELIA system, developed by Redmond [Red90]. In CELIA snippets were linked in order to preserve the structure of reasoning and indexed using both internal and global problem solving context. Barletta and Mark [BM88] stated that links are necessary to maintain coherence and consistency of actions. Each snippet stores three types of information [Red90]. Firstly, information about a goal and the information related to the pursuit of that goal. Namely, each snippet is organized around one particular goal. Secondly, information about the current problem solving context at the time the goal was initiated. This includes the initial problem description. Thirdly, information about the links to other related snippets. Each snippet is linked to the snippet for the goal that suggested it and to the snippets for the goals it suggests. Snippets facilitate the retrieval and identification of relevant sub-cases based on the goals of the problem at hand. Snippets can be accessed either directly, matching the current situation and the goal and context that a snippet stores, or sequentially, following links between snippets. One of the main drawbacks of snippets as defined by CELIA is that they have to be pre-computed

before being stored for further reuse. Sequential cases in our approach bear certain resemblance to snippets. However, sequential cases can be dynamically decomposed to provide a partial explanation or compounded on-the-fly to provide a composite explanation. An additional drawback of snippets arise due to the fact that they are indexed using the global problem solving context what may not be feasible in a multi-agent setting where agents only have a partial view of the global problem solving context as stated by Prasad [PLL95].

There are two interrelated tasks where CBR systems have been shown to be more applicable and successful when they use multiple cases [DK93, HKS95]: *case-based design* and *case-based planning* [MAN01]. A case-based designer provides a new design based on previous designs that were useful in the past. The objective of these systems ranges between two extremes: routinely configuring a new solution based on the parametric variation of existing designs [SGK92] and, on the contrary, creatively providing radical customized solutions [AMS98]. Hinrichs developed JULIA for autonomous design [Hin92]. JULIA addressed issues of case segmentation and indexing of case chunks but it was limited to represent a moderate numbers of relatively simple artifacts. For example, Alba, our research prototype, takes into account more than 1800 basic individual cases (as of this writing), each one corresponding to one of the alerts that a SNORT sensor can evoke providing a huge number of possible combinations. Sequential cases can be compounded together using both parallel or serial relations as we show in Chapter 3 of the thesis. Domeshek and Kolodner developed ARCHIE a case-based aid for conceptual design in architecture [DK93]. They addressed issues related to the complexity of cases and how they can be segmented into chunks for future reuse and how those chunks should be indexed for subsequent retrieval. Branting and Aha also discussed multiple case reuse by means of cases that were represented using several levels of abstraction [BA95]. Likewise, Smith and Keane proposed DÉJÀ VU a CBR system for plant-control software design that was able to compound a new solution using different parts of multiple cases at various levels of abstraction [SK96].

A case-based planner constructs a new plan based on previously generated plans (plan cases) saving considerable time over planning from scratch [MAN01]. A plan case is a sequence of actions to meet predefined specifications. These specifications or goals are decomposable into subsets of specifications or subgoals allowing cases to be decomposed into sub-cases. This hierarchical decomposition enables a case-base planner to constructs new plans in terms of sub-cases (subparts of multiple cases) increasing its versatileness due to greater number of possible combinations. Goel et al proposed a multi-strategy navigation planning system called ROUTER [GAD94]. ROUTER was also able to retain cases that previously have been broken up into small pieces. Veloso proposed PRODIGY/ANALOGY, a system that was also able to retrieve and subsequently combine the results of an arbitrary number of totally ordered plans [Vel92, Vel96]. Ram and Francis presented NICOLE-MPA, a multi-plan adaptor algorithm that allowed a case-based least-commitment planner to retrieve and adapt relevant pieces of multiple past experiences [RA96]. They used plan clippings that were similar in essence to snippets but that can be constructed

dynamically during problem solving rather than pre-computed beforehand as we mentioned above.

Macedo et al proposed the representation of structured plan cases as tree-like structures built from goals and actions [MPG96]. We refer to that representation as Hierarchical and Temporal Related Case Pieces (HTRCPs). HTRCPs are useful in structured planning domains. Macedo et al applied HTRCPs at music composition reformulating, previously, the musical composition task as a planning task [MPG96]. In HTRCPs each node corresponds to a case piece whose goal may be broken up into subgoals and represented by case pieces in its subnodes. Case pieces are made up of a set of attribute-value pairs describing its properties. Arcs represent causal justifications or explanations (hierarchical and temporal relationships) between case pieces. They distinguished six types of arcs depending on the relatioship induced between case pieces: implicit/explicit arcs, temporal/hierarchical arcs, and antecedent/consequent arcs. This classification in turn determines eight different types of case piece contexts—the neighboring case pieces: ranging from antecedent-hierarchical-implicit context to consequent-temporal-explicit context. Moreover, they use a pseudo-date scheme to represent time that is based on Allen's temporal intervals [All83]. Each case piece in a case plan is given a period of time and represented by a pseudo-date to facilitate the obtention of temporal relations. The likeliness between Macedo et al's HTRCPs and sequential cases in our approach is notable. The main difference comes from the fact that sequential cases represent plans from the point of view of the recognizer. That is to say, sequential cases are devised for the recognition of plans in mind whereas HTRCPs are useful for plan generation. Thus, HTRCPs lack predictive capabilities while sequential cases allows one to make inferences based on statistical regularities. We discuss the differences between case-based sequence recognition and case-based sequence generation later on in Section 2.5.

The combination of multiple sub-cases has also be addressed from a multi-agent perspective. In multi-agent scenarios, the problem at hand is solved compounding past cases that are logically or physically distributed and managed by different agents. Prasad et al [PLL95] presented a new model of response based on cooperative retrieval and composition of a case whose sub-cases are distributed across several agents in a multi-agent setting. In this way, the response to a query is derived after composing partial responses from distributed case bases. Since no single agent contains the complete information to respond to a query, piecing together related partial responses from disparate agents is necessary. However a case derived from the summation of best sub-cases may not give rise to a good overall case. Thus, to smooth inconsistencies they introduced the *Negotiation Retrieval* (NR) algorithm that allowed to cooperatively retrieve responses while negotiating compromises to solve conflicts. Schematically, a complex query is presented to several agents. Then each agent is responsible for retrieving information related to a part of the query. Afterwards, agents negotiate to piece together an acceptable response. Prasad et al viewed the NR algorithm as "an asynchronous parallel distributed constraint optimization search to obtain a good overall episode assembled from case pieces". This can be considered as the first work concerned with distributed case bases in a multi-agent setting. Plaza et al have investigated different cooperation and learning methods among dis-

tributed CBR agents [PAM97, MPA99, OP03]. Leake and Sooriamurthi have recently provided some insights related with the use of multiple case bases [LS01, LS03].

Up to now, in this Chapter, we have summarized previous CBR works that opened four of fundamental issues that we deal with. However, to the best of our knowledge those issues still remain open and nobody has seized them together. Thus, our approach is arguably different in essence from other major CBR proposals. Next Sections are devoted to provide background on some relevant issues for the fully comprehension of this document and review in summary form some additional works.

One of the hallmarks of human intelligence is having available in long-term associative memory a number of recurring sequential patterns that are useful for recalling past situations and making predictions when they are partially recognized. Hence, many common human activities ranging from language and reasoning to complex problem solving such as planning are based on previously learned sequences [SG01a, SG01b]. In next Section we will overview some of relevant works concerned with case-based sequence generation, recognition, prediction, and learning.

2.5 CASE-BASED SEQUENCE ANALYSIS

The automated, sequential and time series analysis of sensed data collected over time has been applied in many different disciplines under various guises and for different purposes. For example, discovery of black holes in large-scale astrophysical data [ZS03a], forecasting conflicts in international event analysis [Sch00], fraud detection in cellular telephones [FP97], intrusion detection [LB99], etc. We refer to the general task that underpins such multiple approaches and paradigms as *sequence analysis*[7]. Sequence analysis encompasses a collection of methods, techniques, and tools that aim at extracting meaning from data with a serial composition and decode it into an understandable structure for further reasoning. Said differently, sequence analysis's goal is to make sense of data with a sequential (temporal or spatial) structure. Several computer science disciplines have been very prolific and produced an outstanding number of methods, techniques, and tools for the automation of sequence analysis (e.g., natural language processing [MS99], computational biology [Gus97], machine learning [SG01a, Die02], speech recognition [Jel97], etc). The applicability, limitations, and complexity of sequence analysis depends on a number of factors mainly concerned with (i) the purpose of the analysis (e.g., recognizing if a sequence is legitimate, predicting elements of a sequence, discovering new sequential patterns, etc); (ii) how the analysis is performed (e.g., on-line, off-line, etc); (ii) the nature of the data itself (e.g., noisy, sparse, incomplete, numerical, categorical, structured, etc); and (iv) the characteristics of the environment (e.g., Markovian or non-Markovian, deterministic or stochastic, stationary or non-stationary, etc). Ceaseless CBR aims at

[7]We prefer to call such task sequence analysis rather than *sequence learning* since we consider that it also covers other aspects of the operationalization of the sequences learned and not only the process of learning (constructing) them [SG01a].

analyzing on-line unsegmented (noisy) sequences of complex objects that stem from multiple coincidental sources with the purpose of recognizing similar past situations and predicting undesired situations in plenty of time. These particular characteristics and the assumptions often required (e.g., *iid*, *Markovian*, *stationary*, etc) make infeasible the adoption of a large number of current sequence analysis methods, techniques, and tools. Next, we provide a concise characterization of several sequence analysis tasks according to the purpose of the analysis and concisely examine some case-based approaches to those tasks [SG01a, Die02]. Then we enumerate some existing and extended techniques and speculate on their appropriateness for our particular problem domain.

2.5.1 Case-Based Sequence Generation

The goal of sequence generation is to produce or continue a succession of elements whose order is crucial for the success of the task at hand. That is to say, given a sequence of length $j \geq 0$, generate a k more elements in the succession[8]:

$$[\vec{x}_1, \vec{x}_2, \cdots, \vec{x}_j] \rightarrow [\vec{x}_{j+1}, \cdots, \vec{x}_{j+k}]$$

The most striking example of case-based sequence generation is case-based planning[9] [BMV98, Spa01]. In fact *automated planning* is one of the problem-solving arenas where the CBR research community has been more prolific from the very beginning and where CBR has been successfully applied [SA77, Ham89, Sug95, MAN01, BM93, RA96, MPG96, BMV98, Spa01, DAN02]. As mentioned above, a case-based planner constructs a new plan—a sequence of actions to meet predefined specifications—based on previously generated plans saving considerable time over planning from scratch [MPG96, MAN01]. Nonetheless, as we will note in Chapter 8, case-based plan recognition—the case-based planning counterpart—has not deserved the same attention and has been less explored.

2.5.2 Case-Based Sequence Recognition

Sequence recognition aims at deciding whether a sequence conforms a known pattern or not. Said differently, given a sequence of complex objects of length n, the goal is to predict a single label y that classifies such sequence[10]:

$$[\vec{x}_1, \vec{x}_2, \cdots, \vec{x}_j] \rightarrow \text{yes or not}$$

As we noted in the above Sections, most CBR systems have dealt with instantaneous situations rather than with situations that require to recognize a temporally-

[8]When $k > 1$ the task is known as open-loop, otherwise it is called close-loop [SG01a].
[9]Sequence generation through actions is also referred as *sequential decision making* [SG01a].
[10]Notice that this prediction can be also stochastic rather than deterministic. As a matter of fact, sequence generation, recognition, and prediction can also be specified stochastically rather than using a deterministic formulation [SG01a, Die02].

evolving sequence of events. Ceaseless CBR, on the contrary, focuses on such situations. A high-level task that involves these kind of situations is *plan recognition*. Hence, we also consider Ceaseless CBR as the basic mechanism for the construction of *plan recognition* systems in multi-agent settings. We consider this one of the main frontiers to explore in the future such as we will point out in Chapter 8.

2.5.3 Case-Based Sequence Prediction

Sequence prediction (or time-series prediction) tries to foretell the next elements of a sequence. That is, given the preceding elements of a sequence this task attempts to predict the successive k elements[11]:

$$[\vec{x}_1, \vec{x}_2, \cdots, \vec{x}_j] \rightarrow [\vec{x}_{j+1}, \cdots, \vec{x}_{j+k}]$$

Notice that this task coincides with the sequence generation task. Hence, the different task formulations above can be transformed one into each other [SG01a]. Likewise, we can also turn the solution to one of these tasks into a solution for another [Die02]. Furthermore, in the real-world, most problems consist of combinations and alterations of these tasks. For example, intrusion detection aims at (i) recognizing suspicious sequence of actions and (ii) predicting the next steps of an attacker so that (iii) the proper sequence of countermeasures can be generated. As pointed out before, we only focus on the two first steps above. Ceaseless CBR attempts to recognize sequences of complex objects (alerts) that have been previously deemed of interest to the user (security manager) so that their priority can be assigned accordingly. In addition, in our problem domain, when the sequence of alerts corresponds to a malicious activity that could affect the operations of the network under surveillance the objective is to advert it in plenty of time. Many of the approaches that we have seen in the above Sections can be classified as case-based sequence prediction approaches [RS93a, Jac97, MV03, Zeh03].

2.5.4 Case-Based Sequence Analysis Tasks

Generally speaking, two interrelated tasks sustain almost every sequence analysis undertaking:

The creation or learning of sequence models that establish what sequences are recognizable or legitimate. For example, Hidden Markov Models (HMMs) [Rab89, Bil02], stochastic grammars (SGs) [Ney92], recurrent neural networks (RNNs) [Pea95], sparse n-grams [Esk02, Pfl02], sliding-window techniques [Die02], Conditional Random Fields (CRFs), [LMP01], sequential case-bases, etc. Several machine learning paradigms (supervised, unsupervised, reinforcement-based, etc) aim at constructing such sequence models (HMMs, RNNs, etc)

[11] When $k = 1$ we call it single-step prediction and when $k > 1$ multi-step prediction.

based on a number of independent and identically distributed (*iid*) training exemplars [HT01b, RN03]. For example, using the expectation-maximization (EM) algorithm [MK96], gradient-descent methods, clustering, etc. Let us take the example of HMMs. Hidden Markov Models (HMMs) provide a flexible statistical model that has been successfully applied more than any other technique for sequence generation, recognition, and prediction in multiple disciplines [Rab89, Bil02]. For example, intrusion detection [Lan00], forecasting international conflicts, [Sch00], predicting protein secondary structure [AHH93], model-based diagnosis [YKP00], text recognition [AB95], computer vision [RB91], etc. See Cappé's list of references for a complete bibliography [Cap01]. However, HMMs present several disadvantages that have to be considered above all when coping with dynamics environments (e.g., imagine a hand-written recognition system that not only had to cope with the variability introduced by each individual writer but also with a dynamic alphabet). The most common problems with HMMs are [Kad02]: (i) the large number of parameters required even when modeling a small number of states; (ii) the Markovian assumption and its secondary effects (e.g., the probability of staying a number of time-steps in a given state turns to decrease exponentially instead of linearly as required in many domains); (iii) HMMs only use positive data (i.e., the presence of exemplars of other classes does not decreases the probability of observation); (iv) the absence of a method to determine the number of states and transitions; and, as consequence of the above, (v) HMMs require a very large amount of data to be trained. Moreover, the concepts learnt by HMMs (transition and emission probabilities) are not easily-understandable by human experts and they are also hard-to-craft by human experts in certain domains that do not admit the statistical assumptions or where there is not enough statistical support to learn them.

The comparison of sequences to determine whether a given sequence is legitimate or not [Kru83]. The most extended technique for sequence comparison is dynamic programming. It appears at the core of many well-known algorithms (e.g., Viterbi algorithm [G 73], Levenshtein distance [Lev66], alignments [Gus97], time-warping algorithms [SK83], etc A few CBR approaches have used dynamic programming algorithms in the recognition of sequences. For example, to recognize static sequences in domains such gene finding and music [LZ99, CW03a, AGM03] or temporally-evolving sequences of data for diagnosis purposes in a laboratory plant [CMG02].

The main tendency of statistical learning has been to develop classifiers neglecting two important aspects of the real world: *structure* and *time*. Therefore, statistical classifiers have been developed focusing on individual objects with a fixed set of static attributes rather than focusing on the evolution of its attributes over time and its relationships with other objects in the domain. In real-world scenarios this is not often the case and focusing on "flat-data" (i.e., data consisting of identically-structured objects assumed to be *iid*) or using static attributes rather than temporally-evolving attributes may be out of question [TAK02, Kad02]. For example, generative models

(HMMs, SGs, etc) require very strict independence assumptions to achieve tractability and therefore are not recommendable for representing multiple interacting features or long-range dependencies [LMP01]. Conditional models (Markov Networks [Pea88], CRFs [LMP01], etc) overcome this shortcoming but their current training methods still present low convergence. Generally speaking, it could be said that although general-purpose probabilistic models are well understood and widely used they are still restricted to small "explicit" state spaces or require a pre-established number of states variables [FGS02]. The recent burst of works on (Dynamic) Probabilistic Relational Models (PRMs) announce a shift on the above machine learning tendency [FGK99, SDW03].

In real-world domains where the above statistical learning models are often hard to apply and where an automated system expects to cope with related problems that recur over its lifetime, the case-based paradigm elicits as an elegant approach to surmount the complexity barrier and facilitate a mechanism that allows an automated system to improve its performance with experience.

2.6 CASE-BASED TROUBLESHOOTING

Automate troubleshooting aims at minimizing the time and cost required to fix a system's problem or to recover a system from a faulty state. Automate troubleshooting can be seen as composed of three interlaced processes: *problem detection*, *root cause identification*, and *problem repair*. We consider the system under observation as a complex system made up of several components. The time and cost to fix a problem is the sum of the time and cost to detect the problem and determine its root causes (problem determination or fault diagnosis) and the time and cost required to execute the actions that can restore the system to a healthy state again (problem repair). The time and cost of problem determination depends on the time and cost required to isolate the problem (problem detection and root cause identification) proposing a number of competing hypotheses and the time and cost spent on testing these hypotheses (diagnostic tests) [KS95]. When the number of possible faulty components is large, an effective strategy in time and cost for both problem determination and problem repair becomes of paramount importance. A number of works in different problem domains have investigated methods for the selection of the optimal set of diagnostic tests in a cost-effective way [LP93, BRM02, Bay03]. We, in this thesis, are only interested in problem detection and root cause identification issues and leave the automation of diagnostic tests and problem repair issues for further work. Providing systems with autonomous problem determination and full remediation capabilities (i.e., self-healing systems) is an incipient area of research that constitutes the focus of our future work such as we will explain in Chapter 8 of the thesis.

As systems grow in number of components and complexity (e.g., in large-scale manufacturing and service equipment, thousands of electro-mechanical, hydraulic, and digital electronic components are assembled together to perform an intended function [Gup98]), knowledge-based (KB) troubleshooting tools are becoming significantly more necessary to identify the ever-increasing number of components

that could be responsible for a fault. However, traditional KB techniques such those used by rule-based systems [Cal89, MCV89, Mar89] or model-based systems [KW87, BMM93, JW93] are limited by their brittleness and their lack of flexibility [FGT91, Gup98]. These techniques cannot precisely capture the dynamic complexity of large systems [CKF02]. Said differently, those techniques are not resilient to continuous changes or manage easily the presence of noise, imprecise, or adversarial conditions. Moreover, they not only require experts to constantly input extensive knowledge about the systems but also continuous human oversight and the domain expertise of their end users. Lewis grouped the limitations of those techniques under the following four headings [Lew93, Lew95]: (i) the knowledge representation problem (i.e., the mismatch between a real-world task and its corresponding representational scheme accentuates due to constant changes in the environment); (ii) the knowledge acquisition problem (i.e., continuously compiling details of how problems are solved and articulating the reasoning that experts use becomes infeasible); (iii) the brittleness problem (i.e., those systems do not recover gracefully after confronting a problem that they were not designed to cope with); and (iv) the problem of learning and adapting in evolving domains. CBR emerges as a suitable paradigm to confront the dynamic complexity of large systems [Lew95, Gup98]. A number of automated troubleshooters have already shown the feasibility of CBR in complex and dynamics domains [Goo91, BL91, RR91, AW92, Sim92, RNC93, BH95, Lew93, Lew95, WTM95, RTT96, Gup98, PNM99, MT00]. For example, Lewis extended a Ticket Troubleshooting System (TTS) system with CBR methods that aided in computer network alarm management [Lew95], Gupta introduced SPOTLIGHT, a CBR tool for complex equipment troubleshooting [Gup98], and Ram et al presented a framework for introspective multi-strategy learning approach in the context of electronics assembly manufacturing through which a troubleshooter was able to introspect about its own performance and identify what it needed to learn to improve its own performance [RNC93]. They introduced *meta-explanation patterns* (Meta-XP) as causal, introspective explanation structures that explains how and why and agent reasons. They distinguished between *Trace Meta-XPs* that are able to record a declarative trace of the reasoning performed by a system and *Introspective Meta-XP* that facilitate the analysis of the declarative reasoning trace and help to explain the reasoning process and determine what to learn and how to learnt it [RNC93]. Furthermore, (case-based) explanation models have been largely studied by the CBR community [SKR94, Aam94, Lea95]. We overview some case-based explanation approaches in Section 2.6.2 and see how they lay the groundwork of a significant part of our approach. Nonetheless, as mentioned throughout this document, many of the CBR approaches above as well as the dominant mainstream CBR model [AP94] only contemplate instantaneous situations rather than time-extended situations that are only partially expressed in terms of unsegmented sequences of observational data stemming from multiple coincidental sources. Thus, as pointed out in Chapter 1 of the thesis, the problem determination issues that Ceaseless CBR deals with are significantly more challenging. In next Section we briefly describe a particular problem domain (network event management) that widely subsumes the problem domain we

cope with (alert triage in intrusion detection) showing the wide-spectrum of application that our approach could reach.

2.6.1 Network Event Management

Network event management is one of the problem domains where different research communities have been more prolific offering different troubleshooting approaches that aim at mitigating the ever-increasing number of possible faults [FGT91, LF93, BCF94, KS95, RH95, KYY95, YKM96, Mei97, SS01b, SS02a, SS02b, SS03].

With some necessary simplification, network event management systems could be classified according to the fault localization techniques that they employ:

Rule-Based Reasoning systems represent knowledge and expertise by means of rules [Lie89, Cal89, MCV89, Mar89, BMM93, LMY99, HSV99]. These systems are unable to automatically learn from experience. A number of approaches have proposed to automatically discover rules from databases of past alerts but discovered rules need to be constantly regenerated after small changes in system configuration [Kle99, SS01a, ZXL02].

Model-Based Reasoning system that represent both physical and logical components in the system by means of a model that describe the particular behavior of each component and the relationships with other components [KW87, FGT91, BMM93, JW93, Rie93, BLB99]. These systems look for discrepancies between the modeled and observed behavior of the system under supervision. Most times such discrepancies are originated due to noise in the observed behavior or to a wrong model. However most of these systems model neither noise nor wrong models. These systems as well as expert systems work well in small static settings but both types of systems suffer the drawbacks signaled above when applied to a large system in an environment of noise and uncertainty.

Fault-Propagation Model-Based Reasoning systems represent the relationship between faults and symptom events using a *dependency model* [BCF94, KS95, RH95, KYY95, YKM96, Gru98, HSV99, SS01b, SS02a, SS02b, SS03]. A dependency model is a causality graph that describes the relationships among problems and observable symptom events. Given a series of events, a dependency model is used to work the way back to their cause (e.g., a component failure, a perpetrated attack, etc) and determine which components (or attacks) might be responsible for the symptoms observed so far. A dependency model can be represented using different formalisms such as: context-free grammars, code-books, belief networks, or dependency graphs. We review some of these formalisms below.

Case-based Reasoning systems are able to make decisions based on their experience on past situations [Lew93, Lew95, PNM99, MT00]. CBR systems lessen the constant modeling efforts in dynamic enviroments and are able to learn new correlation patterns over time [Lew95]. CBR systems are able to adapt

past cases to solve outstanding situations updating their case-bases with the proposed solution for further reuse.

Ceaseless CBR is, in some sense, an hybrid approach given that we use a special dependency (fault-propagation) model (i.e., actionable trees) to represent sequential cases. This feature makes Ceaseless CBR particularly different from other approaches before. An exception is Breese and Heckerman's work [BH95]. We see the differences with their proposal later on in this Section. Next, we succinctly overview some of the systems referred above.

Katzela and Schwartz designed a collection of fault localization algorithms relying on dependency models that were able to find the best explanation of a set of alerts [KS95]. Their algorithms used a fixed set of alerts as input and defined finding the best explanation as the process of searching a set with (i) the minimum number of components that explain all the alerts; and (ii) the maximum probability that at least one of the components in it is a primary fault. Katzela and Schwartz shown that the problem of finding the best explanation of the received alerts is NP-complete translating the problem into a generalization of the set cover problem [KS95]. Ceaseless CBR also searches the best explanation but it does incrementally and therefore the greedy algorithms proposed by Katzela are not directly applicable. Given that alerts often result in a set of possible alarm sequences, Rouvellou and Hart proposed to model each fault by means of a probabilistic finite state machine (PFSM) [RH95]. PSFMs were automatically built from historic data associating sequence of alerts to fault occurrences. An PSFM output sequences correspond to the possible sequence of alerts that results from the corresponding fault. They designed an on-line correlation algorithm that was able to receive as input an interleaving of sequences of alerts stemming from multiple faults. Ceaseless CBR is very similar, in essence, to this approach. Although, actionable trees are not so powerful as PFSMs they are more easily-learnable. Moreover, Ceaseless CBR also learns new cases on-line. Rouvellou and Hart's alert model took into account the presence of noise what differed from other previous probabilistic approches [PR87, PR90, BCF94]. A number of works have also considered the occurrence of corrupted (lost and spurious) symptom events [KYY95, ZXL02, SS02a]. Ceaseless CBR is also resilient to a noisy sequence of alerts.

Kliger et al introduced a coding approach in which problems are viewed as messages generated by the system that are encoded using the sequence of alerts that they cause into a code-book [KYY95, YKM96]. Correlation is interpreted as decoding such alerts to identify the message [KYY95, YKM96]. This approach produced a substantial improvement in real-time correlation due to optimized symptom events sets and fast decoding mechanisms. Kliger et al modeled causal likelihood by means of a general framework defined as semi-ring that provided an abstract measure of causality that could be instantiated with distinct models *deterministic*, *temporal*, *fuzzy logic*,etc. We have followed a similar approach as the underlying mechanism to model part-of strength relationships in actionable trees as we will see in the next Chapter given the independence advantages that brings to do so [BMR97]. In domains where the components of the system and possible problems can be completely specified beforehand,

the codebook approach is general, scalable, and resilient to noise. However, this turns to be the downside of this approach since each code-book has to be pre-computed and stored beforehand. Thus when this approach is used to manage the events of current computer networks (where changes abound) or when all possible problems or components are unknown beforehand (as in intrusion detection), code-books need to be continuously updated with the corresponding overhead that this implies [KYY95]. Steinder and Sethi proposed the utilization of *positive information* (i.e., the lack of any disorder in some system components) to improve the accuracy of fault localization. They developed a non-deterministic fault-localization process resilient to the existence of spurious and/or lost alerts based on their previous research on applying belief networks to fault localization [SS01b, SS02a, SS02b, SS03]

However, not always it is feasible to obtain a complete characterization of every possible problem and provide a perfect model establishing all possible relationships between problems and symptom events. CBR approaches require less modeling effort and are not subjected to the list of problems mentioned above. The most relevant CBR approach in network management is due to Lewis [Lew93, Lew95]. He proposed to enhance Trouble Ticket Systems (TTSs) with CBR capabilities that aided in alert management. A TTS tracks a problem along is life cycle into a network fault management system [MT00]. A TTS forwards problems but does not analyze or solve them.That is a TTS is usually used to expedite a service request by sending a trouble ticket to right channel in the network operation [Lew95]. Trouble tickets are finally received by human repairers who try to determine an explanation and remedy for the trouble. Lewis proposed to transform a TTS into a CBR problem solver able to automatically manage alerts conveyed by trouble tickets. He proposed CRITTER, a network CBR Trouble Ticket System. CRITTER [Lew93, Lew95] was a precursor of many network management approaches including the small number of other CBR approaches that we have found in the literature [MT00]. CRITTER was inspired by five other CBR systems: PRISM—a case-based telex classifier [Goo91], CEBRUM—a case-based tool for large-scale manufacturing [BL91], CANASTA—a crash analysis troubleshooting assitant [RR91], SMART—a CBR call support system [AW92], and MASTER another CBR TTS system [DV95]. CRITTER was one out of five components that composed the architecture for network management proposed by Lewis [Lew95]. The rest of components were: a network management platform (NMP), a TTS, an automatic trouble ticket generator (ATTG), and a (human) network troubleshooter (NT). The workings of this architecture are as follows. Alerts are detected by the NMP, subsequently alerts are collected by the ATTG that is responsible for creating the corresponding trouble tickets in the TTS. Finally, alerts are analyzed by CRITTER that sends its output to the NT. CRITTER used master cases (i.e., case structures that subsume experiences with similar individual cases) and implemented diverse adaptation strategies such as *null adaptation, parameterized adaptation, critic-based adaptation.*

Null adaptation is used when the solution found in the retrieved cases can be directly mapped to new cases [Lew95]. A null adaptation strategy is a good way to start

the learning process when the system is initiating its operations and only relies on the seed case base [Lew95].

Adaptation by substitution is used when the solution is unworkable but substituting several pieces may make it workable. Therefore, adaptation by substitution strategy replaces several pieces of the solution of a retrieved case by new pieces.

Parameterized adaptation is based on the relationship between the attributes that describe a problem and the attributes of the corresponding solution and is useful when the relevant variables in a case are numerical, similar cases exists but the parameters does not scale, and we can infer the formula through which the numeric solution is computed from the values in the problem. The CBR system compares the problem at hand when the solution in a past case and uses the relation between both to derive a new solution [Lew95].

Critic-Based Adaptation is based on the troubleshooter's feedback on the proposed solution. Thus, the burden of adaptation lies on the troubleshooter who can add/modify adaptation rules. When the NT knows beforehand that a solution proposed by the system will not work then the NT tweaks manually the solution (so that it can be applied) and files a new case. This adaptation method requires a language to specify the differences between the new problem description and the problem description that was altered [Lew95] (e.g., specifying a different role).

Ceaseless CBR also uses master cases but opposed to Lewis' master cases we additionally store statistical information on the occurrence of the individual cases. Ceaseless CBR also implements the null adaptation and critic-based adaptation strategies allowing the user to tweak the priority corresponding to received alerts. Additionally, as we see in Chapter 5, Ceaseless also provides a compositional adaptation strategy that allows new sequential cases to be compounded based on small recurrent subcases. In Ceaseless CBR, sequential cases not only can be hand-crafted by an expert based on a number past problem solving experiences but also through continuous operation.

Melchiors and Tarauco extended the CINEMA TTS with CBR and created DUMBO [MT00] that was able to propose diagnostic actions and learn new relevant features that appeared in unpredicted situations. For further details on event management see Oshie and Kliger's report where a number of reasoning methods and architectures are contrasted [OK94]. Malheiros' thesis also provides a complete survey on alert correlation in communication networks [Mei97]. Malheiros proposed a correlation model based on a *recursive multi-focal correlation* principle that establishes how a network can be recursively partitioned into smaller sub-networks that facilitate to focalize the search of faulty components. Steinder and Sethi provided a comprehensive review of event correlation future challenges [SS01a].

The advent of communication networks during the last few years has made that event correlation systems gain special attention. In fact they are recently being transformed from mere tools for monitoring computer network exceptions to event management tools responsible of supervising businesses' complete supply chain. In other

words, their responsibilities are being expanded from low-level resource availability problems in the lower layers of network protocols to performance problems in business processes. A number of comercial event management systems are today available: ENTUITY, OPENVIEW, SMARTS INCHARGE (that implements the Kliger et al's coding approach [KYY95]), TIVOLI, SPECTRORX (that employs CBR), etc. However, these systems commonly not only lag behind the pace of network size and complexity growth but also behind the demanding requirements posed by huge amounts of distinct events that a single enterprises daily generates [Lew93]. Thus, investigating new techniques, such as Ceaseless CBR, that are able to make sense of large amount of distributed and heterogeneous events is, nowadays, of paramount importance for enterprise management. New approaches are starting to address problem determination in large and dynamic systems [CKF02].

Ceaseless CBR starts with any of a world observable symptom events and searches the best explanation, the combination of sequential cases that are most highly correlated with the origin of alerts received so far, under the belief that the attacks associated to these sequential cases are causing such alerts. The best explanation prioritizes most urgent alerts minimizing costs in terms of both network administrator's interventions and risk that would convey a wrong explanation. Thus, Ceaseless CBR can also be seen as a reminding-based explanation model that facilitates the generation of plausible explanations in dynamic, imprecise, and adversarial environments (i.e., domains that are complex and imperfectly understood [Lea95]) based on past explanations of similar situations. Next Subsection briefly overviews some case-based explanation models.

2.6.2 Case-Based Explanation

Explanation is making sense of evidence (i.e., finding causes for observed facts [CS94, Lea95]). Explanation and its imitation by computers has been largely studied within the AI community [SA77, SL89, SKR94, PR87, CS94, Aam94, Lea95, TD97]. Case-based explanation generates new explanations to current situation by reusing relevant past explanations that were useful in similar past situations instead of generating them starting from scratch [SKR94]. A number of works addressed different issues of the generation process of case-based explanations in complex domains with imperfect domain knowledge and incomplete information [SL89, Kas94, Lea94]. ABE [Kas94], ACCEPTER [Lea94], AQUA [Ram94], and SWALE [SL89], are four well-known case-based explanation systems. Owens investigated retrieval issues [Owe94], Leake analyzed how to evaluate explanations [Lea92, Lea94], and Kass how past explanations could be adapted [Lea95]. Subsequently, Leake analyzed the commitments and contributions of the case-based approach to explanation generation [Lea95]. Leake discussed models that used case-base reasoning to generate abductive explanations of anomalous events in everyday understanding guided by prior experience and current information needs. He identified six fundamental issues that used to characterized different approaches to abductive explanation: *the nature of explanatory chains*, *when to explain*, *what to explain*, *how explanations are generated*, *how*

to select the best explanation. We focus on each of these issues later on in Chapter 5 of the thesis.

Succinctly, the four most important differences between Leake's case-based explanation model and Ceaseless CBR are: (i) the case-based explanation model reflects plausible reasoning by means of reasoning chains supporting belief in a state to be explained like many other approaches [JJ94, CS94]. On the contrary, we follow an approach closer to Peng and Reggia's *parsimonious covering theory* [PR90]. For us, explanations are combinations of disorders (computer attacks) that provide a covering for a set of manifestations (alerts) according to the pre-established part-of links in the corresponding sequential cases [PR87, PR90]. As we explain below, *parsimonious covering theory* lays the groundwork of part of Ceaseless CBR explanatory process; (ii) the case-based model automatically generates an explanation whenever an *anomaly* arises during the understanding process. However, we have defined a measure of urgency that prompts explanation when possible accumulated risk is above a given threshold in similar way to Huang and Schachter's approach [HS97]; (iii) we not only select best candidates based on minimality criteria such as Occam's razor and judge plausibility using similarity-based methods as the case-based model, but also use application and environment-dependent measures such as risk; (iv) our model provides composite explanations since many concurrent *anomalies* could be causing different but coincidental problems at a time.

The three main advantages of case-based explanation model signaled by Leake are: (i) better candidate explanations since they are started from explanations that were supported by prior experience; (ii) that are created more efficiently than generating explanations from scratch; and (iii) more precise explanations that are likely to be useful since they have been retrieved and adapted focusing on system needs through an integrated process of generation and evaluation. Finally, the case-based model represents explanations as *explanation patterns* (XPs). XPs were introduced by Schank to encode causal schemas in his script-based theory of understanding [SA77, Sch86]. As stated by Aamodt, explanations in KBS has two distinct interpretations: (i) the explanation that a system produces for the benefit of the user; and (ii) the explanation that a system generates for itself during problem solving. As we show in Chapter 5 of the thesis, we adopt the second approach since explanations constructed in such a way are also susceptible of being transformed into good explanations for the user [Aam94].

An essential component for the generation of hypotheses in case-based explanation as well as in troubleshooting in general is *abduction*. Abduction was introduced by Peirce as an additional inference method to induction and deduction [Pei48]. Abduction or abductive inference generates plausible explanations (hypotheses) for the data at hand [PR90]. Abduction constitutes the basis of most diagnostic problem solving methods [PR90]. Abductive inference is best explained through the pattern of reasoning shown in Figure 2.1 [JJ94]. Abduction, or inference to the best explanation, is an inference process that goes from observed data to a hypothesis that best explains the observed data [JJ94]. Conclusions generated by abduction are only plausible rather than logically implied by the premises. There are several models of abductive explanation (e.g., the case-based explanation model reflects plausible reasoning by means

```
D is a collection of data (facts, observations, givens),
Hypotheses H explains D (would, if true, explain D),
No other hypothesis explains D as well as H does.

Therefore, H is probably correct.
```

Fig. 2.1 Abductive Reasoning Pattern [JJ94].

of explanatory chains rather than using deductive proofs that depend on additional abductive assumptions). However, many of these models are hard to apply to problems where imperfect knowledge abound and where reasoning resources are constrained. For further details see Streiter classification of interpretations of abduction [Str02]. In the context of adversarial conditions of intrusion detection systems, abductive inference generates new hypotheses that allows problem solving to follow up with a convinient reasoning [CAM02, GG01]. Ceaseless CBR uses abduction within two of its processes. In Ceaseless Retrieve, hypothesizing on the presence of alerts that have not been observed, and in Ceaseless Reuse hypothesizing on the completion of sequential cases that have been only partially observed.

Peng and Reggia introduced a formal theory of diagnostic inference named *parsimonious covering theory* that was extended to incorporate probability theory [PR90]. In a nutshell, parsimonious covering theory is able to formalize many imprecise and intuitive aspects of abduction providing a good theoretical foundation for automated diagnostic problem-solving [PR90]. Parsimonious covering theory distinguishes two kinds of entities: *manifestations* and *disorders*. Manifestations are directly observable whereas disorders are causes of manifestations. Domain specific knowledge is codified by means of causal networks that make explicit the causal associations among manifestations and disorders. Causal associations take the direction from disorders to manifestations in such a way that the presence of a manifestation evokes all its possible causative disorders (alternative or competing hypotheses). The inference mechanism of the parsimonious covering theory follows a sequential *hypothesize-and-test* cycle. Given a sequence of manifestations the "hypothesize phase" sequentially merges the sets of evoked disorders for each manifestation providing a set of alternative hypotheses or explanations. Each hypothesis in this set covers all manifestations analyzed so far and is *parsimonious*. The hypothesis that contains the minimum number of disorders is called the *minimum cover*. The test phase is performed by a question-answering process to explore additional manifestations that help to discriminate hypotheses. The hypothesize-and-test cycle continues manifestation-by-manifestation until all the manifestations in the sequence have been processed. The final hypotheses could refer to several disorders. The Ceaseless CBR inference process can be considered as a model of the inference process parsimonious covering theory. However, we deal with an infinite sequence of manifestations (alert stream) and a measure of urgency that prompts when to explain. Moreover, the test phase in Ceaseless CBR consists of simply waiting the arrival of new alerts. We see further details of these issues in Chapter 5 of the thesis.

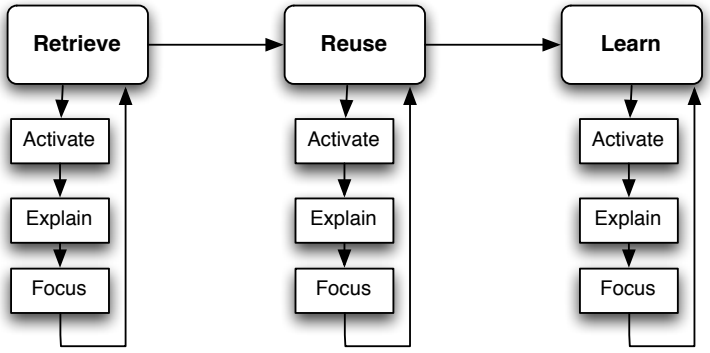


Fig. 2.2 Aamodt's Explanation Driven Model[Aam91]

Ceaseless CBR can also be seen as a *explanation-driven model* similar to the model used by the CREEK CBR system. CREEK addressed problem solving and learning in weak theory, open, changing domains [Aam91]. CREEK combined cases and generalization-based domain knowledge within a single semantic network. Later on, Aamodt specialized the CREEK architecture with an explanation engine, the so-called *explanation-driven* CBR model [Aam94]. This model is a general reasoning scheme that consists of three sequential tasks: *activate*, *explain*, and *focus*.

Activate takes as input the observed facts and returns a set of activated concepts of the semantic network. This process establishes the limits for further reasoning to take place within. Concepts are activated using the pre-established relationships in the semantic network or through links to past cases.

Explain takes as input a set of activated concepts of the semantic network and returns a set of hypotheses supported by explanations.

Focus takes as input those hypotheses that are supported by explanations that are above a given strength and returns the best explanation according to a collection of application and environment-dependent priorities (i.e., the least risky solution first).

This generic scheme is then instantiated for the retrieval, reuse, and learning processes as sketched in Figure 2.2. The Ceaseless CBR explanation model can be seen as an instantiation of Aamodt explanation scheme. However, as we see in Chapter 5 of the thesis, our approach differs operationally from Aamodt's model since the above explanation tasks are distributed along the Ceaseless CBR processes. In other words, Ceaseless Retrieve activates the knowledge required (sequential case activations) to represent the current situation, Ceaseless Reuse combines different sequential case activations to generate explanations of the current situation and proposes the best explanation, and Ceaseless Revise receives user's feedback on the best explanation

to constraint and focus further explanations. This cyclic process repeat ceaselessly as long as new alerts are received given that Ceaseless CBR deals with situations that are extended over time as opposed to CREEK that only deals with instantaneous situations. As we mentioned in Section 2.3, CREEK was subsequently extended to cope with time-extended situations by Jaere [JAS02]. Huang and Shachter introduced a general framerwork for designing alarms that aid in process monitoring specially devised for intensive-care units (ICUs) although it could be applied in may real world contexts. They defined an alarm as a subordinate agent that monitors an environment on behalf of a decision maker (DM) evoking alerts that request the DM's attention. An ideal alert is characterized as an action-based alert that communicates urgency They presented a decision-theorectic measure of the alert urgency. We have used a similar measure to determine when Ceaseless CBR has to provide an explanation.

Next we succinctly overview a troubleshooting approach that bears a striking resemblance in some aspects with Ceaseless CBR.

2.6.3 Decision Theoretic Case-Based Reasoning

Breese and Heckerman defined a decision-theorectic methodology for developing diagnosis and troubleshooting applications based on CBR [HBR94, BH95, HBR95]. They represented diagnostic cases by means of a specific belief network structure where nodes represented *issues*, *causes* and *symptoms*. An issue is a conflict among a set of causes. A cause is a contributing factor, activity or configuration item, and a symptom is a particular behavior caused by an issue. The case-base is modeled as a big belief network hand-crafted by an expert who also initially assesses Dirichlet distributions. Each issue, cause or symptom node stores a textual description. New problem descriptions are provided in plain text that is used by a probabilistic information-retrieval similarity metric [SAB94] to generate a collection of salient causes and symptoms. That collection is presented to the user. Using the user's indications a new belief network that models the current situation is created. The new network only represents the relevant parts of the problem at hand (i.e., a subset of the case-base) and is constructed using an efficient method for finding sets of d-separated nodes [GVP90]. The constructed belief network is then used to (i) compute the probabilities of distinct problems; (ii) to generate a recommendation for components repair; and (iii) to recommend further information-gathering actions based on cost analysis. Finally, the new case is retained updating the Dirichlet distributions corresponding to observed nodes whose parents were also observed. When new issues, causes, or symptoms are found then they are also inserted into the case-base.

This approach is similar in essence to Ceaseless CBR. However, a sequential case only stores part of the complete model for problem determination that helps to determine its plausibility given a collection of alerts. Since we store sequential cases individually, we avoid the on-the-fly construction for each new problem. Moreover, we consider a number of distinct problems (attacks) occurring coincidentally whereas they solved problems sequentially (one by one) supposing the occurrence of only one problem at a time. Their input is provided by an user and they use a myopic approximation (i.e., they presuppose that the user will make just one observation at

a time) whereas we received the input from an automated process and deal with a sequence of interlaced observations corresponding to simultaneous problems.

Belief networks constitute a mathematically sound formalism for probabilistic inference. A number of efficient algorithms and its successful and demonstrated applicability in different domains establishes an incentive for investigate its suitability for many other domains. However, belief network have to been hand-crafted by expert and usually require the intervention of more than an expert to properly model a concrete domain. In addition, they require constant expert oversight to properly model the domain in the long-term. When the environment is continuously changing this process obviously grows unmanegeable. As we show in Chapter 3, the inference model for belief updating of actionable trees is a special case of a *poly-tree model*. A poly-tree model or simple a poly-tree is a singly connected *Belief Network* (BN) (i.e., it contains no undirected loops). In other words, a BN whose underlying undirected graph has no cycles [Pea88, Dec98, KZ03]. The main particularity of poly-trees, and therefore of actionable trees, is that inference can be done in linear time. (i.e., the time and space complexity of exact inference in poly-trees is linear in the size of the network [RN03]). Therefore BELIEF ASSESSMENT, and the MOST-PROBABLE EXPLANATION (MEP) and MAXIMUM A-POSTERIORI HYPOTHESIS (MAP) tasks can be accomplished efficiently [Pea88, Dec98].

A few other works have also investigated the integration of case-based reasoning and decision-theoretic techniques under a unified framework [TCW97, GS01]. Tsatsoulis et al investigated how decision theory can help CBR deal with uncertainties in the problem domain and how CBR can help decision theory handle complex problems with a large number of variables [TCW97]. Gilboa and Schmeidler have provided a theory of case-based decisions based on the observation that people make decisions by analogies to past cases [GS01].

Intrusion detection can be considered as a special troubleshooting task where the symptom events are produced by information security devices (e.g., firewalls, IDSes, etc) due to a confluence of two factors: the presence of a number of vulnerabilities or exposures in our computer network and the presence of an external[12] agent (human, artificial or a combination of both) with malign intentions that exploit such vulnerabilities or exposures. The key point here is that we can not control such external factor as much as we can control an internal component (e.g., likewise a power delivery management system cannot control a lightning strike that completely burns one of its components and probably its consequent thunderbolt effects).

[12]Notice that by external we do not refer to the relationship with the target system (e.g., an employee or not) but an element whose actions are out of the control of the system.

2.7 CASE-BASED INTRUSION DETECTION

The main task of IDSes can be generalized as the monitoring of *computer activities* looking for deviations from normal behavior or *nonintrusive* behavior [DG01]. Computer activities can be monitored at different level of detail: system calls traces [KH97], operating system logs, audit trail records [ESB96], resources usage, network connections, etc. Therefore, nonintrusive behavior can be defined as the use of computer resources according to the corresponding security policy whereas whatever other behavior there is should be treated as *intrusive*. Spafford et al [SZ00] and previously Heady et al [HLM90] defined intrusion as "any set of actions that attempt to compromise the integrity, confidentiality, or availability of a resource", disregarding the success or failure of those actions. An IDS aims to discover intrusion attempts and whenever an intrusion (or intrusive behavior) is detected the IDS notifies to the proper authority by means of *alerts*. Usually, alerts take the form of emails, database or log entries, etc and their format and content vary according to the particular IDS. Notification to the proper authority can be considered as the basic response to an intrusion. On top of notification more sophisticated mechanisms to provide automatic response to intrusions can be developed. Our work is mainly centered around the alert management and notification tasks. Notice that there are several surveys and reports that cover, better than we could ever done here, many of the different techniques and tools that we mentioned throughout this Section [ACF99, Axe98, Jac99, JS00, NWY02, CL02].

Intrusion detection has been investigated for almost twenty five years since Anderson's first seminal article [And80]. A number of diverse techniques and systems have been developed along this time. Loosely speaking , these techniques have been traditionally classified according to two possible detection strategies: *anomaly-based detection* and *signature-based or misuse detection*. Anomaly-based detection prior to detection declares what is known as nonintrusive behavior whereas signature-based detection declares a priori what is known as intrusive behavior. To perform their task anomaly-based IDSes make use of profiles of normal behavior that have been learnt statistically whereas signature-based IDSes decide beforehand what behavior is intrusive using patterns of misuse that have been codified using an attack language [VEK00]. Both strategies for intrusion detection present several shortcomings. Mainly, signature-based detection can hardly detect unknown intrusions whereas anomaly-based detection is highly dependent on the enviroment in which profiles of nontintrusive behavior were learnt. It is widely accepted that intrusion detection can not be solved easily with a single strategy. As a matter of fact, hybrid approaches were early proposed [Axe98]. Nowadays, an assumed fact by the IDS research community is that there is no silver bullet against computer network intrusions (i.e., no single IDS is able to detect all possible attacks). Thus, the broad consensus reached by the IDS community is that the best solution is to design a perimeter defense with different layers of protection combining multiple prevention and detection devices and techniques. Currently, contrasting or correlating output coming from different IDSes constitutes the current goal of many network security endeavors. The fusion and correlation of information stemming from different infor-

mation security devices not only permits to decrease the number of false positives since many of them can be automatically disregarded but also increases the number of façades covered what diminishes in turn the number of false negatives. However, although high-level correlation systems are gaining popularity, more effective fusion and correlation techniques as well as evaluation techniques are still required most due to the ever-increasing sophistication of attacks in addition to the heterogeneity and volume of the information being managed.

Generally speaking, two kinds of components can be distinguished in the current state of the art IDSes [JS00, DW01]:

Probes are intrusion detection systems available either as commercial or in the public domain [DW01]. Probes compile information using host-based sensors as well as network-based sensors and evoke an alert whenever suspicious activity is detected. A couple of examples of probes could be: SNORT and RealSecure. Probes can be considered low-level sensors such as firewalls or integrity checkers [HRT03]. Probes usually do not produce a common output format as recommended by the Intrusion Detection Working Group (IDWG) what adds difficulties to the aggregation and correlation process and increases the complexity of ACCs [CD03, WE03, FMW03].

Aggregation and Correlation Components (ACC) ACCs or alert triage systems take as input alerts from probes or other ACCs covering complementary portions of the overall attack space [DW01]. An ACC after analyzing and correlating received alerts determines whether to send such alerts to the network administrator or not [GHH01a, HRT03]. Alba, our research prototype described in Chapter 7, can be seen as one of these components. An aggregation or correlation component or alert triage system emits a judgment on the malignancy of each alert on an IDS alert stream. Such judgment determines whether an alert is finally notified or not to the site security officer. Thus, the decision of notifying or not to the site security officer is finally made by the alert triage system. Ceaseless CBR aims at increasing the performance of such decisions. When ACCs are provided with a *mission* model then they can produce more savvy decisions. A mission model is a detailed description of the network under surveillance including the resources are vital and how different vulnerabilities could affect their operations [PFV02, HRT03].

One of the main obstacles for the rapid development of higher-level ACCs stems from the absence of a common ontology that not only impedes to deal with computer security incidents at a higher level of abstraction but also the collaboration among different intrusion detection systems. Next, we briefly describe some ACCs that appear in the literature.

2.7.1 Alert Correlation

Alert correlation can be defined as a conceptual interpretation of multiple alerts such that a new meaning is assigned to these alerts [JW93]. Although the major applica-

tion of alert correlation is problem determination, it has other purposes such as alert triage (or filtering), alert abstraction or generalization (i.e., creating a composite alert [LMY99]), or problem prediction [JW93]. In Section 2.6.1, we have seen several approaches to alert correlation in network management. Alert correlation has also application in diverse domains such as patient-care monitoring [HS97], air-traffic control, etc. Loosely speaking, alert correlation in intrusion detection do not differ so much from alert correlation in communication networks. As mentioned above, intrusion detection alerts stem from information security devices (e.g., firewalls, IDSes, etc) and are cause due to a confluence of two factors: the presence of a number of vulnerabilities or exposures in our computer network and the presence of an external agent (human, artificial or a combination of both) with malign intentions that exploit such vulnerabilities or exposures. The spatial and temporal aspects of alert correlation are similar in both domains. Jiang and Cybenko discussed several aspect of time and space correlation for network security [JC]. Previously, Mathonet et al did the same in the context of network troubleshooting [MCV89].

The correct interpretation of an IDS alert stream is an active area of research in the intrusion detection community [GHH01b, VS01, CM02, CAM02, MMD02, NCR02, PFV02]. We divide the exposition of relevant works in the following interrelated categories: *alert correlators*, *distributed IDSes*, *ontologies*, and *attack languages*. As mentioned above, for further details in many of these systems see the referred surveys [ACF99, Axe98, Jac99, JS00, NWY02, CL02].

2.7.1.1 Alert Correlators

Valdes et al extended EMERALD with probabilistic methods (based on Bayes inference) for sensor correlation [PN97, VS00] and demonstrated the performance (increased sensitivity and specificity) of sensor fusion [VS00, VS01]. Then, they adopted notions from the field of multi-sensor data fusion and examined situations involving heterogeneous sensors on diverse platforms [VS01, PFV02]. They introduced *expectation of similarity* as a way to correlate alerts that match closely but not perfectly. Our dynamic similarity, as we explain in Chapter 4 of the thesis, allows a similar capability using sequences of alerts instead of individual alerts. Porras et al proposed M-correlator, a prototype system, part of EMERALD [PN97], that correlates and priorizes alerts coming from multiple information security devices such as IDSes, firewalls, authentication services, etc [PFV02]. M-correlator filters an alert stream by assigning to each alert an *incident rank* that indicates the priority of such alert and its likelihood of success. The incident rank is based on a *relevance score* and a *priority calculation* that are computed taking into account the topology and operational objectives of the protected network and an *incident handling fact base*. The relevance score is computed comparing the topology of the protected network and the vulnerability requirements provided by the incident handling fact base whereas the priority calculation indicates the degree to which the alert is targeted at operational objectives and the security officer's interest for this alert type. Finally, M-correlator is also able to combine related alerts using an *alert clustering algorithm*. The topology database of the protected network and the incident-handling fact base constitutes

the main sources of knowledge of M-correlator. The topology database is automatically updated by means of Nmap [Fyo97, Hil01] and provides information about OS type and version, hardware type, network services, etc. The incident-handling fact base contains more than one thousand types of incidents compiled from Realsecure, SNORT, Checkpoint, and EMERALD. Such incidents are structured according to fields such as unique incident code, commercial off-the-shelf code, incident class, description, vulnerable OS and hardware, bound ports and application, cluster list, and references.

Debar and Wespi proposed an aggregation and correlation algorithm that formed groups of alerts notifying a small number of composite alerts rather than an unmanageable number of raw alerts [DW01]. Their algorithm, based on an unified alert data model, used rules to create correlationships between different alerts according to two different types of relationships: *duplicates* (i.e., common alerts provided by different sensors) and *consequences* (i.e., alerts that follow a causal or temporal chain and should occur together). They introduced the concept of situations to aggregate alerts that have certain characteristics in common (e.g., alerts with the same source, the same target and belonging to same alert class can be aggregated). Sequential cases, in our approach, are able to express the same relationships and situations that can been described in Debar and Wespi's approach.

Ning et al also proposed to focus on the logical steps behind attacks rather than on low-level attacks or anomalies [NC02, NCR02]. Ning et al's approach is mainly based on the observation that the objective of the first stages of a multi-stage attack is to *prepare* the way for the later stages [NCR02]. They establish the notions of *prerequesites*, *consequences*, *hyper-alert types*, and *hyper-alert correlation graph* as the means to represent the knowledge about attacks. A prerequesite of an attack is a necessary condition for the attack to be successful. The consequence of an attack is defined as the possible outcome of the attack. Prerequisites and consenquences are expressed in terms of predicates of a first order logic. A hyper-alert type is composed of a *fact*, a prerequisite, and consequences. The fact provides the alert information in terms of attribute-value pairs. Both prerequisites and consequences are expresed by means of formulae whose free variables are all in fact. A hyper-alert graph represents correlated alerts. It is a connected graph whose nodes are hyper-alerts and the directed edges represent the *prepare* relationship mentioned above. Based on these notions Ning et al introduced three utilities to interactively analyze large sets of correlated alerts [NCR02] *adjustable graph reduction*, *focused analysis*, and *graph decomposition*. The adjustable graph reduction utility allows an user to choose the degree to which the number of nodes and edges of a hyper-alert correlation graph should be reduced. The focused analysis utility allows an user to specify the hyper-alerts of interest. Finally, the graph decomposition permits hyper-alert correlation graphs to be decomposed in smaller graphs according to a cluster algorithm similar to the proposed by Porras et al and mentioned above [PFV02].

2.7.1.2 Ontologies for Intrusion Detection

Morin et al proposed M2D2, a formal model based on the B-Method [Abr96, MMD02]. The B-Method is an extension of the Z notation [Spi92] that covers the development part of the design process of safety-critical applications. Z is a formal method that uses predicate logic and set theory for the specification of discrete digital software and hardware systems proposed by Abrial [Abr96]. The M2D2 model considers four relevant sources of information to accomplish the alert correlation task: the monitored information system, vulnerabilities, security tools for the monitoring, and events. The monitored information system are modelized by Morin et al using a hyper-graph, where nodes are the network interfaces and edges are subsets of two kinds of interfaces: hosts and links. This model is an adoption of the TCP/IP network topology proposed by [Vig03]. As a main part of the monitored information system and for correlation purposes, they also modeled products—logical entities executed by hosts. Vulnerabilities are modelized and built from the ICAT CVE based vulnerability database (see Chapter 7 of the thesis for further details). Security tools are also modelized as part of the topology of the monitored system. Finally, the last source of information modelized is composed of events, alerts and scans. They modeled all events occurring in a network such as *IPevents*, *TCPevents*, *UDPevents*, *HTTPevents*. Events produced by security tools are modeled as *scans* whereas *alerts* are modeled as a kind of events that reflect the state of an IDS.

We introduced SOID, a Simple Ontology for Intrusion Detection, that allows our agent-aided intrusion detection tool called Alba (ALert BArrage) to reason about computer security incidents at a higher level of abstraction than most current intrusion detection systems do [MP03e]. We dentified four key sources of knowledge to be conceptualized (*networks*, *incidents*, *vulnerabilities*, and *alerts*) and built a separate ontology for each of them using the knowledge representation language Noos [AP96]. Finally, we merged those partial ontologies in a more global ontology that we have called SOID. SOID can be seen as a domain-specific language for intrusion detection alert triage that (i) facilitates the conceptualization of the environment in which the IDS probles operate; and (ii) enables the use of several automated problem solving paradigms such as Ceaseless CBR. We throughly describe SOID in Chapter 7 of the thesis.

M2D2 bears some resemblance with SOID. The most significative difference between both approaches is that M2D2 uses the B formal method to model the different sources of information for the alert management task whereas we are using a description logic like language. Goldman used CLASSIC, another description logic language [BBM89, GHH01b]. Formal methods, such as Z, constitute an excellent notation for formal specification and have a great deal of expressive power but are not well integrated with common development methods and are hard to apply to inerently inconsistent domains. On the other hand, knowledge representation languages like Noos or description logic languages are less formal. That is, they have lesser expresivity and lack semantics needed for extensive analysis. However, they are good for the development part of the design process and make the necessary jump to program code less problematic. The B-method tries to minimize the drawbacks of formal specification. Nevertheless, B is overkill for real world domains where expensive

verification takes months to write and even more to understand [13]. Thus, it is only recommendable for safety-critical applications. Therefore, for IDSes we think that for research purporses where a variety of different and novel methods have to be devised, our approach is good enough. For real world applications, once the applicability of novel methods have been shown, time and effort investments in models like M2D2 can have much more sense.

Different taxonomies of computer security incidents have been proposed [UP02]. An ontology centered on computer attacks was introduced in [GPK99]. That ontology provides a hierarchy of notions specifying a set of harmful actions in different levels of granularity—from high level intentions to low level actions. An ontology based on natural language processing notions is proposed in [RHT01] as the theoretical foundation to organize and unify the terminology and nomenclature of information security (in general). A target-centric ontology for intrusion detection based on the features of the target (system component, means of attack, consequences of attack and location of attack) has been proposed in [UP02]. IDMEF tries to establish a common data model for intrusions mainly based on XML [CD03, WE03]. Thus, has a limited capability to describe the relations among objects and requires each IDS to interpret the data model programmatically [UP02]. In [UP02] RDFS was proposed as an alternative to IDMEF and in [Noe01] an ontology for computer attacks was provided on top of DAML+OIL (now renamed OWL) [DCH02]. The representation we use in SOID is equivalent to OWL Lite [DCH02].

2.7.1.3 Distributed IDSes

Research interest in distributed IDSes have recently increased since these systems offer greater coverage and security and not are limited by scalability issues as centralized IDS. Three striking examples of this kind of systems are AAFID [SZ00], EMERALD [PN97], and GRIDS [SCC96]. Crosbie and Spafford were the first to propose autonomous agents in the context of intrusion detection. Their initial proposal evolved to become AAFID [SZ00]. Other works such as Cooperating Security Managers have proposed a multi-agent system to handle intrusions instead of only detecting them [WFP96]. However, in these works agents lack reasoning capabilities and are used for mere monitoring. More sophisticated agents with richer functionality were introduced by [GPK99]. It has been argued in [CHS00] that a collection of heterogenous software agents can reduce risks during the window of vulnerability introduced between when an intrusion is detected and the security manager can take an active role in the defense of the computer system.

2.7.1.4 Attack Languages

[13] These conclusions were derived from a synthesis of ideas shared in a personal communication with Dr. Richard V. Benjamins.

Attack description languages or attack languages for short provide an abstract set of concepts in terms of which computer attacks can be described. It can be said that every IDS has its own attack language in terms of which it codifies misuse signatures, statistics of normal behavior, or both depending on the method or mix of methods followed by the IDS. Thus, there is a number of diverse attack languages to describe known attacks. Moreover, the level of detail at which attacks are described depends again on the method followed by the IDS. Vigna et al [VEK00] characterized attack languages according to the following different classes: event languages, response languages, reporting languages, correlation languages, correlation languages, exploit languages and detection languages. In that classification, our approach can be seen as a correlation language based on a description logic-like language. The ever increasing frequency of new coordinated and multi-stage attacks motivates the need for better attack models [DLD02]. Doyle et al proposed a richer language that subsumes and extends the expressive capabilities of current attack languages [DSS00, DKL01]. The benefit of dealing with events at higher level of abstraction is twofold: (i) it allows irrelevant details to be removed; and (ii) the differences between heterogeneous system to be hidden. Abstraction-based intrusion detection proposed by Ning et al [NJW01] provided a framework for distributed attack specification and detection based on a set of autonomous but cooperative components and event abstraction. One of the most relevant advantages of generic signatures provided by Ning et al is the possibility of accomodating different variants of known attacks. Sequential cases share the same property.

Templeton et al proposed JIGSAW, a language for specifying models of multi-stage attacks [TL00]. This language allows one to describe more ellaborate scenarios since attacks are described in terms of (*abstract attack) concepts* instead of only events as other approaches. In JIGSAW, each stage that conforms a multi-stage attack is described in terms of abstract situations defined by concepts. Each concept specifies a set of capabilities that are required in order to occur. Capabilities describe the state that must be satisfied for a particular stage of an attack to occur. Thus, each concept requires a set of capabilities to occur. Once that capabilities are met, the concept provides additional capabilities that can be useful for other concepts to hold. A key concept in JIGSAW is that multiple events can provide equivalent capabilities. For instance, there are different tools by which an attacker can achieve his or her objective. JIGSAW models can be used to discover new attacks. This constitutes a fundamental advantage with respect to other previous approaches.

Ning et al's approach is a variation of JIGSAW [TL00, NCR02]. However, three interesting differences can be appreciated: (i) the utilities provided by Ning et al are interactive whereas JIGSAW does not present such capability; (ii) Ning et al allow the aggregation of alerts to be performed before and after correlation has taken place while JIGSAW does not allow correlation at any time; and (iii) Ning et al allow prerequisites to be partially satisfied whereas JIGSAW requires a complete match of prerequisites or required capabilities. Finally, SCYLLARUS, MIRADOR, and MAFTIA are other well-known previous projects that also addressed the reduction in the number of alerts that a human security officer has to handle.

Goldman et al [GHH01a] introduced SCYLLARUS, a prototype of an architecture for integrating multiple IDSes into an unified framework for overall security assessment. SCYLLARUS is mainly based on the intrusion reference model (IRM) in turn composed of a network entity/relationship database (NERD), security goal database, and event dictionary. NERD allows one to modelize the site's hardware and software. The security goal database allows site's objectives and security policy to be modelized whereas the event dictionary provides the terms to describe harmful and benign events. One of the sub-ontologies that underpins SOID is NERD.

Cuppens et al introduced, as part of the MIRADOR project [Cup01], a cooperative module for IDSes called CRIM [CM02]. This module provide functions for alert clustering, alert merging, alert correlation, and intention recognition. Alerts and attacks are modeled using a first order logic. Concretly, alerts are modeled using the IDMEF format, but for automatic reasoning purposes alerts are automatically translated into a set of logical facts of a first order logic. Attacks are specified using the LAMBDA attack language. The fields that define an attack are attack pre-condition, attack post-condition, attack scenario, detection scenario, and verification scenario. The pre-condition and post-condition fields of an attack have respectively a similar meaning to the prerequesites and consenquences of an attack within the Ning et al's approach mentioned above [NC02]. The scenario attack field expresses the set of events produced by an intruder when the attack occurs. The combination of events needed to detect an attack are specified in the detection scenario field. Finally, the verification scenario field indicates the combination of events that allow to check if the attack suceeds. To describe the system's state through pre-conditions and post-condtiont they use L_1, a language based on the logic of predicates with some restrictions. They only use conjunction, that is, they do not contemplate disjunction in the pre-conditions or post-conditions of an attack. Moreover, negation is only applied to predicates not to conjunctive expressions. Attack, detection, and verification scenarios are specidifed using event calculus algebra that allows operators such as sequential composition, parallel unconstrained execution, non deterministic choice, etc. They introduced semi-explicit correlaction as a method to automatically generate correlation rules that then are applied in an on-line process. Performing the analysis of attack descriptions specified in LAMBDA and trying to identify if an attack A contributes to the execution of another attack B. If this is the case then they formalize it using the post-contidion field of A and the pre-condition field of B through the following formula $post(A) \rightarrow pre(B)$. They establish the notions of direct (simple and general case) and indirect correlation based on the logical unification of pre-conditions and post-conditions of attacks. Based on these notions, they defined an offline correlation process that generates a set of correlation rules. Then these rules are applied to the alerts produced by the IDS following an online correlation process that aggregate alerts corresponding to the same attack scenario. Finally, an abdutive correlation process allows one to forecast the next step of an attack scenario. Michel et al introduced ADELE [MM01]. ADELE has many aspects in common with the LAMBDA language. The main difference between both approaches is that ADELE follows a procedural against the declarative approach followed by Lambda. ADELE was devised to allow all the aspects of an attack to be described in only one language

given that most attack languages center around one aspect of the attack. Adele provides a machine readable (XML-like syntax) model of attack. Attacks in ADELE are expressed from the point of view of both attacker and defender. Sequential cases only represent attacks from the point of view of the defender.

MAFTIA (Malicious and Accidental Fault Tolerance for Internet Applications) was an IST Programme RTD Research Project that studied the tolerance paradigm to construct dependable distributed systems. MAFTIA combined notions borrowed from the fault tolerance field and notions coming from security areas such intrusion detection and cryptographic protocols. In MAFTIA, concepts such faults, synchrony, topology and security goals have rigorously been formalized using CSP (Communicating Sequential Processes) [Ros98]. An important part of MAFTIA has been devoted to the creation of a taxonomy of intrusion detection systems and attacks [ACD01]. MAFTIA describes IDSes in terms of the capabilities that they have. In order to conceptualize the capabilities of IDSes, firstly an actvity taxonomy is created. This taxonomy allows one to unify the input of different IDSes. Moreover, using this taxonomy and the notion of activity assumptions and activity scope a taxonomy of attacks is established. Furthermore, MAFTIA describes harmful and harmless activities in terms of the IDS capabalities required for their detection. In this way, for example, an IDS can be evaluated indepently of its signature database.

In spite of the fact that intrusion detection is a task highly amenable to AI solutions and particularly to CBR approaches [Fra94], the number of approaches to intrusion detection using case-based reasoning techniques is really scarce [ESB96, SSY02, MP03d] as we see in next Section.

2.7.2 Case-Based Reasoning for Intrusion Detection

To the best of our knowledge, only a few case-based approaches to intrusion detection have been published [ESB96, SSY02]. The first case-based reasoning model for intrusion detection was published by Esmaili et al in 1996 [ESB96]. They argued that in spite of the fact that a number of intrusion detection experts systems (e.g., IDES [Lun90], NADIR [HHM90], MIDAS [SSH88], USTAT [Ilg93], etc) behave conveniently for recognizing single dangerous events or even sequences of events, rule-based expert systems present difficulties in acquiring and representing knowledge. Since their knowledge-bases can neither easily be created nor updated and they lack of flexibility for representing and recognizing minor variations of the same intrusion. Our reasons in favor of investigating CBR in these domains are based on a close argumentation (see Chapter 1).

Esmaili et al proposed a Case-Based Intrusion Detection System (CBIDS) whose input is the audit trail produced by an operating sytem[14] and whose output is a collection of countermeasure actions that the system performs based on the severity of the intrusion detected so far. A response table stores all the available countermeasures

[14]CBIDS used the Audit Event Log file generated by the C2-Basic Security Module, the audit collection tool provided by Sun-OS 4.1.3.

that the system could take depending on several conditions. They first constructed a taxonomic hierarchy of *action classes* and defined a *risk factor* for each action class. The action classes are formed grouping together commands in Unix that share the same semantic. For example, the action class FILE CREATION whose semantic is *creating new files* encompasses BSD Unix commands such as touch, vi, vim, jove, emacs, etc. Another example is the action class DUPLICATE whose semantic is *copying files* and whose elements are the commands cp, cat, more, less, zcat and their arguments origin, destine. Action classes are also provided with a category depending on various factors such as: whether they are explicitly forbidden by the security policy not, or whether they may pass through the protection mechanism or not. A *risk factor* (RF) represents the degree of risk associated with each possible action in the system. A RF provides a measure of the potential threat to the system that each action entails. A RF is computed in terms of each action category using a fuzzy-logic approach (i.e., a risk factor may take values ranging from very-low through middle to very-high).

In CBIDS cases store previously known intrusion scenarios. They used a case-base of 20 intrusion scenarios collected from several theses [Por92, Kum95]. Figure 2.3 shows an example of such scenarios that exploits a well-known mail vulnerability. In such attack scenario a sequence of five actions could allow an attacker to get root privileges. A past intrusion scenario is composed of a collection of alternative sequences of commands that resulted in the same unauthorized access. Two different alternatives to the attack scenario above are shown in Figure 2.3. Attacks can not only be altered using different actions but also changing the order of the actions. However, as we see below reasoning in terms of action class increases the number of false positives. In addition a case stores a *description* and an *example scenario*. A description provides a definition of the different action classes that intervene in the scenario together with their required values for arguments and option fields (e.g., for the last scenarios DUPLICATE, ACCESS-CONTROL, CREATE-FILE, MAIL, and EXECUTION). An example scenario is a concrete instantiation of the scenario with the corresponding commands. Precisely, what we have depicted in Figures 2.3 and 2.4 are three example scenarios. Scenarios are constructed using action classes since it guarantees the robustness and portability of the cases. A CBR engine constantly compares the n most recent commands in the sequence of commands that an user executes against the case-base. All matched cases are instantiated and kept into the working memory. A case is matched if it stores a sequence of action classes such that its first element subsumes one of the n most recent commands. This retrieval process is dynamic just like Ceaseless Retrieval but unlike traditional CBR systems that suppose full descriptions of problems. CBIDS continuously keeps a set of cases that match the current window in working memory and refreshes them upon the arrival of new audit record. This facet is similar to what we have called in our approach *sequential case activations*. In CBIDS, cases are subsequently ranked according to their *combined risk factor* (CRF) that is computed using the combination operation for *belief functions* introduced by MYCIN [BS84]. The case with the highest CRF determines the set of recommend actions (e.g., NO-ACTION, RAISE-WARNING-LEVEL, INFORM-ADMINISTRATOR, etc).

Mail Root Attack				
Step	**Command**	**Argument 1**	**Argument 2**	**Comment**
1.-	cp	/bin/csh	/usr/spool/mail/root	# assumes no root mail file
2.-	chmod	4755	/usr/spool/mail/root	#make setuid file
3.-	touch	x		# create empty file
4.-	mail	root	$<$ x	# mail the empty file to root
5.-	/usr/spool/mail/root			# execute setuid-to-root shell

Fig. 2.3 Mail Root CBIDS attack scenario [ESB96]. (1) The attacker firstly creates a copy of csh and renames it as root's mail file. (2) Then the attacker activates the setuid bit on the fake file. Subsequently, (3) the attacker creates an empty message and (4) sends it to root. if mail fails to reset the setuid bit of /usr/spool/mail/root before it sets its owner to root then a flaw arises and consequently (5) the attacker only needs to execute that file to escalate privileges to root.

Mail Root Attack Variant I	**Mail Root Attack Variant II**
cat /bin/csh /usr/spool/mail/root	cp /bin/csh /usr/spool/mail/root
chmod 4755 /usr/spool/mail/root	touch x
jove x	chmod 4755 /usr/spool/mail/root
elm root $<$ x	mail root $<$ x
exec /usr/spool/mail/root	/usr/spool/mail/root

Fig. 2.4 Mail Root CBIDS interchangeable attack scenarios [ESB96]. The first variant consists of an alternative sequence of commands with respect to the sequence of commands in Figure 2.3. The second variant is constructed swapping commands 2 and 3.

The main coincidences and differences between CBIDS and Ceaseless CBR are summarized in the following. In terms of case representation, we use a graphical model (actionable trees) to represent all the alternative sequences that conforms a case whereas CBIDS uses a simple set of sequences. In our approach, partially matched actionable trees provide a measure of confidence on the occurrence of the complete sequential case (following a pre-established likelihood model). CBIDS only uses a measure of risk to rank cases what surely results in a large number of false positives (i.e., a case whose risk is high but its evidence is too low does not need to be reported). The taxonomic hierarchy of Unix commands provided by CBIDS and the alert model that Ceaseless CBR uses share a similar purpose. As mentioned above, the abstraction provided by CBIDS action classes results in more robust and portable cases. However, such abstraction also has a negative effect in the performance of the system since the number of false positives could increase enormously. In our approach, *sequential abstract cases* as mentioned above in Section 2.4 are only used as a back-up when no other explanation is available. We consider this aspect indispensable in order to reduce the number of false positives. With respect to case matching and retrieval, CBIDS uses a simple retrieval mechanism based on the matching of one command out of a window containing the n most recent commands against the first element of one of the sequences of action classes in the intrusion scenario represented by each case. This mechanism is in fact an command-driven retrieval (see Section 3.5 for a discussion of different window models). That is to say, CBIDS operates on an command-by-command basis. Ceaseless CBR on the contrary matches the current window of alerts against the set of whole sequences that represents each sequential case and retrieves,

for example, sequential cases independently of whether the alert(s) that initiate(s) the sequence is/are matched or not. This is useful when considering noise, the possibility that the attacker uses a unknown variant to initiate the corresponding attack, or a sneaking attacker that hides part of his actions, etc. CBIDS uses a simple risk model to rank retrieved cases and subsequently select the case that is finally presented to the user or used to generate a convenient response. This model operates on case-by-case basis. That is, the case with the highest case is selected and then successively the second one, etc. Ceaseless CBR tries to find the best explanation for the complete set of pending alerts, alerts in the current window and alerts that were not explained in preceding windows. This is required given that we have to produce an effective triage which in turns requires explaining malicious, innocuous and false positive alerts. Moreover, Ceaseless CBR selects the final explanation not only considering the risk (computed as a combination of hostility, exposure, and cost) but also the possible outcomes of each explanation. Thus, we define a measure of utility that is computed for each explanation that Ceaseless CBR makes and a measure of urgency that obligates to Ceaseless CBR to make prompt decisions in those occasions that require so.

Other differences between CBIDS and Ceaseless CBR stem from the fact that CBIDS uses host-based commands directly as input whereas we use the alerts provided by network-based sensor. As a matter of fact, we work at a higher-level of abstraction and therefore our results depend in part on the quality of the associated network-based sensor. Finally, as Esmaili et al already pointed out a major difference between CBR models for intrusion detection (such as CBIDS and Ceaseless CBR) and CBR systems for other domains such as design, planning or diagnosis stem from the fact that intrusion detection demands to work on a real-time basis.

Recently Schwartz et al proposed to improve the capabilities of Snort IDS [Roe99]. (see Section 7.2) using a case-based approach [SSY02]. They re-implemented Snort treating each Snort rule as a case that was codified using XML. We have performed a similar translation of Snort rules into Noos lingo but using the intermediate format provided by ACID. Schwartz el al proposed a similarity measure based on a collection of distinct comparators for each feature (Snort rule properties) rather than using a complete match on all features as Snort. There are two main differences to our approach. First, we work at higher-level of abstraction using alerts provided by Snort as input and providing a priority as output whereas they used directly suspect network packets as input and provided alerts as output. They tried to substitute Snort detection engine with a reflective engine that used the comparators mentioned above. That is, they (apparently) used the Snort sniffer and its pre-processors but neither its detection engine nor its output postprocessors. Second, their approach was stateless since they assessed the danger of each suspect network packet (case) individually whereas our approach can be considered stateful and considers a whole sequence of alerts before determining the priority of alerts.

Next Section overviews some few works concerned with the evaluation of alert correlation and alternatives methods to ROC analysis that constitute the basic technique of the evaluation framework that we will propose in Chapter 6.

2.8 EVALUATION OF ALERT TRIAGE

The evaluation of intrusion detection systems (IDSes) has gained attention over the last years [DM02b, DCW99, GU01, LFG00, McH00]. First, the 1998 and 1999 DARPA evaluations conducted by the Massachusetts Institute of Technology (MIT) Lincoln Laboratory [LFG00, HLF01] and then Mchugh's critique [McH00] of such experiments have shown that this area needs much more research and experimentation before a framework for the evaluation of IDSes can be widely accepted. Moreover, in the industry, IDS benchmarks provide misleading results rather than useful independent evaluations what accentuates this need [DM02b, Ran01]. In Chapter 6 of the thesis we pursue a less ambitious objective. We intend at providing a framework for the evaluation of *alert triage*—just a cog in the intrusion detection machinery. As we have seen in Chapter 1 of the thesis, alert triage is the process of rapid and approximate prioritization for subsequent action of an IDS alert stream [MP01]. Alert triage is the main task to be performed by IDS modules that, as we have seen above, have been named differently in the literature such as alert correlators [NC02], d-boxes [SS98], analyzers [WE02], ACCs [DW01], assistants [MP03e], etc. Indepently of the name, the correct interpretation of an IDS alert stream constitutes an active area of research in the intrusion detection community [NC02, SS98, WE02, DW01, MP03e, MP03d, CM02, GHH01b, MMD02, NCR02, PFV02]. Nonetheless, to the best of our knowledge nobody has provided a formal method of analysis for the evaluation of this particular component of IDSes. As a matter of fact the first experimental validation of correlation systems was recently published [HRT03]. Up to this work researches had not previously compared the performance of correlation systems [HRT03]. An explanation for the lack of literature on the evaluation of these systems could be that up to now they did not reach the adequate level of maturity to assess the collective progress in the field [HRT03].

One of the most difficult issues when measuring the performance of an IDS is defining a meaningful measure unit. That is, in terms of what we count the number of times that the IDS makes the right or wrong decision. This was one of the main drawbacks of MIT Lincoln Laboratory experiments. It was impossible to say formally how many times an IDS was right or wrong and therefore the use of ROC analysis made no sense. In alert triage the unit is clear. It is "alert". That is, the performance of an alert triage system depends on the number of times that it disturbs the SSO unnecessarily and the number of times that it does not advert about malicious alerts.

Next, we briefly overview the use of ROC analysis in intrusion detection.

2.8.1 ROC Analysis in Intrusion Detection

The beginnings of ROC analysis in intrusion detection were misleading. ROC analysis was originally introduced in the evaluation of IDSes in the 1998 DARPA experiments carried out by MIT Lincoln Laboratory [LFG00, McH00]. MIT researchers performed a cost insensitive evaluation of DARPA funded IDSes. However their

investigations were based on some misconceptions of ROC analysis such as signaled in [McH00]:

1. They did not employ a comprehensive unit measure to properly construct confusion matrices.
2. The evaluation of parametric IDSes were performed without previously selecting an operating point for those IDSes.
3. The interpretation of the results did not use a standard ROC space.

Stolfo et al proposed a method based on cost metrics as an alternative to ROC analysis to study the performance of IDSes [SFL00]. However, in their evaluations they did not use all the valuable information provided by ROC analysis such as Gaffney and Ulvila demonstrated later [GU01]. To the best of our knowledge, Gaffney and Ulvila's work has been the soundest proposal of ROC analysis for the intrusion detection community [GU01]. They provide a method grounded in decision, cost and ROC analysis to compare the performance of intrusion detectors. However, they did not provide any method for the construction of the ROC curves that they analyzed given that as mentioned above it is hard to find a comprehensive unit of measure when a complete intrusion detection system is evaluated. Tanachaiwiwat et al proposed complete models for the assessment of risk in intrusion detection [THC02]. However, they use simple performance metrics instead of the more powerful analysis capabilities of ROC curves.

Next, we briefly overview two ROC alternatives. See [GTD01] for an additional comparison between ROC analysis and the Taguchi method for quality engineering.

2.8.2 ROC Alternatives

DET (Detection Error Tradeoff) curves were introduced in [MDK97] as a variant of ROC curves. The main difference with respect to ROC curves is that a DET curve shows the tradeoff between the two types of error involved in a detection task. A DET curve plots a performance curve showing the range of possible operating characteristics. As ROC curves the abcissa axis shows the false positive fraction however the ordinate axis shows the false negative fraction instead of the true positive fraction. Figure 2.5 depicts three ROC curves and their alternative representation using DET curves. A DET curve gives equal treatment to false negatives and false positives what produces quasi-linear plots. When the curves are straight lines means that the underlying likelihood distributions are normal. DET curves have been used in large vocabulary speech recognition tasks [MDK97].

Other alternative to ROC representation is to explicitly represent the cost of a detection system such as was proposed in [DH00]. An explicit representation allows one to directly seize the range of frequencies and the range of costs where a detection system surpass the others and to quantify how much better it is. Explicitly representing expected cost sets a cost space where the abcissa axis designates the probability-cost function for positive examples $PFC(C+)$ and the ordinate axis indicates the expected cost normalized (NEC) with respect to the cost of the worst detection system.

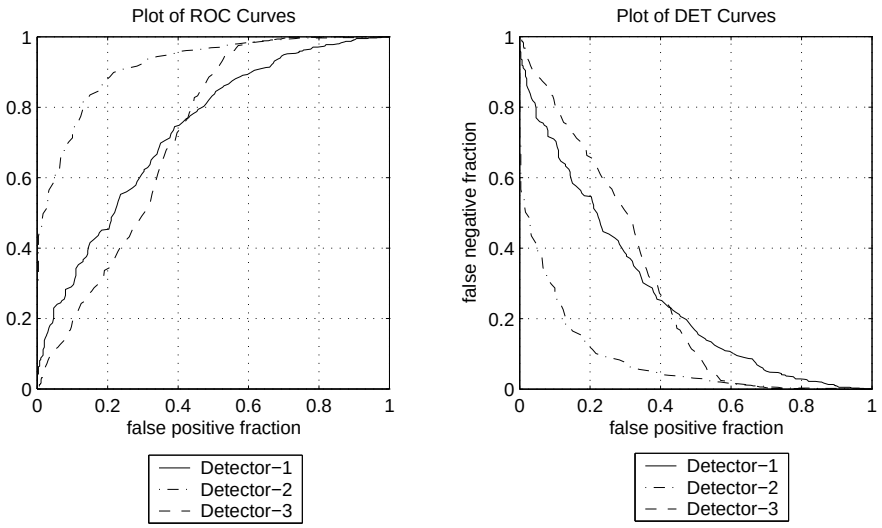


Fig. 2.5 ROC Curves and DET Curves Comparison. DET curves plot error fractions on both axes abcissa and ordinate.

$$PFC(C+) = \frac{P(C+) \cdot C(D- \mid C+)}{P(C+) \cdot C(D- \mid C+) + P(C-) \cdot C(D+ \mid C-)} \tag{2.1}$$

$$NEC = (1 - TPF - FPF) \cdot PCF(C+) + FPF \tag{2.2}$$

An important feature of explicitly representing expected cost is its duality with respect to ROC space. That is, a line in the ROC space can be directly transformed to a point in the cost space whereas a point in the ROC space is converted into a line in the cost space.

Bidirectional point/line duality

$$\begin{array}{ll} NEC & = (1 - TP_o) \times PCF(C+) \\ PFC(C+) & = \frac{1}{1+S} \end{array} \tag{2.3}$$

Where S is the slope of the line and TP_o the intersection with the abcissa axis. Figure 2.6 depicts 3 ROC points and their dual representation whose error type weighting is 1:10 (i.e., a cost of 1 for each false positive and a cost of 10 for each false negative). We will be back to Fig. 2.6 later on in Chapter 6 of the thesis. The main advantage of explicitly representing expected cost is its comprehensible visual interpretation.

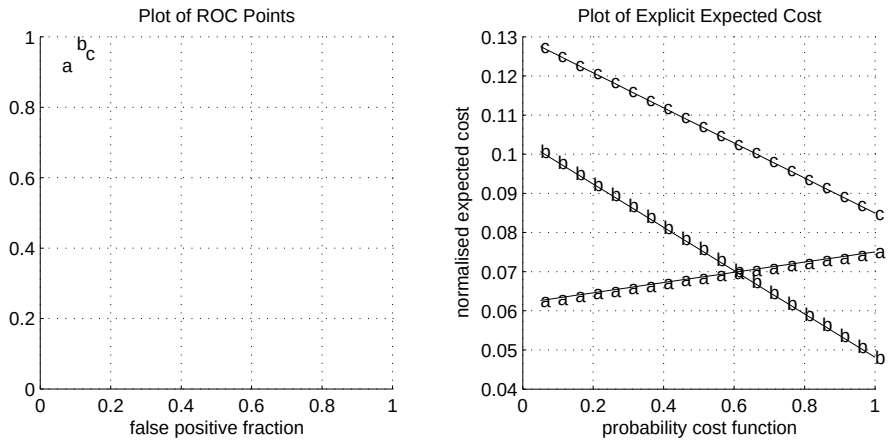


Fig. 2.6 Three ROC points and their dual representation explicitly representing expected cost.

This Chapter places the thesis within the larger research landscape and has served to review in summary form much of the relevant material from the literature.

3

Sequential Cases

This Chapter introduces the knowledge structures and techniques that sustain sequential cases. We commence our exposition in Section 3.1 pointing out the relevance of hierachical knowledge structures, the unbalanced attention that compositional hierarchies have received with respect to taxonomic hierarchies in the Artificial Intelligence literature, and the importance of partonomic knowledge for properly reasoning on composite or aggregated objects. In Section 3.2 we introduce the alert model that makes our approach independent of the alert device and define most of the terms needed through the rest of the thesis. In Section 3.4 we throughly describe the so-called *actionable trees*. Actionable trees are predictive compositional hierarchies that are useful for dealing with sequences of complex objects and reasoning in terms of partial sequences that continuously arrive over time. Finally, in Section 3.5 we overview several window models to compute aggregates of interest over a sequence of alerts.

3.1 HIERARCHICAL KNOWLEDGE STRUCTURES

Artificial Intelligence (AI) allows computers to achieve human-level performance on a variety of specialized reasoning and learning tasks. Several AI techniques allow one to perform tasks that otherwise would be difficultly doable by human beings, at least in an admissible amount of time [Gem99]. Often, these tasks reflect structure—smaller and common parts to build from—according with the number of regularities that constantly appears in the natural world. It is apparent that human beings exploit these regularities to solve complex tasks [Gem99]. As a matter of fact, structure appear

throughout human knowledge supporting many aspects of commonsense reasoning [Why69]. The natural world (both the physical and the biological) are also plenty of examples that exhibe hierarchical structure [Why69]. Hierarchical structure is not only omnipresent in natural systems but also in artificially created systems [LGW02].

Hierarchical structures have played a lead role in the problem of modeling representations within the Artificial Intelligence and Cognitive Science fields [WCH87, Mac89, Rin89, RK91, G00]. Understanding the workings of hierarchical structures in natural systems and devising artificial software devices that accurately simulate the real world mimicking the emergence of comparable structures also constitutes an arena of research within the Artificial Life community [LGW02]. In knowledge representation and object-centered formalisms, hierarchies are key for bridging the gap between low-level, fine-grained representations and high-level concepts [Pfl02].

There exist two well-known types of hierarchies: *taxonomic hierarchies* and *compositional hierarchies*. It is often said that taxonomic hierarchies control the level of abstraction whereas the level of granularity is controlled by compositional hierarchies [Pfl02]. Respectively, these structural hierarchies are also known as subsumption or *is-a* relation and *part-of* relation:

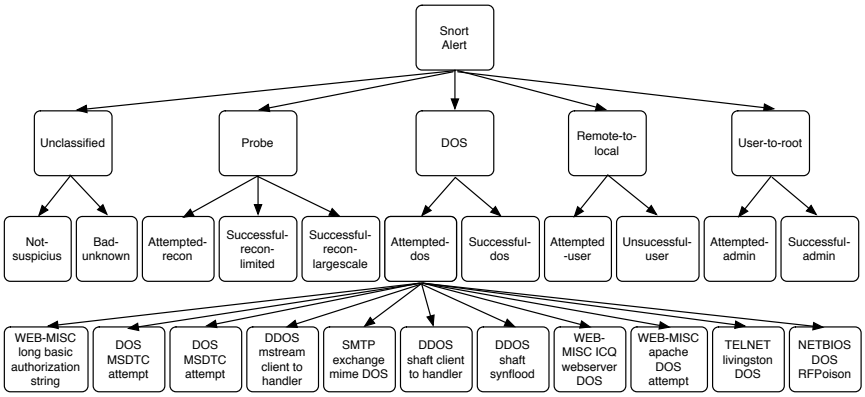


Fig. 3.1 Partial view of a taxonomic hierarchy for Snort alerts. The lowest level shows alerts indicanting a *denial of service* (DoS)

Is-a Relation The is-a relation allows objects with similar properties in a given domain to be organized into classes [Bra83, Woo91]. For example, Homo Sapiens species occupies a particular place whithin a well established taxonomic hierarchy (Homo Sapiens, Hominidae, Catharrhini, Primates, Eutheria, Mammalia, Therapsida, Synapsida, Amniota, Terrestral Vertebrates, and so on) [GD02]. Given that taxonomic hierarchy we can make interpretations such as Homo Sapiens is a Hominidae or Mammalia subsumes Homo Sapiens. Figure 3.1 depicts a partial view of the taxonomic hierarchy that we have constructed for modeling alerts in computer networks. In Chapter 7, we will

describe SOID our particular ontology for intrusion detection that provides us with taxonomic hierarchies for alerts, incidents, computer networks, and vulnerabilities.

Part-whole Relation The part-of (part-whole) relation facilitates the organization of objects in the domain in terms of composite or aggregated objects—individualised entities which themselves are composed of and can be analysed into other individual entities [WCH87, PL94, RN95, LS97, Mor98, Lam00, Sat00, GD02]. A well-understood example is HTN (Hierarchical Transition Network) planning. Usually, a HTN planner starts with an abstract network representing the whole task to be solved, and proceeds by expanding the abstract network into more detailed sub-tasks lower down into a hierarchy, until the sub-tasks only contain executable actions [KJN94, MPG96]. In intrusion detection, a composite or multi-stage attack can be identified by the sequence of intrusion alerts that generates. This sequence of alerts (the whole) can be divided into smaller chunks or subsequences of alerts (the sub-parts) that represent specific portions of the attack. Figure 3.2 shows a compositional hierarchy that represents a composite attack. In this example, a sequence of four intrusion alerts are grouped together into a composite attack (one main part) called the Mitnick attack. This attack (the whole) can in turn be decomposed into two levels of subparts. Each sub-part groups together alerts that correspond to portions of the attack that are logically coupled. If we consider the TCP Sequence-Number Prediction attack independently of the compositional hierarchy of Figure 3.2 we can say that such attack only contains one main part and one level of atomic subparts (nmap fingerprint attempt and SYN half-open connection). Now, this attack could be part of a TCP Hijacking attack that in turn could be part of the Mitnick attack.

These two types of relations enable respectively two basic operations: *abstraction* and *combination* [VC98]. Both operations sustain a variety of methods for pattern recognition: *syntactic pattern recognition*, *mathematical linguistics*, *structural pattern recognition*, and *statistical pattern recognition* (see [VC98] for further details). Although the philosophical concerns about the interpretations of both relations have their origins a long time ago by part of linguistics, mathematicians, and philosophers [Sim87, Har02], the attention that each type has been dispensed is different, at least, in the *data and knowledge engineering* and *machine learning* communities [CV02]. Only few years ago Lambrix claimed that “the is-a relation has received a lot of attention and is well-understood, while part-of has not been studied as extensively. Also the interaction between these two relations has not been studied in any detail” [Lam00]. Recently, Pfleger stated “the fact that vastly more learning effort has been devoted to taxonomic structure, strongly indicates a need for work on learning compositional structure” [Pfl02]. Neither object-centered sytems have been provided with appropiate semantics and specific inferential mechanishms for the part-whole relation as noted by Artale et al “rarely do current formalisms and methodologies give it (part-whole relation) a specific *first-class* dignity” [AFG96b]. The following

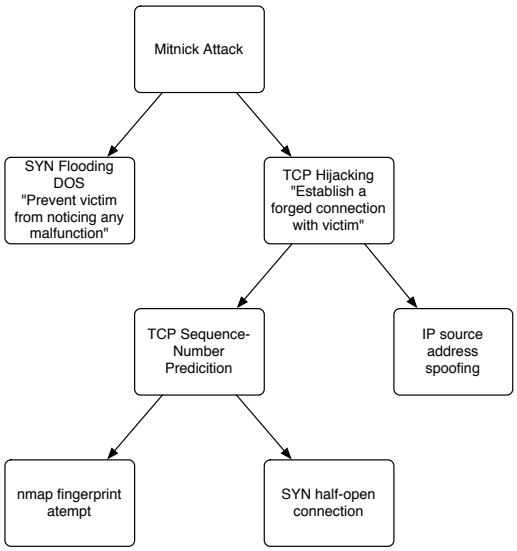


Fig. 3.2 A compositional hierarchy for the well-known Kevin Mitnick attack.

are a representative sample of works tackling this critical deficiency [Sat95, Ber96, Ber96, Fra93, Var96, JN98, PL94, AFG96a, Sat00, Lam00, ABT99, AFG96b].

One of the issues that we are mostly concerned with in this thesis is hierarchical structuring of sequences by means of compositional hierarchies[1]. A compositional hierarchy (part-whole hierarchy) is a graph (see Figure 3.2) that represents a composite object as a hierarchical composition of parts with the semantics that nodes represent parts and substructures and the arcs represent the structure of the object [Uta94]. In a compositional hierarchy the nodes at the bottom refer to individual parts of the composite object whereas nodes at the higher levels reflect substructures (the higher the level the larger the substructure represented). Sequences as many other types of data exhibit hierarchical compositional structure [Pfl02]. Hence, sequences are made up of subsequences that in turn are made up of sub-subsequences and so on [SG01b]. A sequence of lower-level entities can be represented using subsequences of high-level entities or *chunks*. The following examples try to give an intuitive picture of how both types of hierarchies (taxonomic and compositional) can be used when analyising sequences. Figure 3.3 shows an example sequence of letters that conform the word thinking. Taking this sequence as a unique exemplar it could be said that the occurrence of a letter i is always immediately followed by the occurrence of a letter n. Following the taxonomic hierarchy depicted by Figure 3.4 the word thinking can

[1]Compositional Hierarchies have been widely studied in the context of model-based recognition systems [Uta94]. Recently, Plfeger's thesis provided a significant advance in the learning of these structures [Pfl02].

Fig. 3.3 An example sequence, the word thinking.

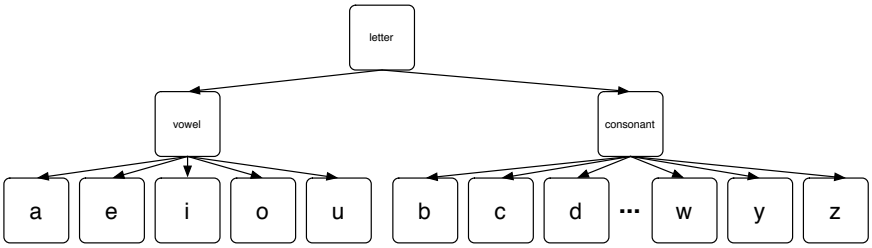


Fig. 3.4 An example taxonomic hierarchy of letters.

be represented, in a more abstract way, as a sequence of vowels and consonants such as shown in Figure 3.5. Using such abstraction an additional pattern could be formulated: a vowel always occurs among blocks of two consonants. Therefore, higher level inferences can be done using abstraction as illustrated with the patterns that can be extracted before and after abstraction.

On the other hand, if we look at the compositional structure of the word thinking it could be broken down as shown in Figure 3.6. An important issue in compositional hierarchies is how to allocate substructures [Uta94]. Taking the example introduced by Utans, Figures 3.7, 3.8, and 3.9 show three different compositional hierarchies of the same object. The compositional hierarchy in Figure 3.7 have been constructed using substructures that emphasize parallel segments whereas the compositional hierarchy in Figure 3.9 uses substructures that emphasize the use of angles. Finally, the substructures in the compositional hierarchy shown by Figure 3.8 emphasize the reuse of parts. We are mainly concerned with compositional hierarchies that emphasize the use of decoupled substructures (just as in Figure 3.7). That is, each substructure is assigned a local context reflecting independence from the rest of substructures at the same level in the hierarchy.

We propose to represent sequential cases by means compositional hierarchies. Particularly, we introduce *actionable trees* to represent sequences of complex objects and reason in terms of partially observed sequences. An actionable tree is a a Multi-Rooted Acyclic Graph (MAG) with the semantics that (i) roots symbolize observable symptom events, (ii) intermediate nodes (in the trunk and crown) embody sequences of events (chunks), and (iii) the arcs represent part-whole relationships as well as the likelihood of occurrence of the whole (sequence) given the part (subsequence). We

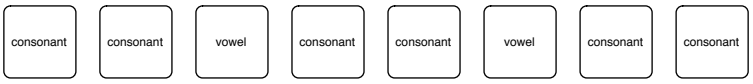


Fig. 3.5 An example abstraction of the word thinking following hierarchy of Figure 3.4.

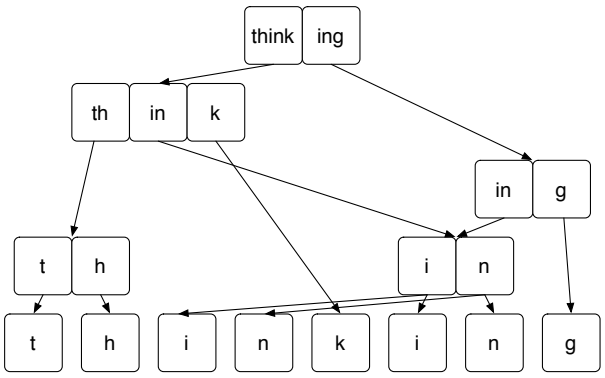


Fig. 3.6 An example compositional hierarchy, in which individual letters of the word thinking are pieced together into chunks such as th or in, which in turn are pieced together to conform the word thinking [Pfl02].

throughly describe actionable trees in Section 3.4. Next Section introduce the alert model and basic structures that sustain the techniques presented in the rest of this Chapter.

3.2 ALERT MODEL

We suppose alerts that are triggered by automated real-time systems that collect and interpret sensor data in real-time[2]. In other words, we deal with processes that have already been automated but whose output still provide an unmanageable number of alerts. Remember the separation between probes and ACCs that we did in Chapter 2 and that we will make explicit again in Chapter 7. As a result of such previous automation, alerts come in a standardized form. Alerts are complex objects (data structures) made up of a set $\mathcal{F}$ of numeric, qualitative and structured features that can be modeled as tuples over the *Cartesian* product $D_{f_1} \times D_{f_2} \times \cdots D_{f_{n-1}} \times D_{f_n}$ where

[2] As noted in Chapter 1, we use indistinctly the terms alert, event, manifestation, etc.

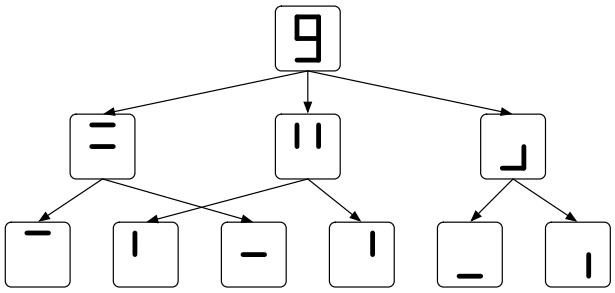


Fig. 3.7 A compositional hierarchy for the digit 9 for a seven-segment LED display emphasizing parallel lines.

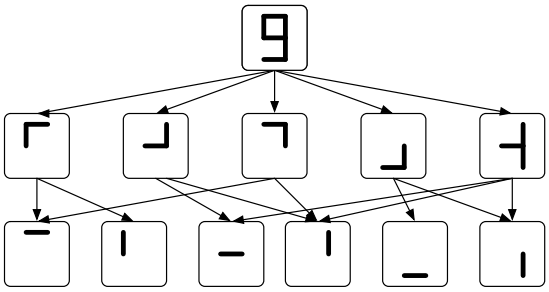


Fig. 3.8 A compositional hierarchy for the digit 9 for a seven-segment LED display emphasizing reuse.

$\mathcal{F} = \{f_1, f_2, \cdots, f_n\}$ and D_{f_i} is the range of possible values (domain) of feature f_i [JD02]. More formally, we model alerts using feature terms[3].

Next, we define the basic elements that form an *alert model.*

Definition 1 (Alert Model) *An alert model $\mathbb{A}$ is defined as an ordered pair $\mathbb{A} = \langle \vartheta, \Sigma \rangle$ where ϑ is a set of variables and Σ is an alert signature defined as follows.*

Definition 2 (Alert Signature) *An alert signature $\Sigma = \langle \mathcal{S}, \perp, \mathcal{F}, \preceq \rangle$ is a four-tuple where $\mathcal{S}$ is a set of sort symbols; $\mathcal{F}$ is a set of feature symbols; and $\preceq$ is a decidable partial order on $\mathcal{S}$ such that $\perp$ is the least element.*

Definition 3 (Alert) *Given an alert model $\mathbb{A} = \langle \vartheta, \Sigma \rangle$, an alert is an expression of the form $\psi ::= X : s[f_1 \doteq \Psi_1 \ldots f_n \doteq \Psi_n]$ where X is a variable in ϑ called the*

[3]Feature terms organizes concepts into a hierarchy of *sorts*, and represent *terms* or *individuals* as collections of *features* (functional relations). For further details and examples see [Pla95]

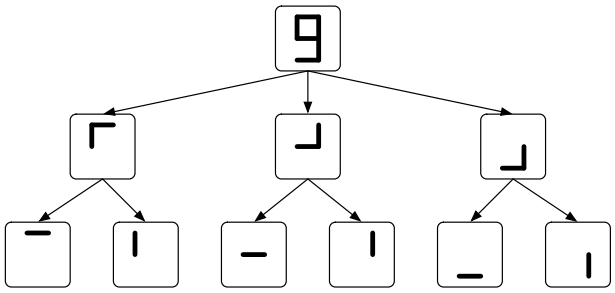


Fig. 3.9 A compositional hierarchy for the digit 9 for a seven-segment LED display emphasizing angles.

root *of the alert,* s *is a sort in* $\mathcal{S}$*,* $f_1 \ldots f_n$ *are features in* $\mathcal{F}$*,* $n \geq 0$*, and each* Ψ_i *is an alert and/or variable* [4].

We denote the set of variables occurring in ψ as ϑ_ψ. We also define the function *root(*ψ*)* that returns the sort of the root. Notice that when $n = 0$ we are defining a variable without features.

Definition 4 (Dummy Alert) *A* dummy *alert is an alert with no features and whose root is* any. *We indistinctly denote it by* ψ_0 *or by overloading* $\perp$ *symbol.*

An important notion to distinguish is *path equality* since it allows actionable trees to specify constraints among the distinct alerts that compound a sequential case as we will see later on. We formally define a *path* and *path equality* as follows.

Definition 5 (Path) *Given an alert model* $\mathbb{A} = \langle \vartheta, \Sigma \rangle$*, a* path $\rho(X, f_i)$ *is a sequence of features going from the variable* X *to the feature* f_i*.*

We can concatenate *paths* to conform longer paths $\tau = \rho_1 \cdots \rho_n$.

Definition 6 (Path Equality) *There is a* path equality *when two paths* $\rho(X, f_i)$ *and* $\rho(Y, f_j)$ *point to the same value (i.e.,* $\rho(X, f_i) = \rho(Y, f_j)$*).*

Path equality is equivalent to variable equality in first order terms. Figure 3.10 shows a formal representation of an alert. Alerts are also represented graphically by means of labeled directed graphs. See Figure 7.13 in Chapter 7 for an example of an alert evoked by a Snort sensor such as it is represented in the Noos object-centered language.

Next, we introduce two crucial concepts: *subsumption* and *sequences*.

[4]Notice that we consider that every possible value is a singleton rather than a set. The motivation for this characteristic is twofold: first, it makes much more comprehensible the notion of path defined below; and, second, it avoids exponential explosion caused when computing subsumption according to the definition introduced subsequently.

3.2.1 Informational Order Among Alerts

Given an alert model $\mathbb{A} = \langle \vartheta, \Sigma \rangle$, sorts in $\Sigma.\mathcal{S}$ have by definition an informational order relation ($\preceq$) among them, where $s \preceq s'$ means that s carries less information than s'—or equivalently that s is more general than s'. The minimal element ($\perp$) is called *any* and it represents the minimum information (i.e., $\forall_{s \in \Sigma.\mathcal{S}} \perp \preceq s$). When a feature has an unknown value it is represented as having the value *any*. All other sorts are more specific than *any*. Based on that order among sorts, intuitively, we say of two alerts $\psi, \psi' \in \mathbb{A} = \langle \vartheta, \Sigma \rangle$ that ψ subsumes ψ' $(\psi \sqsubseteq \psi')$ when all that is true for ψ is also true for ψ'. More formally:

Definition 7 (Alert Subsumption) *Given an alert model $\mathbb{A} = \langle \vartheta, \Sigma \rangle$, an alert $\psi \in \mathbb{A}$ subsumes other alert $\psi' \in \mathbb{A}$ $(\psi \sqsubseteq \psi')$ if:*

1. $root(\psi) \preceq root(\psi')$*, i.e., the sort of ψ' is a subsort of ψ's sort.*
2. $\forall f \in \Sigma.\mathcal{F} : \psi.f \neq \perp \Rightarrow \psi.f \neq \perp$*, i.e., every defined feature in ψ is also defined (has a value different from* any*) in ψ'.*
3. $\forall f \in \Sigma.\mathcal{F} : \psi.f = v \neq \perp \Rightarrow v \sqsubseteq v' = \psi'.f$
4. *path equality is satisfied downwards:*

$$\rho(root(\psi), f_1) = \rho((root(\psi), f_2) = v$$
$$\Rightarrow$$
$$(\rho(root(\psi'), f_1) = \rho((root(\psi'), f_2) = v') \wedge (v \sqsubseteq v')$$

Alert subsumption is essential in our approach to compare both individual alerts and sequence of alerts and determine their similarity. We now proceed to the definition of *sequence of alerts*.

3.2.2 Sequences of Alerts

The basic goal here is to provide some concepts that allows us to model conveniently the manipulation of sequences of alerts. We firstly distinguish between *sequence of alerts* and *sequence of sorts*. Often this separation is valid for dealing with sequences of simple objects and therefore gaining simplicity and minoring the complexity and increasing the efficiency of several algorithms.

Definition 8 (Temporal Sequence of Alerts) *Given an alert model $\mathbb{A} = \langle \vartheta, \Sigma \rangle$, a temporal sequence of alerts, or simply a sequence of alerts, $\vec{S}$ is an ordered collection of alerts $\vec{S} = [\psi_1, \psi_2, \ldots, \psi_n]$ such that each $\vec{S}_i \in \mathbb{A}$ has at least a common feature $t \in \Sigma.\mathcal{F}$ (usually a time-stamp) whose values are totally ordered i.e., $\psi_1.t \leq \psi_2.t \leq \cdots \leq \psi_n.t$. We denote an individual alert within a sequence $\vec{S}$ by $\vec{S}_i^{t_i}$ or $\psi_i^{t_i}$ or without t_i superscripts when the specific value of t is not of interest. We say that $\vec{S} \in \mathbb{A}^*$.*

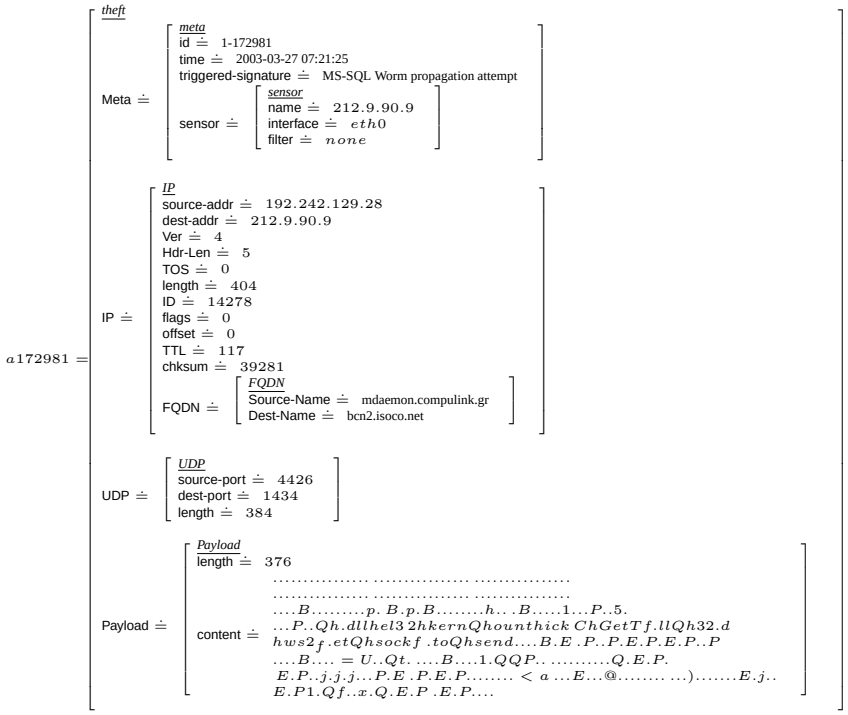


Fig. 3.10 MS-SQL Worm propagation alert represented using feature terms.

The length of a sequence $\vec{S} = [\psi_1, \psi_2, \ldots, \psi_n]$, written $|\vec{S}|$, is its cardinality (n). Two sequences of alerts can be linked together using the $\bullet$ operator.

Definition 9 (Sequence of Sorts) *Given an alert model $\mathbb{A} = \langle \vartheta, \Sigma \rangle$, a temporal sequence of sorts, or simply a sequence of sorts, $\vec{T}$ is an ordered collection of sorts $\vec{T} = [s_1, s_2, \ldots, s_n]$ such that $\vec{T}_i \in \mathbb{A}.\Sigma.\mathcal{S}$. We say that $\vec{T} \in \mathbb{A}.\Sigma.\mathcal{S}^*$*

Given a sequence of alerts $\vec{S} \in \mathbb{A}^*$ we can obtain the corresponding sequence of sorts $\vec{T} \in \Sigma.\mathcal{S}^*$ such that $\vec{T}_i = root(\vec{S}_i)$. The length of a sequence of sorts $\vec{T} = [s_1, s_2, \ldots, s_n]$, written $|\vec{T}|$, is its cardinality (n). Two sequences of sorts are linked together by means of the operator $\circ$.

Definition 10 (Simple Sequence Subsumption) *Given an alert model $\mathbb{A} = \langle \vartheta, \Sigma \rangle$, and let $\vec{S}$ and $\vec{P}$ be two sequences of sorts in $\Sigma.\mathcal{S}^*$, such that $\vec{S} = [s_1, s_2, ..., s_n]$, $\vec{P} = [p_1, p_2, ..., p_m]$, $|\vec{S}| = n$, $|\vec{P}| = m$, and $n \geqslant m$ we say that $\vec{P}$ subsumes $\vec{S}$ if there exists a sequence of indices $1 \leq i_1 < i_2 < \cdots < i_m \leq n$ such that: $p_1 \preceq s_{i_1}, p_2 \preceq s_{i_2}, \cdots, p_m \preceq s_{i_m}$.*

We overload the operator $\sqsubseteq$ to denote that a sequence of sorts $\vec{S}_i$ subsumes another sequence of sorts $\vec{S}_j$ (i.e., $\vec{S}_i \sqsubseteq \vec{S}_j$).

Notice that simple sequence subsumption entails a subsequence comparison where not only a perfect matching between the elements of the pattern and the text are allowed but also a partial match based on the informational order established among sorts. Notice also that subsequence comparison has higher complexity than substring matching [FGS01]. Sequence subsumption is defined as follows.

Definition 11 (Sequence Subsumption) *Given an alert model $\mathbb{A} = \langle \vartheta, \Sigma \rangle$ and let $\vec{S}$ and $\vec{P}$ be two sequences of alerts in $\mathbb{A}^*$ such that $\vec{S} = [\psi_1, \psi_2, ..., \psi_n]$ and $\vec{P} = [\psi'_1, \psi'_2, ..., \psi'_m]$, $|\vec{S}| = n$, $|\vec{P}| = m$, and $n \geqslant m$. We say that $\vec{P}$ subsumes $\vec{S}$ if there exists a sequence of indices $1 \leq i_1 < i_2 < \cdots < i_m \leq n$ such that: $\psi'_1 \sqsubseteq \psi_{i_1}, \psi'_2 \sqsubseteq \psi_{i_2}, \cdots, \psi'_m \sqsubseteq \psi_{i_m}$.*

When it is clear that both operands are sequences of alerts we use the overloaded $\sqsubseteq$ operator. Given a sequence of alerts of length n it can be segmented according to the following definition.

Definition 12 (Sequence Segmentation) *Given an alert model $\mathbb{A} = \langle \vartheta, \Sigma \rangle$, let $\vec{S}$ be a sequence of alerts in $\mathbb{A}^*$ such that $\vec{S} = [\psi_1, \psi_2, ..., \psi_n]$ and $|\vec{S}| = n$; then a sequence segmentation $\mathbb{S}(\vec{S}, m)$ of degree m, or simply m-segmentation, is a sequence of $m + 1$ segment boundaries in $[1, n]$ such that $\mathbb{S}(\vec{S}, m) = [\mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_m, \mathbf{s}_{m+1}]$ and $1 = \mathbf{s}_1 < \mathbf{s}_2 < \mathbf{s}_3 < \cdots < \mathbf{s}_m < \mathbf{s}_{m+1} = n + 1$. We denote by $\mathbf{Seg}^m(\vec{S})$ the set of all m-segmentations of $\vec{S}$.*

A m-segmentation yields a sequence of m segments $[\mathbf{S}_1, \cdots, \mathbf{S}_m]$: $[[\psi_1, \cdots, \psi_{\mathbf{s}_2-1}], [\psi_{\mathbf{s}_2}, \cdots, \psi_{\mathbf{s}_3-1}], \cdots, [\psi_{\mathbf{s}_m}, \cdots, \psi_n]] = [[\psi_{\mathbf{s}_i}, \cdots, \psi_{\mathbf{s}_{i+1}-1}]]_{i=1}^m$ such that $\sum_{i=1}^m |\mathbf{S}_i| = n$. Notice therefore that given a sequence of alerts there exist multiple possible segmentations. The sequence segmentation definition likewise applies to a sequence of sorts.

A crucial operation between sequences for comparison purposes is *alignment* as we will see in the next Chapter of the thesis. We define a sequence alignment as follows. The same definition also applies to sequences of sorts.

Definition 13 (Sequence Alignment) *Given an alert model $\mathbb{A} = \langle \vartheta, \Sigma \rangle$ and two sequences $\vec{S}_1, \vec{S}_2 \in \Sigma^*$. An alignment of sequences $\vec{S}_1$ and $\vec{S}_2$ is a pair $\langle \vec{S}'_1, \vec{S}'_2 \rangle$ attained by insertion of a number of dummy feature terms ($\perp$) in both sequences such that: $|\vec{S}'_1| = |\vec{S}'_2|$ and $\forall_{1 \leq i \leq |\vec{S}'_1|}$ $\vec{S}'_1[i]$ is aligned with $\vec{S}'_2[i]$ and either $\vec{S}'_1[i]$ or $\vec{S}'_2[i]$ is not a dummy feature term.*

Finally, we define the notion of *occurrence* as follows:

Definition 14 (Occurrence) *Given an alert model $\mathbb{A} = \langle \vartheta, \Sigma \rangle$ and two sequences $\vec{S}_1, \vec{S}_2 \in \Sigma^*$. We say that $\vec{S}_1$ occurs in $\vec{S}_2$ if $\vec{S}_1$ subsumes $\vec{S}_2$ (i.e., $\vec{S}_1 \sqsubseteq \vec{S}_2$).*

We will subsequently use these definition for introducing the strength of a part-of relation in Section 3.4.4. Next we provide a partial introduction to our case base model that we complete in Chapter 5.

3.3 CASE BASE MODEL

Let's first define the notions of compositional and sequential (serial and parallel) cases.

Compositional Cases A compositional (or composite) case is an assemblage of several cases that lies in a hierarchical structure. The cases on the upper levels are made up of sub-cases that in turn are compositional. The lowest level is made of indivisible cases. The highest level is made up of only one case that refers to the whole compositional hierarchy. Intermediate compositional cases (cases that lie between the highest level and the lowest level) are considered as part of a larger ensemble solution rather than as individual solutions to the case at hand.

Sequential Cases A sequential case is a compositional case where a temporal order is established among all the parts that comprise it. If all the sub-cases that make up a sequential case are totally-ordered then we say that the sequential case is serial. If the order of all the sub-cases are interchangeable then we say that the sequential case is parallel. Otherwise, we say that it is a partially-ordered sequential case (i.e., an arbitrary sequential case made up of serial and parallel subcases).

Additionally, we distinguish between *master cases* and *abstract cases* as we will see in Chapter 5. Now we concentrate on the compositional aspects of sequential cases. We can say that among the sub-cases that compound a sequential case there exist a part-of relation defined as follows.

3.3.1 Part-Of Representation

Let us define the notion of *direct part* as follows.

Definition 15 (Direct Part) *We say that a case C_i is a* direct part *of a case C_j, denoted by $C_i \lhd C_j$, iff $C_i \subset C_j \wedge \nexists C_k \neq C_i : C_i \lhd C_k \wedge C_k \lhd C_j$ (i.e., they are one step away).*

Notice that we could also name the part-whole relation. For example we could write $C_i \lhd_{reconnaissance} C_j$ indicating that case C_i constitutes a direct *reconnaissance* part of case C_j. Naming part-of relations allows one to distinguish between different kinds of relations [Lam00]. However, as we will see below, we are more interested in the order between parts and consider only two types of relations (serial and parallel) thus we obviate the name of part-of relations from now on.

A case that has direct parts is a compositional case. A case that has no parts is a simple case.

Definition 16 (Serial Part) *We say that case C_i is a* serial part *of case C_j, denoted by $C_i \trianglelefteq C_j$, iff C_i is a direct part of C_j and C_j is a serial case.*

Definition 17 (Parallel Part) *We say that case C_i is a* parallel part *of case C_j, denoted by $C_i \lhd_{\|} C_j$, iff C_i is a direct part of C_j and C_j is a parallel case. Whenever there is no place for confusion we simply denote parallel parts by $\lhd$.*

Definition 18 (Part-of) *We say that case C_i is a* part *of case C_j, denoted by $C_i \lhd^* C_j$, iff there exist $n \geq 0$ cases $C_{k+1}, \cdots, C_{k+n}$ such that $C_i \lhd C_{k+1} \lhd \cdots \lhd C_{k+n} \lhd C_j$.*

The part-of relation is the transitive closure over all direct parts. If C_i is a direct part of C_j it is also a part of C_j. Following Lambrix part-of representation basis we make the following assumptions [Lam00]:

1. A case can not be a part of itself:

$$\forall_{C_i} \neg (C_i \lhd^* C_i) \tag{3.1}$$

   This assumption implies that there are no cycles in the part-of relation. Therefore the part-of relation is also antisymmetric given that it is also transitive.

2. A composite case is made up of at least two direct parts:

$$\forall_{C_i, C_j} C_i \lhd C_j \rightarrow \exists C_k : C_k \neq C_i \wedge C_k \lhd C_j \tag{3.2}$$

3. If a case C_i is a direct part of another case C_j then there is no other case C_k such that C_i is part of C_k and C_k is part of C_j.

$$\forall_{C_i, C_j} C_i \lhd C_j \rightarrow \nexists C_k : C_i \lhd^* C_k \wedge C_k \lhd^* C_j \tag{3.3}$$

   Therefore if a case C_j is composed-of a composite case C_i then direct parts of C_i are not direct parts of C_j.

3.4 ACTIONABLE TREES

An actionable tree is a predictive compositional hierarchy—a knowledge structure with the ability to easily capture the hierarchical structure of sequences, reasoning at higher levels in terms of part-whole relationships, and making predictions. An actionable tree is represented using a Multi-Rooted Acyclic Graph (MRAG) with the semantics that roots (defined below) represent observable events, intermediate nodes represent composite (serial or parallel) cases and the arcs represent part-whole relationships. An actionable tree allows one to specify:

- Sorts to which each part of a composite case belongs.
- Constraints between the different parts conforming of a composite case by means of path equalities (see Definition 6).
- A (partial) order among the different parts that comprise a composite case.

Moreover, an actionable tree facilitates one to reason and make inferences in terms of part-whole relations. For example, inferring the existence of a whole based on the presence of a number of its parts or determining whether a new whole could be built from available parts that belong to a particular type. In the next Subsection we provide definitions of basic terms and structures which we shall need throughout the rest of this Section.

3.4.1 Basics

For the sake of completeness we first define graph and some related terms[5]. If the reader is familiar with graph basics can skip this subsection without loss of continuity.

Definition 19 (Graph) *A graph G is an ordered pair $\langle V, E\rangle$ such that $V \neq \emptyset$ is a finite set and E is a finite collection of two-element subsets of V. The elements of V are called nodes or vertices. Each unordered pair $\{u, v\} \in E$ with end nodes u and v is called an arc, edge, link, or connection.*

Given an arc $a = \{u, v\}$, we say that u and v are *adjacent* each other and that both are *incident* with a. The *neighbors* of a node u are its adjacent nodes.

Definition 20 (Degree) *Given a graph $G = \langle V, E\rangle$ the degree of $u \in V$, denoted by $deg(u)$ is the number of its neighbors.*

When every node has degree d then we say that a graph is *regular* of degree d. An arc $a = \{u, v\}$ where $u = v$ is called a *self-loop*. When the same arc occurs more than once in E is called a *multiple arc* [BET99].

Definition 21 (Simple Graph) *A simple graph has no self-loops and no multiple edges.*

A graph is *connected* if there is a path between *u* and *v* for each pair $\{u, v\}$ of vertices. From now on, we only consider simple connected graphs.

Definition 22 (Subgraph) *Given a graph $G = \langle V, E\rangle$, a graph $G' = \langle V', E'\rangle$ is a subgraph of G iff $V' \subseteq V$ and $E' \subseteq E \cap (V' \times V')$*

Definition 23 (Path) *Given a graph $G = \langle V, E\rangle$ a path is a sequence $[v_1, v_2, \cdots, v_k]$ such that $v_i \in V$ and $\{v_i, v_{i+1}\} \in E$ for $1 \leq i \leq k-1$. A path is a cycle if additionaly $\{v_k, v_i\} \in E$.*

We say that a graph is *connected* if each pair of vertices is joined by path. A graph $G = \langle V, E\rangle$ is a *weighted graph* if there is a weight function $w : E \rightarrow \mathbb{R}$ associated with it. The weight of a path $[v_1, v_2, \cdots, v_k]$ is defined as the sum of the weights of each $\{v_i, v_{i+1}\}(i.e., \{v_1, v_2\} + \{v_2, v_3\} + \cdots + \{v_{k-1}, v_k\})$.

[5] See Jungnickel's book for further details on graphs and trees [Jun99].

Definition 24 (Acyclic Graph) *A graph $G = \langle V, E \rangle$ is acyclic if it has no cycles.*

An acyclic graph is also known as a *forest*.

Definition 25 (Directed Graph) *A directed graph or digraph $G = \langle V, E \rangle$ is a graph where the elements of E are directed arcs. That is ordered pairs of nodes $\langle u, v \rangle$ instead of unordered pairs $\{u, v\}$.*

Given a directed graph, the *underlying undirected graph* can be constructed by forgetting the directions of the arcs. The directed arc $\langle u, v \rangle$ is an*outgoing* arc of u and an *incoming* arc of v.

Definition 26 (Out-degree) *Given a digraph $G = \langle V, E \rangle$ the out-degree of $u \in V$, denoted by $outdeg(u)$ is the number of its outgoing arcs.*

Definition 27 (In-degree) *Given a digraph $G = \langle V, E \rangle$ the in-degree of $u \in V$, denoted by $indeg(u)$ is the number of its incoming arcs.*

A node u whose out-degree is 0 $outdeg(u) = 0$ (i.e., without outgoing arcs) is called a *sink*. Conversely, a node u whose in-degree is 0 $indeg(u) = 0$ (i.e., without incoming arcs) is called a *source*.

Definition 28 (Directed Path) *Given a digraph $G = \langle V, E \rangle$ a directed path is is a sequence $[u_1, u_2, \cdots, v_k]$ such that $v_i \in V$ and $\langle v_i, v_i + 1 \rangle \in E$ for $1 \leq i \leq k - 1$. A directed path is a directed cycle if additionaly $\langle v_h, v_i \rangle \in E$.*

Definition 29 (Directed Acyclic Graph (DAG)) *A digraph $G = \langle V, E \rangle$ is acyclic if it has no directed cycles.*

The following theorem allows us to introduce the notion of *tree*. Its *proof* can be found elsewhere [Jun99].

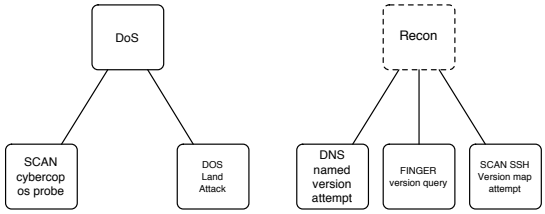
Theorem 1 *Given a graph $G = \langle V, E \rangle$ such that $|V| = n$. Then any two of the following conditions imply the third:*

- *G is connected.*
- *G is acyclic.*
- *G has $n - 1$ arcs.*

A graph G that accomplishes the conditions of Theorem 1 is called a *tree*. Therefore, a tree is a *connected acyclic graph* [Jun99]. An *ordered tree* is a tree where the children of every node are ordered, that is, there is a first child, second child, third child, etc.

Definition 30 (Multi-Rooted Acyclic Graph) *A Multi-Rooted Acyclic Graph (MRAG) consists of a graph $G = \langle V, E \rangle$ and a set of distinguished nodes $R \subset V$. Each node $u \in R$ is called a root of G.*

Notice that a MRAG is also a *multi-rooted tree* or *poly-tree*. Now we are ready to introduce actionable trees.



(a) (b)

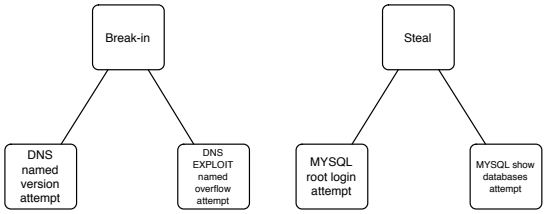
Fig. 3.11 (a) An actionable tree of a DoS attack. The first alert SCAN cybercop os probe is a reconnaissance attempt generated when an attacker uses Cybercop to ascertain which ports the server is listening on and whether the ports are filtered by a firewall or not, and to fingerprint which operating system is running. Once he detects that a victim is using an unpatched version of Windows NT he launches a DOS Land attack. This attack exploits that some TCP/IP stacks crash or hang when sent a spoofed TCP SYN packet with the same source and destination host and the same source and destination port (see CVE-1999-0016 for further information). (b) A parallel actionable tree of a Reconnaissance attack. The first alert DNS named version attempt is a reconnaissance attempt generated when an attacker attempts to determine the version of a BIND DNS server (see http://www.whitehats.com/info/IDS278 for further details). The second alert FINGER version query is generated by an attempt of ascertaining which version of fingerd is running on a host. The third alert SCAN SSH Version map attempt is evoked when an attempt has been made to determine if a vulnerable version of ssh is being used.

3.4.2 Actionable Trees

An actionable tree[6] is an ordered tree in which a single special (distinguished) node is designated as the *crown*, a number of nodes are designated as the *roots*, and the intermediate nodes in the path between the crown and a root node are designated as the *trunk*[7]. Each node in an actionable tree is either a *root* or an *non-root*. A root node has no child nodes. A non-root node is an internal node that has one or more *child nodes* and is called the *parent* of its child nodes. All children of the same node are *siblings*. In an actionable tree any node v on the unique path from the crown to u is called an *ancestor* of u. We say that if v is an ancestor of u, then u is *descendant* of v. Non-root nodes can be classified into *serial* or *parallel* according to the order that they establish among their child nodes. Formally we define an actionable tree as follows:

[6] We often refer indistinctly to actionable trees as alert trees.

[7] A anecdotic particularity of actionable trees is that, conversely to almost all tree graphical representations that we (computer scientists) have been using for years, they are graphically drawn with the crown and trunk above the roots such as in the natural world instead of pictured as inverted trees with the root node at the top and the leaves at the bottom.



(a) (b)

Fig. 3.12 (a) An actionable tree of a Break-In attack.The first alert DNS named version attempt is a reconnaissance attempt generated when an attacker attempts to determine the version of a BIND DNS server (see http://www.whitehats.com/info/IDS278 for further details). The second alert DNS EXPLOIT named overflow attempt is generated by an attempted buffer overflow associated with incorrect validation of NXT records. This attack leads the attacker up to compromise the DNS server gaining access with the privileges of the user running BIND (see http://www.cert.org/advisories/CA-1998-05.html for further information). (b) An actionable tree of a Steal attack. The first alert MYSQL root login attempt is generated when an attempt to log in to as root from a remote source is detected. The second alert is evoked MYSQL show databases attempt when a remote user uses the command show to gather the list of databases being served. The attacker may then continue to gain sensitive information from any database in the system.

Definition 31 (Actionable Tree) *Given an alert model $\mathbb{A} = \langle \vartheta, \Sigma \rangle$ and a collection of alerts $\Psi = \{\psi_1, \cdots, \psi_n\}$ such that $\psi_i \in \mathbb{A}$, an actionable tree is a 5-tuple $\langle G, \mu, \tau, \kappa, \lhd \rangle$ where:*

- *G is a multi-rooted acyclic graph $G = \langle V, E \rangle$ where V is partitioned in three mutually exclusive sets: R (the set of roots), T (the set of trunk nodes), and C (the singleton set containing the crown).*
- *$\mu : R \rightarrow \Psi$ is a mapping that associates each root node with an alert in Ψ.*
- *$\tau : T \cup C \rightarrow \{\mathbf{serial}, \mathbf{parallel}\}$ is a mapping that associates each non-root node with an order type.*
- *$\kappa : T \cup C \rightarrow 2^{\Sigma.\mathcal{F}}$ is a mapping that associates each non-root node with a subset of features (constraints) in $\Sigma.\mathcal{F}$.*
- *E defines part-of relationships $\lhd$ among the subsets of R that in turn induces an order among the elements of R.*

The crown C represents the set of sequences composed of all alerts appearing in R ordered according to $\lhd$ and satisfying the constraints imposed by κ. Non-root nodes $T \cup C$ represent meaningful sequences of the alerts mapped by R. Thus, each node in the tree corresponds to a subsequence of alerts represented by C and the child nodes of each node represent a segment of the parent sequence. We say that an actionable

tree is serial if all its non-root nodes are serial and therefore the order induced among root nodes by $\lhd$ is total. Figures 3.11(a), 3.12(a), and 3.12(b) show three examples of serial actionable trees. In those examples the total order imposed by the order in which they are depicted is crucial for the attacks to succeed (remember that an actionable tree is an ordered tree). Likewise, we say that an actionable tree is parallel if all its non-root nodes are parallel. Figure 3.11(b) shows a parallel actionable tree representing a sequence of alerts that may be the prelude to an attack. The order in which the three alerts are evoked is not important. Since the attacker's objective is just to determine which vulnerable versions of different services are being used on a host, before proceeding to exploit one of those vulnerabilities. Notice that we depict parallel nodes by means of dashed lines.

An actionable tree is injective if the mapping τ establishes a relation one-to-one between root nodes and the sorts of alerts in Ψ. Formally, for every $s \in \mathcal{S}$ there exists at most one $u \in R$ with $\tau(u) = \psi_i \wedge root(\psi_i) = s$. In other words, given $u, v \in R$, when $root(\tau(u)) = root(\tau(v))$ then it follows that $u = v$. From now on we will focus only on injective actionable trees. We suppose that a computer attack does not generates twice the same alert or that our sensors are able to aggregate alerts that correspond to the same attacker's action.

Actionable trees are useful to model a partially ordered collection of events occurring together. Root nodes model individual events whereas non-root nodes model composite events. Roots (or event-level nodes or evidence nodes) represent observable symptom events (i.e., alerts). Inference is engaged upon their individual observation thus we call them roots. Nodes in the trunk represent composite events (i.e., sequences of alerts), and the crown represents a composite event made up of a sequence of all the events in the roots. Roots store alerts whereas non-root nodes store constraints among the alerts of their respective child nodes (including their temporal order).

The most simple actionable tree consists only of a node. That node is simultaneously a root and the crown. Therefore it stores both an alert and information that qualifies that alert. Basic actionable trees are actionable trees that are only composed of roots and crown. That is to say, the roots are directly neighbors of the crown (i.e., they are one step away). Actionable trees shown in Figures 3.11(a), 3.11(b), 3.12(a), and 3.12(b) are basic actionable trees. Basic actionable trees can be compounded together creating larger actionable trees. Figure 3.13 depicts an actionable tree that have compounded piecing actionable trees in Figures 3.11(a), 3.12(a), and 3.12(b) together. Notice that in an actionable tree in addition to the crown node the remaining set of zero or more nodes are themselves actionable trees. The arcs of actionable trees are undirected. However, we consider that actionable trees are *bipolar*. That is to say, each and every arc in an actionable tree is either pointing from the crown to the roots for generation purposes or from the roots to the crown for recognition and prediction purposes. In the first situation, the parent, child, ancestor, and decendant relationships are as mentioned above and an actionable tree corresponds to a *simple tree* (i.e., between every pair of nodes there is at most one path and each node has at most one parent). In the second situation, those relationships are reverse and therefore

an actionable tree corresponds to a *poly-tree* (i.e., although there is still at most one path between every pair of nodes, nodes can have more than one parent).

Let us now define a concept of *yield* that is of paramount importance for the understanding of the rest of the thesis.

3.4.3 Yields of An Actionable Tree

An useful way to think about an actionable tree is as generator of sequences of its root nodes (alerts) that meet the temporal restrictions and constraints imposed by its non-root nodes. When used for generation purposes arcs represent the *composed-of* relation (inverse of part-of) and an actionable tree can be mapped into a context-free grammar that generates the set of sequences of alerts that it represents. We say that an actionable tree produces a set of ordered, fixed-length sequences or *yields*. We formally define a yield as follows.

Definition 32 (Yield) *Given an actionable tree $AT = \langle G, \mu, \tau, \kappa, \lhd \rangle$, a yield is each one of the possible combinations of root nodes in $R \subseteq G$ that meet the temporal restrictions imposed by τ and satisfy the constraints mapped by κ.*

Each actionable tree has associated a context-free grammar that allows one easily compute its yields. For example, the set of sequences generated by the actionable tree in Figure 3.11(b) is:

[DNS named version attempt, FINGER version query, SCAN SSH Version map attempt],

[DNS named version attempt, SCAN SSH Version map attempt, FINGER version query],

[FINGER version query, DNS named version attempt, SCAN SSH Version map attempt],

[FINGER version query, SCAN SSH Version map attempt, DNS named version attempt],

[SCAN SSH Version map attempt, DNS named version attempt, FINGER version query],

[SCAN SSH Version map attempt, FINGER version query, DNS named version attempt]

That is to say, each and every one of the possible combinations. The actionable tree in Figure 3.13 only produces a unique combination:

[SCAN cybercop os probe, DOS Land Attack, DNS named version attempt,
DNS EXPLOIT named overflow attempt, MYSQL root login attempt,
MYSQL show databases attempt]

We also call *episode* to each possible combination or yield of an actionable tree.

As we have indicated in the first Chapter of the thesis most of our knowledge about the situation at hand is neither absolutely true nor completely available but only true with some degree of certainty and partially available. So how to express uncertainty with actionable trees? We have provided actionable trees with a general framework for representing likelihood according to different theories.

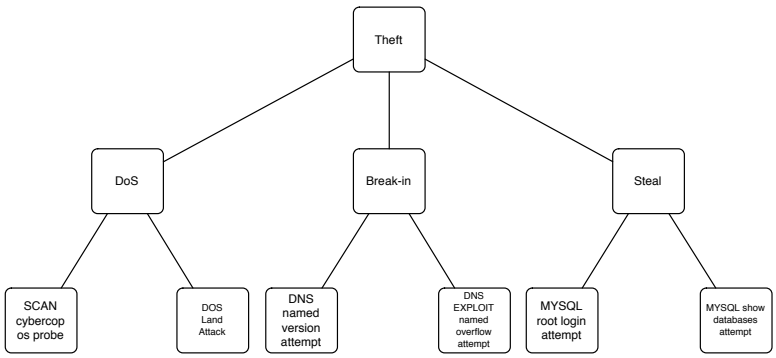


Fig. 3.13 An actionable tree of a Theft attack. This actionable tree is composed of the actionable trees of Figures 3.11(a), 3.12(a), and 3.12(b). An attacker launches a DoS against a machine running an IDS, subsequently breaks in to a DNS server, spawns a remote shell, and steals some information from a MySQL database.

3.4.4 Predictive Actionable Trees

A predictive actionable tree embodies an actionable tree in a representation that facilitates predictive inference. For this purpose we use a general model of likelihood that allows several schemes to model the strength of part-of relation. This model is based on associating a semiring structure with the relationships entailed by the arcs of actionable trees.

A semi-ring is defined as follows:

Definition 33 (Semiring) *A semiring S is an algebra $\langle L, \otimes, \oplus, 0, 1\rangle$ such that L is a set, $0, 1 \in L$ are constants, $\otimes$ is a multiplicative operation, and $\oplus$ is called an additive operation that satisfy the following conditions:*

1. *Multiplicative unit element:*

$$\forall a \in L, a \otimes 1 = a = 1 \otimes a \tag{3.4}$$

2. *Additive unit element:*

$$\forall a \in L, a \oplus 0 = a = 0 \oplus a \tag{3.5}$$

3. *Left and right annihilator:*

$$\forall a \in L, a \otimes 0 = 0 = 0 \otimes a \tag{3.6}$$

i.e., 0 is multiplicative absorbing element.

4. *Multiplicative associativity:*

$$\forall a, b, c \in L, (a \otimes b) \otimes c = a \otimes (b \otimes c) \tag{3.7}$$

5. *Additive associativity:*

$$\forall a, b, c \in L, (a \oplus b) \oplus c = a \oplus (b \oplus c) \tag{3.8}$$

6. *Additive commutativity:*

$$\forall a, b \in L, a \oplus b = b \oplus a \tag{3.9}$$

7. *Left and right distributivity:*

$$\forall a, b, c \in L, a \otimes (b \oplus c) = (a \times b) \oplus (a \times c) \wedge (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c) \tag{3.10}$$

Therefore $\langle L, \otimes, 1\rangle$ is a monoid and $\langle L, \oplus, 0\rangle$ is a commutative monoid. Moreover we define the additive operation to be *idempotent* over L:

$$\forall a \in L, a \oplus a = a \tag{3.11}$$

We also extend the additive operation to be used over sets of elements of L as follows[8]:

$$\forall a \in L, \sum(\{a\}) = a \tag{3.12}$$

$$\sum(\emptyset) = 0 \tag{3.13}$$

$$\sum(L) = 1 \tag{3.14}$$

$$\sum(\bigcup_{i \in I} L_i) = \sum(\sum_{i \in I}(L_i)) \tag{3.15}$$

Given a semiring S with the above additional characteristics we can compare different elements of the semiring using a partial order $\leq_S$ defined as follows:

$$\forall a, b \in L, a \leq_S b \leftrightarrow a \oplus b = b \tag{3.16}$$

The following theorem holds:

Theorem 2 *The relation $\leq_S$ on L defined as $a \leq_S b \leftrightarrow a \oplus c = b$ is a partial order (i.e., $\leq_S$ is reflexive, transitive, and antisymmetric).*

[8] A semiring so defined is called a *c-semiring* [BMR97].

Proof:

- $\forall a \in L, a \oplus a = a$ *given that* $\oplus(\sum)$ *is idempotent and therefore* $a \leq_S a$ *by definition of* $\leq_S$*. Thus,* $\leq_S$ *is reflexive.*
- *Assuming that* $a \leq_S b$ *and* $b \leq_S c$ *we have that* $a \oplus b = b$ *and* $b \oplus c = c$*. Therefore, using Eq. 3.8,* $a \oplus c = a \oplus (b \oplus c) = (a \oplus b) \oplus c = b \oplus c = c$*. Thus,* $\leq_S$ *is transitive.*
- *Assuming that* $a \leq_S b$ *and* $b \leq_S a$ *we have that* $a \oplus b = b$ *and* $b \oplus a = a$*. Therefore, using Eq. 3.9,* $a = b \oplus a = a \oplus b = b$*. Thus,* $\leq_S$ *is antisymmetric.*

The elements of L provide a measure of part-of strength ranging from 0 to 1. 0 indicates the lowest strength and 1 indicates the highest strength. The multiplicative operation computes the strength in part-of transitive closures whereas the additive operation computes the strength of multiples part-of relations. The intuition behind $a \leq_S b$ is that b is preferable.

Now we define formally a predictive actionable tree.

Definition 34 (Predictive Actionable Tree) *Given an alert model* $\mathbb{A} = \langle \vartheta, \Sigma \rangle$ *and a collection of alerts* $\Psi = \{\psi_1, \cdots, \psi_n\}$ *such that* $\psi_i \in \mathbb{A}$*, a predictive actionable tree is a 6-tuple* $\langle G, \mu, \tau, \kappa, \phi, \lhd, L \rangle$ *where* G*,* μ *,* τ*,* κ *and* $\lhd$ *are defined as in actionable trees and* L *and* ϕ *as follows:*

- L *is a likelihood model that assigns a measure of strength to the part-whole relation.* L *can be instantiated according to the models below* $D, T, P, PT, etc.$
- $\phi : E \rightarrow L$ *is a mapping that labels each arc* $e \in E$ *to a part-of strength in* L*.*

A number of models L could be used to instantiate the likelihood framework ranging from a simple deterministic model to uncertainty models such as Shafer-Dempster through combinations of probabilistic and temporal models. Take the following models as an example:

Deterministic This is the basic model. We could label arcs in an actionable tree using a deterministic model D instantiating the above likelihood framework as follows. $L = \{0, 1\}$, $\otimes = \wedge$, $\oplus = \vee$ and $\langle L, \wedge, 1 \rangle$ and $\langle L, \vee, 0 \rangle$ are respectively a monoid and a commutative monoid composing the semiring [KYY95]:

$$D = \langle \{0, 1\}, \wedge, \vee, 0, 1 \rangle \tag{3.17}$$

That is to say, the multiplicative operation is the Boolean $\wedge$ (**and**) whereas the additive operation is the Boolean $\vee$ (**or**) with the respective unit elements 1 and 0 and the order $0 \leq_D 1$. Following this model all the arcs of an actionable tree are labeled with 1 (since all the arcs labeled with 0 are purportedly not considered in the construction of the actionable tree). Figure 3.14 shows an example of a deterministic actionable tree. In this example the mere occurrence of one of the alerts allow us to infer the final presence of the complete sequence. Although a pure deterministic model would

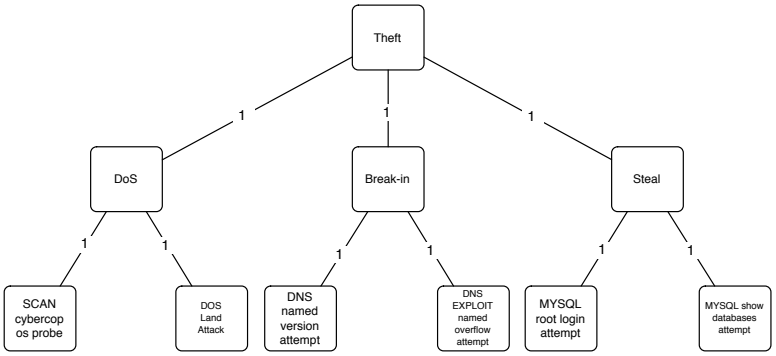


Fig. 3.14 A deterministic actionable tree.

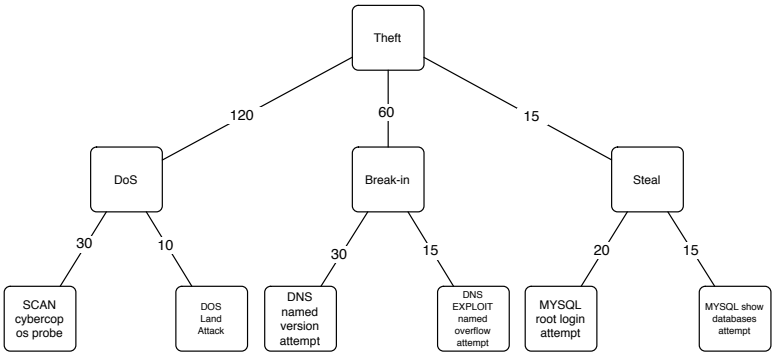


Fig. 3.15 A temporal actionable tree.

decrease the number of false negatives, the number of of false positives will become unmanageable since the presence of only an alert will fully activate the actionable tree.

Temporal In this case the multiplicative operation is the sum of times along the part-of transitive closure and 0 its unit element. The additive operation is the $\mathbf{min}$ operation over real numbers and its unit element is ∞. The order $\leq_T$ is the real numbers order. The temporal semi-ring is:

$$T = \langle [0, \infty[, +, \mathbf{min}, 0, \infty \rangle \tag{3.18}$$

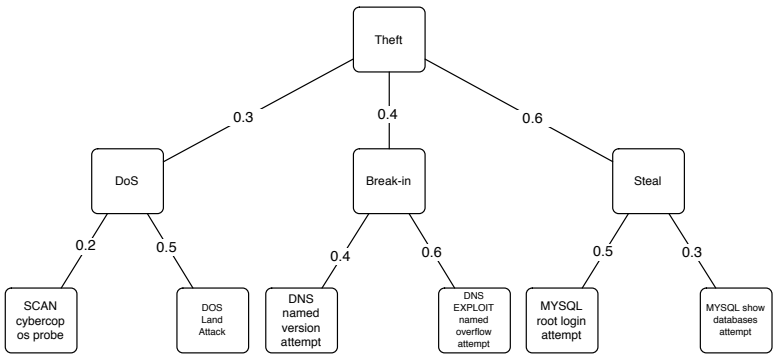


Fig. 3.16 A probabilistic actionable tree.

Intuitively, when the arcs of an actionable tree are labeled using this model the time units express the expected time of occurrence of the whole given the part. ∞ can be interpreted as that the part is unlikely to occur in a finite time whereas 0 indicates immediate ocurrrence. For example, Figure 3.15 depicts a predictive actionable tree using a temporal likelihood model. In this case, the presence of a SCAN cybercop os probe alert would allow us to infer that the occurrence of a Theft attack will take place in at most 150 seconds. If, additionally, we observed a DNS EXPLOIT named overflow attempt alert then we would conclude that a Theft attack will occur in only 75 seconds.

Probabilistic The likelihood framework could also be instantiated using the probabilistic model given by the following semi-ring:

$$P = \langle [0,1], \cdot, 1 - \prod_{i=1}^{k}(1 - l_i), 0, 1 \rangle \tag{3.19}$$

That is the multiplicative operation is the product of probabilities (i.e., $\otimes = \cdot$) and the additive operation is defined as:

$$l_1 \oplus l_2 \oplus \cdots \oplus l_k = 1 - \prod_{i=1}^{k}(1 - l_i) \tag{3.20}$$

The order $\leq_P$ is the order of the real numbers. Figure 3.16 shows the same actionable tree as Figure 3.15 but now using a probabilistic model. In this case, the presence of of a SCAN cybercop os probe alert would allow us to infer that the probability of occurrence of a Theft attack is 0.06. If, additionally, we observed a DNS EXPLOIT named overflow attempt alert then we would conclude that the probability of occurrence of a Theft attack is 0.2856, i.e., $1 - ((1 - (0.2 \cdot 0.3)) \cdot (1 - (0.6 \cdot 0.4)))$.

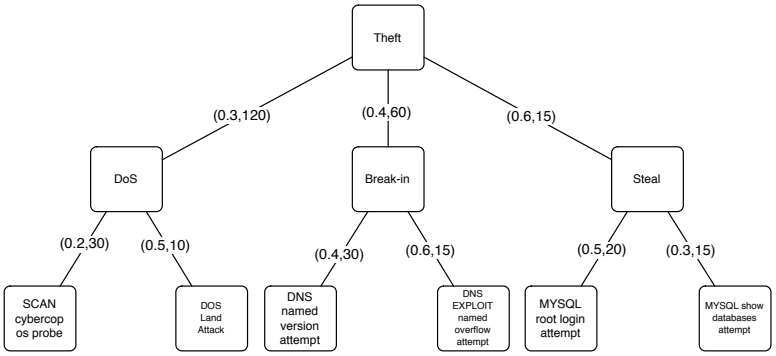


Fig. 3.17 A probabilistic-temporal actionable tree.

In Chapter 5, we will employ this probabilistic model as the underlying part-of strength model used by Ceaseless CBR processes to manage uncertainty about the completion of the sequential cases at hand.

Moreover, combinations of several semi-ring can also be used to instantiate the above likelihood framework. For example, we can consider the following combination of a probabilistic and temporal model.

Probabilistic-Temporal This model allows on to measure the part-of strength in terms of pairs (l_i, t_i) such that the first element l_i refers to the confidence on the occurrence of the relation and the second term t_i refers to expected time of occurrence. The composite model is given by the following semi-ring.

$$PT = \langle [0,1] \times [0,\infty[, \Big(\prod_{i=1}^{k} l_i, \sum_{i=1}^{k} t_i\Big), \\ (1 - \prod_{i=1}^{k}(1-l_i), min_{i=1}^{k}(t_i)), (0,\infty), (1,0)\rangle \quad (3.21)$$

Notice that in this case the multiplicative operation is a composition that inherits the multiplicative operations of the above probabilistic and temporal models.

$$(l_1,t_1) \otimes (l_2,t_2) \otimes \cdots \otimes (l_k,t_k) = (\prod_{i=1}^{k} l_i, \sum_{i=1}^{k} t_i) \quad (3.22)$$

Likewise the additive operation is computed based on the above models.

$$(l_1, t_1) \oplus (l_2, t_2) \oplus \cdots \oplus (l_k, t_k) = \left(1 - \prod_{i=1}^{k}(1 - l_i), min_{i=1}^{k}(t_i)\right) \tag{3.23}$$

Figure 3.17 shows an example of a predictive actionable tree that instantiates this model. In this case, the presence of of a SCAN cybercop os probe alert would allow us to infer that the probability of a Theft attack occurring in the next 150 seconds is 0.06. If, additionally, we observed a DNS EXPLOIT named overflow attempt alert then we would conclude that the probability of occurrence of a Theft attack in the next 75 seconds is 0.2856.

The order in this model is given by the following expression:

$$(l_1, t_1) \leq_{PT} (l_2, t_2) \leftrightarrow l_1 \leq_P l_2 \wedge t_1 \leq_T t_2 \tag{3.24}$$

Next, we will see how to assign a particular strength to each part-of relation provided by an actionable tree.

3.4.4.1 Part-of Strength

The general semi-ring structure introduced above allow us to use distinct likelihood models using the same hierarchical structure provided by actionable trees. Let's see now how we assign a particular part-of strength (likelihood) to each and every one of the part-of relations provided by actionable trees (i.e., l_i's in the above models). We need to introduce first the notion of *support*[9].

Using the definition of occurrence (Definition 14), the support of a sequence of alerts can be defined as follows:

Definition 35 (Support) *Given an alert model $\mathbb{A} = \langle \vartheta, \Sigma \rangle$ and a sequence of alerts $\vec{S}_2$ in $\mathbb{A}$, the support* $\mathbf{s}$ *of a sequence of alerts $\vec{S}_1$ is defined as the probability that a randomly-chosen window of size t in $\vec{S}_2$ contains an occurrence of $\vec{S}_1$*[10].

$$\mathbf{s}(\vec{S}_1, \vec{S}_2, t) = \frac{\textit{\#windows of size } t \textit{ where } \vec{S}_1 \textit{ occurs}}{\textit{\#windows in } \vec{S}_2} \tag{3.25}$$

The part-of strength or confidence of a part-of relation is defined in terms of the support of the part and the support of the whole.

[9]Notice that the support-strength framework used for capturing the dependence among parts and wholes carried out by actionable trees can be seen as the support-confidence framework used by *association rules* to model dependences among items represented in a database or events into a sequence[AS95, MTV97, Ada00, ZZ02].

[10]Notice that size t can refer either to a temporal or to a spatial dimension. For example in a sequence of 31483 alerts that extends over more than 111 days (such our Rustoord data-set has) there are either 315 non-overlapping windows of size 100 alerts or 2680 time-based windows of size 3600 seconds. See next Section for further details.

Definition 36 (Part-of Strength) *Given an alert model $\mathbb{A} = \langle \vartheta, \Sigma \rangle$ a sequence of alerts $\vec{S}_2$ in $\mathbb{A}$, a window of size t, and two sequential cases C_1 and C_2 in $\mathbb{A}$, such that $C_1 \lhd^* C_2$ the strength of an part-of relation $l_{C_1 \lhd C_2}$ is defined as the support of the occurrence of the whole over the support of the part.*

$$l_{C_1 \lhd^* C_2} = \frac{\mathbf{s}(C_2, \vec{S}_2, t)}{\mathbf{s}(C_1, \vec{S}_2, t)} \tag{3.26}$$

As we mentioned before, we will see in Chapter 5 how Ceaseless CBR uses sequential cases represented by means of predictive actionable trees (that use a probabilistic model) to compute the likelihood of the occurrence of an attack (the whole) given some of the intrusion alerts that it generates (its parts). Predictive actionable trees have been devised with the main purpose in mind of providing a measure of confidence we have in the occurrence of a sequence of alerts given a number of observations. Said differently, they provide a measure of the strength with which the existence of the whole can be determined in terms of the existence of some of its parts.

In our domain the intuition behind part-of strength is that if during an attack a sensor evokes the alerts in C_1 then it is also likely to evoke the alerts C_2 in a period of time that does not exceeds t. Notice that part-of strength assignations could also be done by a human expert. However, part-of strengths not only vary depending on the concrete scenario but also over time within the same scenario. Thus, human expert assessment can be significantly more costly and less accurate than automatic data-driven assessment. Therefore, an actionable tree allows one to engage inference upon the occurrence of alerts that are subsumed by alerts appearing in its root nodes. In the construction of an actionable tree, alerts in its roots follow a process of abstraction in order to increase the number of alerts that will engage inference. The constraints imposed by non-root nodes allows one to reduce the number of alerts that produce subsequent inferences.

Each predictive actionable tree has associated a degree of belief (likelihood) of occurrence in terms of the alerts observed so far. Such degree of belief is updated as long as more alerts are observed. The inference model for belief updating of actionable trees is a special case of a *poly-tree model*. A poly-tree model or simple a poly-tree is a singly connected *Belief Network* (BN) (i.e., it contains no undirected loops). In other words, a BN whose underlying undirected graph has no cycles [Pea88, Dec98, KZ03]. The inference model of an actionable tree is a poly-tree where each observed node (observable symptom events) has no parents and both observed nodes and latent nodes have only one child. Moreover, in an actionable tree there is only a node whose degree of belief is of interest (i.e., the crown). The main particularity of poly-trees, a therefore of actionable trees, is that inference can be done in linear time. (i.e., the time and space complexity of exact inference in poly-trees is linear in the size of the network [RN03]).

The on-line analysis of temporally-evolving sequences of alerts requires to select an adequate window model to compute aggregates of interest over a number of alerts. Next section examines several window models.

3.5 WINDOW MODEL

This section discusses four different window models for aggregate analysis of sequences. In Chapter 6, we will experimentally analyze how they intervene in the performance of our model. Our objective is to guarantee the robustness of the CBR model proposed independently of the window model chosen. In some sense, a concrete window model determines how large is the context is considered each time that inference is invoked upon the arrival of new events. Window models vary according to size of the window. The size of a window can be measured in terms of the number of elements considered or in terms of the time interval that the window represents.

Given a sequence of n alerts $\vec{S} = [\psi_{1,w_1}^{t_1}, \psi_{2,w_2}^{t_2}, \cdots, \psi_{i,w_i}^{t_i} | \psi_{i+1,w_{i+1}}^{t_{i+1}}, \cdots, \psi_{n,w_n}^{t_n}]$ where $i \leq n$, t_i superscripts represent each alert timestamp, w_i represents the weight (importance) that each alert will receive (these subscripts are omitted when they are equal for all alerts). Finally, $|$ establishes the frontier between the alerts analyzed so far and pending alerts—alerts that were received but have not been considered for analysis yet. We will contemplate the following models:

Alert-driven The first model consists of a window of size one. That is to say, not to use a window model at all. Thus, a system following this model operates on an alert-driven basis and its inference is engaged upon a new event arrival. This model is not subject to the inaccuracies resulting from an incorrect specification of the size of the window (in terms of space or time) [SS03] and can be easily adapted to perform *point monitoring*—i.e., only the latest element in the sequence is of interest [ZS03b]. We will denote by w_j^e the j^{th} alert of $\vec{S}$ (i.e., $\psi_j^{t_j}$). The next alert to analyze at a given time t is computed as follows:

$$w^{e(t)} = \begin{cases} \psi_{i+1}^{t_{i+1}} & \textbf{when } i < n \\ \text{wait until the next alert arrival } \psi_{n+1}^{t_{n+1}} & \textbf{otherwise} \end{cases} \tag{3.27}$$

Landmark Window This model establishes a specific point-in-time λ named the landmark and then returns alerts between the landmark and the current time t.

$$w_\lambda^{l(t)} = [\psi_j^{t_j}, \cdots, \psi_k^{t_k}] : t_j \geqslant \lambda \ \wedge \ t_k \leq t \tag{3.28}$$

In this model the sequence of alerts to analyze becomes longer and longer in every interation. However, the landmark may be periodically changed increasing it in t_λ time units. In this case we would obtain windows of duration t_λ, i.e., $[t - t_\lambda, t_\lambda]$ from the immediately-preceding landmark $t - t_\lambda$ until the current point-in-time t. Then each window, for example, aggregates alerts on a monthly or daily or hourly basis, etc. Nevertheless, the number alerts aggregated by each window may vary over time depending on the alert inter-arrival frequency. In this case, the next alerts to analyze at a given time t are given by:

$$w_{t_\lambda}^{l(t)} = [\psi_j^{t_j}, \cdots, \psi_k^{t_k}] : t_j \geqslant t - t_\lambda \ \wedge \ t_k \leq t \tag{3.29}$$

Sliding Windows This model uses a moving window of fixed size that always considers the most recent alerts in the sequence. Depending on how its size is determined (in terms of space or time) the following two models are possible:

Space-based Sliding Window A window in this model consists of the k most recent alerts to arrive.

$$w_k^{sbw(t)} = [\psi_i^{t_i}, \cdots, \psi_l^{t_l}] : l = \mathbf{min}(i + k, n) \tag{3.30}$$

Time-based Sliding Window A time-based sliding window of duration t_w considers those alerts that arrived no more than t_w time units ago.

$$w_\iota^{tbw(t)} = [\psi_i^{t_i}, \cdots, \psi_l^{t_l}] : t_l = t_i + t_w \tag{3.31}$$

Other type of sliding window that deserves special attention is *recurrent sliding window* (i.e., with overlap > 0). In this case, a sliding window with a smaller size is running again within each window. This model is specially useful for learning and mining purposes. When the overlap between two consecutive windows is 0 we will refer to these window models as space-based and time-based respectively.

Damped Window This model considers the most recent alerts the most interesting. Thus in a damped window model the weights corresponding to alerts into the past are decreased exponentially. An example of a damped space-based sliding window model will be:

$$w_k^{dsbw(t)} = [\psi_{i,w*p_i}^{t_i}, \cdots, \psi_{l,w*p_l}^{t_l}] : \\ l = \mathbf{min}(i + k, n) \wedge p_i < \cdots < p_l \wedge \sum_i^l p_i = 1 \tag{3.32}$$

Sometimes the context required to properly analyzed a sequence is greater than the one provided by a specific window model. For example, an attack (pattern) to be detected could appear partially divided into several consecutive windows impeding its recognition but causing the same malicious effect. In these cases elastic window models can solve this kind of problems by extending the size of the window as far as needed. To select a proper window model and its correct size constitutes by itself an arena of research. Our approach solves this problem by maintaining a working memory where a number of *pending alerts* in the form of *case activations* are kept over several iterations such as we will see in Chapter 5 of the thesis. In the experiments described in Chapter 6 we will only consider the space-based, time-based, and alert-driven models.

3.6 SEQUENTIAL CASES RECAP

This Chapter introduced the terms needed to know to read the rest of the thesis. We have addressed the conceptual and representational issues concerned with the knowledge structures that allow a case-based reasoner to analyze unsegmented sequences of complex objects in terms of sequential cases. We have described an unified data model to represent alerts and introduced the notion of sequential case. A sequential case is a compositional case where additionally a temporal order is established among their parts. We have proposed to represent sequential cases by means compositional hierarchies. Particularly, we have introduced *actionable trees* to represent sequences of complex objects and reason in terms of partially observed sequences. An actionable tree is a a Multi-Rooted Acyclic Graph (MRAG) with the semantics that (i) roots symbolize alerts (observable symptom events), (ii) intermediate nodes (in the trunk and crown) embody sequences of alerts (chunks), and (iii) the arcs represent part-whole relationships as well as the likelihood of occurrence of the whole (sequence) given the part (subsequence). We used a general model of likelihood that allows several schemes (deterministic, probabilistic, temporal, etc) to model part-of strengths. This model is based on associating a semiring structure with the part-whole relationships entailed by the arcs of actionable trees. In Chapter 5, we will see how Ceaseless CBR uses sequential cases represented by means of predictive actionable trees that use a probabilistic model to compute the likelihood of the occurrence of an attack (the whole) given some of the intrusion alerts that it generates (its parts).

Next Chapter is devoted to a dynamic sequence similarity that allows a case-based reasoner to retrieve relevant sequential cases.

4

A Dynamic Sequence Similarity

This Chapter introduces a similarity measure to compare the sequences of alerts yielded by actionable trees (yields or episodes) against the current window of alerts in the alert stream. Our similarity measure has two outstanding characteristics: continuously adaption to data seen so far and promotion of rareness. We first overview in Section 4.1 some concepts about similarity. Then, we introduce in Section 4.2 the notion of sequence similarity and overview two approaches for its computation—*edit distances* and *alignments*. Then, in Section 4.3 we formally define our dynamic sequence similarity as a semi-global alignment and describe several of its components: (i) a dynamic subsumption scoring scheme based on the reciprocal of *Odds* that uses a taxonomic hierarchy to properly compute the score given to each pair of alerts; (ii) an operation of abduction that automatically injects an alert in arbitrary positions of the alert stream; (iii) a operation of neglection that ignores an alert at an arbitrary position in the alert stream; and (iv) a system of dynamic programming recurrence equations that returns the score of the optimal alignment between a suffix and a prefix of the respective sequences being compared. These two operations allow us to deal respectively with lost and spurious alerts in the alert stream. Section 4.4 illustrates how our dynamic sequence similarity works using a simple example. Finally, in Section 4.5 we show how those components behave over time using our real-world data-sets.

4.1 INTRODUCTION

At the heart of a case retrieval system is always the computation of a similarity measure between a new case and previous cases in a case base. A similarity measure provides a useful approximation of human-determined similarity [AAD03]. Similarity measures have been largely studied both within and outside the CBR research community. As a matter of fact, over the years, similarity measures have been invented and reinvented a number of times at a variety of fields. In CBR, similarity measures have been categorized according to:

- several models [Ric92, OB97]: either *absolute*, *relative* or *metric* similarity; and
- several approaches to their realization [Por89, OB97]: either *representational* or *computational*.

On the one hand, an absolute similarity function judges two terms to be either similar or not similar whereas a relative similarity assigns a degree of similarity to each pair of terms. Often this degree or score is normalized and restricted to lie in some interval (e.g., [0, 1]). Sometimes it is an arbitrary value. Absolute similarity functions are simple but counterintuitive since people is commonly used to deal with degrees of similarity while relative similarity functions are more intuitive but more complex too since arbitrary degrees of similarity have to be quantified. Metric similarity functions have been proposed to circumvent both absolute and relative similarity drawbacks. A metric similarity is a lattice-valued function that generalizes over absolute and relative similarity. A metric similarity function returns values (degrees of similarity) from a total order rather than real numbers and in addition provides a lattice that makes such values comparable [OB97].

On the other hand, in CBR systems that follow a representational approach the mechanism responsible for seeing similarity is hard-coded (i.e., similarities are computed at case base update time rather than on every retrieval [OB97]) whereas CBR systems that implement a computational approach compute similarity on each retrieval. We will introduce in this Chapter a relative similarity for CBR systems that follow a computational approach.

When dealing with individual objects, where each object is made up of a number of numerical, qualitative, and structured values, there are a number of well-known similarity measures (such as Euclidean, Manhattan, L'Eixample, etc) that, the great majority, have their roots in the well-established Tversky's theory of similarity [TK70]. For instance, following Tversky, we can conceptualize the similarity of two alerts as the combination of pairwise comparisons of the individual feature-structure-values in both alerts as follows [Tve77]:

Definition 37 (Alert Similarity) *Given the following alert signature* $\Sigma = \langle \mathcal{S}, \bot, \mathcal{F}, \preceq \rangle$*, the similarity of two alerts* $\psi_i \sim \psi_j$ *is computed as follows:*

$$\psi_i \sim \psi_j = \gamma f(\pi(\psi_i) \cap \pi(\psi_j)) - \alpha f(\pi(\psi_i) - \pi(\psi_j)) - \beta f(\pi(\psi_j) - \pi(\psi_i))$$

where:

- $\pi(\psi) \subset \mathcal{F}$ *is the set of all features of alert* ψ.
- $\pi(\psi_i) \cap \pi(\psi_j)$ *is the set of common features to events* ψ_i *and* ψ_j.
- $\pi(\psi_i) - \pi(\psi_j)$ *is the set of features that are present in alert* ψ_i *but not in alert* ψ_j*, i.e., the distinctive features of* ψ_i.
- $\pi(\psi_j) - \pi(\psi_i)$ *is the set of features that are present in alert* ψ_j *but not in alert* ψ_i*, i.e., the distinctive features of* ψ_j.
- f *is a measure that represents the salience of a set of features.*
- α *indicates the relevance of the distinctive features of alert* ψ_i.
- β *indicates the relevance of the distinctive features of alert* ψ_j.
- γ *indicates the relevance of the common features of both alerts.*

Likewise, the computation of the similarity of two cases c_1 and c_2 can be conceptualized as the combination of pairwise comparisons of the individual feature-structure-vales in both cases [OB97]. When cases are represented by feature terms, their anti-unification also gives a similarity metric with a lattice based on subsumption [Pla95]. See Gebhardt and Jantke's respective works for other approaches exploiting geometrical relations between cases or other nonstandard concepts of similarity in CBR that take structural properties into account, with the aim of making more flexible and expressive comparisons [Jan94, Geb96]. As we will see later on, we measure the similarity of two alerts in terms of their respective sort's position in a taxonomic hierarchy and their respective frequency of occurrence. The rationale behind our approach is clear. First, to gain efficiency and, second, to reduce complexity compared with the above definition.

We are concerned with the problem of defining similarity not only between individual objects but also between sequences of objects (e.g., where each object and additionally has an associated time of occurrence). As pointed out in Chapter 2, computing similarity of sequences of cases is much less developed than computing similarity between individual cases. Without a shadow of a doubt, one of the most deployed techniques for comparing sequences is *Levenshtein* distance (and the like) [Lev66]. By far, both Sankoff and Kruskal's book and Gusfield's book are the key sources for finding throughly discussions about all the problems surrounding sequence comparison [SK83, Gus97]. In the AI literature, most work in sequences is based on the Schank and Abelson concept of scripts [SA77]. A number of non-linear methods such as genetic algorithms and neural networks have also been proposed for sequence comparison in a variety of domains such as international event analysis [Sch00]. See Chapter 2 for more references.

However, almost all the above similarities and techniques can be considered as non-adaptive. Broadly speaking, they return the same score independently of the

context, the history of the system, or the specific point in time at which a comparison is occurring. In this Chapter, we propose and describe a similarity measure for computing the similarity of two sequences of objects that continuously adapts to data seen so far. We argue that the context and the history in which comparison occurs is fundamental to pinpoint some aspects of interests (such as rareness) of the sequences being compared. We take a simple approach to solving this problem. We propose a semi-global alignment that uses a time-varying subsumption scoring scheme. This scoring scheme uses a taxonomic hierarchy and the frequency of occurrence to compute the score for each pair of terms in the semi-global alignment. As we will see, to promote the rareness of terms being compared the score is computed using the reciprocal of the *Odds*. Additionally, we have defined two operations: *abduction* and *neglection* that allow us to deal respectively with lost and spurious alerts in the alert stream. Another feature that distinguishes our approach is the fact that normalized and non-normalized versions behave differently. The non-normalized similarity computation gives preference to rare occurrences of alerts that are more indicative that something new (unknown) is undergoing. When normalized the similarity lies between 0 and 1 and gives preference to complete occurrences. That is, sequences of alerts that are completely included in the alert stream are promoted (ranked first) versus those that only occur partially.

Our sequence similarity can be seen as a dynamic similarity. Keane et al proposed *dynamic similarity* as an alternative to static similarity theories arguing that past, present and future contribute to the perceived similarity of the things [KS01]. The idea behind dynamic similarity is treating the computation of similarity from a process-oriented perspective rather than as a process acting over static representations. That is, in situations where the mental representations of two events are similar the whole cognitive processing context—past, present, or prospective—contributes to the conceived similarity of the events. However, this elegant concept lacks the operational model that allows one to compute the similarity between two sequences of events. In this Chapter we try to provide an operational model. Next, Section succinctly describes some alternatives to compute the similarity between two sequences.

4.2 SEQUENCE SIMILARITY

Computing the similarity between two sequences can be interpreted like the search of evidence that both sequences share a common serial structure. For example, in biological sequence analysis comparing two sequences is looking for the testimony that they have diverged from a common ancestor through continuos mutation and selection processes [DEK98]. In geology the sequences being compared represent the stratigraphic structure of core examples [EGG93]. In our case, we look for the evidence that some alerts in the alert stream and a sequential case are derived from a common attack pattern (perhaps altered using new undetectable actions or exploiting new vulnerabilities).

Generally speaking, there are two kinds of differences between two sequences:

- those that arise due to the differences between their serial structure; and
- those that arise due to the individual differences among the elements of each sequence.

Using the terms of the alignment theory of similarity [MG01] it could be said that two sequences $\vec{S}_1$ and $\vec{S}_2$ always share a collection of:

Commonalities Comparable elements that appear at the same position in both sequences:

$$\mathcal{C}(\vec{S}_1, \vec{S}_2) = \{(\vec{S}_1[i], \vec{S}_2[i])\}_{0<i\leq min(|\vec{S}_1|,|\vec{S}_2|)} : \vec{S}_1[i] \sim \vec{S}_2[i] \tag{4.1}$$

Alignable differences Comparable elements that occur in both sequences but at different position:

$$\mathcal{A}(\vec{S}_1, \vec{S}_2) = \{(\vec{S}_1[i], \vec{S}_2[j])\}_{0<i\leq|\vec{S}_1|,0<j\leq|\vec{S}_2|,i\neq j} : \vec{S}_1[i] \sim \vec{S}_2[j] \tag{4.2}$$

Non-alignable differences Non-comparable elements that appear in one sequence but not in the other:

$$\mathcal{N}(\vec{S}_1, \vec{S}_2) = \{\vec{S}_1[i]\}_{0<i\leq|\vec{S}_1|} : \forall j\ \vec{S}_1[i] \nsim \vec{S}_2[j] \cup \{\vec{S}_2[j]\}_{0<j\leq|\vec{S}_2|} : \forall i\ \vec{S}_2[j] \nsim \vec{S}_1[i] \tag{4.3}$$

Computing the similarity between two sequences entails the definition of a scoring model that determines how commonalities, alignable differences and non-alignable differences are weighted when a comparison is performed. Moreover, given that depending on how both sequences are aligned the commonalities and alignable differences differ, we also need to establish a strategy for aligning both sequences. Next, we overview two possible strategies: *edit distances* and *alignments*.

4.2.1 Edit Distances

Edit distances and *alignments* have been used in a multitude of fields of study such as biology, geology, telecommunications, speech recognition, etc to compare sequences [Kru83, EGG93, Gus97, MR97, DEK98, Ron98]. An edit distance uses dynamic programming to minimize the cost of the *operations* needed to transform a sequence into another. Those operations are able to eliminate non-alignable differences and to align alignable differences. We formally define the distance between two sequences of alerts as follows:

Definition 38 (Sequence Edit Distance) *Given the following alert signature* $\Sigma = \langle \mathcal{S}, \perp, \mathcal{F}, \preceq \rangle$*, the distance between two sequences of alerts* $\vec{S}_1$ *and* $\vec{S}_2$ *can be computed as follows:*

$$\vec{S}_1 \sim_D \vec{S}_2 = \ min_{T \in \mathcal{T}^*}\{C(T) - \sum_{i=1}^{|\vec{S}_2|}(\vec{S}_1^T[i] \sim \vec{S}_2[i])\} \tag{4.4}$$

where:

- $\mathcal{T}^*$ *is the language of transformations, usually over the alphabet* {*deletion (*`D`*), insertion (*`I`*), match (*`M`*), or substitution (*`S`*)*}.
- $T \in \mathcal{T}^*$ *is a sequence of transformations that describes how* S_1 *can be transformed into* S_2
- $\vec{S}_1^T$ *is the result of applying the transformations* T *on* $\vec{S}_1$.
- $C(T)$ *is the cost of the transformations (i.e.,* $\sum_{i=1}^{|T|} C(T[i])$.
- $\sim$ *is the similarity between alerts of Definition 37.*

Notice that the above definition is a distance and not a similarity and therefore the higher the score the more dissimilar are the terms being compared. The above definition has associated a key issue: how to weigh up the cost of transformations (operations). Commonly, only a small set of edit operations is used such as *substitution*, *insertion*, *deletion*, etc. The cost of each operation can be the same or distinct depending on the application. A common assignation of cost is -1 for insertions, deletions, and substitutions and 0 for matches. The cost can also vary depending on the specific element of the alphabet that is being edited (inserted, deleted, substituted, etc). For example, in computational biology, alphabet-weight edit distances use scoring matrices to differentiate the cost of replacement between different elements. However, the most common is that each operation has an associated cost that does not varies over time and is independent of the elements involved as input in the operation. In our approach, on the contrary, the cost of each operation not only depends on the pair of elements being compared but also changes over time depending on the own history of the system that makes the comparisons.

4.2.2 Alignments

Another approach to estimate the similarity between two sequences is using an *alignment*. Alignments are used in molecular biology instead of edit distances to compute similarity measures between strings of DNA, proteins, etc. We have already introduced formally the notion of alignment between sequences of alerts in Definition 13. Once an alignment between two sequences has been established, dynamic programming is employed to find the score of the alignment that maximizes the similarity (i.e., the optimal alignment). Alignments follow a similar dynamic programming formulation to edit distances. However, an alignment tries to maximize the scoring given to commonalities between both sequences. That is, instead of measuring

the cost of transformations we need look for the optimal alignment that maximizes the contribution of commonalities. Usually a scoring matrix is used to determine the score given two every possible pair of matching elements. This matrix does not vary over time. For example, in biology a number of sensitive scoring schemes have been developed based on known sequences where the probabilities of mutation can be computed empirically for example BLOSUM, PAM or BLAST[Gus97, DEK98]. Alignments that try to match a short sequence (a subsequence of a long a one) against a long one are called global. Those alignments that match a complete sequence with part of the other are called semi-global alignments. Finally, the alignments that only try to match pieces of the two sequences are called local alignments. As we see below, our similarity measure is a semi-global alignment that seeks the optimal alignment between the suffix and the prefix of the sequences of involved. Moreover, in our approach the scoring matrix used is computed dynamically since it changes over time.

4.3 DYNAMIC SEQUENCE SIMILARITY

Dynamic sequence similarity is a function S, denoted in infix notation by $\sim_S$, that produces a number expressing the degree of similarity between two sequences of alerts. The criteria of closeness between two sequences used to computed S is based on four basic components:

1. A dynamic subsumption scoring scheme that dynamically computes the score given to each pair of aligned alerts.

2. An operation of abduction that enables the generation of new alerts in the alert stream.

3. An operation of neglection that ignores alerts from the alert stream.

4. A collection of dynamic programming recurrence equations that allows one to find efficiently the score of the optimal alignment.

As we have said before, we seek the optimal alignment between the suffix of the first sequence given as input (that we refer indistinctly as $\vec{S}_1$ or alert stream) and the suffix of the second sequence (that we refer indistinctly as $\vec{S}_2$ or sequential case). The motivation for this specific alignment is that we are interested in anticipating as much as possible the detection of a sequential case. Thus, if we recognize the prefix of a sequential case then we are able to immediately predict its suffix.

4.3.1 Dynamic Subsumption Scoring Scheme

We use a dynamic subsumption scoring scheme as an efficient way to compute the likeliness of two alerts and obtain the score for the subsumption replacements (the score given to alerts that are aligned together). This subsumption scoring scheme

tries to maximize the score given to the subsumption of rare alerts. The score given to each pair of alerts is based on the following assumptions. The score of aligning like alerts should be different to the score of aligning unlike alerts. Likewise, the score of aligning frequent alerts should be different to the score of aligning rare alerts.

We define a dynamic subsumption scoring scheme as scoring scheme that given a taxonomic hierarchy employs the subsumption relation and the frequency of occurrence of each sort to compute an score that measures the relatedness or similarity of two alerts that are aligned. This score will vary over time as long as the frequency of sorts changes. Our dynamic subsumption scoring scheme receives as input two alerts (or their sorts directly). If the first alert does not subsume the second then the score returned is $-\infty$ (i.e., the alerts are not related at all)[1]. But when it does (the first alert subsumes the second) it returns a score that is based on the reciprocal of the *Odds*[2]. That is to say, when the Odds of a subsumption are high the scoring returns a low value and when the Odds are low the scoring returns a high value. This behavior tries to leverage the importance of the occurrence of rare alerts. Formally:

Definition 39 (Dynamic Subsumption Scoring Scheme) *Given the following alert signature $\Sigma = \langle \mathcal{S}, \bot, \mathcal{F}, \preceq \rangle$, a subsumption scoring scheme $\mathcal{M}$ is a square $|\mathcal{S}| \times |\mathcal{S}|$ matrix such that $\mathcal{M}_{i,j} = \begin{cases} \frac{1-q_i}{q_j} \text{ if } i \preceq j \\ -\infty \text{ otherwise} \end{cases}$ where q_i and q_j are respectively the frequencies of the alerts in the alert stream whose sorts are $i \in \mathcal{S}$ and $j \in \mathcal{S}$.*

Notice that we have measured the similarity of two alerts in terms of their respective sort's position in a taxonomic hierarchy and their respective frequency of occurrence. The rationale behind our approach is clear. First, to gain efficiency and, second, to reduce complexity:

1. The computation of subsumption that we realize only takes linear time (using a subsumption matrix) whereas the computation of the similarity according to Definition 37 is polynomial in the number or attributes or even exponential depending on the complexity (e.g., sets) of the values of each attribute.

2. We only require a taxonomic hierarchy and the frequency of occurrence of each sort rather than demanding to parameterize (or learn) a number of artificial values (e.g., α, β, and so on) as needed in other well-known aforementioned similarity measures.

Next we define an operation of abduction on the alert stream. This operation is mainly oriented to provide a method resilient to noise—lost alerts.

[1] Notice our dynamic subsmption scoring scheme is asymmetric.
[2] The Odds in favor of an event e are the number of ways the event can occur compared to the number of ways the event can fail.

4.3.2 Abduction

We define an operation of abduction such that **Abduction**($\vec{S}$, ψ, i) injects an alert ψ in alert stream $\vec{S}$ at position i. When aligning two sequence of alerts, this operation inserts missing alerts so that an alignment can take place. Following information theory we can say that abducting an alert whose sort occurs frequently in the alert stream should cost less than abducting a rare alert and that abducting a more general alert should be more costly than a more specific one. Therefore, we compute the cost of an abduction using two parameters:

1. the α-rarity of the alert that is being abduced; and
2. the place that the sort of the alert ($root(\psi)$) occupies in the hierarchy of sorts $\mathcal{S}$.

The cost of abduction is therefore given by the equation:

$$C^{a}(\psi) = - \sum_{\psi' \in \mathcal{S}: root(\psi) \sqsubseteq \psi'} \rho_{\alpha}(\psi') \tag{4.5}$$

where $\rho_{\alpha}(\psi)$ is the α-rarity of alert ψ that is defined as:

$$\rho_{\alpha}(\psi) = \frac{\alpha\text{-rare}(\psi)}{\sharp distinct} \tag{4.6}$$

.

We say that alert ψ is α-rare if its sort appears exactly α times in alert stream. $\sharp distinct$ denotes the number of different sorts in the alert stream [DM02a].

This operation of abduction automatically generates virtual alerts while computing the similarity of the alert stream and a sequential case. These virtual alerts can be seen as the automatic generation of hypotheses that will entitle plausible explanations from the current alert stream [GG01, CM02]. This special feature of our approach is useful for dealing with unobserved actions (i.e. missing alerts that our sensors are not able to detect because the corresponding signature has not been published yet or because the intruder is using a new and unknown vulnerability or simply an alteration of a known attack). Moreover, abduced alerts also allows one to predict alerts that are likely to occur.

4.3.3 Neglection

We define an operation of neglection such that **Neglection**($\vec{S}$, i) ignores an alert at position i from alert stream $\vec{S}$. We compute the cost of the neglection $C^{n}(\psi)$ of an alert ψ as the inverse of its α-rarity:

$$C^{n}(\psi) = -\rho_{\alpha}(\psi)^{-1} \tag{4.7}$$

This operation is useful given that we deal with coincidental or parallel sources and thus we have to count on intervening non-matching elements that need to be interpreted as noise. We define using the subsumption scoring scheme and abduction

and neglection costs that we introduced above the similarity between two sequences as follows.

4.3.4 Dynamic Programming Recurrence Equations

We define the dynamic sequence similarity $S \sim S$, the similarity between sequences of alerts, as the scoring of the optimal (semi-global) alignment between between the suffix of the first and the prefix of the second. Formally:

Definition 40 (Dynamic Sequence Similarity) *Given an alert model $\mathbb{A} = \langle \vartheta, \Sigma \rangle$ and let $\vec{S}_1$ and $\vec{S}_2$ be two sequence of alerts in $\mathbb{A}^*$. The similarity between $\vec{S}_1$ and $\vec{S}_2$ is the score of the optimal alignment between a suffix of $\vec{S}_1$ and a prefix of $\vec{S}_2$:* $\vec{S}_1 \sim_s \vec{S}_2 = max_{1 \leq j \leq |S_2|} S(|\vec{S}_1|, j)$

The score of the optimal alignment is computed based on the following dynamic programming recurrence relation:

$$\begin{array}{lll} S(0,0) & = 0 & \\ S(i,0) & = S(i-1,0) & \\ S(0,j) & = S(0,j-1) + C^a(\vec{S}_2[j]) & \\ S(i,j) & = max \left\{ \begin{array}{ll} S(i-1,j) & + C^n(\vec{S}_1[i]) \\ S(i,j-1) & + C^a(\vec{S}_2[j]) \\ S(i-1,j-1) & + \mathcal{M}(\vec{S}_1(i), \vec{S}_2(j)) \end{array} \right. & \end{array} \tag{4.8}$$

Where C^n and C^a are respectively the costs of neglecting and abducing the corresponding alert and $\mathcal{M}$ is the dynamic subsumption scoring scheme defined above. The general condition will select in function of the corresponding costs to neglect an alert in the alert stream, to abduce an alert from the sequential case of interest or to subsume an alert of the sequential case with the corresponding alert in the alert stream. Notice that the second base condition of the recurrence relation $S(i,0) = S(i-1,0)$ states that we will neglect all the alerts in the alert stream whereas the second base condition $S(0,j) = S(0,j-1) + C^a(\vec{S}_2[j])$ means that all the alerts that compose the sequential case will be abduced in the alert stream. Given our special alignment, we do not assign any cost for neglecting alerts at the beginning of the alert stream and at the end of the sequential case. Thus, the second base condition does not assign any cost at all. We look for the optimal alignment through the last row of the score matrix in the similarity definition [Gus97] (i.e., $max_{1 \leq j \leq |S_2|} S(|\vec{S}_1|, j)$).

These operations compute the similarity in time $O(mn)$ where m and n are the lengths of the two sequences compared (i.e., $m = |\vec{S}_1|$ and $n = |\vec{S}_2|$). They also require $O(mn)$ space to trace back the elements that were aligned and took place in the scoring computation.

Two important aspects need to be taken into account from the above formulation.

1. First, our dynamic sequence similarity gives an arbitrary quantitative indication of the relatedness of two sequences of alerts that ranges from 0 to ∞ (i.e.,

a	A	B	A	A	B	C	D	E	C	C	B	A	B	A	C	C	E	A	B
s	Z	Y	Z	Z	Y	Y	X	X	Y	Y	Y	Z	Y	Z	Y	Y	X	Z	Y
t	1	1	2	3	4	4	5	5	5	6	7	10	10	10	10	11	11	12	12

Table 4.1 Illustrative sequence of alerts. **a** = alerts; **s** = super-sorts; and **t** = time-stamps

without an upper bound). Thus, as we see in the next Chapter, we need to normalize it if we want to establish a fixed threshold to cut-off less similar sequences to limit the size of the set of sequential cases retrieved. We define the normalized dynamic sequence similarity between two sequences as follows:

$$\|\vec{S}_1 \sim_s \vec{S}_2\| = \frac{\vec{S}_1 \sim_s \vec{S}_2}{\vec{S}_2 \sim_s \vec{S}_2} \tag{4.9}$$

That is the fraction of the dynamic sequence similarity between $\vec{S}_1$ and $\vec{S}_2$ and the dynamic sequence similarity between $\vec{S}_2$ and $\vec{S}_2$. The normalized dynamic sequence similarity ranges between 0 and 1. Given that the maximum (arbitrary) value of the similarity between a sequence and another sequence is obtained when compared with itself.

2. Second, our similarity measure gives higher values to rare dissimilar sequences of alerts than to frequent similar sequences. That is to say, rare dissimilar sequence receive a higher calculated similarity than common similar sequences. There is a clear justification for this. Rare sequences of alerts can be more dangerous than frequent ones. Thus, we say that our similarity provides a mechanism to advert their presence.

Therefore, our dynamic sequence similarity can be seen as a ranking (that has also been called *excess* in the literature [OB97]). As was noted by Osborne and Bridge [OB97], "it is often easier for a user to 'rank' objects, and even to say how much 'better' one object is compared with another, than it is for a user to decide on the degrees of similarity of objects". Osborne and Bridge defined excesses and then defined similarity measures in terms of these. In our case, the similarity between two sequences of alerts indicates that both were caused by similar sources mechanisms. However, since a rare sequence of alerts could bring more peril and the risk of disaster than a frequent one our similarity ranks it first. We say that our dynamic similarity has the particularity of ranking according the rareness and behaving as a regular relative similarity when it is normalized. In order to understand this particularity, we a use a simple but striking example, in next Section, to illustrate how our dynamic sequence similarity works.

4.4 HOW IT WORKS

Table 4.1 shows a simple example of an alert stream that we will use to illustrate how our dynamic similarity works. The alert stream contains 19 alerts of sorts A, B, C, D,

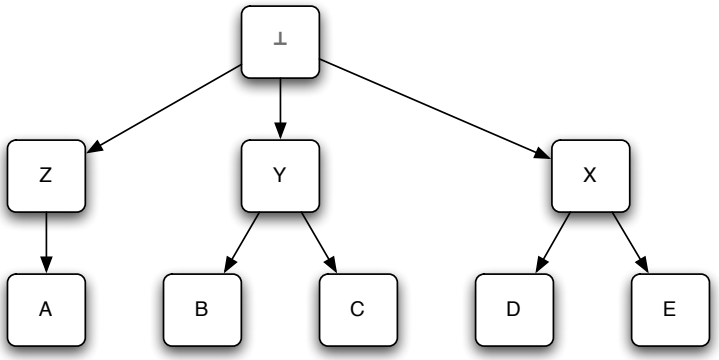


Fig. 4.1 Illustrative taxonomic hierarchy of alerts.

and E. The first row indicates the sort of each alert. The second row references the super-sort of each alert according to the taxonomic hierarchy depicted by Figure 4.1. For example, sort Y subsumes sorts B and C. There are three super-sorts X, Y, and Z. $\perp$ is the highest sort in the hierarchy and subsumes the rest of sorts. We consider a time-based window model of size 3 seconds. Therefore, we will use 4 windows containing 4, 6, 1 and 8 alerts respectively to analyze the complete sequence.

Assume that we are at the forth iteration and therefore analyzing alerts whose time-stamp is greater or equal to 10 and lower or equal to 12. That is to say, the current window of alerts W_4 is [A B A C C E A B]. Table 4.2 shows for each sort the values of abduction and neglection cost as well as it α-rare, α-rarity and relative frequency. Table 4.3 shows subsumption scoring scheme at the same point in time. For example, for sort A $M_{A,A} = \frac{1-0.3158}{0.3158} = 2.1667$. Notice that highest score goes to sort D that is the least frequent (the rarest).

Now, let us suppose that we have only two sequences of alerts (two episodes) to compare with the current window of alerts: e_1 =[E B D] and e_2 =[A B A]. Tables 4.4 and 4.5 show respectively the trace of the dynamic programming computation for both sequences. The maximum value at the last row is written in boldface (i.e., the score of the optimal alignment). For sequence [E B D], sequence similarity with current window of alerts is $max_{1\leq j\leq 3}S(8,j) = 10.4667$ whereas for sequence [A B A] is $max_{1\leq j\leq 3}S(8,j) = 4.9667$. Notice that in spite of the fact that sequence of alerts [A B A] even occurs within the current window of alerts it receives a lower score since dynamic sequence similarity promotes the rareness of sequence [E B D] and ranks it first.

Now we will show what happens when we normalize both similarities (rankings or excesses). Tables 4.6 and 4.7 show the trace for the computation of the sequence similarity of each episode and itself. The maximum scores are 29.3 and 7.1333 for episodes [E B D] and [A B A] respectively. If we normalize, $\|W_4 \sim_s e_1\| = \frac{10.4667}{29.3} =$

	α-rare(i)	α-rarity(i)	$\mathbf{q_i}$	$\mathbf{C^a(i)}$	$\mathbf{C^n(i)}$
A	6.0000	1.2000	0.3158	-1.2000	-0.8333
B	5.0000	1.0000	0.2632	-1.0000	-1.0000
C	5.0000	1.0000	0.2632	-1.0000	-1.0000
D	1.0000	0.2000	0.0526	-0.2000	-5.0000
E	2.0000	0.4000	0.1053	-0.4000	-2.5000
Z	6.0000	1.2000	0.3158	-2.4000	-0.8333
Y	10.0000	2.0000	0.5263	-4.0000	-0.5000
X	3.0000	0.6000	0.1579	-1.2000	-1.6667
$\perp$	19.0000	3.8000	1.0000	-11.4000	-0.2632

Table 4.2 Dynamic Sequence Similarity walkthrough.

	A	**B**	**C**	**D**	**E**	**Z**	**Y**	**X**	$\perp$
A	2.1667	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
B	$-\infty$	2.8000	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
C	$-\infty$	$-\infty$	2.8000	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
D	$-\infty$	$-\infty$	$-\infty$	18.0000	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
E	$-\infty$	$-\infty$	$-\infty$	$-\infty$	8.5000	$-\infty$	$-\infty$	$-\infty$	$-\infty$
Z	2.1667	$-\infty$	$-\infty$	$-\infty$	$-\infty$	2.1667	$-\infty$	$-\infty$	$-\infty$
Y	$-\infty$	1.8000	1.8000	$-\infty$	$-\infty$	$-\infty$	0.9000	$-\infty$	$-\infty$
X	$-\infty$	$-\infty$	$-\infty$	16.0000	8.0000	$-\infty$	$-\infty$	5.3333	$-\infty$
$\perp$	0	0	0	0	0	0	0	0	0

Table 4.3 Dynamic Subsumption Scoring Scheme.

			E	**B**	**D**
		$\mathbf{S_2[0]}$	$\mathbf{S_2[1]}$	$\mathbf{S_2[2]}$	$\mathbf{S_2[3]}$
	$\mathbf{S_1[0]}$	0	-0.4000	-1.4000	-1.6000
A	$\mathbf{S_1[1]}$	0	-0.4000	-1.4000	-1.6000
B	$\mathbf{S_1[2]}$	0	-0.4000	2.4000	2.2000
A	$\mathbf{S_1[3]}$	0	-0.4000	1.5667	1.3667
C	$\mathbf{S_1[4]}$	0	-0.4000	0.5667	0.3667
C	$\mathbf{S_1[5]}$	0	-0.4000	-0.4333	-0.6333
E	$\mathbf{S_1[6]}$	0	8.5000	7.5000	7.3000
A	$\mathbf{S_1[7]}$	0	7.6667	6.6667	6.4667
B	$\mathbf{S_1[8]}$	0	6.6667	**10.4667**	10.2667

Table 4.4 Trace of dynamic sequence similarity computation for a window of 3 seconds and episode EBD.

0.3572 while $\|W_4 \sim_s e_2\| = \frac{4.9667}{7.1333} = 0.6962$. Therefore, if we used a normalized sequence similarity sequence [A B A] would be ranked first.

Continuing with the above example, let us show now how costs and subsumption scoring values evolve over time and conveniently adapt to data seen so far. See Table 4.8 and Figure 4.2. Figure 4.2 plots the data in Table 4.8. The first row in Table 4.8 (plot (a) in Figurea 4.2) shows the cumulative absolute frequency of alert A along the 4 windows. The second row (plot (b) in Figure 4.2) shows the exact number of occurrences of alert A in each window. The third row (plot (c) in 4.2) shows the cumulative absolute frequency of all the alerts. The forth and fifth row show the cost of abduction and neglection respectively (plots (d) and (e) in Figure 4.2). Finally, the

			A	**B**	**A**
		$\mathbf{S_2[0]}$	$\mathbf{S_2[1]}$	$\mathbf{S_2[2]}$	$\mathbf{S_2[3]}$
	$\mathbf{S_1[0]}$	0	-1.2000	-2.2000	-3.4000
A	$\mathbf{S_1[1]}$	0	2.1667	1.1667	-0.0333
B	$\mathbf{S_1[2]}$	0	1.1667	4.9667	3.7667
A	$\mathbf{S_1[3]}$	0	2.1667	4.1333	7.1333
C	$\mathbf{S_1[4]}$	0	1.1667	3.1333	6.1333
C	$\mathbf{S_1[5]}$	0	0.1667	2.1333	5.1333
E	$\mathbf{S_1[6]}$	0	-1.2000	-0.3667	2.6333
A	$\mathbf{S_1[7]}$	0	2.1667	1.1667	1.8000
B	$\mathbf{S_1[8]}$	0	1.1667	**4.9667**	3.7667

Table 4.5 Trace of dynamic sequence similarity computation for a window of 3 seconds and episode ABA.

			E	**B**	**D**
		$\mathbf{S_2[0]}$	$\mathbf{S_2[1]}$	$\mathbf{S_2[2]}$	$\mathbf{S_2[3]}$
	$\mathbf{S_1[0]}$	0	-0.4000	-1.4000	-1.6000
E	$\mathbf{S_1[1]}$	0	8.5000	7.5000	7.3000
B	$\mathbf{S_1[2]}$	0	7.5000	11.3000	11.1000
D	$\mathbf{S_1[3]}$	0	2.5000	6.3000	**29.3000**

Table 4.6 Trace of dynamic sequence similarity computation between episode EBD and itself.

			A	**B**	**A**
		$\mathbf{S_2[0]}$	$\mathbf{S_2[1]}$	$\mathbf{S_2[2]}$	$\mathbf{S_2[3]}$
	$\mathbf{S_1[0]}$	0	-1.2000	-2.2000	-3.4000
A	$\mathbf{S_1[1]}$	0	2.1667	1.1667	-0.0333
B	$\mathbf{S_1[2]}$	0	1.1667	4.9667	3.7667
A	$\mathbf{S_1[3]}$	0	2.1667	4.1333	**7.1333**

Table 4.7 Trace of dynamic sequence similarity computation between episode ABA and itself.

window	1	2	3	4
Q(A)	3.0000	3.0000	3.0000	6.0000
N(A)	3.0000	0.0000	0.0000	3.0000
Q	4.0000	10.0000	11.0000	19.0000
$\mathbf{C^a(A)}$	-1.5000	-0.6000	-0.6000	-1.2000
$\mathbf{C^n(A)}$	-0.6667	-1.6667	-1.6667	-0.8333
$\mathbf{M_{A,A}}$	0.3300	2.3300	2.6667	2.1667

Table 4.8 Costs and subsumption scoring dynamics for alert A.

last row (plot (f) in Figure 4.2) shows the subsumption score given to alert A along the four windows. Notice how this score varies over time. It increases with alert A rareness and decreases when alert A becomes more common. Similarly, the cost of abduction decreases with rareness and increases with commonness. On the contrary, the cost of neglection increases with rareness and decreases with commonness[3]. The adaptation of the subsumption score, neglection, and abduction costs to data seen so far is a singularity of our sequence similarity that allows us to conveniently pinpoint those situations (sequence of events) that are rarer and could entail more peril.

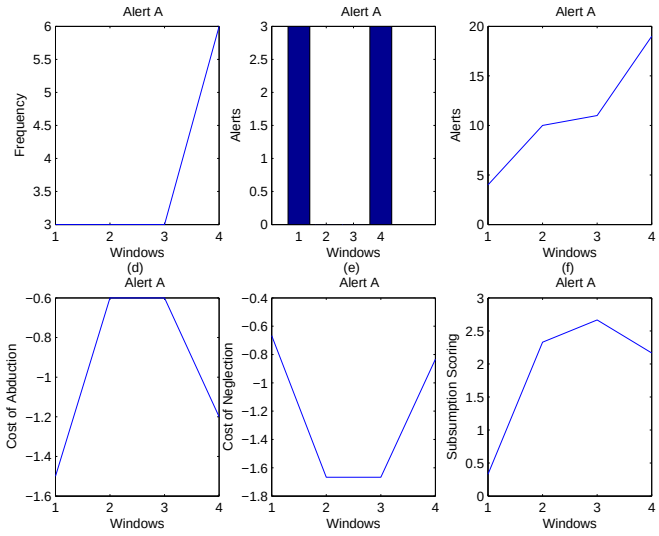


Fig. 4.2 Costs and scoring dynamics for alert A in illustrative example of Table 4.1 over 4 time-based windows of size 3. Plot (a) shows the cumulative absolute frequency of alert A. Plot (b) shows a histogram of the number of occurrences per window. Plot (c) shows the cumulative absolute frequency of all the alerts. Plot (d) shows the cost of abduction. Plot (e) depicts the cost of neglection over the 4 windows. (f) shows the evolution of the subsumption scoring scheme over time.

[3]Notice that costs are negative.

Next Section, we will use the same graphics that we depicted for alert A to analyze in further detail the evolution of the dynamic sequence similarity using a number of alerts in our real-world data-sets.

4.5 COSTS AND SCORING DYNAMICS

This Section tries to make explicit the intuitions behind the costs and the scoring scheme employed by our dynamic sequence similarity. We will show how these values vary over time for a number of alerts in the distinct data-sets that we have collected for experimental purposes. Specifically, we will analyze cumulative (absolute) frequency, the number of occurrence on each window, the cumulative number of alerts, the evolution of the cost of abduction, the cost of neglection, and the subsumption scoring for a frequent alert, a rare alert, and an arbitrary alert in distinct data-sets and using different window models. Table 4.9 lists the nine alerts analyzed below.

Dataset	wm	ws	alert	sid	Fig.
Rustoord	0	100	WEB-IIS cmd.exe acces	1002	4.3
Rustoord	1	3600	WEB-IIS cmd.exe acces	1002	4.4
NaxPot	1	3600	WEB-MISC http directory traversal	1113	4.5
Rustoord	0	100	WEB-CGI campus access	1653	4.6
NaxPot	1	3600	SCAN SSH Version map attempt	1638	4.7
Huckleberry	1	3600	WEB-IIS CodeRed v2 root.exe access	1256	4.8
Rustoord	1	3600	WEB-IIS CodeRed v2 root.exe access	1256	4.9
NaxPot	1	3600	MS-SQL Worm propagation attempt	2003	4.10
Huckleberry	1	3600	MS-SQL Worm propagation attempt	2003	4.11

Table 4.9 Alert Analyzed. **Dataset**= dataset name; **wm** = window model; **ws** = window size; alert= Snort signature; sid = Snort identifier. **Fig.** = figure number.

We first analyze the most frequent alert in Rustoord data-set. See Appendix A for further details on the data-sets. Plot 4.3(a) depicts the cumulative (absolute) frequency of alert WEB-IIS cmd.exe access against the number of windows for a total of 315 windows[4]. In the last example, we used a space-based window of size 100 alerts with null overlapping (see Section 3.5). Plot 4.3(b) shows a histogram of the alert over the 315 windows (i.e., exactly the number of occurrences on each window). Plot 4.3(c) represents the cumulative number of alerts that allows us to compare the pace at which the alert of interest increases its occurrence versus the number total of alerts that occur in the system. As it can be seen alert WEB-IIS cmd.exe access frequency contributes significantly to the total number of alerts in the system. A number of times more than 80 over 100 alerts are of this sort. Thus, its cost of abduction continously decreases as shown in Plot 4.3(d). This drop stabilizes as soon as the frequency of occurrence increases with lower slope. The cost of neglection depicted by Plot 4.3(e)

[4]This alert indicates that an attempt has been made to exploit potential weaknesses in a host running Microsoft IIS. This alert could correspond to either the prelude of an attack (i.e., an attacker trying to compile information on the IIS implementation) or an attack to get root access to the host.

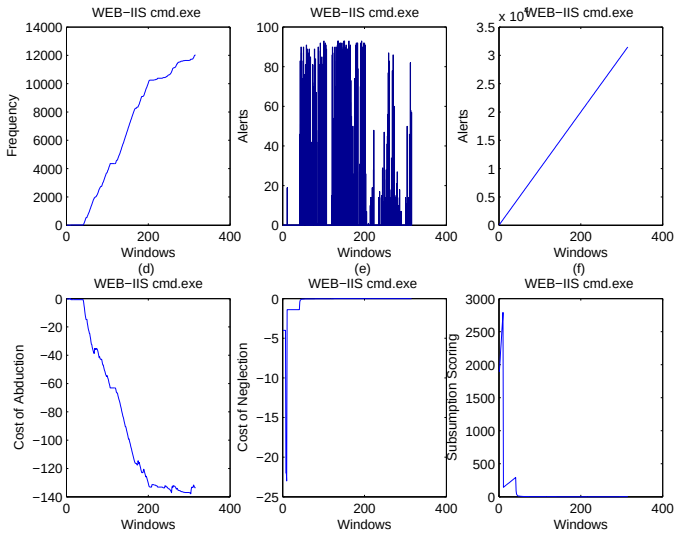


Fig. 4.3 Costs and scoring dynamics for alert WEB-IIS cmd.exe access (SID 1002) in Rustoord data-set over 315 space-based windows of size 100.

becomes almost zero. This means that this alert is so common that can practically be ignored and treated as noise (i.e., ignoring its occurrence does not intervene in the final performance of the system). Finally, Plot 4.3(f) shows the scoring given to a match between two alerts of sort WEB-IIS cmd.exe access over time. Although initially it receives a high score as soon as the alert starts to be frequent the score decreases and becomes practically zero. Said differently, it does not contribute to the total similarity in a sequence neither positively nor negatively.

Figure 4.4 shows the same analysis as above but using a time-based window of 3600 seconds. Notice the similar behavior of costs and subsumption in spite of using a distinct window model. Plot 4.4(b) shows the cadency of this alert. Every hour it occurs between 10 and 70 times. It is important to notice that not appearing in the plot does not mean that the alert did not occur but that the inner intrusion detection was not working. Thus, the existence of this kind of alerts reinforces the importance of ignoring frequent alerts (cost of neglection 0 or close to 0) as the best way to concentrate vigilance on rare ones.

Figures 4.5 shows the evolution of alert WEB-MISC http directory traversal, the most frequent alert in the NaxPot data-set[5]. This alert occurs around 20 times every hour along, approximately, the first 2000 windows. Then, its presence suddenly

[5]A number of web servers and CGI scripts are vulnerable to directory traversal attacks. Often a web server application may intend to allow access to a particular portion of the filesystem. But without proper checking of user input, a user could often simply add ".." directories to the path allowing access to parent directories, possibly climbing to the root directory and being able to access the entire filesystem. This leads to information disclosure and possible exposure of sensitive system information.

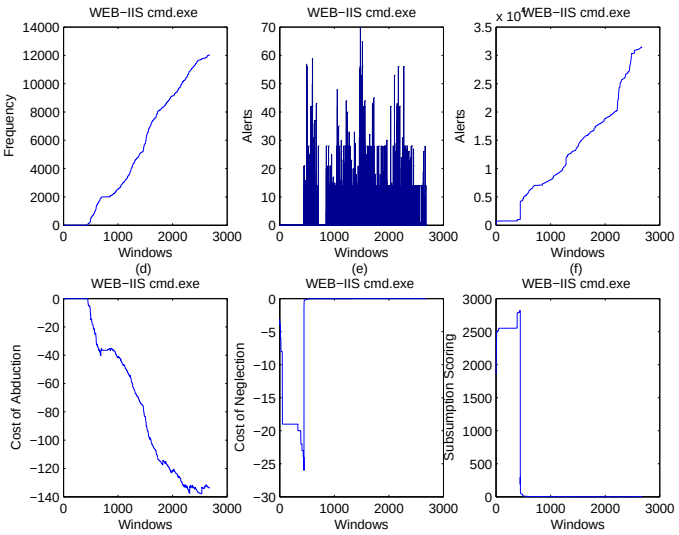


Fig. 4.4 Costs and scoring dynamics for alert WEB-IIS cmd.exe access (SID 1002) in Rustoord data-set over 2680 time-based windows of 3600 seconds.

increases and occurs around 100 times every hour along the following 3000 windows with several peaks that exceed 150 alerts per hour. Observe that this alert even once occurs more than 500 hundreds times in only a hour. The cost of abduction of this alert increases dramatically whereas the cost of neglections as well as the score given by the subsumption scoring scheme practically become 0 from the very beginning windows. We have seen by now that our dynamic similarity measure behaves similarly using different windows models and that minimizes the score given to frequent alerts. Let us now show the same analyses for rare alerts.

Figure 4.6 shows the plots for alert WEB-CGI campus access, that occurred only once in the Rustoord data-set[6]. First notice that the alert only occurs once although the cumulative frequency is 2. The reason is that in our analyses we used Laplace's adjustment to properly compute the subsumption scoring scheme of those alerts that never occur and added an extra occurrence for each possible alert. This alert occurred uniquely in the first window. Thus, as long as the analysis proceeds the cost of abduction tends to zero whereas the cost of neglection increases significantly. The score given by subsumption scheme also increases over time. Thus, if this alert occurred again it would receive a significant score. Figure 4.7 shows the same analysis for alert SCAN SSH Version map attempt that occurs 5 times in the NaxPot data-set[7]. This rare alert, one of the least frequent in NaxPot data-set, receives similar

[6]These alerts indicate that an attempt has been made to gain unauthorized access to a CGI application running on a web server.
[7]This alert indicates the prelude to an attack. An attacker tries to ascertain if a vulnerable version of ssh is being used on the target host.

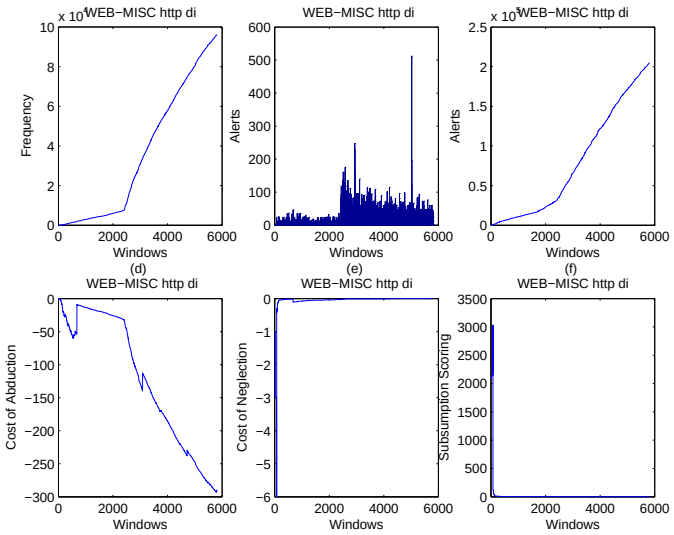


Fig. 4.5 Costs and scoring dynamics for alert WEB-MISC http directory traversal (SID 1113) in NaxPot data-set over 5816 time-based windows of 3600 seconds.

treatment as above. That is to say, the cost of abduction is kept around 0 while the cost of neglection stabilizes around -50. The score returned by our dynamic subsumption scheme is also very high. These scores contribute to point out the importance of the occurrence of this type of alert. As we have been shown, our dynamic similarity promotes rareness. Let's now show how it behaves with a number of arbitrary alerts.

We will depict the same graphics for an arbitrary alert CodeRed v2 root.exe access in two different data-sets: Huckleberry in Figure 4.8 and Rustoord in Figure 4.9. Comparing both figures we see how the similarity of the same alert is considered differently by each data-set depending on the exact number of occurrences. Interestingly, through the analysis of the number of occurrences of this alert every hour one can imagine the impact of the prevalence of Internet worms in intrusion detection systems. The CodeRedII worm started to spread on August 4, 2001[8]. Our analyses are dated in 2002 and 2003. At this rate, it can take months before Code Red goes away or perhaps it may never stop completely. See CAIDA analyses[9] or [ZGT02] analysis for a thoroughly discussion. Figures 4.10 and 4.11 depict the analyses for alert MS-SQL Worm propagation attempt that corresponds to the Sapphire Worm (also called Slammer) that began to spread on January 25th, 2003[10]. Notice that even though these worms propagate using randomly-generated IP addresses the number

[8]This alert corresponds to an attack that exploits a buffer-overflow vulnerability in Microsoft's IIS web-servers. See Exampe 1 for further information.
[9]http://www.caida.org/analysis/security/code-red/
[10]See http://www.caida.org/analysis/security/sapphire/ for the corresponding CAIDA analysis.

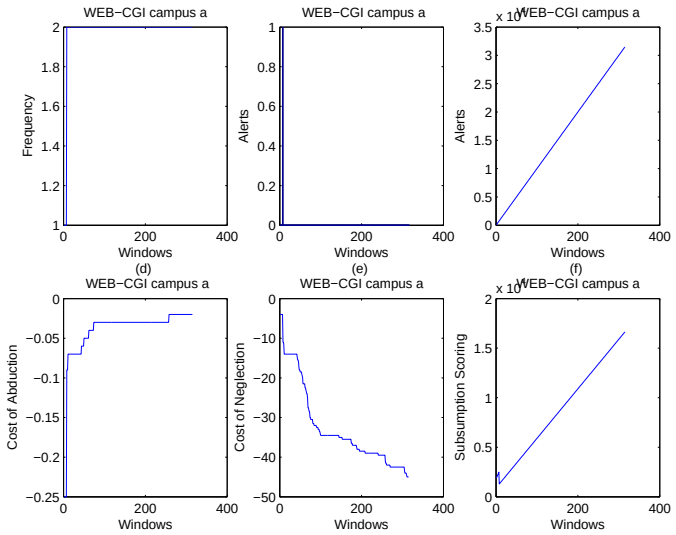


Fig. 4.6 Costs and scoring dynamics for alert WEB-CGI campus access (SID 1653) in Rustoord data-set over 315 space-based windows of size 100.

of occurrences of the corresponding alerts on each data-set depends on the number public IP addresses that the target network had open.

The analyses of the above nine alerts have served two main purposes: to show how our dynamic sequence similarity behaves over time and to gain some insights about the population of alerts that we deal with.

In this Chapter, we introduced a dynamic sequence similarity that allows us to compare the input alert stream against the stored sequential cases as we see in the next Chapter. Our dynamic sequence similarity measure has two main characteristics: continuously adaption to data seen so far and promotion of rareness. Thus we say that our dynamic sequence similarity measure is an useful mechanism to provide an automated indication of similarity that approximates what an SSO could be interested in.

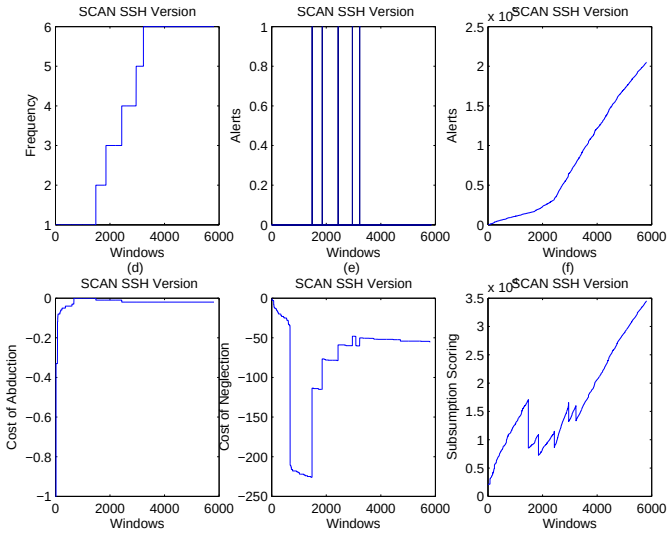


Fig. 4.7 Costs and scoring dynamics for alert SCAN SSH Version map attempt (SID 1638) in NaxPot data-set over 5816 time-based windows of 3600 seconds.

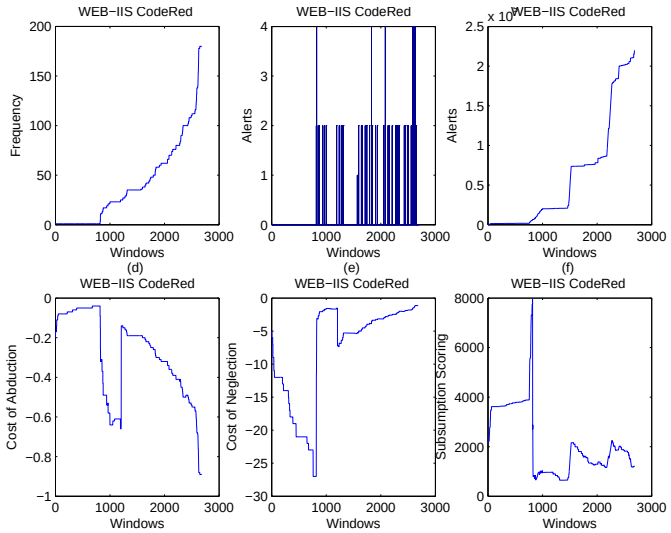


Fig. 4.8 Costs and scoring dynamics for alert WEB-IIS CodeRed v2 root.exe access (SID 1256) in Huckleberry data-set over 2685 time-based windows of 3600 seconds.

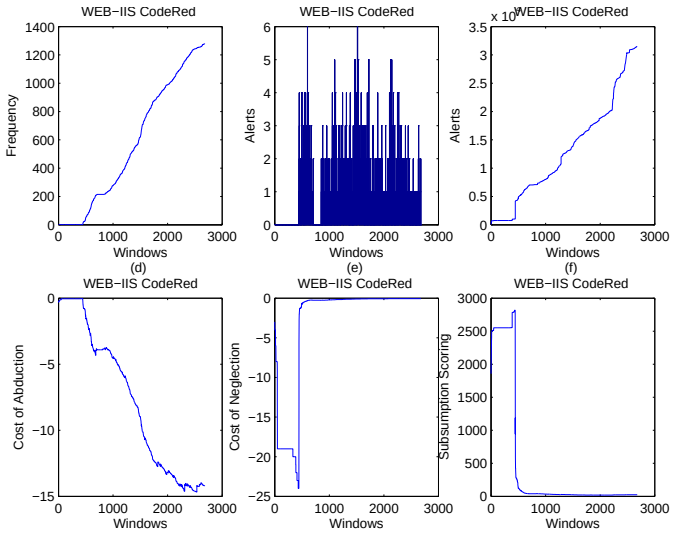


Fig. 4.9 Costs and scoring dynamics for alert WEB-IIS CodeRed v2 root.exe access (SID 1256) in Rustoord data-set over 2680 time-based windows of 3600 seconds.

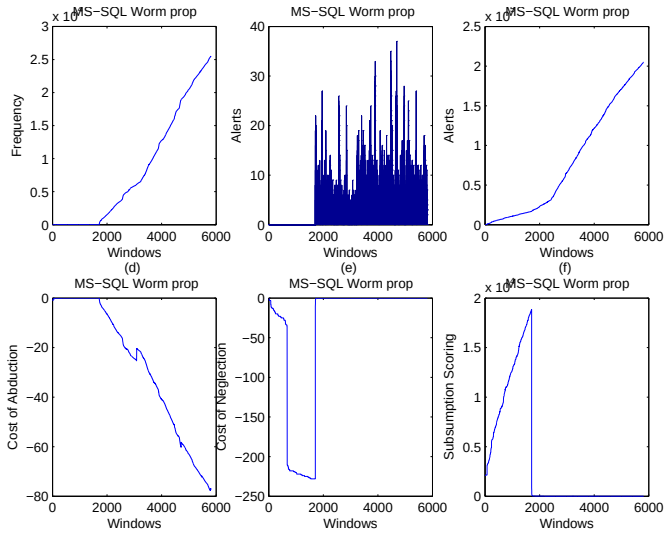


Fig. 4.10 Costs and scoring dynamics for alert MS-SQL Worm propagation attempt (SID 2003) in Naxpot data-set over 5816 time-based windows of 3600 seconds.

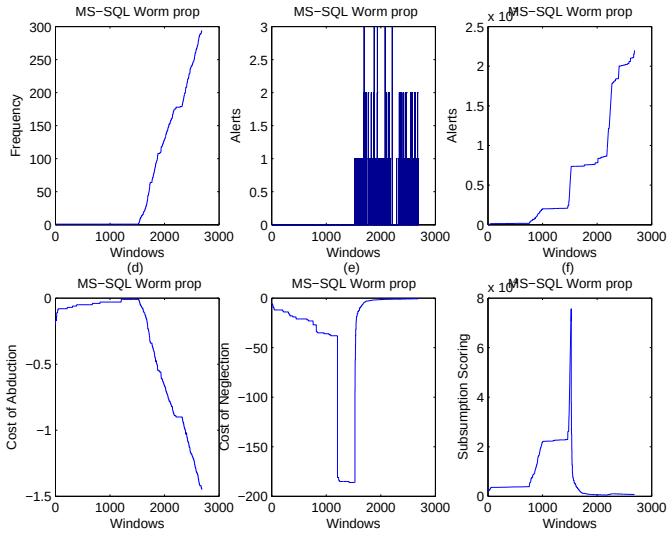


Fig. 4.11 Costs and scoring dynamics for alert MS-SQL Worm propagation attempt (SID 2003) in Huckleberry data-set over 2685 time-based windows of 3600 seconds.

5

Ceaseless CBR

This Chapter throughly describes Ceaseless CBR. Ceaseless CBR can be seen as a constructive situation awareness process governed ceaselessly by *observational data*, a *sequential case base*, and *case activations*. Such concepts are first described in Section 5.1. Then, Section 5.2 explains how Ceaseless Retrieve uses dynamic sequence similarity to retrieve similar sequential cases that subsume the sequence of alerts at hand and create *case activations* that anticipate as much as possible all the situations that might likely occur. Section 5.3 describes how Ceaseless Reuse construct explanations—combinations of case activations that completely explain the sequence of alerts at hand—and provides a likelihood assessment for each explanation considering its likely consequences. We also see how to tradeoff risk versus efficiency estimating a degree of *urgency* that finally determines when the prioritization of each alert has to be done. Section 5.4 explains how Ceaseless Revise allows users to supervise the explanations provided by Ceaseless Reuse what guides its search towards the best explanations in future iterations. Section 5.5 describes how Ceaseless Retain constantly updates the sequential case base with the revised sequential cases and with the frequencies of alerts seen so far. Finally, Section 5.6 sums up some of the relevant characteristics of our model.

5.1 INTRODUCTION

Ceaseless CBR aims at finding the best explanation of an unsegmented sequence of alerts with the purpose of pinpointing whether undesired situations (an attack, fault, etc) have occurred or not and, if so, indicating the multiple responsible sources (if

more than one intervened) or at least which ones are the most plausible. Moreover, Ceaseless CBR prioritizes each individual alert according to the proposed explanations. Loosely speaking, the reasoning performed by Ceaseless CBR to accomplish such task can be cast as *plausible reasoning*. Plausible reasoning can be understood as the process of reasoning (and taking decisions) as best as we can in spite of the presence of incomplete information and has an evident correspondence with common sense. Based on current gathered evidence and past experience, Ceaseless CBR:

1. anticipates as much as possible all the situations that might occur generating hypotheses based on sequential cases;

2. emits a judgment on how likely each situation is, and makes the decision on whether to prioritize an alert or not according to its urgency that is computed judging each alert's further consequences;

3. creates an explanation composing hypotheses together that completely explain all urgent alerts and prioritizes them accordingly;

4. revises the decision and its outcome; and finally stores the solution for further reuse.

Conversely to the mainstream CBR model, we do not consider a unique process, composed of four sequential sub-processes, initiated upon a new problem description arrival. Instead we have broken up this process into four sub-processes running ceaselessly in parallel: Ceaseless Retrieve, Ceaseless Reuse, Ceaseless Revise, and Ceaseless Retain. By parallel processes we mean that their order of execution is not restricted. The description of those processes constitutes the bulk of this Chapter. Before explaining them in detail we describe first some concepts of interest. We see Ceaseless CBR as a constructive situation awareness process governed ceaselessly by *observational data*, a *sequential case base*, and *case activations*. We discuss each of these concepts in detail through the next Subsections.

5.1.1 Observational Data

We assume that at a given point in time t there is a sequence of n alerts (alert stream) in the system. We denote by $\vec{S}^{(t)}$ the sequence of alerts received so far. We use $\vec{W}^{(t)}_{wm}$ to represent the most recently received alerts according to a specific window model wm. We denote by $\vec{P}^{(t)}$ the sequence of pending alerts at time t. That is, alerts that have not been prioritized yet either because they have just arrived or they were not prioritized in a previous iteration because they had a low *urgency*. We discuss this issue in detail later on in Section 5.3.2. In an ever-changing environment, recalling the history of the system can be the only way to reduce uncertainty. Our model considers that as the analysis of the alert stream proceeds, it produces a probability distribution over the set of all received alerts. This probability distribution constitutes the basis of our similarity between sequences as well as the foundation that allows us to go from observations to hypotheses and from hypotheses to explanations. Each alert ψ_j in

$\vec{S}^{(t)}$ belongs to a pre-specified alert signature $\Sigma = \langle \mathcal{S}, \perp, \mathcal{F}, \preceq \rangle$ within a given alert model $\mathbb{A} = \langle \vartheta, \Sigma \rangle$ according to the definitions introduced in Chapter 3. For each sort i in $\mathcal{S}$ we denote by $q_i^{(t)}$ the relative frequency of sort i at time t. We use the relative frequency of a sort to estimate its a priori probability $P^{(t)}(i) = q_i^{(t)}$. When there is no risk of confusion with the instant of time that we are referring to we simply use $P(i)$ and q_i. We say that the probability of occurrence of an alert ψ_j whose sort is $i = root(\psi_j)$ is $P(\psi_j) = q_{root(\psi_j)} = q_i$. Notice that given two sorts $i, j \in \mathcal{S}$ such that $i \preceq j$ then $P(i) \geqslant P(j)$ and that $P(\perp) = 1$. Likewise, we denote by $Q_i^{(t)}$ the absolute frequency of sort i at time t. The number of alerts received so far is given by $|\vec{S}^t|$. Therefore, $q_i^{(t)} = \frac{Q_i^{(t)}}{|\vec{S}^t|}$.

5.1.1.1 Noise Model

We consider that current monitoring systems (probes) are unable to capture all possible alerts (observable symptoms events) that affect the system under supervision. Alerts may be not evoked due to a number of causes. For example, because the corresponding network sensors cannot detect an attacker's action that corresponds to a new and unknown vulnerability. Alerts could also be lost before reaching the alert correlation component because they are transmitted through unreliable or corrupted communication channels. We define the alert loss ratio as the probability that an alert of a given sort is lost and denote it by $L(i)$. This value is adjusted based on the knowledge about the system under supervision. For example, if we know that our sensor are able to detect almost any kind of reconnoissance activity we should set up the probability of alerts of sort attempted-recon by 0. We could also adjust that value using for example the packet loss rate in the communication channel or other parameters that allow us to derive the reliability of the different components that underpin the correlation component [SS03].

5.1.2 Sequential Case Base

We suppose that there initially exists a sequential case base made up of a number of sequential cases $C^{(0)} = \{C_1, \cdots, C_n\}$ with $n \geqslant 0$ that have been either handcrafted by a human expert or learnt through a sequential case discovery mechanism. A sequential case provides a prototype (pattern) of sequences of alerts that were caused by the same source during a specific situation. We consider two types of sequential cases: *sequential master cases* and *sequential abstract cases*.

5.1.2.1 Sequential Master Cases

A sequential master case is an abstract representation of a sequential case that allows one to cast several similar sequential cases that only differ in very specific attributes. For example, the same computer attack launched from different IP address sources, against different IP address destinations, at different times, or altering the order of

$$attack11 = \left[\begin{array}{l} \underline{\textit{theft}} \\ \text{alerts} \doteq \left[\begin{array}{l} \underline{\textit{serial-node}} \\ \text{alerts} \doteq \left[\begin{array}{l} \underline{\textit{SCAN-cybercop-os-probe}} \\ \text{likelihood} \doteq \left[\begin{array}{l} \underline{\textit{probabilistic}} \\ \text{likelihood} \doteq 0.2 \end{array}\right] \end{array}\right] \\ \qquad\quad \left[\begin{array}{l} \underline{\textit{DOS-land-attack}} \\ \text{likelihood} \doteq \left[\begin{array}{l} \underline{\textit{probabilistic}} \\ \text{likelihood} \doteq 0.5 \end{array}\right] \end{array}\right] \\ \text{constraints} \doteq \left[\begin{array}{l} \underline{\textit{feature}} \\ \text{name} \doteq \text{source-addr} \end{array}\right] \left[\begin{array}{l} \underline{\textit{feature}} \\ \text{name} \doteq \text{dest-addr} \end{array}\right] \\ \text{likelihood} \doteq \left[\begin{array}{l} \underline{\textit{probabilistic}} \\ \text{likelihood} \doteq 0.3 \end{array}\right] \end{array}\right] \\ \qquad\quad \left[\begin{array}{l} \underline{\textit{serial-node}} \\ \text{alerts} \doteq \left[\begin{array}{l} \underline{\textit{DNS-named-version-attempt}} \\ \text{likelihood} \doteq \left[\begin{array}{l} \underline{\textit{probabilistic}} \\ \text{likelihood} \doteq 0.4 \end{array}\right] \end{array}\right] \\ \qquad\quad \left[\begin{array}{l} \underline{\textit{DNS-exploit-named-overflow-attempt}} \\ \text{likelihood} \doteq \left[\begin{array}{l} \underline{\textit{probabilistic}} \\ \text{likelihood} \doteq 0.6 \end{array}\right] \end{array}\right] \\ \text{constraints} \doteq \left[\begin{array}{l} \underline{\textit{feature}} \\ \text{name} \doteq \text{source-addr} \end{array}\right] \left[\begin{array}{l} \underline{\textit{feature}} \\ \text{name} \doteq \text{dest-addr} \end{array}\right] \\ \text{likelihood} \doteq \left[\begin{array}{l} \underline{\textit{probabilistic}} \\ \text{likelihood} \doteq 0.4 \end{array}\right] \end{array}\right] \\ \qquad\quad \left[\begin{array}{l} \underline{\textit{serial-node}} \\ \text{alerts} \doteq \left[\begin{array}{l} \underline{\textit{MYSQL-root-login-attempt}} \\ \text{likelihood} \doteq \left[\begin{array}{l} \underline{\textit{probabilistic}} \\ \text{likelihood} \doteq 0.5 \end{array}\right] \end{array}\right] \\ \qquad\quad \left[\begin{array}{l} \underline{\textit{MYSQL-show-database-attempt}} \\ \text{likelihood} \doteq \left[\begin{array}{l} \underline{\textit{probabilistic}} \\ \text{likelihood} \doteq 0.3 \end{array}\right] \end{array}\right] \\ \text{constraints} \doteq \left[\begin{array}{l} \underline{\textit{feature}} \\ \text{name} \doteq \text{source-addr} \end{array}\right] \left[\begin{array}{l} \underline{\textit{feature}} \\ \text{name} \doteq \text{dest-addr} \end{array}\right] \\ \text{likelihood} \doteq \left[\begin{array}{l} \underline{\textit{probabilistic}} \\ \text{likelihood} \doteq 0.6 \end{array}\right] \end{array}\right] \\ \text{risk} \doteq \left[\begin{array}{l} \underline{\textit{loss-expectancy}} \\ \text{rate-of-occurence} \doteq \left[\begin{array}{l} \underline{\textit{risk}} \\ \text{threat} \doteq 0.5 \\ \text{exposure} \doteq 0.9 \end{array}\right] \\ \text{single-loss-expectancy} \doteq 5000 \end{array}\right] \\ \text{priority} \doteq 0.8 \end{array}\right]$$

Fig. 5.1 Sequential case representation of a Theft attack using feature terms.

some attack steps. Sequential master cases are able to efficiently summarize a set of similar situations. A sequential master case stores the structures that subsume experiences with similar concrete sequential cases. Sequential master cases group together cases that can be represented using the same *actionable tree* and therefore are stored only once but constantly updated with the statistical information that reflect their different activations and how likely they are to occur. Along this Chapter we assume that the underlying predictive actionable trees used by sequential cases are based on the probabilistic likelihood model introduced in Section 3.4.4. Ceaseless

CBR seeks to maintain a compact case base since it would not be feasible to store the whole sequence of complex objects over time. Notice that our approach does not disregard concrete sequential cases. In fact, concrete sequential cases are stored for subsequent *post-mortem* analyses[1]. However, only sequential master cases are considered for retrieval purposes. This increases efficiency whereas it does not affect to performance detecting attacks. Figure 5.1 shows a sequential master case corresponding to the probabilistic actionable tree shown in Figure 3.16 represented using *feature terms* (i.e., the same representation formalism that we have used to describe alerts in Chapter 3).

5.1.2.2 Sequential Abstract Cases

Ceaseless CBR additionally uses sequential abstract cases that provide an additional level of abstraction to cope with unknown situations. Sequential abstract cases allow Ceaseless CBR to find explanations for those alerts corresponding to attacks that have never occurred in the system before. Sequential abstract cases are constructed based on the informational order provided by the taxonomic hierarchy used to represent the alerts (symptom events) at the roots of the actionable tree. Sequential abstract cases are used as a back-up for the explanation of a group of unknown alerts and also allow Ceaseless CBR to discover new sequential cases. Sequential abstract cases offer an explanation to previously unknown situations. Thus, when Ceaseless CBR uses a sequential abstract case means that the current case-base cannot properly define the current situation and a new sequential case has to be created. The priority associated with this new sequential case requires operator's oversight. We use the predicate $\mathbf{abstract}?(C_i)$ to determine whether a case C_i is abstract or not. We call *sequential case discovery* to the whole process by which Ceaseless CBR compounds new abstract cases together (assembling case activations) and, after user revision, stores them for further reuse. As we will see later on, this process is interleaved among the rest of Ceaseless CBR processes.

In Ceaseless CBR, sequential cases not only can be hand-crafted by an expert based on a number past problem solving experiences but also through continuous operation and through an off-line methods to mine alert databases [JD02].

5.1.2.3 Risk Model

A sequential case allows one to define different fields storing relevant information to the concrete domain of application. A field that is mandatory in the family of domains we cope with is the *risk cost*. Risk cost expresses the expenses that the occurrence of a particular undesired situation (attack, fault, etc) brings. It can be measured, for example, in terms of the information lost, extent of system downtime,

[1]The goal of *post-mortem analysis* is threefold. First, to gather forensic evidence (contemplating different legal requirements) that will support legal investigations and prosecution. Second, to compile experience and provide or improve documentation and procedures that will facilitate the recognition and rapid repelling of similar intrusions in the future, and, third, to validate the current security policy.

or business impact that the undesired situation causes. Understanding the effects on critical infrastructures and the number of available countermeasures is of paramount importance to properly appraise the risk of each possible undesired situation. It is convenient to differentiate between risk and cost. The cost is measured in Euros indicating the operational losses while risk is measured in terms of the odds of occurrence of the attack. Risk expresses the potential for harm and depends on the concrete system's exposure to particular threats. In other words, anything of variable certainty and impact that threatens the successful achievement of a system's mission. Risk is considered as a weighting factor of the cost that reflects how likely or unlikely an undesired situation is. Therefore, an alert corresponding to an attack with a high cost but against which the system has a low exposure should receive a lower priority than an alert corresponding to a less costly attack but against which the system has a higher exposure. How to tradeoff between risk and cost depends on the concrete security policy and risk aversion of the system under supervision. Although more sophisticated risk functions could be used, for the sake of simplicity, we use a simple risk cost function. We compute the *risk cost* of each sequential case C_i as follows:

$$\begin{aligned}\text{risk-cost}(C_i) = (C_i.\text{risk.rate-of-ocurrence.threat}\times \\ C_i.\text{risk.rate-of-ocurrence.exposure})\times \\ C_i.\text{risk.single-lost-expectancy}\end{aligned} \quad (5.1)$$

Where:

Threat is a measure of the likelihood that the specific type of undesired situation represented by C_i will be initiated against the system under supervision.

Exposure is a measure that specifies the likelihood of the system under supervision to be vulnerable to the corresponding undesired situation.

Single lost expectancy represents the magnitude of losses resulting when the undesired situation occurs.

We assume that alerting an operator on the occurrence of an attack will reduce the exposure to such attack and therefore will additionally reduce its associated costs. Thus, as soon as we have enough evidence to determine that an attack in undergoing we should notify the network administrator. However, if our evidence does not reflect the reality and we notify the network administrator prematurely then we are causing a cost associated with the disturbance created in the network administrator that is called false positive cost. Ceaseless CBR provides a *minimum evidence threshold* θ that can be set up by the corresponding SSO according to either a general security policy, current situation, or his/her own risk preferences. For example, risk-averse SSOs can set it up at its lowest value. Then, the system will be permanently on guard and as soon as some small part of an attack is identified the SSO will be immediately notified. In Chapter 6, we will experimentally analyze the impact on performance for distinct values of this threshold. We also go back to this threshold later on in this Chapter.

5.1.3 Case Activations

One of the main issues we cope with is due to the fact that our problem descriptions are only partially available over time. In other words, in the domains we are dealing with new cases do not arrive assembled but broken up into pieces that arrive over time without fixed boundaries and mixed in with other cases' pieces that correspond to problems that occur coincidentally. This establishes an important difference with standard CBR approaches where a full-fledged case description is available at the beginning. Therefore, an incremental case acquisition process is required. Case acquisition entails piecing together different parts that resemble a past case. This process is sequential and implicitly requires to retrieve cases from the sequential case base ceaselessly. While this happens our model needs to keep a number of plausible hypotheses that continuously best match the different partial descriptions received during a specified time span. These hypotheses, that we have called case activations, are generated by retrieving similar cases from the case base and are constantly updated as soon as new evidence is gathered. Given an alert model, a case base, and a sequence of alerts we formally define a case activation as follows:

Definition 41 (Case Activation) *A case activation is a hypothesis on the occurrence of a similar past case and is represented formally as a 6-tuple* $h = \langle C, \hat{a}, \breve{a}, \varrho, \tilde{e}, t \rangle$ *where:*

- C *is a reference to the sequential case (actionable tree) being activated.*
- $\hat{a}$ *represents a partial binding between the sequence of alerts that occurred* (the subsumees*) and are subsumed by* C *(*the subsumer*).*
- $\breve{a}$ *represents those alerts in* C *that have not been observed yet and that were abduced in the dynamic sequence similarity computation.*
- ϱ *represents the rareness of the occurrence of the subsumees. We compute it as the normalized sequence similarity between* C *and* ϱ *i.e.,* $\varrho = C \sim_s \hat{a}$*.*
- $\tilde{e}$ *measures the level of confidence (evidence) we have in the occurrence of a complete similar sequence of alerts to those that the sequential case* C *represents. We compute it as the normalized sequence similarity between* C *and* $\hat{a}$ *i.e.,* $\tilde{e} = \|C \sim_s \hat{a}\|$ *. We also denote it by* $\|\varrho\|$*.*
- t *is the time at which the last alert on* $\hat{a}$ *occurred.*

Notice that $\tilde{e}$ expresses how relevant the current situation is compared to a sequential master cases that generalizes similar situations that occurred before. Remember that is the rareness of the alerts what determines the score returned by the dynamic sequence similarity such as defined in Chapter 4. That is to say, the rarer the alerts that comprise an attack the higher the score.

New sequential cases are compounded sequentially combining the information provided by different alerts that arrive over time. Consequently, at a given moment in time we would only count on a new case partially. Thereby, we need to provide a

measure that determines the degree of completion of a new case acquisition. We have defined a belief measure that expresses the probability that the complete new case happens with a given confidence within a fixed interval of time. This belief measure is computed using the inference mechanism provided by actionable trees (see Section 3.4.4). We say that this belief function is based on negative symptoms. That is, alerts evoked by sensors after detecting that something dangerous was occurring. As we see later on, Ceaseless CBR also uses *positive symptoms*. That is to say, the fact that some of the alerts of a sequential case have not been observed decreases the belief on the occurrence of the corresponding attack.

We say that there is a number of case activations that are active constantly. Case activations can be partially ordered in terms of either their belief or their evidence or a combination of both. Multiple and different orders can be established using other case features or aggregate functions. For example, in intrusion detection alert triage, case activations can be ordered in terms of the risk that the corresponding attack conveys for the protected network. Said differently, a case activation with less evidence but higher risk could get a higher prioritization. As we will see below, we have defined a measure of *urgency* that tries to identify when a case activation should be considered for further inference.

The number of sequential cases in the case base as well as the number of case activations limits the efficiency of continuous retrievals and comparisons during case acquisition. Therefore, it is interesting to define a policy such that certain case activations which have been active for a long period of time without increasing their evidence are considered obsolete and are "harvested" from the working memory. A similar policy could be applied to the sequential case base. That is, cases that are not retrieved for a long period of time should not be kept constantly under consideration. However, the latter policy could run serious risks. For example, in intrusion detection alert triage we have to be aware of risky, malicious, and rare attacks, being thus necessary to maintain the corresponding cases in spite of the fact that the common hope is that those attacks never happen. A deprecation parameter τ establishes when a case activation becomes obsolete: deprecated case activations are filtered from consideration in successive iterations and presented to the user for its proper revision (or simply disregarded).

5.1.3.1 Assembling Case Activations

We saw in Chapter 3 that a sequential case establishes a collection of constraints among its parts. We define an *equality path checking* process that ensures that such constraints are followed when two case activation are compounded together. That is, that path equality (Definition 6) will be kept for all alerts in the new case activation. Formally:

Definition 42 (Equality Path Checking) *Given an alert model $\mathbb{A} = \langle \vartheta, \Sigma \rangle$ and a temporal sequence of alerts $\vec{S} = [\psi_1, \psi_2, \ldots, \psi_n]$ such that each $\psi_i \in \mathbb{A}$ then we say that the sequence $\vec{S}$ is* ***constrainable?*** *given the a set of constraints $\mathcal{C} =$*

$\{f_1, \cdots, f_m\} : f_i \in \Sigma.\mathcal{F}$ *when the path equality for all features in* $\mathcal{C}$ *is kept for all alerts in* $\vec{S}$*. That is to say:*

$$\textbf{constrainable}?(\mathcal{C}, \vec{S}) = \begin{cases} \textbf{true} & \textbf{if } \forall_{f_i \in \mathcal{C}} \forall_{\psi_j, \psi_k \in \vec{S}} \big(\rho(\psi_j, f_i) = \rho(\psi_k, f_i)\big) \\ \textbf{false} & \textbf{otherwise} \end{cases} \tag{5.2}$$

This process guarantees that all the alerts in a given sequence share a number of common features. This process is part of the *fusion* of case activations that we have defined as follows:

Definition 43 (Case Activation Fusion) *The fusion of two case activations* $h_i = \langle C_i, \hat{a}_i, \check{a}_i, \varrho_i, \tilde{e}_i, t_i \rangle$ *and* $h_j = \langle C_j, \hat{a}_j, \check{a}_j, \varrho_j, \tilde{e}_j, t_j \rangle$*, denoted by* $h_i \uplus h_j$*, is defined as follows:*

$$h_i \uplus h_j = \begin{cases} h_i \odot h_j & \textbf{if compoundable}?(h_i, h_j) \\ \{\langle C_i, \hat{a}_i, \check{a}_i, \varrho_i, \tilde{e}_i, t_i \rangle, & \\ \langle C_j, \hat{a}_j, \check{a}_j, \varrho_j, \tilde{e}_j, t_j \rangle\} & \textbf{otherwise} \end{cases} \tag{5.3}$$

Where $h_i \odot h_j$ *is defined as:*

$$h_i \odot h_j = \langle C_i, \hat{a}_i \bullet \hat{a}_j, \check{a}_i - \hat{a}_j, C_i \sim_s (\hat{a}_i \bullet \hat{a}_j), \|\varrho\|, \textbf{max}(t_i, t_j) \rangle \tag{5.4}$$

and $\bullet$ *is the operator to link together temporal sequences of alerts that we introduced in Section 3.2.2. The evidence* $\tilde{e}$ *of the new fused case activation is* $\|\varrho\| = \|C_i \sim_s (\hat{a}_i \bullet \hat{a}_j)\|$.

We define the composability of two case activations as follows.

Definition 44 (Case Activation Composability) *Composability is the quality of two case activations of being able to be compounded together. We say that two case activations* $h_i = \langle C_i, \hat{a}_i, \check{a}_i, \varrho_i, \tilde{e}_i, t_i \rangle$ *and* $h_j = \langle C_j, \hat{a}_j, \check{a}_j, \varrho_j, \tilde{e}_j, t_j \rangle$ *are compoundable, represented by* $\textbf{compoundable}?(h_i, h_j)$*, when:*

1. *The corresponding sequential cases do not subsume repeated alerts. That is, the observed alerts in both case activations do not intersect.*

$$(\hat{a}_i \cap \hat{a}_j) = \emptyset \tag{5.5}$$

2. *The constraints expressed by the corresponding sequential case are kept.*

$$\textbf{constrainable}?(C.alerts.constraints, \hat{a}_i \bullet \hat{a}_j) = \textbf{true} \tag{5.6}$$

3. *Either:*

(a) Both case activations correspond to the same sequential case.

$$C_i = C_j \tag{5.7}$$

(b) One of the case activations corresponds to a new abstract case.

$$\begin{gathered}(\mathbf{abstract}?(C_i) \wedge \neg\mathbf{abstract}?(C_j)) \\ \vee \\ (\neg\mathbf{abstract}?(C_i) \wedge \mathbf{abstract}?(C_j))\end{gathered} \tag{5.8}$$

(c) Both case activations correspond to a new abstract case and there exists a sequential master case that can be abstracted to subsume the corresponding composition:

$$\begin{gathered}\mathbf{abstract}?(C_i) \wedge \mathbf{abstract}?(C_j) \\ \wedge \\ \exists C_k \in C^{(t)} : C_k \sqsubseteq \hat{a}_i \cup \hat{a}_j\end{gathered} \tag{5.9}$$

Thus:

$$\mathbf{compoundable}?(h_i, h_j) = \begin{cases} \mathbf{true} & \mathbf{if}\ (5.7 \vee 5.8 \vee 5.9) \wedge 5.5 \wedge 5.6 \\ \mathbf{false} & \mathbf{otherwise} \end{cases} \tag{5.10}$$

Therefore, those case activations that are compoundable can be fused together. As we see later, this is useful to reduce the number of hypotheses under consideration. The above definition can be easily extended to the union of n case activations. Given a set H made up of n fused case activations $H = \{h_{i_1}, \cdots, h_{i_n}\}$ and an additional case activation h_j then we define their fusion as follows:

$$H \uplus h_j = \begin{cases} H \cup \{h_j\} & \mathbf{if}\ \nexists h_i \in H : \mathbf{compoundable}?(h_i, h_j) \\ \{h_{i_k} \uplus h_j\}_{k=1}^{n} & \mathbf{otherwise} \end{cases} \tag{5.11}$$

Notice that each additional case activation is checked for composability against all previous case activations and therefore a number of case activations can be pieced together in the same operation. Likewise, given two sets H and H' of n and respectively m fused case activations $H = \{h_{i_1}, \cdots, h_{i_n}\}$ and $H' = \{h_{j_1}, \cdots, h_{j_m}\}$ we define their fusion as follows:

$$H \uplus H' = \bigcup_{k=1}^{m} (H \uplus h_{j_k}) \tag{5.12}$$

As we will see below, creating new case activations and assembling them together are the core processes that allow Ceaseless CBR to discover new cases. That is, at each iteration Ceaseless CBR not only tries to recognize past sequential cases but compound new ones together using new abstract cases generated for those alerts that are not covered by the current sequential case base (i.e., whose domains are empty). Next Sections are devoted to discuss in detail each one the Ceaseless CBR processes.

5.2 CEASELESS RETRIEVE

Ceaseless Retrieve continuously compares the sequence of alerts at hand with sequential cases in the case base and keep updated a collection of case activations that represent the current *situation*.

Algorithm 1 Ceaseless Retrieve

[htb]

Require: $\mathbf{C}^{(0)}$, $S^{(0)}$, θ, τ, wm;
Local: $\mathbf{H}$, $\mathbf{R}$, $\mathbf{A}$, C_i, $\mathbf{h}_i$
1: $\mathbf{H}^{(0)} = \emptyset$;
2: **while** true **do**
3: $\mathbf{A}^{(t)} = \emptyset$;
4: $\mathbf{R}^{(t)} = \mathbf{retrieve}(root(\vec{W}_{wm}^{(t)}(S^{(t)})), \mathbf{C}^{(t-1)}, \theta)$;
5: **for each** $C_i \in \mathbf{R}^{(t)}$ **do**
6: $\mathbf{h}_i = \langle C_i, \hat{a}_i, \check{a}_i, \varrho_i, \tilde{e}_i, t \rangle$;
7: $\mathbf{A}^{(t)} = \mathbf{A}^{(t)} \uplus \{\mathbf{h}_i\}$;
8: **end for**
9: **for each** $\psi_i \in \vec{W}_{wm}^{(t)} : \mathbf{D}^{(t)}(\psi_i) = \emptyset$ **do**
10: $\mathbf{h}_i = \langle \perp, \psi_i, \emptyset, \varrho^*, 1, t \rangle$;
11: $\mathbf{A}^{(t)} = \mathbf{A}^{(t)} \uplus \{\mathbf{h}_i\}$;
12: **end for**
13: $\mathbf{H}^{(t)} = \mathbf{H}^{(t-1)} \uplus \mathbf{A}^{(t)}$;
14: **for each** $\mathbf{h}_i \in \mathbf{H}^{(t)}$ **do**
15: **if** $\mathbf{h}_i.t - t \geqslant \tau$ **then**
16: $\mathbf{H}^{(t)} = \mathbf{H}^{(t)} - \{\mathbf{h}_i\}$;
17: **end if**
18: **end for**
19: **send(** $\mathbf{H}^{(t)}$, _CEASELESSREUSE); /* non-blocking call */
20: $[\mathbf{H}^{(t)}, \vec{P}^{(t)}]$ = **recv(**_CEASELESSREUSE);
21: **end while**

Ceaseless Retrieve proceeds as follows. Upon new alerts arrival, it retrieves similar sequential cases from the case base that best adjust to the current input. That is, it matches the sequence of alerts provided by the current window against the collection of sequences generated by each sequential master case. Ceaseless Retrieve establishes a case retrieval policy based on the frequency of occurrence of alerts. This policy

promotes rareness. Those cases that subsume alerts that are very common receive a low score whereas those cases that subsume rare alerts receive a high score. This helps our system to notice those situations that apparently convey more peril since the system is less used to dealing with them. The match is carried out using the dynamic sequence similarity measure introduced in Chapter 4 that behaves according to such policy. However, as we saw in Section 4.3 our dynamic sequence similarity returns an arbitrary value that varies over time based on the frequency of the alerts involved. In order to make the similarity value returned for each sequence equally comparable we use a normalized sequence similarity (as we will see later on). An elasticity threshold θ establishes how many cases are retrieved. We call this threshold *minimum evidence* since it sets up the minimum threshold above which a hypothesis is formulated. Therefore, cases that differ from the current sequence above a given minimum evidence threshold are removed. For each sequential case that is similar above this threshold a *case activation* is created. A case activation represents a hypothesis that could explain part of the current sequence of alerts. Ceaseless Retrieve pieces together case activations that fulfill the constraints specified by sequential cases. The path equality checking process described above ensures that such constraints are followed when two case activation are compounded together. This composition reduces the number of elements to be posteriorly considered when constructing overall *explanations*. The set made up of all current case activations defines the current *situation*.

Let's now discuss about Ceaseless Retrieve in more detail. The tasks performed by Ceaseless Retrieve, sketched by Algorithm 1, are as follows. Assume that a case base is initially composed of $n \geqslant 0$ sequential cases $\mathbf{C}^{(0)} = \{C_1, \cdots, C_n\}$. $\mathbf{H}^{(t)}$ denotes the set of current case activations. Inititally $\mathbf{H}^{(0)} = \emptyset$. $\mathbf{A}^{(t)}$ denotes the set of all new case activations at iteration t. It is set to $\emptyset$ at the beginning of each iteration (line: 3). $\vec{W}_{wm}^{(t)}$ extracts the next sequence of alerts from the alert stream $S^{(t)}$according to a given window model wm. We denote by $\mathbf{R}^{(t)}$ the set of sequential cases retrieved at iteration t (line: 4). Using the sequence of sorts returned by $root(\vec{W}_{wm}^{(t)}(S^{(t)}))$ and our dynamic similarity measure $\sim_s$, those cases that are similar to the sequence above a user-defined threshold $0 < \theta \leqslant 1$ are retrieved.

$$\mathbf{R}^{(t)} = \mathbf{retrieve}(root(\vec{W}_{wm}^{(t)}(S^{(t)})), \mathbf{C}^{(t-1)}, \theta) \tag{5.13}$$

Therefore:

$$\mathbf{R}^{(t)} = \left\{ C_i \in C^{(t-1)} : \frac{root(\vec{W}_{wm}^{(t)}(S^{(t)})) \sim_s C_i}{C_i \sim_s C_i} \geqslant \theta \right\} \tag{5.14}$$

We normalize the value of the dynamic sequence similarity function (between the current sequence of alerts and the corresponding sequential case) dividing by the value resulting from the dynamic sequence similarity function of the sequential case and itself. Since the value returned by our dynamic sequence similarity is arbitrary this is needed to fix the value boundaries for the threshold θ.

A case activation $\mathbf{h}_i = \langle C_i, \hat{a}_i, \check{a}_i, \varrho, \tilde{e}_i, t \rangle$ is created for each retrieved case containing observed and abduced alerts as well as an estimation of its rareness and

evidence and the time of the last observed alert (lines: 5–8). The new case activation is then fused with previous case activations generated during the same iteration (line: 7).

$$\mathbf{A}^{(t)} = \{\mathbf{h}_i : \mathbf{h}_i.C_i \in \mathbf{R}^{(t)}\} \tag{5.15}$$

We denote the domain of an alert ψ_i over time by $\mathbf{D}^{(t)}$:

$$\mathbf{D}^{(t)}(\psi_i) = \{C_j \in \mathbf{C}^{(t-1)} : root(\psi_i) \lhd^* C_j\} \tag{5.16}$$

That is, the set of sequential cases including an alert of the same sort. We say that an alert is *uncovered* when its domain is $\emptyset$. For each uncovered alert in $\vec{W}_{wm}^{(t)}$, a new case activation $\mathbf{h}_i = \langle \psi_i, \psi_i, \emptyset, \varrho^*, 1, t\rangle$ is created using a simple actionable tree composed uniquely of the observed alert (lines: 9–12):

$$\forall_{\psi_i \in \vec{W}_{wm}^{(t)} : \mathbf{D}^{(t)}(\psi_i) = \emptyset} \mathbf{A}^{(t)} \uplus \langle \psi_i, \psi_i, \emptyset, \varrho^* 1, t\rangle \tag{5.17}$$

The evidence of this kind of case activation is originally set to 1 and its rareness at its maximal value (i.e., $\varrho^* = \mathbf{max}\ \varrho_i, \forall \mathbf{h}_i$). Therefore, if it was not fused with other previous case activations it would promptly be prioritized since both its evidence and rareness are the highest. New case activations generated at the current iteration are fused with case activations created in previous iterations considering the constraints imposed by each sequential case (line: 13). For example, the same source and destination IP address in the whole sequence of alerts.

$$\mathbf{H}^{(t)} = \mathbf{H}^{(t-1)} \cup \mathbf{A}^{(t)} \tag{5.18}$$

Those case activations that have not been altered during a certain period of time (given by the paremeter τ) are filtered out from consideration (lines: 14–17):

$$\mathbf{H}^{(t)} = \mathbf{H}^{(t)} - \{\mathbf{h}_i \in \mathbf{H}^{(t)} : \mathbf{h}_i.t - t \geqslant \tau\} \tag{5.19}$$

Therefore, we say that $\mathbf{H}^{(t)}$ always keeps a number of up-to-date case activations for each pending alert. That is, alerts that have not received an explanation/prioritization yet. We denote the sequence of indexes of pending alerts by $\vec{P}^{(t)}$. We also say that, $\mathbf{H}^{(t)}$, the set composed of all current case activations (hypotheses on the occurrence of the corresponding attacks) defines the current situation. The current situation is then sent to the Ceaseless Reuse (line: 19). Ceaseless Reuse decides on which alerts to explain/prioritize first and returns those case activations and associated alerts for which it estimates that more evidence is needed before the corresponding alerts can be prioritized conveniently (line: 20). Next Section describes Ceaseless Reuse.

5.3 CEASELESS REUSE

Ceaseless Reuse receives as input a set of case activations and uses their information to produce a situation assessment. Ceaseless Reuse constantly searches the combination

of case activations that best explains the sequence of alerts most recently received and those that did not find an explanation in previous iterations (pending alerts). Ceaseless Reuse constantly hypothesizes about the risk and cost that the overall situation represents computing an estimation on the completion for each case activation using the inference mechanism provided by actionable trees.

Algorithm 2 Ceaseless Reuse

Local: $\mathbf{H}$, $\mathbf{h}_i$, $\mathbf{b}$, $\mathbf{B}$, $\mathbf{E}$, $\mathbf{e}$, $\mathbf{e}^*$, $\vec{U}$,
1: **while** true **do**
2: $\mathbf{H}^{(t)}$ = **recv**(_CEASELESSRETRIEVE);
3: **for each** $h_i^{(t)} \in \mathbf{H}^{(t)}$ **do**
4: $\mathbf{b}^{(t)+}(\mathbf{h}_i) = 1$;
5: **for each** $\psi_j \in \mathbf{h}_i.\breve{a}$ **do**
6: $\mathbf{b}^{(t)+}(\mathbf{h}_i) = \mathbf{b}^{(t)+}(\mathbf{h}_i) \times \big(L(\psi_j) + ((1 - L(\psi_j)) \times (1 - P(\psi_j|\mathbf{h}_i)))\big)$;
7: **end for**
8: $\mathbf{b}^{(t)-}(\mathbf{h}_i) = P(\mathbf{h}_i)P(\vec{P}^{(t)}|\mathbf{h}_i)$;
9: $\mathbf{b}^{(t)}(\mathbf{h}_i) = \mathbf{b}^{(t)+}(\mathbf{h}_i) \times \mathbf{b}^{(t)-}(\mathbf{h}_i)$;
10: **end for**
11: $[\mathbf{H}^{(t)}, \mathbf{H}_U^{(t)}, \vec{P}^{(t)}, \vec{U}^{(t)}] = \mathbf{rank}(\mathbf{H}^{(t)}, \mathbf{b}^{(t)})$;
12: **send**($[\mathbf{H}^{(t)}, \vec{P}^{(t)}]$, _CEASELESSRETRIEVE); % non-blocking call
13: $\mathbf{E}^{(t)} = \{\mathbf{e}_i \subseteq \mathbf{H}_U^{(t)} : \forall_{\psi_i \in \vec{U}^{(t)}} \exists \mathbf{h}_i \in \mathbf{e}_i \ : \mathbf{h}_i.C_i \sqsubseteq \ \psi_i\}$;
14: **for each** $\mathbf{e}_i \in \mathbf{E}^{(t)}$ **do**
15: $\mathbf{B}^{(t)+}(\mathbf{e}_i) = 1$;
16: **for** $\mathbf{h}_j \in \mathbf{e}_i$ **do**
17: $\mathbf{B}^{(t)+}(\mathbf{e}_i) = \mathbf{B}^{(t)+}(\mathbf{e}_i) \times \mathbf{b}^{(t)+}(\mathbf{h}_j)$;
18: **end for**
19: $\mathbf{B}^{(t)-}(\mathbf{e}_i) = P(\mathbf{e}_i)P(\vec{U}^{(t)}|\mathbf{e}_i)$;
20: $\mathbf{B}^{(t)}(\mathbf{e}_i) = \mathbf{B}^{(t)+}(\mathbf{e}_i) \times \mathbf{B}^{(t)-}(\mathbf{e}_i)$;
21: **end for**
22: $\mathbf{e}^{*(t)} = \mathbf{e}_i \in \mathbf{E}^{(t)} : \mathbf{B}^{(t)}(\mathbf{e}_i)$ is $\mathbf{maximal}$;
23: **send**($[\mathbf{e}^{*(t)}, \vec{U}^{(t)}]$, _CEASELESSREVISE); % non-blocking call
24: **end while**

Ceaseless Reuse proceeds as follows. It constantly combines case activations to form *explanations*. Explanations are ensembles of case activations that explain the whole sequence of alerts at hand. Ceaseless Reuse uses a belief function to determine which explanation is susceptible of being used to prioritize the corresponding alerts. However, if this process prioritizes alerts too soon, that is, without being completely sure of the presence of a (possible) exceptional situation, the number of false positives will be high and the ultimate objective (to triage the alert stream) will not be achieved. On the contrary, if it prioritizes too late and an exceptional situation is really occurring, then the time to enable a prompt response is reduced. Thus we take a decision-theoretic approach that maximizes the overall utility of each complete explanation and define a measure of urgency that guides the decisions of this process over time. That utility

takes into account the distinct costs that could be incurred after each prioritization including possible damage costs on the system under supervision. The utility values assigned to each explanation under consideration are estimated based on the success of similar decisions in previous situations. This process enables the segmentation of the sequence of alerts seen so far providing the best (sub-) explanation for each segment. Algorithm 2 sketches the tasks performed by Ceaseless Reuse.

At each iteration Ceaseless Reuse receives a number of case activations that define the current situation (line: 2): a number of competing hypotheses expressed in terms of case activations that explain alerts that have been observed so far. We say that a case activation $\mathbf{h}_j$ explains an alert ψ_k if the corresponding sequential case $h_j.C$ subsumes ψ_k (i.e., $\mathbf{h}_j.C_j \sqsubseteq \psi_k$ or $root(\psi_k) \lhd^* \mathbf{h}_j^{(t)}.C_j$). We explain the rest of tasks performed by Ceaseless Reuse in detail through the next Subsections.

5.3.1 Ranking Case Activations

For each case activation Ceaseless Reuse computes a belief function as the product of two other belief components (lines: 3–10): a negative component (that takes into account observed alerts[2]) and a positive component (that takes into account those alerts that have not been observed yet[3]):

$$\mathbf{b}^{(t)}(\mathbf{h}_i) = \mathbf{b}^{(t)+}(\mathbf{h}_i)\mathbf{b}^{(t)-}(\mathbf{h}_i) \tag{5.20}$$

On the one hand, the positive component is computed in terms of the alerts that have been abduced during the sequence similarity computation as follows:

$$\mathbf{b}^{(t)+}(\mathbf{h}_i) = \prod_{\psi_j \in \mathbf{h}_i.\breve{a}} \big(L(\psi_j) + ((1 - L(\psi_j))(1 - P(\psi_j|\mathbf{h}_i))\big) \tag{5.21}$$

For each abduced alert we consider every possible alternative. That is to say, we consider the probability that the alert is lost $L(\psi_j)$ and the probability that the alert is not lost $(1 - L(\psi_j))$ but it was not observed given the sequential case corresponding to the case activation at hand $(1 - P(\psi_j|\mathbf{h}_i))$. Later we will show through Equation 5.30 how to compute the probability that the alert was in fact observed given such case activation $P(\psi_j|\mathbf{h}_i)$.

On the other hand, the negative belief on a case activation $\mathbf{h}_i$ is computed as the posterior probability of the case activation given the current sequence of pending alerts (i.e., alerts that did not receive an explanation in previous iterations plus the alerts received in the current iteration).

$$\mathbf{b}^{(t)-}(\mathbf{h}_i) = P(\mathbf{h}_i|\vec{P}^{(t)}) \tag{5.22}$$

Using Bayes' theorem the posterior probability can be computed as follows:

[2]Negative symptoms.
[3]Positive symptoms.

$$P(\mathbf{h}_i|\vec{P}^{(t)}) = \frac{P(\mathbf{h}_i)P(\vec{P}^{(t)}|\mathbf{h}_i)}{P(\vec{P}^{(t)})} \tag{5.23}$$

The probability of the sequence of pending alerts $\vec{P}^{(t)}$ can be computed as follows:

$$P(\vec{P}^{(t)}) = \prod_{\psi_i \in \vec{P}^{(t)}} P(\psi_i) \tag{5.24}$$

Notice that Equation 5.24 is a constant for all case activations at the current iteration. Therefore the relative rank produced will be the same if we only use the numerator of Equation 5.23. Thus, the computation of $\mathbf{b}^{(t)-}(\mathbf{h}_i)$ can be approximated as follows:

$$P(\mathbf{h}_i|\vec{P}^{(t)}) \propto P(\mathbf{h}_i)P(\vec{P}^{(t)}|\mathbf{h}_i) \tag{5.25}$$

The probability of a case activation $P(\mathbf{h}_i)$ represents the probability of occurrence of the associated sequential case that in turn represents the probability of occurrence of the corresponding undesired situation (attack, fault, etc). This probability is computed using the inference mechanism provided by the predictive actionable trees that we saw in Chapter 3. From Equation 3.20:

$$P(\mathbf{h}_i) = 1 - \prod_{\psi_j \in \mathbf{h}_i.\hat{a}} \left(1 - \prod_{e \in path(\psi_j, \mathbf{h}_i.C_i)} \phi(e)\right) \tag{5.26}$$

Where, according to Definition 34, ϕ maps each arc e to a part-of strength computed using Equation 3.26.

The second part of the product of Equation 5.25 (i.e., the probability that given the occurrence of a sequential case we observe the sequence of alerts $\vec{P}^{(t)}$) is computed as follows:

$$P(\vec{P}^{(t)}|\mathbf{h}_i) = \prod_{\psi_i \in \vec{P}^{(t)}} (1 - P(\psi_i|\mathbf{h}_i)) \tag{5.27}$$

Using Bayes' theorem to compute $P(\psi_i|\mathbf{h}_i)$:

$$P(\psi_i|\mathbf{h}_i) = \frac{P(\psi_i)P(\mathbf{h}_i|\psi_i)}{P(\mathbf{h}_i)} \tag{5.28}$$

By actionable trees the probability of occurrence of a sequential case given an alert is:

$$P(\mathbf{h}_i|\psi_i) = \prod_{e \in path(\psi_j, \mathbf{h}_i.C_i)} \phi(e) \tag{5.29}$$

Substituting Equations 5.26 and 5.29 in Equation 5.28 we get:

$$P(\psi_i|\mathbf{h}_i) = \frac{P(\psi_i)\prod_{e\in path(\psi_j,\mathbf{h}_i.C_i)}\phi(e)}{1-\prod_{\psi_j\in\mathbf{h}_i.\hat{a}}\left(1-\prod_{e\in path(\psi_j,\mathbf{h}_i.C_i)}\phi(e)\right)} \tag{5.30}$$

Therefore $\mathbf{b}^{(t)-}(\mathbf{h}_i)$ can be approximated using Equations 5.25, 5.26, 5.27, and 5.30 as follows:

$$\mathbf{b}^{(t)-}(\mathbf{h}_i) \propto \left(1-\prod_{\psi_j\in\mathbf{h}_i.\hat{a}}\left(1-\prod_{e\in path(\psi_j,\mathbf{h}_i.C_i)}\phi(e)\right)\right)\times \prod_{\psi_i\in\vec{P}^{(t)}}\left(1-\frac{P(\psi_i)\prod_{e\in path(\psi_j,\mathbf{h}_i.C_i)}\phi(e)}{1-\prod_{\psi_j\in\mathbf{h}_i.\hat{a}}\left(1-\prod_{e\in path(\psi_j,\mathbf{h}_i.C_i)}\phi(e)\right)}\right) \tag{5.31}$$

Notice that a belief on a case activation does not need to be computed again and again at each new iteration. That is, we can cache beliefs on case activations and use them in later iterations. We only need to recompute them when they have varied at the current iteration (i.e., when $\mathbf{h}_i.t$ is equal to t).

Once we have computed the belief on each case activation we rank them and select a number of alerts to build an overall explanation (line: 11). The motivation for not considering all the alerts at each iteration is twofold. First, to reduce the combinatorial explosion in successive steps. The larger the number of alerts considered, the longer it will take to build an overall explanation. Second, it does not make sense to consider alerts for which our belief in their corresponding sequential case is too low, since it increases the probability of making a wrong judgment and therefore decreasing the expected utility. Different criteria could be applied to rank and select which alerts to explain first. For example, ranking them using either their evidence or belief and select always a fixed number of case activations or selecting all those case activations whose belief is above a given threshold. Depending on the criteria applied we could be trading off efficiency for risk. Our approach it to use instead a measure of *urgency* in the same way that it is applied to healthcare patient monitoring [HS97] and in mass casualty incidents (MCI) where patients are screened based on probable needs for immediate care and the resources available [BKS96, Str98].

5.3.2 Alert Urgency

Let us articulate the following question to explain the concept of alert urgency: what is the need for prioritizing an alert right away and what would occur if we waited to do it later? We answer that question using the notion of urgency. Urgency is the degree to which an immediate prioritization is required [HS97]. Therefore, we say that we do need to prioritize an alert only if it is *urgent*, while we could wait to do it later at no risk when the alert is not urgent. We compute the urgency of each alert in terms of the expected utility of prioritizing it right now, using our current degree of belief on the hypotheses that explain it, versus the expected utility of waiting to do

it after more evidence has been gathered. In the latter case, a higher degree of belief is obtained but at the risk of alerting too late about a undesired situation. Urgency provides a measure of the relative benefits of increased performance making more accurate judgments (selecting the appropriate case activation) versus the cost that could imply the occurrence of a undesired situation. As we will see experimentally in Chapter 6, alert urgency has a direct impact on the number of false positives and false negatives that our system generates.

Since $\mathbf{H}^{(t)}$ denotes the current situation (i.e., all the case activations that represent possible occurrences of cases), for each case activation $\mathbf{h}_i \in \mathbf{H}^{(t)}$ two particular decisions are possible:

$$d^{(t)}(\mathbf{h}_i) \rightarrow \{\mathbf{wait}, \mathbf{prioritize}\} \tag{5.32}$$

We denote by $d_0^{(t)}(\mathbf{h}_i)$ the act of leaving the case activations alone for further iterations (i.e., to wait and not prioritize the corresponding alerts yet) and by $d^{*(t)}(\mathbf{h}_i)$ the *best decision* at each iteration t. The best decision is that which causes the situation with the highest *utility*. We define the utility as a function that estimates a real number for each possible situation and denoted it by U. Given that it is almost impossible to know for sure how the next situation is influenced for each decision we compute for each decision its *expected utility*.

$$E[U(\mathbf{H}^{(t+1)}|d^{(t)}(\mathbf{h}_i)] = \sum_{\mathbf{h}_{i_{t+1}} \in \mathbf{H}^{(t+1)}} P(\mathbf{h}_{i_{t+1}}|d^{(t)}(\mathbf{h}_i))U(\mathbf{h}_{i_{t+1}}|d^{(t)}(\mathbf{h}_i)) \tag{5.33}$$

The expected utility of an individual case activation $\mathbf{h}_i$ is computed as follows:

$$\begin{array}{l} E[U(\mathbf{h}_i)] = \\ \quad \mathbf{b}^{(t)}(\mathbf{h}_i) \quad \times \quad \Big(\quad P(\mathbf{h}_i.C) \quad \times \quad B(D+|C+) \quad + \\ \qquad\qquad\qquad\quad (1-P(\mathbf{h}_i.C)) \quad \times \quad C(D+|C-)\Big) \\ \quad (1-\mathbf{b}^{(t)}(\mathbf{h}_i)) \quad \times \quad \Big(\quad \text{risk-cost}(\mathbf{h}_i.C) \qquad\qquad\quad + \\ \qquad\qquad\qquad\quad (1-P(\mathbf{h}_i.C)) \quad \times \quad B(D-|C-)\Big) \end{array} \tag{5.34}$$

where risk-cost$(\mathbf{h}_i.C)$ is computed according to Equation 5.1, $P(\mathbf{h}_i.C) = \mathbf{h}_i.C.$risk.rate-of-occurrence, $B(D+|C+)$ represents the benefit obtained for correctly prioritizing when the alert is dangerous (i.e., positive decision when the condition of interest is present), $B(D-|C-)$ represents the benefit obtained for correctly not prioritizing an innocuous alert (i.e., negative decision when the condition of interest is absent), and $C(D+|C-)$ is the cost caused by a false positive. In Section 6.2.2 we will consider the evaluation of these scenarios where correct decision outcomes have associated a benefit B and incorrect decision outcomes have associated a cost C.

Therefore, the best decision is that which maximizes the expected utility:

$$d^{*(t)}(\mathbf{h}_i) = \mathbf{argmax}_{d^{(t)}} E[U(\mathbf{H}^{(t+1)}|d^{(t)}(\mathbf{h}_i)] \tag{5.35}$$

However, it is easy to appreciate that the next situation not only depends on the decision made not just on a single case activation but over all the case activations $\mathbf{H}^{(t)}$. We denote by $D^{(t)}$ the set of all decisions made over all the case activations $\mathbf{H}^{(t)}$. Moreover, decisions $D^{(t)}$ made at time t affect all the sequence of subsequent situations $[\mathbf{H}^{(t+1)}, \cdots, \mathbf{H}^{(\infty)}]$. Likewise, a given situation $\mathbf{H}^{(t)}$ is influenced by the corresponding sequence of previous decisions $[D^{(t_0)}, \cdots, D^{(t-1)}]$, where t_0 is the time at which supervision was started. Obviously, we only consider finite sequences of situations that extend along a maximum of T iterations (i.e., $[\mathbf{H}^{(t+1)}, \cdots, \mathbf{H}^{(t+T)}]$). We call T the decision horizon. When $T = 1$ we refer to the iteration horizon as *myopic*. The best decision can then be defined as the argument that maximizes the *maximum expected utility*:

$$d^{*(t)}(\mathbf{h}_i) = \mathbf{argmax}_{d^{(t)}} \mathcal{U}(D^{(t+1)}|d^{(t)}(\mathbf{h}_i), \mathbf{H}^{(t)}) \tag{5.36}$$

where the maximum expected utility is defined as follows:

$$\mathcal{U}(D^{(\mathbf{t})}|d^{(\mathbf{t-1})}(\mathbf{h}_i), \mathbf{H}^{(\mathbf{t-1})}) = \\ \mathbf{max}_{d^{(\mathbf{t})}} \begin{cases} \mathcal{U}(D^{(\mathbf{t+1})}|d^{(\mathbf{t})}(\mathbf{h}_i), \mathbf{H}^{(\mathbf{t})}) & \textbf{if } t+1 \leqslant \mathbf{t} \leqslant t+T-2 \\ E[U(\mathbf{H}^{(t+1)}, \cdots, \mathbf{H}^{(t+T)}|d^{(t)}(\mathbf{h}_i)] & \textbf{if } \mathbf{t} = t+T-1 \end{cases} \tag{5.37}$$

We can finally define the urgency $\upsilon^{(t)}(\mathbf{h}_i)$ of a case activation $\mathbf{h}_i$ as follows:

$$\upsilon^{(t)}(\mathbf{h}_i) = \frac{\mathcal{U}(D^{(t_{j+1})}|d^{*(t_j)}, \mathbf{H}^{(t_j)}) - \mathcal{U}(D^{(t_{j+1})}|d_{\mathbf{0}}^{(t_j)}, \mathbf{H}^{(t_j)})}{t_{j+1} - t_j} \tag{5.38}$$

Urgency measures the expected utility of prioritizing now against the utility obtained if the prioritization is delayed. Given that not all alerts require to be treated with the same urgency, we say that urgency allows us to trade off in prioritizing an alert versus continuing computation as well as choosing among competing case activations. Therefore, we can dedicate resources to those alerts that seem more urgent. Thus, we say that given the set of current case activations $\mathbf{H}^{(t)}$ and their current beliefs $\mathbf{b}^{(t)}$ the function $\mathbf{rank}(\mathbf{H}^{(t)}, \mathbf{b}^{(t)})$ (line: 11) partitions alerts and their corresponding case activations into those that are urgent (i.e., need an immediate prioritization) and and those that will remain pending (i.e., waiting for further evidence). This function also uses the rareness of the occurrence of the subsumees $\mathbf{h}_i.\varrho$ to rank alerts. Thus, when several alerts have the same urgency then the rarer the alert, the higher the rank it gets. We denote by $\vec{U}^{(t)}$ the alerts that are urgent and need to be explained at the current iteration. Alerts that can wait to be prioritized form the sequence of pending alerts $\vec{P}$ for the next iteration. Likewise, $\mathbf{H}_U^{(t)}$ denotes the set of case activations that explain urgent alerts and that will be used at the current iteration to compound explanations whereas $\mathbf{H}^{(t)}$ denotes the set of case activations that remains for further iterations. Both case activations that remains for further iterations and pending alerts are sent back to Ceaseless Retrieve (line: 12).

Then, Ceaseless Reuse creates explanations using the set of case activations that explain urgent alerts $\mathbf{H}_U^{(t)}$ and select the explanation whose belief is maximal to

propose it to the user as the most plausible explanation. In next Subsection we will see how to compound and rank explanations.

5.3.3 Ranking Explanations

An explanation $\mathbf{e}_i$ is a subset of $\mathbf{H}_U^{(t)}$ that explains all alerts in $\vec{U}^{(t)}$. An explanation $\mathbf{e}_i$ is said to explain an alert ψ_k if it contains at least a case activation $\mathbf{h}_j$ that explains ψ_k. $\mathbf{E}^{(t)}$ represents the set of all explantions—combinations of case activations that explain completely the current sequence of urgent alerts $\vec{U}^{(t)}$. $\mathbf{E}^{(t)}$ is computed following a parsimonious principle [PR90]. Based on the observation that the probability of multiple coincidental sources is low we induce the following heuristic: $\mathbf{e}'$ is not included in $\mathbf{E}^{(t)}$ if it contains a case activation that is already contained by $\mathbf{e}'' \in \mathbf{E}^{(t)}$ such that its size is lower:

$$\mathbf{E}^{(t)} = \{\mathbf{e}' \subseteq \mathbf{H}_U^{(t)} : \forall_{\psi_i \in \vec{U}^{(t)}} \exists \mathbf{h}_i \in \mathbf{e}' \ : \mathbf{h}_i.C_i \sqsubseteq \ \psi_i \\ \wedge \\ \nexists \mathbf{e}'' : |\mathbf{e}''| < |\mathbf{e}'| \wedge (\mathbf{e}' \cap \mathbf{e}'') \neq \emptyset\} \quad (5.39)$$

Therefore those explanations that contain case activations that appear in other explanations that are already in $\mathbf{E}^{(t)}$ and whose size is lower are not contemplated (line: 13). The next step is to compute an estimation of the goodness for each explanation in $\mathbf{E}^{(t)}$ (lines: 14–21). We use a likelihood measure to conveniently rank explanations. The explanation with the highest likelihood (the most probable) can be chosen as the problem solution. We define $\mathbf{B}^{(t)}(\mathbf{e}_i)$ as a belief function that represents the likelihood that all cases in $\mathbf{e}_i$ have occurred and $\mathbf{e}_i$ explains all alerts in $\vec{U}^{(t)}$. $\mathbf{B}^{(t)}$ is computed using the beliefs $\mathbf{b}^{(t)}$ previously computed for each case activation $\mathbf{h}_i$. We use again a belief function based on a double component.

$$\mathbf{B}^{(t)}\Big(\mathbf{e}_i\Big) = \mathbf{B}^{(t)+}\Big(\mathbf{e}_i\Big)\mathbf{B}^{(t)-}\Big(\mathbf{e}_i\Big) \quad (5.40)$$

The belief $\mathbf{B}^{(t)+}$ (based on positive symptoms) gives a degree of suitability for each explanation that is based on the intuition that when some of the expected alerts have not occurred yet it is a positive symptom that allows us to decrease our belief on the hypotheses that compound the explanation at hand.

$$\mathbf{B}^{(t)+}(\mathbf{e}_i) = \prod_{\mathbf{h}_i \in \mathbf{e}_i} \prod_{\psi_j \in \mathbf{h}_i.\breve{a}} \mathbf{b}^{(t)}(\mathbf{h}_i) \quad (5.41)$$

Using Equation 5.21:

$$\mathbf{B}^{(t)+}(\mathbf{e}_i) = \prod_{\mathbf{h}_i \in \mathbf{e}_i} \prod_{\psi_j \in \mathbf{h}_i.\breve{a}} \big(L(\psi_j) + ((1 - L(\psi_j))(1 - P(\psi_j|\mathbf{h}_i))\big) \quad (5.42)$$

The belief component based on negative symptoms determines the relative likelihoods of multiple case activations according to their posterior probabilities.

$$\mathbf{B}^{(t)-}(\mathbf{e}_i) = P(\mathbf{e}_i|\vec{U}^{(t)}) \tag{5.43}$$

By Bayes' theorem we know that

$$P(\mathbf{e}_i|\vec{U}^{(t)}) = \frac{P(\mathbf{e}_i)P(\vec{U}^{(t)}|\mathbf{e}_i)}{P(\vec{U}^{(t)})} \tag{5.44}$$

In order to compare posterior probabilities it is only necessary to compare the joint probabilities since the normalization factor $P(\vec{U}^{(t)}) = \prod_{\psi_i \in \vec{U}^{(t)}} P(\psi_i)$ is a constant for all competing explanations at iteration t. Therefore:

$$P(\mathbf{e}_i|\vec{U}^{(t)}) \propto P(\mathbf{e}_i)P(\vec{U}^{(t)}|\mathbf{e}_i) \tag{5.45}$$

The a priori probability of a explanation $\mathbf{e}_i$ is given by:

$$P(\mathbf{e}_i) = \prod_{\mathbf{h}_i \in \mathbf{e}_i^{(t)}} P(\mathbf{h}_i) \tag{5.46}$$

that can be estimated using Equation 5.26 as follows:

$$P(\mathbf{e}_i) = \prod_{\mathbf{h}_i \in \mathbf{e}_i} \left(1 - \prod_{\psi_j \in \mathbf{h}_i.\hat{a}} \left(1 - \prod_{e \in path(\psi_j, \mathbf{h}_i.C_i)} \phi(e)\right)\right) \tag{5.47}$$

The conditional probability of $\vec{U}^{(t)}$ given $\mathbf{e}_i$ is computed as follows:

$$P(\vec{U}^{(t)}|\mathbf{e}_i) = \prod_{\psi_i \in \vec{U}^{(t)}} \left(1 - \prod_{\mathbf{h}_i \in \mathbf{e}_i} (1 - P(\psi_i|\mathbf{h}_i))\right) \tag{5.48}$$

Then, by Equation 5.30,

$$P(\vec{U}^{(t)}|\mathbf{e}_i) = \prod_{\psi_i \in \vec{U}^{(t)}} \left(1 - \prod_{\mathbf{h}_i \in \mathbf{e}_i} \left(1 - \frac{P(\psi_i) \prod_{e \in path(\psi_j, \mathbf{h}_i.C_i)} \phi(e)}{1 - \prod_{\psi_j \in \mathbf{h}_i.\hat{a}} \left(1 - \prod_{e \in path(\psi_j, \mathbf{h}_i.C_i)} \phi(e)\right)}\right)\right) \tag{5.49}$$

Finally, from Equations 5.45, 5.47, and 5.49:

$$\mathbf{B}^{(t)-}(\mathbf{e}_i) \propto \prod_{\mathbf{h}_i \in \mathbf{e}_i} \left(1 - \prod_{\psi_j \in \mathbf{h}_i.\hat{a}} \left(1 - \prod_{e \in path(\psi_j, \mathbf{h}_i.C_i)} \phi(e)\right)\right) \times$$
$$\prod_{\psi_i \in \vec{U}^{(t)}} \left(1 - \prod_{\mathbf{h}_i \in \mathbf{e}_i} \left(1 - \frac{P(\psi_i) \prod_{e \in path(\psi_j, \mathbf{h}_i.C_i)} \phi(e)}{1 - \prod_{\psi_j \in \mathbf{h}_i.\hat{a}} \left(1 - \prod_{e \in path(\psi_j, \mathbf{h}_i.C_i)} \phi(e)\right)}\right)\right) \tag{5.50}$$

All explanations $\mathbf{e}_i$ in $\mathbf{E}^{(t)}$ are ranked according to $\mathbf{B}^{(t)}$. The best explanation $\mathbf{e}^{*(t)}$, the one that is maximal, among all competing explanations, is chosen as the problem solution and sent to Ceaseless Revise for user's revision (line: 23). Assuming that all non-urgent alerts are originated by the same source we could say that the best explanation $\mathbf{e}^{*(t)}$ produces a m-segmentation $\mathbb{S}$ (according to Definition 12) of the current sequence of alerts $\vec{W}_{wm}^{(t)}(S^{(t)})$ such that $\mathbb{S}(\vec{W}_{wm}^{(t)}(S^{(t)}), m) \in \mathbf{Seg}^m(\vec{W}_{wm}^{(t)}(S^{(t)}))$ and $m \geqslant |\mathbf{e}^{*(t)}| + 1$ since each alert is explained by either a sequential case that was included in the best explanation $\mathbf{e}^{*(t)}$ or is a non-urgent alert.

5.4 CEASELESS REVISE

Ceaseless Revise continuously provides a human (expert) operator with the set of most likely explanations given the alerts received so far (instead of presenting a solution periodically) [4]. The user can at any moment revise whatever combination of case activations that is being proposed as the best explanation and propose modifications. The revision process conveys the identification of new cases (most times including alerts that are not considered by other case activations) and new classification of pre-existing cases—altering their risk , cost or priority factors. Ceaseless Revise proceeds according to Algorithm 3.

Ceaseless Reuse receives a collection of urgent alerts $\vec{U}^{(t)}$ and an optimal explanation $\mathbf{e}^{*(t)}$ that explains all alerts in $\vec{U}^{(t)}$ (line: 2). Then, Ceaseless Reuse requires user's feedback to get a supervised version of the optimal explanation (line: 3). The operator's feedback may create a completely new sequential case or update a past sequential case (adding, deleting or altering observable events or constraints among them), altering its risk (threat, exposure, or cost) or the corresponding prioritization. Given the explanations, the user may decide to perform further observations or repairs before reporting the results to the system. The operator's feedback produces a set of revised solutions that in turn produces the triage of the corresponding alerts and initiates a back-propagation process that automatically updates the current case

[4] Notice that this means that the user is able to interrogate the model at any time to see what is happening but at the application level, as we will see in Chapter 7, the user receives, for feasibility and usability purposes, either a periodical email informing that nothing new has occurred or a spontaneous email notifying that some extremely urgent is undergoing.

Algorithm 3 Ceaseless Revise

Local: $\mathbf{e}^*$, $\mathbf{s}^*$,$\vec{U}$, $\mathbf{p}$, $\mathbf{f}$, $\mathbf{h}_i$
1: **while** true **do**
2: $[\mathbf{e}^{*(t)}, \vec{U}^{(t)}]$ = **recv**(_CEASELESSREUSE);
3: $\mathbf{s}^{*(t)} = \mathbf{userfeedback}(\mathbf{e}^{*(t)}, \vec{U}^{(t)})$
4: **for each** $\mathbf{h}_i \in \mathbf{s}^{*(t)}$ **do**
5: **for each** $\psi_i \in \vec{U}^{(t)} : \psi_i \in \mathbf{h}_i.\hat{a}$ **do**
6: $\mathbf{p}^{(t)}[\psi_i] = \mathbf{h}_i^{(t)}.C_i.priority$;
7: **end for**
8: **for each** $\psi_i \notin \vec{U}^{(t)} : \psi_i \in \mathbf{h}_i.\check{a}$ **do**
9: $\mathbf{f}^{(t)}[\psi_i] = \mathbf{h}_i.C_i.priority$;
10: **end for**
11: **end for**
12: **send**($\mathbf{p}^{(t)}$, $\mathbf{f}^{(t)}$, _USER); % non-blocking call
13: **send**($[\mathbf{s}^{*(t)}, \vec{U}^{(t)}]$, _CEASELESSRETAIN); % non-blocking call
14: **end while**

activations. Notice that user's feedback can also be provided by an automatic process that keeps a model of the system under supervision. For example, in our application (see Chapter 7) we used a model created and kept up-to-date automatically by a network scanner such as Nessus. The SSO can carefully define a threshold such that individual explanations whose likelihood is above it either produce an automatic triage of the corresponding alerts or require his or her intervention.

In any case, as soon as Ceaseless Reuse receives feedback it proceeds to prioritize alerts accordingly. For each alert ψ_i in $\vec{U}^{(t)}$ that belongs to the subsumees of a case activation $\mathbf{h}_i$ in $\mathbf{e}^{*(t)}$ the priority corresponding to the subsumer $\mathbf{h}_i^{(t)}.C_i.priority$ is assigned (lines: 5–6). Moreover, for each alert ψ_i that does not belong to $\vec{U}^{(t)}$ but belongs to the abduced alerts of a case activation $\mathbf{h}_i$ in $\mathbf{e}^{*(t)}$ a new alert prediction is immediately generated and its corresponding priority assigned (lines: 8–10). All prioritized alerts $\mathbf{p}^{(t)}$ as well as all predicted alerts $\mathbf{f}^{(t)}$ are then sent to the user for its further processing (line: 12). Likewise, this process rather than a human user could be an automatic process that is able to initiate a convenient response or a more sophisticate reasoning module that uses predictions as input. For example, a plan recognizer as we will propose as part of our future work in Chapter 8. In the next Chapter, we will evaluate Ceaseless CBR *predictiveness*. That is to say, its capability of of predicting in advance alerts that will posteriorly occur.

The sequential case base must now be updated with the revised solution. This is the task performed by Ceaseless Retain. Thus, Ceaseless Revise sends the revised solution, prioritized alerts, and the sequence of urgent alerts to Ceaseless Retain (line: 13).

5.5 CEASELESS RETAIN

Once a solution has been revised by the user, the Ceaseless Retain process updates the sequential case base accordingly. For each hypothesis that conformed the final explanation distinct information is retained depending on the sequential case that was employed. If the sequential case was completely new or adapted from a previous sequential case (i.e., it is abstract) then a new sequential case is created in the sequential case base. For each sequential case that already existed in the case base its statistics are updated accordingly. Moreover, Ceaseless Retain continuously updates the likelihood of occurrence of each sequential case as well as the likelihood of occurrence of each alert in each sequential case. Thus, we say that the retain process is also performed on a continuous basis. Those sequential cases whose probability of occurring together is above the probability of occurring separately are merged together in a new sequential case. Algorithm 4 sketches the tasks performed by Ceaseless Retain.

Algorithm 4 Ceaseless Retain

Require: $\mathbf{s}^*$,$\vec{U}$;
Local: t_i, t_f,C_i, $\mathbf{h}_i$;
1: **while** true **do**
2: $[\mathbf{s}^{*(t)}, \vec{U}^{(t)}]$ = **recv**(_CEASELESSREVISE);
3: **for each** $\mathbf{h}_i \in \mathbf{s}^{*(t)}$ **do**
4: $C_i = \mathbf{h}_i.C$;
5: $C_i.\mathbf{occurrence} = C_i.\mathbf{occurrence} + 1$;
6: $C_i.\mathbf{epidoses} = C_i.\mathbf{epidoses} \cup \mathbf{h}_i.\hat{a}$;
7: **if** $\mathbf{abstract}?(C_i)$ **then**
8: $\mathbf{C}^{(t)} = \mathbf{C}^{t-1} \cup \mathbf{h}_i.C$;
9: **end if**
10: **end for**
11: **for each** $\psi_i \in \vec{W}_{wm}^{(t)}$ **do**
12: $Q_{root(\psi_i)}^{(t)} = Q_{root(\psi_i)}^{(t)} + 1$;
13: **end for**
14: **for each** $C_i \in \mathbf{C}^{(t)}$ **do**
15: **for each** $\psi_j \in C_i$ **do**
16: **for** $e \in path(\psi_j, C_i)$ **do**
17: $\phi(e) = \frac{\mathbf{s}(C_i,\vec{S},wm}{\mathbf{s}(root(\psi_j),\vec{S},wm)}$;
18: **end for**
19: **end for**
20: **end for**
21: **end while**

Ceaseless Retain receives from Ceaseless Reuse a collection of revised hypothesis $\mathbf{s}^{*(t)}$ according to which current urgent alerts in $\vec{U}$ have been prioritized (line: 2). For each sequential case that formed part of a hypothesis, if the sequential (master) case previously existed Ceaseless Retain updates it number of occurrences (see Sections

3.2.2 and 3.4.4) (lines: 3–9). Moreover, if the the order of the subsumees differ from the current episodes that the sequential case represents then the new episode is also stored and the corresponding actionable tree updated in accordance (line: 6). If no sequential master case for the case at hand previously existed, then additionally a new sequential master is created and stored in the case base (lines: 7–9):

$$\mathbf{C}^{(t)} = \mathbf{C}^{t-1} \cup \mathbf{h}_i.C$$

This process generates a new master case where all the irrelevant details corresponding to the occurrence at hand have been dismissed. Finally, Ceaseless Retain recomputes the likelihood of occurrence (part-of strength) for each alert sort within each sequential case (lines: 11–20). That is, the absolute frequency of all alerts in the current window is updated (lines: 11–13) as well as the relative frequency of all alerts (not shown in the algorithm) and the part-of-strength within each sequential case according to Equation 3.26 (lines: 14–20).

Therefore, we say that Ceaseless Retain continuously adds useful information to the existing knowledge.

5.6 CEASELESS CBR RECAP

Ceaseless CBR translates the general goal of recognizing an undesired situation into an on-line composition of a collection of individual and predefined sequences of symptom events (alerts). Therefore, Ceaseless CBR can be seen as a constructive situation awareness process governed ceaselessly by: (i) observational data—current gathered evidence, and (ii) a sequential case base—past experience. The sequence of alerts received so far pushes towards a situation (exceptional or not) whereas the sequential case base pulls towards the best explanation expressed in terms of partially-matched sequential cases (case activations). Ceaseless CBR tries to find the best explanation to the current situation interpreted in terms of past sequential cases and prioritizes alerts accordingly. Those alerts that cannot be explained using the current sequential case base are proposed to the user in the form of new abstract cases. The user assesses their risk and assigns a proper prioritization. Then, new cases are stored for subsequent reuse in further analyses. Ceaseless CBR is decomposed into four parallel sub-processes that run forever in an infinite loop. Ceaseless Retrieve instantiates several plausible hypotheses for the current sequence of alerts using similar sequences provided by a sequential case base. Ceaseless Reuse assesses the goodness of each hypothesis, generates explanations compounding hypothesis, and discerns between competing explanations. Ceaseless Reuse decides on how immediately an alert needs to be prioritized selecting for prioritization only those alerts that are urgent. On the contrary, it makes non-urgent alerts to wait to be prioritized until additional support for refining the associated hypotheses is gathered. Ceaseless Reuse determines appropriate decisions by assigning an utility value to each hypothesis under consideration and uses a measure of urgency to maximize the expected utility and minimize the overall risk of each decision. In some general sense, it could be said that Ceaseless

Retrieve generates explanations considering only the similarity between the current window and the past cases, whereas Ceaseless Reuse takes into account additional criteria such as rareness and uncertainty on the hostility, exposure, and cost of the cases at hand. Ceaseless Revise provides the user with an explanation of the current situation and obtains his or her feedback that is subsequently used to adapt the explanation proposed. Human intervention is not mandatory, since a process providing an up-to-date model of the system under surveillance could take the human out of the loop. For example, indicating whether the system is vulnerable or not to the threats signaled by alerts. Ceaseless Retain, the process that completes the CBR cycle, stores new sequential cases and keeps up-to-date the frequency of occurrence of each alert within each sequential case so that Ceaseless Retrieve and Reuse can act in accordance to the concrete distribution of alerts of the system under supervision.

Notice that Ceaseless CBR carries out two interlaced tasks at each iteration. On the one hand, the recognition of past situations by means of previously stored sequential cases, and, on the other hand, the discovery of new sequential cases. We call sequential case discovery the process by which:

- Ceaseless Retrieve generates a case activation with its corresponding abstract case for each previously unknown alert (i.e., alerts whose domain is empty).
- Ceaseless Retrieve fuses several case activations together providing new large abstract cases.
- Finally, and once revised by Ceaseless Revise, these new abstract cases are stored by Ceaseless Retain as a master case that will generalize particular sequential cases in the future. In this process, some superfluous details of the concrete sequential case that generated the abstract case are dismissed. For example, the concrete source IP address or destination.

Our sequential case discovery process is comparable to mining approaches that search association rules in streams of events. As a matter of fact, in a separate branch of our research we implemented and tested Mannila's algorithms (MINEPI and WINEPI) [MTV97]. However, those algorithms (and the like) show an excessive number of drawbacks from our point of view. First of all, they run off-line. Second, they require a high number of passes over the data stream what is quite inefficient; an arbitrary parameters (such as support, confidence, and window size) have to be adjusted finely; and worst of all they generate an unmanageable number of rules. Julisch and Dacier drawn similar conclusions and instead they proposed AOI (Attribute-Oriented Induction) for mining alert databases [JD02].

An additional feature of Ceaseless CBR is its ability to handle burst situations efficiently. Ceaseless CBR is able to identify burst situations controlling the number of distinct alerts in the current window. When a burst situation arises (e.g., the number of distinct alerts is below a pre-specified threshold or the number of unique alerts in the current window is 1), Ceaseless CBR retrieves a special sequential case that subsumes the whole window of alerts and proposes it as the only possible explanation. This allows Ceaseless CBR to consume thousands of alerts in near real-time.

Finally, notice that Ceaseless CBR have been devised bearing the following policies in mind. In increasing order of priority:

1. *Explain first rare alerts*. Since alerts seldom occurring are unusual cases for which the system hardly had got practical experience to deal with. Saying this in the other way around, it is supposed that the system have plenty of experience to deal with common alerts and they do not require prompt attention.

2. *Explain first uncovered alerts*. Alerts for which there is no previous experience should be presented to the user as soon as possible. That is, when no sequential cases subsuming them can be found in the case base.

3. *Explain first urgent alerts*. Since these alerts are more likely to entail peril and therefore they must take precedence over any other alert in the explanation/prioritization process.

The objective of these policies is to dedicate computational resources to those alerts whose corresponding prioritization produce the *maximum expected utility*.

This Chapter has described the processes that compose Ceaseless CBR in detail. In the next Chapter, we will establish a framework for the evaluation of alert triage systems that we subsequently will use to evaluate experimentally the performance of Ceaseless CBR.

6

Performance Evaluation

This Chapter is devoted to evaluation of the techniques proposed through the thesis. We will describe our experimental setting and evaluate Ceaseless CBR along five different dimensions: (i) performance using a new formal framework for the evaluation of alert triage (ii) capability of discovering new sequential cases; (iii) CPU time requirements; (iv) capability of predicting future alerts (or preemptive ratio); and last but not least important, alert load reduction achieved. This Chapter is divided into three sections. Section 6.1 overviews ROC analysis and exposes some well-known concepts in the context of detection systems. Section 6.2 describes the construction of a decision-theoretic framework for the evaluation of alert triage that is based on a combination of ROC analysis and computational geometry. We show how this framework not only allows one to select the best alert triage system but also to make practical choices when assessing different components of alert triage. We introduce *t-area* a new measure for computing the performance of non-parametric systems in ROC spaces. The performance of alert triage depends on the environment where the evaluation is carried out (i.e., whether misdetection costs are considered or not and whether are known beforehand or not). We contemplate three possible environments (scenarios) in increased order of uncertainty and therefore of complexity: *ideal environments*, *cost-based environments*, and *imprecise environments*. Finally, Section 6.3 explains the set of experiments that we have conducted using the data-sets described in Appendix A and draws some interesting conclusions on the performance achieved. Our evaluations demonstrate how a Ceaseless CBR-enhaced IDS system provides improvement in both the number of alerts that could be managed by a SSO and the speed with which they could be triaged.

6.1 ROC ANALYSIS

Alert triage can be formulated as a detection or classification task[1] as we saw in Section 1.2.1. ROC analysis is a key evaluation technique for comparing the performance of detection and classification systems [Swe96] and lays the groundwork of this Chapter. ROC analysis was originally introduced in the field of signal detection theory in the 50's [Ega75]. The term Receiver Operating Characteristic refers to the performance (the operating characteristic) of a human or mechanical observer (the receiver) that has to discriminate between radio signals contaminated by noise (such as radar images) and noise alone [Ega75]. Nowadays, ROC analysis has become a common tool in medical decision making to qualify diagnostic tests specially in radiology [MSL76, HHJ97, HM82, HH02] and in psychology for sensory and cognitive processes [SDM00]. Recently, ROC analysis has also been proposed to compare and improve machine learning classifiers [PFK98, PF01, FFH02, WF02] and information retrieval systems [MP01].

Detection is the judgement on the presence of some condition in the present or the occurrence of an event in the future (e.g., an intrusion). A detection system, a person or a device or a combination of both, makes (positive or negative) decisions regarding the presence or absence of some condition [SDM00]. Depending on the output, detection systems can be classified into non-parametric and parametric detection systems.

6.1.1 Non-parametric Detection Systems

When the output of a detection system only considers yes-no alternatives then it has the four possible decision outcomes shown in Table 6.1.

Table 6.1 Four possible outcomes of a non-parametric detection system.

	Condition	
Decision	Present	Absent
Positive	True Positives	False Positives
Negative	False Negatives	True Negatives

On the one hand, a detection system makes two types of correct decisions: true positives (TP) and true negatives (TN). That is, the judgement on the presence and respectively absence of the condition of interest is correct. On the other hand, a detection system can commit two kinds of decision errors: false positives (FP) and false negatives (FN). Respectively, deciding that some condition is present when it is really absent (e.g., an alert is signaled when there is a manifest absence of intrusive

[1] Respectively known is statistics as binary and n-ary hypotheses testing.

behavior), and, conversely, deciding that some condition is absent when it is really present (e.g., an attack is undergoing and no alert is evoked).

The evaluation of a non-parametric detection system can be performed using a *confusion matrix*. A confusion matrix is a two-by-two table that collects the frequency of the four possible decision outcomes for the evaluation of a detection system that made N decisions ($N = TP + TN + FP + FN$). The following fractions and probabilities are of interest.

The *true positive fraction* (TPF) or $P(D+ \mid C+)$ of a detection system is the probability that it makes a positive decision (D+) when the condition is present (C+). It is also called *sensivity* or *recall*.

$$TPF \approx \frac{TP}{TP + FN} \tag{6.1}$$

The *true negative fraction* (TNF) or $P(D- \mid C-)$ of a detection system is the probability that it makes a negative decision (D-) when the condition is absent (C-). It is also called *specificity*.

$$TNF \approx \frac{TN}{FP + TN} \tag{6.2}$$

The *false positive fraction* (FPF) or $P(D+ \mid C-)$ of a detection system is the probability that it makes a positive decision (D+) when the condition is absent (C-).

$$FPF = 1 - TNF \tag{6.3}$$

The *false negative fraction* (FNF) or $P(D- \mid C+)$ of a detection system is the probability that it makes a negative decision (D-) when the condition is present (C+).

$$FNF = 1 - TPF \tag{6.4}$$

Notice that the TPF answers the question: if the condition is present, how likely will the detection system make a positive decision? The TNF answers the question: if the condition is absent, how likely will the detection system make a negative decision? Nevertheless, often the questions of interest are: if the decision was positive, how likely is the condition to be present? and if the decision was negative, how likely is the condition to be absent? These questions are answered using the following concepts.

The *positive predictive value* (PPV) or $P(C+ \mid D+)$ of a detection system is the probability that the condition is present (C+) given that the decision was positive (D+). It is also called *precision*.

$$PPV \approx \frac{TP}{TP + FP} \tag{6.5}$$

The *negative predictive value* (NPV) or $P(C- \mid D-)$ of a detection system is the probability that the condition is absent (C-) given that the decision was negative (D-).

$$NPV \approx \frac{TN}{FN + TN} \tag{6.6}$$

The *prevalence* $P(C+)$ of a detection system is the probability that the condition is present. See Axelsson's work [Axe00] for a discussion on base-rates.

$$prevalence \approx \frac{TP + FN}{N} \tag{6.7}$$

The *accuracy* of a detection system is the percentage of correct decisions that it makes, i.e., the percentage of true positives plus the percentage of true negatives.

$$accuracy \approx \frac{TP + TN}{N} \tag{6.8}$$

Frequently, accuracy has been used as the most basic form of measuring the performance of a detection system when misdetection costs are not contemplated. However, even so it is not a good measure of the performance of a detection system. Imagine that we evaluate a detection system using a total of 1000 alerts, 997 innocuous alerts and only three that correspond to malicious attacks. Then a simple detection system that always makes negative decisions will have 0.997 accuracy even when it has missed all the malicious alerts. Moreover, that level of accuracy will surely be hard to achieve by a more sophisticated detection system [PF01]. We will show later that the performance of a detection system should be measured in terms of not only the number of decisions that it makes correctly but also in terms of the cost of the errors that it commits.

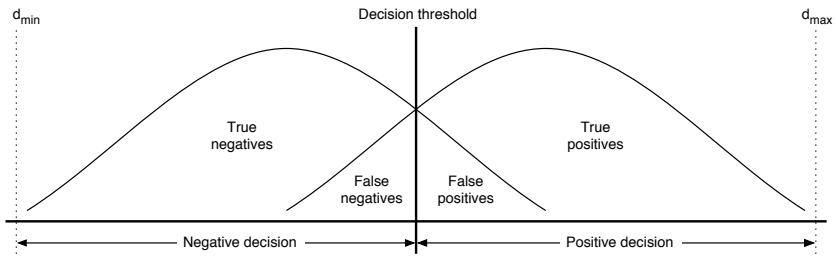


Fig. 6.1 A decision threshold establishes a balance between both types of error.

6.1.2 Parametric Detection Systems

Often, some detection systems are designed to return a value on a continuous measure —i.e., indicating the likelihood of the presence of the condition being inspected— instead of a yes or not. In these detection systems, called parametric detection systems, an arbitrary *decision threshold* (cut-off point) has to be chosen such that above the decision threshold the decision about the condition is positive and below it the

decision is negative. Figure 6.1 uses two Gaussian curves to illustrate the workings of a decision threshold. The Gaussian on the left represents the population where the condition is absent (e.g., alerts corresponding to innocuous attacks) whereas the Gaussian on the right represents the population where the condition is present (e.g., alerts corresponding to malicious attacks). The decision threshold establishes a trade-off between the two types of errors. In other words, if we move the decision threshold to the left on the abcissa axis then the number of false negatives will be decreased but the number of false positives will be increased. On the contrary, if we move the decision threshold on the opposite direction both errors will be altered conversely. So, how can we choose an optimal decision threshold?

ROC analysis allows one to select the optimal decision threshold of a parametric detection system. ROC analysis is best explained in terms of its components.

ROC Space

ROC space is a two-dimensional Cartesian space that lies in a unit square where the ordinate axis designates the true positive fraction and the abcissa axis designates the false positive fraction (see Figure 6.2). The ROC space has four singular points:

1. $(\mathbf{0}, \mathbf{0})$ This point symbolizes a detection system that never makes positive decisions and corresponds to a parametric detection system whose decision threshold has been set to the maximum value d_{max}.

2. $(\mathbf{0}, \mathbf{1})$ This point represents the perfect detection system. It always makes correct decisions and therefore the FPF is 0 and the TPF is 1.

3. $(\mathbf{1}, \mathbf{0})$ This point denotes the worst detection system. FPF is 1 and the TPF is 0 since it always makes wrong decisions.

4. $(\mathbf{1}, \mathbf{1})$ This points symbolizes a detection system that always makes positive decisions and corresponds to a parametric detection system whose decision threshold has been set to the minimum value d_{min}.

A diagonal line from the point $(0, 0)$ to the point $(1, 1)$ (the dotted line from the lower left hand corner to the upper right hand corner in Figure 6.2) represents a detection system with no discriminating power (i.e., a detection system that works no better than chance).

ROC Point

Given a decision threshold d a ROC point (operating characteristic) is a pair of values (FPF_d, TPF_d) such that FPF_d represents the false positive fraction and TPF_d represents the true positive fraction for threshold d. We say that one point in a ROC curve dominates another if its above and to the left. That is, it has a higher or equal TPF and lower or equal FPF.

$$(FPF_{d_1}, TPF_{d_1}) \gg (FPF_{d_2}, TPF_{d_2}) \text{ if } TPF_{d_1} \geq TPF_{d_2} \text{ and } FPF_{d_1} \leq FPF_{d_2} \tag{6.9}$$

ROC Curve

A ROC curve is constructed calculating the true positive fraction and false positive fraction of a detection system for each possible decision threshold from strict to lenient. Figure 6.2 depicts three hypothetical ROC Curves. Notice that the shape of the ROC curve depends on how distinguishable are the underlying distributions (see Figure 6.2 and Figure 6.3). When the decision threshold is set to its strictest value d_{max} then all decisions are negative and therefore the FPF is 0 as well as the TPF. This corresponds to the point $(0, 0)$. On the other hand, the point $(1, 1)$ corresponds to the the most lenient decision threshold d_{min} when all decisions are positive and therefore both FPF and TPF are 1. A ROC curve encapsulates all the information provided by a confusion matrix for all possible values of a decision threshold. As a matter of fact, it is said that A ROC curve makes explicit the inherent tradeoff between sensivity and specificity for a number of different decision thresholds.

Slope of the Curve

At any point along the curve, the slope of the ROC curve S measures the decision threshold used to generate that point [SDM00]. S can be used to determine the optimal decision threshold as we will see in Section 6.2.

Area Under the ROC Curve

The area under the curve (AUC) gives an estimation of the accuracy of a detection system [HM82]. The higher the AUC, the greater the accuracy. A perfect detection system will have an area of 1. A detection system with no discriminating power will have an area of 0.5. There exist two types of methods to compute the area under the ROC curve[2]: parametric and non-parametric methods [HHJ97]. Parametric methods use a maximum likelihood estimator of the AUC [MSL76] whereas non-parametric methods approximate the AUC by means of trapezoids. This last method has be shown to be equal to Mann-Whitney-Wilcoxon test statistic [HM82, HT01a, HH02]. See [HT01a] for a generalization of the area under the ROC curve to multiple class classification problems. As we will see later, these methods allows one to compare one or several detection systems in different populations[3].

[2]Notice that the area under the ROC curve is equivalent to Gini index in Lorenz diagrams [Gas72].

[3]The Department of Radiology of the University of Chigaco provides a number of computer programs to compute the area under the ROC curve. They are available at www-radiology.uchicago.edu/krl/toppage11.htm.

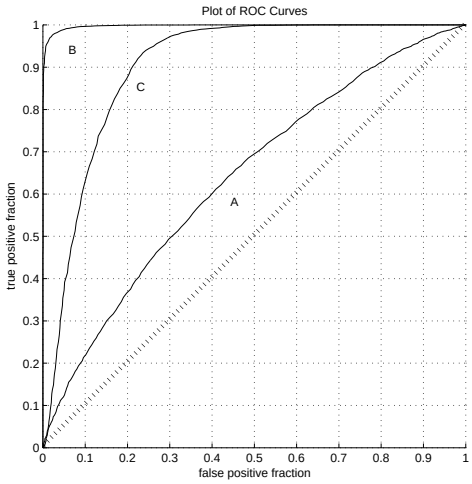


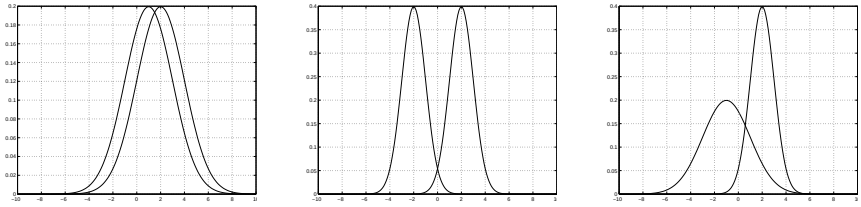
Fig. 6.2 Three illustrative Receiver Operating Characteristic curves corresponding to distributions on Figure 6.3.

Non-parametric detection systems can be represented in a ROC space by a single ROC point. Sometimes in the literature [DCW99, LFG00] it has been suggested to complete a ROC curve for non-parametric detection systems connecting that single point with straight lines to points $(0, 0)$ and $(1, 1)$. The usefulness of this procedure has caused some discussion such as Gaffney and Ulvila pointed out [GU01].

In this section we have introduced some basic concepts that we will use in our framework to evaluate alert triage in three scenarios: when the costs of misdetection are disregarded, when the costs of misdetection are known a priori, and when we cope with imprecise environments and the costs are unknown a priori and can change dynamically.

6.2 A FORMAL FRAMEWORK FOR THE EVALUATION OF ALERT TRIAGE

Alert triage is the process of rapid and approximate priorization for subsequent action of an IDS alert stream [MP01]. Alert triage is a classification task that takes as input the alerts generated by a number of (distributed) intrusion detection sensors (e.g., Snort [Roe99]) and classifies them producing a tag indicating its malignancy (degree of threat) for each alert. Such tags priorize alerts according to their malignancy. Depending on its priorization each alert will require an automatic response, notification,



(a) ROC curve A (b) ROC curve B (c) ROC curve C

Fig. 6.3 Three illustrative pairs of distribution to generate ROC curves in Fig 6.2. (a) Non-distinguishable distributions: N(2,2), N(1,2). (b) Distinguishable distributions: N(2,1), N(-2,1). (c) N(2,1), N(-1, 2)

or manual response from the site security officer (SSO). The tag assigned to alerts can be a (fuzzy) label (e.g. `malicious`, `innocuous`, `falsepositive`, etc) or it can be a continuous value. Alert triage can be explained in terms of the following priority formulation [PFV02] where *mission* is defined in terms of the critical and sensitive assets of the target network.

Definition 45 (Priority Formulation) *Given an alert model $\mathbb{A} = \langle \vartheta, \Sigma \rangle$ and let an alert stream $\vec{S} = [\psi_1, \psi_2, \ldots, \psi_{t_n}]$ such that $\psi_i \in \mathbb{A}$ find:*

$$\mathbf{HighImpact} = \{\psi_\alpha, \psi_\beta, \ldots, \psi_\omega\} \subseteq \vec{S}$$
$$\forall_{\psi_i \in \mathbf{HighImpact}} \mathbf{ThreatRank}(\psi_i, \mathbf{Mission}) > T_{\textit{acceptable}} \quad (6.10)$$

For the sake of brevity, we suppose that once the alerts have been priorized there will be only two possible responses $\{\mathbf{notification}, \neg\mathbf{notification}\}$. In addition, looking at Equation 6.10 it can be said that the decision threshold $T_{\textbf{acceptable}}$ will discern between two (apparently) disjoint classes of alerts (i.e those alerts that require notification to the SSO and those alerts that do not). We, therefore, without loosing generality, proceed in our evaluation as if alert triage was a detection task[4]. We consider the evaluation of alert triage as a tournament process where a ranking is established among competing systems. We proceed as follows:

1. Participants are provided with a detailed model of the target networks and their missions using a common ontology [MP03e].
2. N consecutive alerts is selected from the alert stream (e.g., extracted from a database) and sent to the participants.

[4]Notice that as we mentioned above and we will see below the mathematical methods used in this work (AUC, ROC Convex Hull) can be extended to multiple class classification [HT01a, Sri99].

3. Participants emit their judgement on each alert.

4. The outcomes of each participant are represented by means of a ROC point (non-parametric systems) or ROC curve (parametric systems). The decision threshold (operating point) of parametric systems will be varied in the evaluations so that all possible values of the decision threshold can be evaluated.

5. The system or group of systems that outperforms the rest of participants is selected as the winner.

The performance of the participants will depend on the environment where the evaluation is carried out. We contemplate three possible environments (scenarios) in increased order of uncertainty and therefor of complexity. It is worth to notice here that the first environment is a particular case of the second environment that in turn is subsumed by the third environment.

6.2.1 Ideal Alert Triage Environment

In this environment, we do not contemplate misdetection or misclassification costs at all, neither a utility function over the decisions that are made correctly. This scenario is valid at design time when we want to check new triage techniques and for example fix some kind of requirement such as the maximum number of false positives allowable [Axe00]. The goal of alert triage in this scenario is either to maximize the overall percentage of correct decisions or minimize the overall percentage of incorrect decisions. The performance of parametric systems can be computed using the area under the ROC curve as we saw in Section 6.1 whereas the performance of non-parametric systems can be measured using accuracy (Equation 6.8) but as we indicated in Section 6.1 accuracy is not always a good measure, particularly, if true negatives abound [KHM97]. We will see a clear example later on in our experimental results. Fortunately, there exist other performance measures in the literature such as:

e-distance E-distance is the Euclidean distance from the perfect classifier (point $(0, 1)$) and the ROC point of interest [HGF02].

$$\text{e-distance} = 1 - \sqrt{W \cdot (1 - TPF)^2 + (1 - W) \cdot FPF^2} \tag{6.11}$$

Where W is a parameter that ranges between 0 and 1 and establishes the relative importance between false positives and false negatives.

f-measure F-measure is a combination of recall (TPF) and precision (PPV) [LG94].

$$\text{f-measure} = \frac{(\beta^2 + 1) \cdot TPF \cdot PPV}{\beta^2 \cdot PPV + TPF} \tag{6.12}$$

Where β is a parameter ranging from 0 to infinity that weights recall and precision.

Table 6.2 Confusion matrices for alert triage systems a, b, and c.

a		b		c	
TP	**FP**	**TP**	**FP**	**TP**	**FP**
133	53	156	87	101	116
FN	**TN**	**FN**	**TN**	**FN**	**TN**
12	802	3	754	5	778
TPF	**FPF**	**TPF**	**FPF**	**TPF**	**FPF**
0.9712	0.0620	0.9811	0.1034	0.9528	0.1298

Table 6.3 Accuracy measure results for alert triage systems a, b, and c.

	a	b	c
accuracy	0.9350	0.9100	0.8790
$\mathbf{e-distance}$	0.9269	0.9256	0.9024
$\mathbf{f-measure}$	0.8036	0.7761	0.6254
$\mathbf{g-mean}$	0.9276	0.9379	0.9106
$\mathbf{t-area}$	0.9276	0.9388	0.9115

g-mean Geometric mean is high when both TPF and TNF are high and when the difference between both is small [KHM97].

$$\text{g-mean} = \sqrt{TPF \times TNF} \tag{6.13}$$

However, both e-distance and f-measure performance measures require to explicit an arbitrary parameter to conveniently weight false positives and false negatives whereas g-mean has not a clear interpretation. Thus, we have developed a new measure, called *t-area*, for this purpose.

6.2.1.1 T-area

As we saw above, ROC AUC seems to be an intuitive and robust method to compute the performance of parametric systems. However, computing (or approximating) the area for an arbitrary ROC requires certain skills. For non-parametric systems, it is easier since AUC coincides with the area of a quadrilateral formed by the ROC point of interest (FPF, TPF), the worst detection system (1,0), the detection system that never makes positive decisions (0,0), and the detection system that always make positive decisions (1,1). Based on the observation that the diagonal (that represents detection

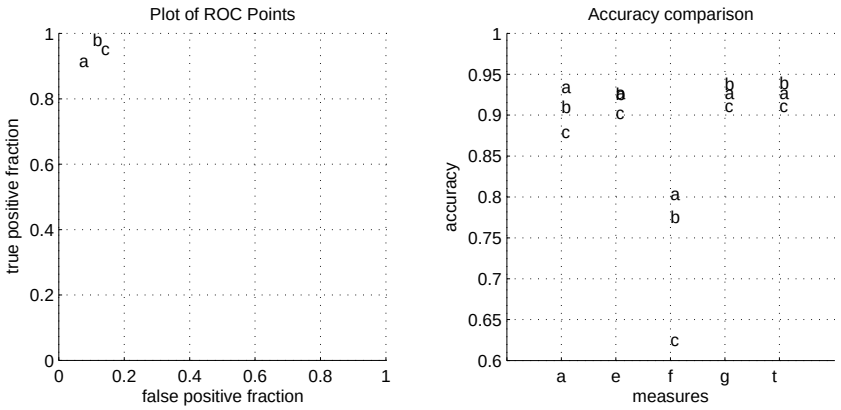


Fig. 6.4 ROC points and ranking corresponding to three alert triage systems.

systems with no discriminating power) and the point of interest always conform a triangle (except when the point of interest lies in the diagonal), we have defined a new performance measure called t-area as follows:

Definition 46 (T-Area) *T-area is the area of the quadrilateral formed by the segments connecting the ROC point of interest and all the singular points of the ROC space except the perfect detection system. We compute T-area using Heron's formula:*

$$\textit{t-area} = \begin{cases} 1/2 + \sqrt{s \cdot (s-a) \cdot (s-b) \cdot (s-c)} & \textit{if } TPF > FPF \\ 1/2 & \textit{if } TPF = FPF \\ 1/2 - \sqrt{s \cdot (s-a) \cdot (s-b) \cdot (s-c)} & \textit{if } TPF < FPF \end{cases} \quad (6.14)$$

Where $s = \frac{a+b+c}{2}$ *i.e., half the perimeter,* $a = \sqrt{2}$, $b = \sqrt{TPF^2 + FPF^2}$, *and* $c = \sqrt{(1-TPF)^2 + (1-FPF)^2}$.

Notice that according to t-area all detection systems with no discriminating power will have always an area of 0.5. T-area does not require arbitrary parameters and has an intuitive interpretation (AUC). An additional advantage of this measure is that it makes parametric and non-parametric systems comparable. Next, we will see how the aforementioned performance measures including accuracy and t-area behave using a simple example.

Table 6.2 shows the confusion matrices of three alert triage systems a, b, and c that have taken part in an evaluation using a stream of 1000 alerts. The corresponding ROC points $(FPF_a = 0.0620, TPF_a = 0.9712)$, $(FPF_b = 0.1034, TPF_b = 0.9811)$,

$(FPF_c = 0.1298, TPF_c = 0.9528)$ are depicted in Figure 6.4. We have used all the above performance measures to rank them. The results are shown in Table 6.3 and Figure 6.4. Clearly, the ROC points of a and b dominate the ROC point of c (according to Equation 6.9). All measures discern between a & b and c. However, there is no consensus among the different accuracy measures to signal a or b as a winner. We advocate for the use of t-area given that it has an intuitive explanation (ROC AUC), serves to compare parametric and non parametric systems, and does not depend on parameters that establish an artificial weight between the errors.

In intrusion detection misdetection costs are asymmetric. That is, the cost of notifying a SSO when an alert corresponds to an innocuous attack (or false positive) is really lower compared with the cost of not adverting the presence of an intruder. Thus, once we have designed an alert triage system we should examine that it will contemplate such difference. Next scenarios allow one to evaluate an alert triage system in a cost sensitive way.

6.2.2 Cost-based Alert Triage Evaluation

We consider now scenarios where correct decision outcomes have associated a benefit B and incorrect decision outcomes have associated a cost C. $B(A \mid A)$ represents the benefit obtained for correctly classifying an alert of type A and $C(A \mid B)$ is the cost incurred if an alert of type B was misclassified as being an alert of class A. These scenarios are valid to test alert triage systems in simulated environments and to determine their optimal decision threshold. In this case, an alert triage detection system outperforms another if it has a lower expected cost. Thus, the goal here is to maximize the expected value of each decision. The expected cost of a non-parametric detection system or a parametric detection system operating at a given decision threshold is given by:

$$\begin{array}{l} \textbf{Expected Cost} = \\ P(C+) \cdot P(D+ \mid C+) \cdot B(D+ \mid C+) \quad + \\ (1 - P(C+)) \cdot P(D- \mid C-) \cdot B(D- \mid C-) \quad + \\ P(C+) \cdot P(D- \mid C+) \cdot C(D- \mid C+) \quad + \\ (1 - P(C+)) \cdot P(D+ \mid C-) \cdot C(D+ \mid C-) \end{array} \tag{6.15}$$

Two consequences can be inferred directly from 6.15:

1. Two detection system will have the same expected value if:

$$\frac{TPF_2 - TPF_1}{FPF_2 - FPF_1} = \frac{1 - P(C+)}{P(C+)} \times \frac{B(D- \mid C-) + C(D+ \mid C-)}{B(D+ \mid C+) + C(D- \mid C+)} \tag{6.16}$$

   Equation 6.16 defines the slope of an *iso-performance* line [PF01].

2. The slope S that corresponds to the optimal decision threshold can be computed as follows:

$$S_{\textbf{optimal}} = \frac{1 - P(C+)}{P(C+)} \times \frac{B(D- \mid C-) + C(D+ \mid C-)}{B(D+ \mid C+) + C(D- \mid C+)} \tag{6.17}$$

In Chapter 2, we described an alternative representation. Figure 2.6 showed the explicit representation of costs using the method introduced in Section 2.8.2 for the alert triage systems $a, b,$ and, c analyzed in Section 6.2.1. Notice that the alert triage system b will be better under certain cost conditions than system a and viceversa. Thus, if the conditions are known a priori we can clearly choose a or b.

Once an intrusion detection system has been developed, its operational costs will depend on the the importance of the target's mission (system under surveillance) and the nature of the possible future attacks (e.g., to be attacked by a CodeRed propagation is quite different from being wounded by a malefactor or experiencing a DoS attack) and the level of hostility [GU01]. The ultimate objective of intrusion detection is to develop robust systems able to face imprecise environments, thus we also have to address this type of scenarios.

6.2.3 Alert Triage Evaluation in Imprecise Environments

In these scenarios, misdetection costs not only will be unknown a priori but also will vary over time. These scenarios are useful for the evaluation of systems for real-world deployment. To decide whether an alert triage systems outperforms others in this type of environment we will use a robust and incremental method for the comparison of multiple detection systems in imprecise and dynamic environments that has been proposed in [PF01]. This method, named ROCCH (ROC Convex Hull) is a combination of ROC analysis, computational geometry, and decision analysis. Thus, once the ROC points or ROC curves have been computed for the different participants we proceed as follows to select the best alert triage systems:

1. Firstly, we compute the convex hull[5] of the set of ROC points that symbolizes a composite alert triage system.
2. Secondly, we compute the set of iso-performance lines corresponding to all possible cost distributions.
3. Finally, for each iso-performance line we select the point of the ROC convex hull with the largest TPF that intersects it.

Figure 6.5 depicts the ROC points and curves for the alert triage systems analyzed above. To select which systems will be of interest for a collection of unknown conditions we compute the convex hull such as it is shown in Figure 6.6. In that figure, the circled points denote the points that forms part of the convex hull and therefore can be optimal for a number of conditions: they form part of the composite alert triage system chosen as a winner. Therefore, we can continue our analysis

[5]The convex hull of a set of points is the smallest convex set that includes the points.

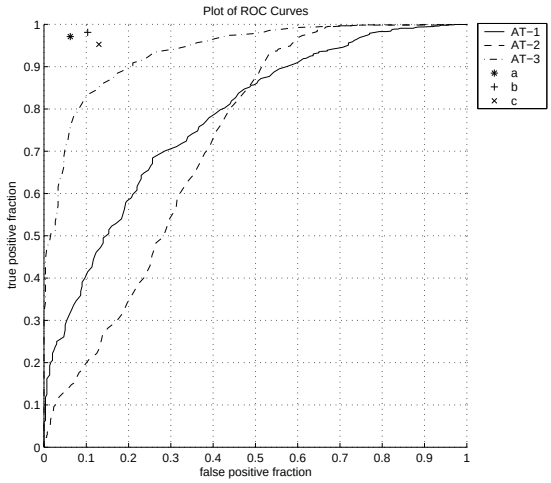


Fig. 6.5 3 ROC points and 3 ROC curves representing 6 alert triage systems.

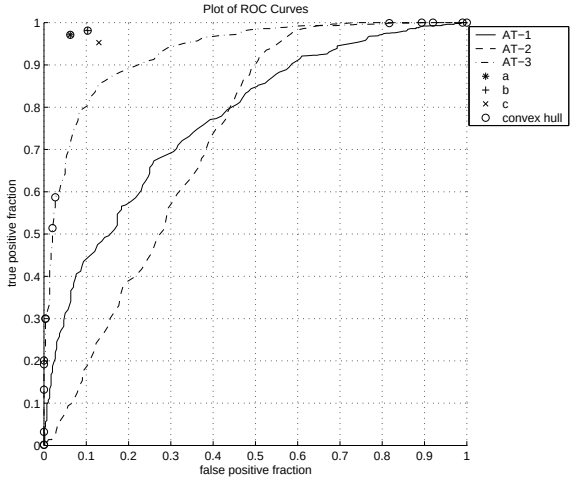


Fig. 6.6 ROC Convex Hull of ROC points and curves of Figure 6.5.

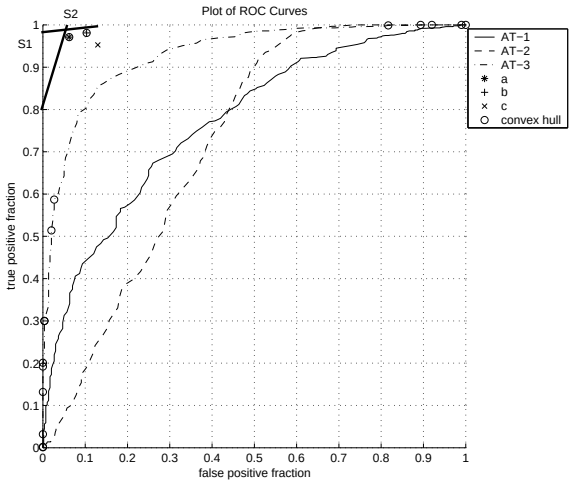


Fig. 6.7 Iso-performance lines for different sets of conditions.

without the non-parametric system c and the parametric systems 2 and 3. Finally, Figure 6.7 shows two illustrative iso-performance lines corresponding to the slopes S_1 and S_2. Thus, for each of the different sets of conditions that determine the values of slopes S_1 and S_2 the alert triage systems a and b will be the optimal respectively.

Next Section describes experiments we did to analyze the performance of our techniques using the above framework and three data sets created using real-world alerts.

6.3 EXPERIMENTAL RESULTS

The exposition of this Section proceeds as follows. Subsection 6.3.1 succinctly describes the three data-sets compiled and their main characteristics. Subsection 6.3.2 presents the experimental setting that we have used to emulate vulnerabilities and provide feedback on new discovered cases. Subsection 6.3.3 introduces a basic algorithm that we have defined to establish a baseline to compare Ceaseless CBR performance with. Subsection 6.3.4 enumerates diverse evaluation criteria that we used to measure the overall performance of our approach. Subsection 6.3.5 defines and enumerates the experiments that we carried out. Subsection 6.3.6 provides Ceaseless CBR results when all the alerts in the current window are explained right away whereas 6.3.7 takes into account the measure of urgency described in Chapter 5. Finally, Subsection 6.3.8 draws some conclusions on the most relevant aspects of the achieved performance.

6.3.1 Data-sets

For a case study of workday IDS load we set up three IDSes into three different scenarios to capture real-world alerts. We collected three alert data-sets in real environments along more than 15 months (in total) of continuous surveillance[6]. We called them respectively Rustoord, Naxpot, and Huckleberry for the place where they were compiled. These environments tried to simulate a realistic situation where a network administrator sets up an IDS and what he or she obtains in return is an unmanageable stream of alerts. A feature that characterizes real world data is that since the conditions are not controlled, there are all sorts of contaminating effects (such misconfigurations, downtimes, etc). Next we briefly describe the alert databases. See Appendix A for further details.

Rustoord is composed of 31483 alerts covering a period of almost 16 weeks. There are 90 unique alerts (i.e., number of alerts with different sort). The most frequent alert was WEB-IIS cmd.exe access that occurred 12050 times, what supposed a 38% of all occurrences. We evaluated the dynamics of this alert in Chapter 4. Alerts usually occur in burst. There are even days with more than 3000 alerts. On the average there are 11 alert every hour. The weekly alert load is on average 1967 alerts per week.

Naxpot is composed of 204977 alerts corresponding to almost 35 weeks of continuous observation. In Naxpot data-set, WEB-MISC http directory traversal, MS-SQL Worm propagation attempt, and SCAN SOCKS Proxy attempt were the most frequent alerts with a 47%, 12%, and 12% of all of the occurrences respectively. On the average there are more than 35 alerts every hour. A number of days the alert load is even over 2000 alerts a day what implies almost 90 alerts per hour. In Naxpot we observed 331 unique alerts. The weekly alert load is on average 5856 alerts per week.

Huckleberry is composed of 219886 alerts occurring in around 16 weeks. On average we observed 82 alerts per hour. There are three days where the number alerts is above 25000 alerts in part due to a misconfigured DSL router that caused a total of 166534 MISC UPNP malformed advertisement alerts. That is, the 75.8% of all alert load. In Huckleberry data-set, 203 unique alerts were identified. The weekly alert load is on average 13742 alerts per week.

We observed in our data-sets that alert load is dominated by alerts corresponding to automated attacks (worms, script kiddies[7], etc) and the presence of truly sophisticated multi-stage attacks is rare. This observation is in good agreement with other studies [OMS03]. Another source of alerts corresponded to miscofigured network elements.

[6]We use two simple honeypots in two different home networks and a IDS running on a real software company.

[7]According to The Jargon Dictionary: "The lowest form of *cracker*; script kiddies do mischief with scripts and programs written by others, often without understanding the *exploit*".

For example, in Huckleberry data-set a cable modem that was incorrectly configured caused multiples bursts of alerts. This kind of errors increased significantly the alert rate.

The distribution of alert sorts is highly skewed. The bottom half of alerts produce $\lesssim 1\%$ of the overall alert load while a small number (5-10) of top alerts is responsible for 80% or more of the total load. That is, a few alerts occur very often while many others occur rarely. These observations allow one to induce that alert load (alert popularity) follows a Zipfian distribution[8]. That is to say, the frequency of occurrence of alerts (P) as function of their rank (r), determined by their frequency of occurrence, is a power-law function $P_r \simeq \frac{1}{r^z}$ where z is close to unity[9]. Figure A.11 shows the plots that we used to check whether our data-sets are driven by Zipfian distributions. It is remarkable that the graph corresponding to Naxpot data-set is the closest to a straight line when plotted using logarithmic axes. Notice that precisely this data-set was compiled over a longer period of time than the others two (i.e., 36 weeks vs 16 weeks). We have used a scatter plot to see how multiple occurrences of alerts spread over time. As shown in Figures A.7, A.8, and A.9, the prevalance of some alerts creates a band of points that follow a horizontal pattern. For example, the top band in Figures A.8 and A.9 corresponds to MS-SQL Worm propagation attempt alerts whose Snort signature identifier (sid) is 2003. A band of data points that follow a vertical pattern indicates the presence of multiple attacks the same day surely caused by automated tools or network scanners such Nessus or Satan. Figure A.10 shows a scatter plot for all three data-sets where we can see the presence of certain alerts extends over data-sets whereas other alerts appear uniquely in one of them.

Each alert provided the following information: id, time, triggered-signature, sensor, ip-source-addr, ip-dest-addr, ip-ver, ip-hdr-len, ip-tos, ip-length, ip-id, ip-flags, ip-offset, ip-ttl, ip-chksum, fqdn-source-name, fqdn-dest-name, protocol, and payload. See Figure 3.10 for a specific example of an alert. In Chapter 7 we will see further details on how this information is abstracted prior to be processed by Ceaseless CBR.

6.3.2 Experimental Setting

Assessing the danger of the events that caused each alert requires to know the specific network environment where the alert was triggered (remember the relative-environment dependency problem that we pointed out in Section 1.2.2). In order to use the same alert load with different network models to which they were captured our experimental framework allows us to simulate which network elements are vulnerable or exposed. A parameter called *overall exposure* determines the simulated percentage of vulnerabilities on the network under supervision. We have tested our techniques with different levels of exposure obtaining the same level of performance. All the

[8] linkage.rockefeller.edu/wli/zipf/

[9] A number of recent studies have shown that popularity in different Internet domains tends to be driven by Zipfian (aka Rank-Size) distributions [BCF99].

data-set	p	tpf	fpf	recall	precision	prevalence
Rustoord	0	1.0000	1.0000	1.0000	0.1046	0.1046
Rustoord	1	0.9997	0.3326	0.9997	0.2598	0.1046
Rustoord	2	0.0000	0.1986	0.0000	0.0000	0.1046
Rustoord	3	0.0000	0.1123	0.0000	0.0000	0.1046
Rustoord	4	0.0000	0.1123	0.0000	0.0000	0.1046
Rustoord	5	0.0000	0.0000	0.0000	N/A	0.1046
Naxpot	0	1.0000	1.0000	1.0000	0.1995	0.1995
Naxpot	1	0.9929	0.9982	0.9929	0.1987	0.1995
Naxpot	2	0.0464	0.1589	0.0464	0.0679	0.1995
Naxpot	3	0.0297	0.0174	0.0297	0.2987	0.1995
Naxpot	4	0.0297	0.0174	0.0297	0.2987	0.1995
Naxpot	5	0.0000	0.0000	0.0000	N/A	0.1995
Huckleberry	0	1.0000	1.0000	1.0000	0.0899	0.0899
Huckleberry	1	0.9945	0.9890	0.9945	0.0903	0.0899
Huckleberry	2	0.0192	0.1467	0.0192	0.0127	0.0899
Huckleberry	3	0.0002	0.1465	0.0002	0.0001	0.0899
Huckleberry	4	0.0002	0.1465	0.0002	0.0001	0.0899
Huckleberry	5	0.0000	0.0000	0.0000	N/A	0.0899

Table 6.4 Performance results for a simple baseline algorithm. **data-set**= data-set name; **wm** = window model; **ws** = window size; **p** = priority threshold; **tpf** = true positive fraction; **fpf** = false positive fraction.

experiment described below were carried out using a simulated overall exposure of 0.2. That is, approximately 2 out of 10 alerts are dangerous and imply a risk for the system under supervision. We used a function called feedback that mimics a SSO providing feedback on the risk entailed by new discovered cases. This allowed us to run the experiments without human intervention at all. In a real world deployment of our approach, as we noticed in Chapter 5, the underlying network model and the actual vulnerabilities can also be automatically compiled using network scanners such as Nessus reducing significantly human intervention[10]. The alert loss ratio L for each and every alert was set up at 0.001 (see Section 5.1.1.1). That is, we estimated the probability of our IDS sensors missing an alert at about 1 in 1000.

6.3.3 Baseline

We used the default priority assigned by Snort to create a simple baseline algorithm with which to compare Ceaseless CBR performance. This simple baseline algorithm uses a threshold ranging from 0 to 5 to prioritize alerts and notify them accordingly. For example, when it is set up to 3, only alerts with a Snort priority higher than 3 are considered dangerous and notified to the SSO. Given a specific exposure level we simulated a number of vulnerabilities for each of the scenarios where we captured our alerts. Then we used the original alert load of each scenario and run our baseline algorithm obtaining the results described in Tables 6.4 and 6.5. Let us now show with these tables a clear example our previous arguments against the accuracy. Notice first the low prevalence 0.1046, 0.1995, and 0.0899 in all the data-sets (i.e., false negatives

[10]Without these tools the ability to analyze the level of risk entailed by each alert remains one of the most pressing challenges to SSOs.

data-set	priority	accuracy	e-distance	f-measure	g-mean	t-area
Rustoord	0	0.1046	0.2929	0.1893	0.0000	0.5000
Rustoord	1	0.7022	0.7648	0.4124	0.8168	0.8336
Rustoord	2	0.7176	0.2791	N/A	0.0000	0.4007
Rustoord	3	0.7948	0.2884	N/A	0.0000	0.4438
Rustoord	4	0.7948	0.2884	N/A	0.0000	0.4438
Rustoord	5	0.8954	0.2929	N/A	0.0000	0.5000
Naxpot	0	0.1996	0.2929	0.3327	0.0060	0.5000
Naxpot	1	0.1996	0.2942	0.3311	0.0428	0.4974
Naxpot	2	0.6826	0.3164	0.0551	0.1976	0.4438
Naxpot	3	0.7925	0.3138	0.0540	0.1707	0.5062
Naxpot	4	0.7925	0.3138	0.0540	0.1707	0.5062
Naxpot	5	0.8005	0.2929	N/A	0.0000	0.5000
Huckleberry	0	0.0899	0.2929	0.1649	0.0000	0.5000
Huckleberry	1	0.0994	0.3007	0.1656	0.1047	0.5028
Huckleberry	2	0.7783	0.2987	0.0153	0.1279	0.4362
Huckleberry	3	0.7768	0.2854	0.0001	0.0114	0.4268
Huckleberry	4	0.7768	0.2854	0.0001	0.0114	0.4268
Huckleberry	5	0.9101	0.2929	N/A	0.0000	0.5000

Table 6.5 Performance measures for a simple baseline algorithm. **data-set**= data-set name; **wm** = window model; **ws** = window size; θ = evidence threshold;

abound). As it can be seen in Table 6.5, when the priority is set to 5 the accuracy of our baseline algorithm in the three data-sets is respectively 0.8954, 0.8005, and 0.9101. This could be considered as good performance in spite of obtaining 0 true positives and 0 true negatives. Notice, however, that t-area is 0.5 for all of them since that algorithm has no discriminating power at all. Clearly, that the best performers in Naxpot and Huckleberry hardly behaved better than chance. The highest t-areas in both data-sets are 0.5602 and 0.5028 respectively. According to t-area the best perfomance in Rustoord data-set as well as in Huckleberry was achieved when the priority threshold was set up to 1. In Naxpot data-set it was achieved when either the priority threshold was set up to 4 or 5. Concluding, this algorithm did unsurprisingly poorly as shown in both Tables. Unfortunately, this approach coincides with the way a high number of real world IDSes operate. Notice that our simple baseline algorithm was only intended to provide a standard comparison of Ceaseless CBR with those IDSes.

6.3.4 Evaluation

We evaluated the overall goodness of our approach analyzing five groups of performance characteristics:

1. First of all, we evaluated the performance of our techniques using the framework introduced throughout this Chapter. For each trial, we provide its true positive fraction (**tpf**), false positive fraction (**fpf**), **recall**, **precision**, **prevalence**, **accuracy** as well as the value given by other performance measures such as **e-distance**, **f-measure**, **g-mean**, and **t-area**. For a number of trials we also provide their ROC curves that we constructed varying the minimum evidence threshold (θ) along several values.

2. Secondly, we evaluated the capability of discovering new sequential cases in terms of the number of cases found, the number of alerts in each case, the number of different episodes of each case, and the number of occurrences of each case. An important parameter to consider here is the number of questions formulated. That is, either the number of human interventions required to properly assess the priority of new sequential cases or the number of queries sent to the model of the network under surveillance created by network scanners such as Nessus.

3. Thirdly, we also measured the CPU time required by each Ceaseless process. Notice that we have grouped together the time taken by Ceaseless Revise and Ceaseless Retain for two reasons: (i) sometimes the time taken by each individual process is too small to be meaningful; and (ii) both processes run interlaced what makes difficult to separate their elapsed execution time.

4. Fourthly, we measured for each trial its *predictiveness* or *preemptive ratio*. That is, the capability of predicting future alerts. The preemptive ratio is defined as the number of alerts whose occurrence was predicted in advance and really occurred afterwards over the the number of alerts whose occurrence was predicted but never occurred. We considered correct predictions those that took place between the following window and a maximum period of time that we fixed at 86400 seconds for all experiments.

5. Finally, we measured the alert load reduction that our techniques were able to achieve in the weekly alert load of each data-set.

6.3.5 Experiments

We measured the performance of our techniques with varying window models, window sizes, and minimum evidence thresholds trying to determine the influence caused in the aforementioned distinct performance characteristics. As a matter of fact, each individual trial is identified by the following four parameters: data-set name (**data-set**), window model (**wm**), window size (**ws**), and minimum evidence threshold (θ).

We have carried out two blocks of experiments:

CCBR In this first block we did not consider urgency at all. That is, Ceaseless CBR was configured to explain all alerts in the current window right away. We denoted by CCBR this first block of experiments.

UCCBR In the second block, denoted by UCCBR, we analyzed the influence of urgency. That is, at each iteration Ceaseless Reuse decides on which alert to prioritize and which alert should wait to be prioritized in further iterations.

This separation allowed us in addition to measure how the fact of prioritizing right now versus to do it only when is urgent affects Ceaseless CBR performance. In all experiments reported below, the sequential case base is constructed from scratch. That is, we initiated each trial with no sequential cases. Remember that one of

data-set	wm	ws	θ	tpf	fpf	recall	precision	prevalence
Rustoord	0	100	0.1	1.0000	0.1452	1.0000	0.4458	0.1046
Rustoord	0	100	0.3	1.0000	0.1448	1.0000	0.4464	0.1046
Rustoord	0	100	0.5	0.9994	0.1442	0.9994	0.4473	0.1046
Rustoord	0	100	0.7	1.0000	0.1445	1.0000	0.4470	0.1046
Rustoord	0	100	0.9	1.0000	0.1443	1.0000	0.4473	0.1046
Rustoord	0	100	1	1.0000	0.1483	1.0000	0.4405	0.1046
Rustoord	0	50	0.5	0.9994	0.1355	0.9994	0.4627	0.1046
Rustoord	0	25	0.5	0.9982	0.1215	0.9982	0.4896	0.1046
Rustoord	0	10	0.5	0.9994	0.0945	0.9994	0.5525	0.1046
Rustoord	0	5	0.5	1.0000	0.0870	1.0000	0.5729	0.1046
Rustoord	0	1	0.5	1.0000	0.0000	1.0000	1.0000	0.1046
Rustoord	1	3600	0.5	1.0000	0.1430	1.0000	0.4496	0.1046
Rustoord	1	86400	0.5	1.0000	0.1486	1.0000	0.4400	0.1046
Naxpot	0	10	0.5	0.9967	0.0408	0.9967	0.8588	0.1995
Naxpot	1	3600	0.5	0.9968	0.0597	0.9968	0.8063	0.1995
Huckleberry	0	10	0.1	1.0000	0.0065	1.0000	0.9387	0.0899
Huckleberry	0	10	0.3	1.0000	0.0056	1.0000	0.9463	0.0899
Huckleberry	0	10	0.5	1.0000	0.0060	1.0000	0.9424	0.0899
Huckleberry	0	10	0.7	1.0000	0.0058	1.0000	0.9448	0.0899
Huckleberry	0	10	0.9	1.0000	0.0055	1.0000	0.9470	0.0899
Huckleberry	0	10	1	1.0000	0.0091	1.0000	0.9155	0.0899
Huckleberry	1	3600	0.5	0.9996	0.0371	0.9996	0.7269	0.0899

Table 6.6 Experimental results for CCBR. **data-set**= data-set name; **wm** = window model; **ws** = window size; θ = evidence threshold; **tpf** = true positive fraction; **fpf** = false positive fraction.

the dimensions through we evaluated Ceaseless CBR is its ability to discover new sequential cases. A logical next step, which we have left for future work, is to measure the impact on performance when the sequential case base is initialized with a number of previously hand-crafted sequential cases. Another future step is to share sequential cases among different data-sets. Said differently, what would be the impact of using sequential cases learnt through the analysis of a data-set within the analysis of other data-set? This would be part of our preliminary work to extend our approach to a multi-agent setting such as we will see in the next Chapter. The set of constraints employed to discover new sequential cases were {ip-source-addr,ip-dest-addr} in all experiments below.

6.3.6 CCBR

We initially carried out a first set of experiments where all alerts in the current window are explained (i.e., in each iteration the number of alerts to explain is the number of alerts in the current window). To analyze the influence of the minimum evidence threshold (θ) used by Ceaseless Retrieve we fixed the window model and size. We run Ceaseless CBR in Rustoord and Huckleberry data-sets fixing a space-based window of size 100 and 10 respectively and varying θ from 0.1 to 1. Likewise, to analyze the influence of the window size we run Ceaseless CBR in Rustoord data-set fixing θ to 0.5 and varying the window size from 100 to 1. Additionally, we carried out other trials with time-based windows and distinct sizes.

data-set	wm	ws	θ	accuracy	e-distance	f-measure	g-mean	t-area
Rustoord	0	100	0.1	0.8700	0.8973	0.6167	0.9246	0.9274
Rustoord	0	100	0.3	0.8703	0.8976	0.6172	0.9248	0.9276
Rustoord	0	100	0.5	0.8708	0.8980	0.6180	0.9248	0.9276
Rustoord	0	100	0.7	0.8706	0.8978	0.6178	0.9249	0.9278
Rustoord	0	100	0.9	0.8708	0.8980	0.6182	0.9251	0.9279
Rustoord	0	100	1	0.8672	0.8951	0.6116	0.9229	0.9258
Rustoord	0	50	0.5	0.8786	0.9042	0.6325	0.9295	0.9319
Rustoord	0	25	0.5	0.8910	0.9141	0.6570	0.9364	0.9383
Rustoord	0	10	0.5	0.9153	0.9332	0.7116	0.9513	0.9524
Rustoord	0	5	0.5	0.9221	0.9384	0.7285	0.9555	0.9565
Rustoord	0	1	0.5	1.0000	1.0000	1.0000	1.0000	1.0000
Rustoord	1	3600	0.5	0.8720	0.8989	0.6203	0.9258	0.9285
Rustoord	1	86400	0.5	0.8669	0.8949	0.6111	0.9227	0.9257
Naxpot	0	10	0.5	0.9667	0.9710	0.9227	0.9778	0.9780
Naxpot	1	3600	0.5	0.9516	0.9577	0.8915	0.9682	0.9686
Huckleberry	0	10	0.1	0.9941	0.9954	0.9684	0.9968	0.9968
Huckleberry	0	10	0.3	0.9949	0.9960	0.9724	0.9972	0.9972
Huckleberry	0	10	0.5	0.9945	0.9957	0.9703	0.9970	0.9970
Huckleberry	0	10	0.7	0.9947	0.9959	0.9716	0.9971	0.9971
Huckleberry	0	10	0.9	0.9950	0.9961	0.9728	0.9972	0.9972
Huckleberry	0	10	1	0.9917	0.9936	0.9559	0.9954	0.9954
Huckleberry	1	3600	0.5	0.9662	0.9738	0.8417	0.9811	0.9813

Table 6.7 Performance measures for CCBR. **data-set**= data-set name; **wm** = window model; **ws** = window size; θ = evidence threshold

Performance Analysis

Tables 6.6 and 6.7 summarize the performance results of the first block of experiments. From these data we can see, for example, that in Rustoord data-set when Ceaseless CBR is configured to use a time-based window model of size 86400 seconds and $\theta = 0.5$ has an improvement of 18.98% in precision, for an overall improvement in accuracy of 16.47% w.r.t. our simple baseline algorithm using priority = 1. Said differently, even the worst Ceaseless CBR configuration (according to t-area) outperforms the best configuration (according to t-area) of our baseline algorithm (see Tables 6.4 and 6.5): a t-area of 0.9257 and 0.8336 respectively. The improvement in the other two data-sets is much more superior.

Let us forget the trial that used an alert-driven model (**wm**=0 and ws=1) by now. We will talk about it later on in this Section. Therefore, without considering the alert-driven model and according to t-area, the best performer in the Rustoord data-set was the configuration using a space-based window of size 5 and $\theta = 0.5$ with a t-area of 0.9565. The rest of performance measures accuracy, e-distance, f-measure, and g-mean also signaled that configuration as the best. Thus, in an ideal alert triage environment we can conclude that this configuration was the winner (outperformed the rest).

Let us now consider what happens when we confront an imprecise environment where misdetection costs are unknown a priori. Figure 6.8 shows the ROC curves and ROC points for the several alert triage configurations that we tested in Rustoord data-set. We can easily distinguish the higher ROC AUC of the curve corresponding to the configuration using a space-based window of size 100 (green solid line) compared against the ROC curve of our baseline algorithm (blue dashed line). We proceeded

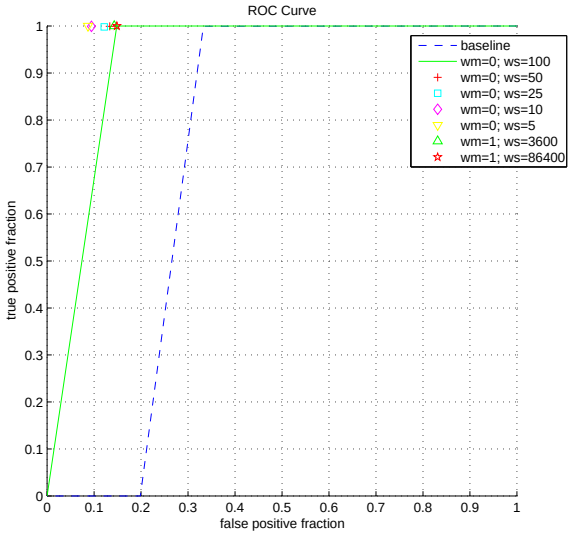


Fig. 6.8 ROC curves and points for several Ceaseless CBR configurations in Rustoord data-set.

as explained in Section 6.2.3 and computed the convex hull of the set of ROC curves and points symbolizing a composite alert triage system. Figure 6.9 depicts the points that conform the Convex Hull (using black circles). Aside the points (0,0) and (1,1) the point (0.087, 1.0000) is the only one in the convex hull. Therefore, among all configurations tested, this configuration is optimal under all possible cost conditions in Rustoord data-set. Notice that this point corresponded to the one signaled by t-area as the best performer (i.e., **wm**=0; **ws**=5; $\theta = 0.5$). Thus, we can conclude that small windows perform better than larger ones.

In Naxpot data-set we used two different configurations: a space-based window of size 10 and a time-based window of size 3600. In both cases we used a minimum evidence threshold θ of 0.5. According to t-area, the best performer in Naxpot data-set was the configuration that used a space-based window of size 10, a t-area of 0.9780 against a t-area of 0.9686. However, we will see below how this is not always true since, depending on the specific cost conditions, the time-based configuration could be better. Figure 6.10 shows the ROC curves and ROC points for the configurations that we tested in Naxpot data-set. Figure 6.11 shows the points (black circles) that conform the convex hull for Naxpot data-set. In this case, both configurations are maximal. Thus, we need to compute the set of iso-performance lines corresponding to all possible cost distributions before we can determine which configuration is the winner. Using Equation 6.16, we can say that both alert triage systems will have the same expected cost when the slope of the iso-performance line is $0.0053 =$

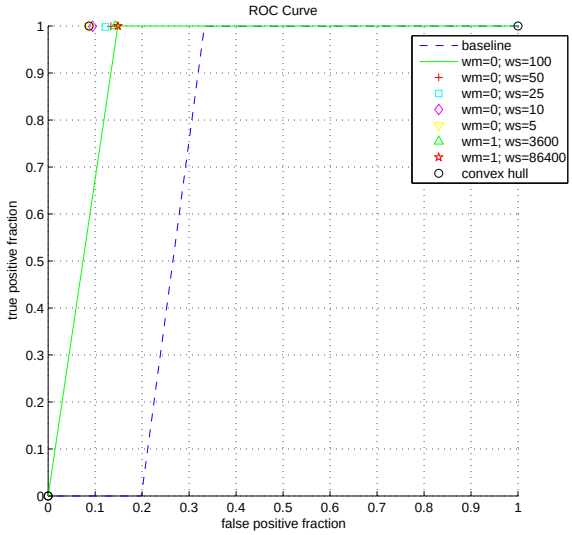


Fig. 6.9 Convex hull for Rustoord data-set.

$\frac{(0.9967-0.9968)}{0.0408-0.0597}$. Approximately, if we would use an error type weighting of 1:200 (i.e., a cost of 1 for each false positive and a cost of 200 for each false negative) then both alert triage configurations would achieve the same performance. Figure 6.12 depicts two iso-performance lines in the former ROC space. The first (magenta solid) line corresponds to an error type weighting of 1:2 and the second (red dashed-dot) line corresponds to an error type weighting of 1:500. Using the first iso-performance line (slope = 0.5) the optimal configuration corresponds to the system that used a space-based window model whereas using the second iso-performance line (slope = 0.002) the optimal configuration corresponds to the system that used a time-based window model. Thus, depending on the concrete cost characteristics of the systems under surveillance a specific configuration is best suited than another.

According to t-area, the best performers in Huckleberry data-set are the configurations that used a space-window model of size 10 and θ equal either to 0.3 or 0.9. Both configurations are tied at 0.9972 t-area. Figure 6.13 shows the ROC curves and ROC points for the configurations that we tested in Huckleberry data-set. Figure 6.14 additionally shows the convex hull where the configuration that used a $\theta = 0.9$ is the only one present and therefore the optimal under all possible cost conditions.

Let us now draw some conclusions about Ceaseless CBR. Generally speaking, we can say that the performance of Ceaseless CBR increases with θ and decreases with the size (number of alerts) of the window model. Therefore, configurations that use small windows (either in time or space) and a high evidence threshold are highly

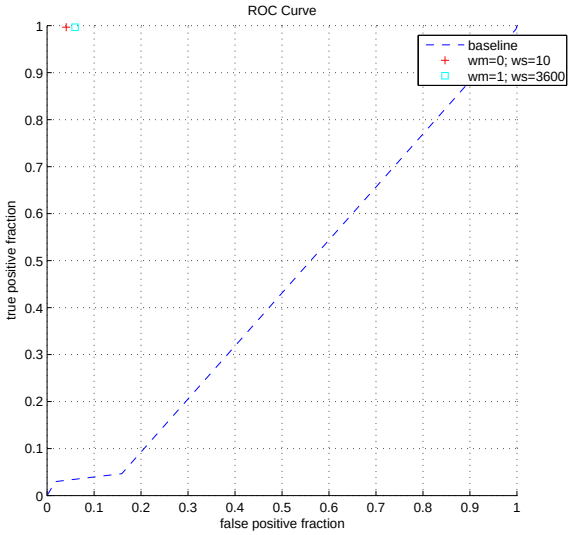


Fig. 6.10 ROC curves and points for several Ceaseless CBR configurations in Naxpot data-set.

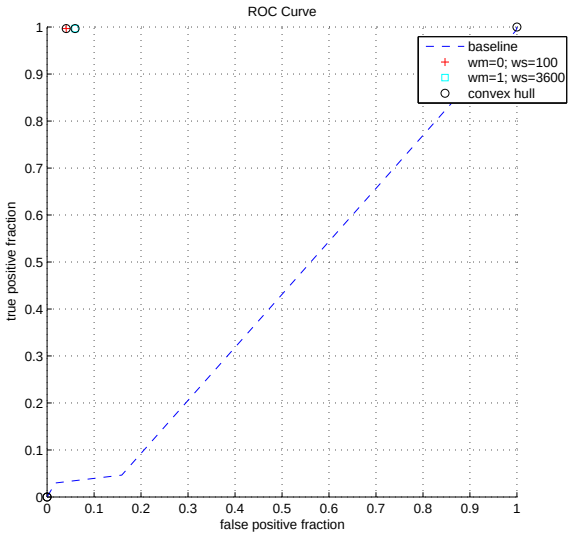


Fig. 6.11 Convex hull for Naxpot data-set.

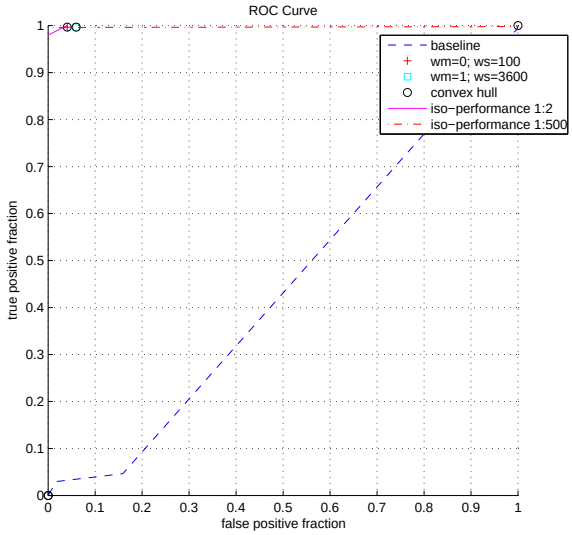


Fig. 6.12 Iso-performance lines for Naxpot data-set.

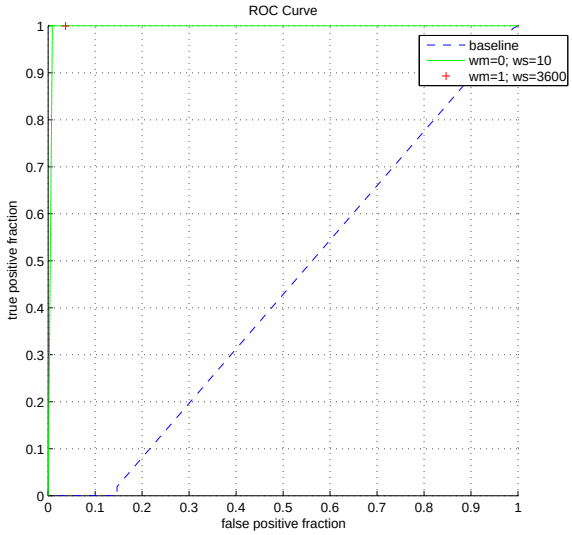


Fig. 6.13 ROC curves and points for several Ceaseless CBR configurations in Huckleberry data-set.

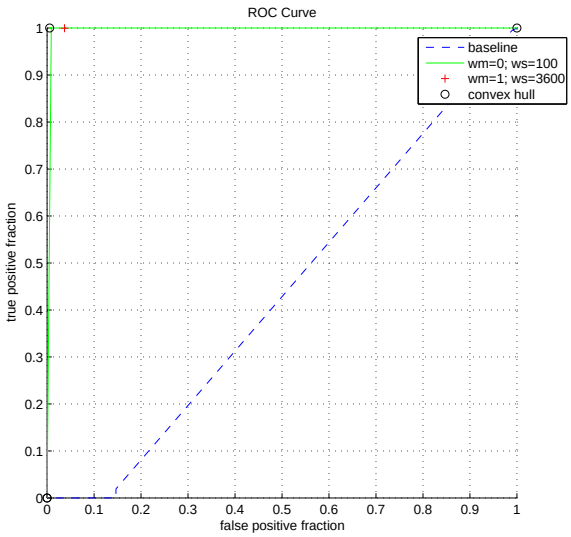


Fig. 6.14 Convex hull for Huckleberry data-set.

recommendable to increase the performance of Ceaseless CBR (i.e., to reduce both the number of false negatives and false positives).

Remember that Ceaseless Retrieve generates a case activation with its corresponding abstract case for each previously unknown alert (i.e., alerts whose domain is empty). Moreover, Ceaseless Retrieve fuses together case activations compounding in turn new large abstract cases together. When these new cases are integrated into an overall explanation presented to the user and accepted by her, then they are stored as as sequential master case. This process, performed by Ceaseless Retain is called sequential case discovery (see Section 5.6).

Let's now see some insights about the sequential cases discovered by Ceaseless CBR.

Sequential Case Discovery

Let us first show three examples of sequential cases that were found in each one of our data-sets:

- In Rustoord, we found a sequential case (that was assigned id 34507) that occurred 27 times. This sequential case was derived from a previous sequential case (with id 152) and was composed of 3 alerts, {882, 895, 1213} ([WEB-MISC backup access WEB-CGI calendar access WEB-CGI redirect access]), that appeared in two different episodes: [1213 882 895] and [882 1213

895]. This sequential case was represented by an actionable tree with a parallel node for alerts 1213 and 882 and a serial node for the former node and alert 895.

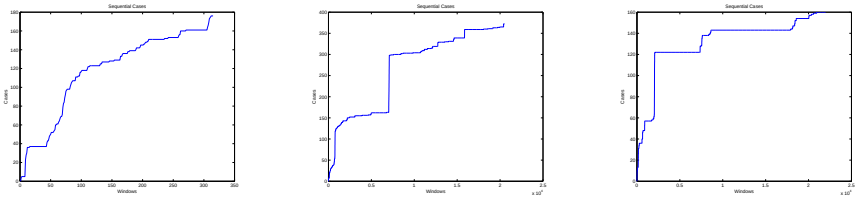
- In Naxpot, we found a sequential case that occurred 84 times (whose assigned id was 10669). It was derived from a previous sequential case (with id 1763) and contained three alerts, $\{982\ 1002\ 1288\}$, that appeared in only one episode: [1002 1288 982] ([WEB-IIS cmd.exe access WEB-FRONTPAGE /_vti_bin/ access WEB-IIS unicode directory traversal attempt]). This sequential case was represented by a serial actionable tree.
- In Huckleberry, we found a sequential case made up of 3 alerts, $\{$[1002 1256 1288$\}$, that occurred exactly in the same order (i.e., only in 1 episode) 89 times: [WEB-IIS cmd.exe access WEB-IIS CodeRed v2 root.exe access WEB-FRONTPAGE /_vti_bin/ access]. This sequential case (with id 10538) was derived from a previous sequential case (with id 1323) and was represented by a serial actionable tree.

Table 6.8 shows some statistics on sequential cases discovered by Ceaseless CBR in our data-sets. In addition to the configuration parameter of each trial. We can see, in this order, the total number of sequential cases discovered, the number of sequential cases that represented burst situations, the median number of alerts per sequential case, the median number of episodes per sequential case, and the median number of occurrences per case. Some other relevant results such as the number of sequential cases that were adapted from previous ones, and the number of queries (either to the user or to an automatic feedback process) needed appear in Table 6.9. Next, we analyze these data in further detail.

Number of sequential cases The number of sequential cases found ranged from 90 to 195 in Rustoord, from 255 to 273 in Naxpot, and from 118 to 160 in Huckleberry. The number of sequential cases seems to be governed by both the size of the window (**ws**) and the minimum evidence threshold (θ). As long as we fix the window size, the number of sequential cases decreases with the increment of θ. On the contrary, if we fix θ, the number of sequential cases increases with the decrement of the size of the window. Figure 6.15(a) shows the cumulative number of sequential cases in Rustoord data-set using Ceaseless CBR with a space-based window of size of 100 alerts and $\theta = 0.9$. Likewise, Figures 6.15(b) and 6.15(c) depict respectively the cumulative number of sequential cases found in Naxpot and Huckleberry data-sets using Ceaseless CBR with a space-based window of size 10 alerts and $\theta = 0.5$ and $\theta = 0.9$ respectively. In the three trials we can observe that the cumulative number of sequential cases grows rapidly at the beginning of the experiment and then more gradually. Approximately, in the first quarter of the experiment Ceaseless CBR discovers the 75% of the sequential cases. From this observation, we can say that an important part of the alert load corresponds to situations that occur again and again as we already anticipated when we depicted the scatter plots of Appendix A.

data-set	wm	ws	θ	Cases	Bursts	Alerts	Episodes	Occ.
Rustoord	0	100	0.1	182	5	2.00	1.00	7.00
Rustoord	0	100	0.3	177	5	2.00	1.00	7.00
Rustoord	0	100	0.5	173	5	2.00	1.00	7.00
Rustoord	0	100	0.7	173	5	2.00	1.00	7.00
Rustoord	0	100	0.9	176	5	2.00	1.00	6.50
Rustoord	0	100	1	159	5	1.00	1.00	6.00
Rustoord	0	50	0.5	175	6	2.00	1.00	8.00
Rustoord	0	25	0.5	186	10	2.00	1.00	8.00
Rustoord	0	10	0.5	193	23	1.00	1.00	9.00
Rustoord	0	5	0.5	195	35	1.00	1.00	9.00
Rustoord	0	1	0.5	90	0	1.00	1.00	17.00
Rustoord	1	3600	0.5	189	26	1.00	1.00	7.00
Rustoord	1	86400	0.5	165	5	2.00	1.00	7.00
Naxpot	0	10	0.5	373	26	3.00	1.00	2.00
Naxpot	1	3600	0.5	255	26	2.00	1.00	3.00
Huckleberry	0	10	0.1	143	10	2.00	1.00	4.00
Huckleberry	0	10	0.3	143	10	2.00	1.00	3.00
Huckleberry	0	10	0.5	146	10	2.00	1.00	3.00
Huckleberry	0	10	0.7	150	10	2.00	1.00	4.00
Huckleberry	0	10	0.9	160	10	2.00	1.00	4.00
Huckleberry	0	10	1	148	10	2.00	1.00	3.00
Huckleberry	1	3600	0.5	118	18	1.00	1.00	3.00

Table 6.8 Sequential cases statistics for CCBR. **data-set**= data-set name; **wm** = window model; **ws** = window size; θ = evidence threshold; **Cases** = total sequential cases found; **Bursts**= Burst situations; **Alerts** = median of alerts per case; **Episodes** = median of episodes per case; **Occ.** = median of occurrences per case.



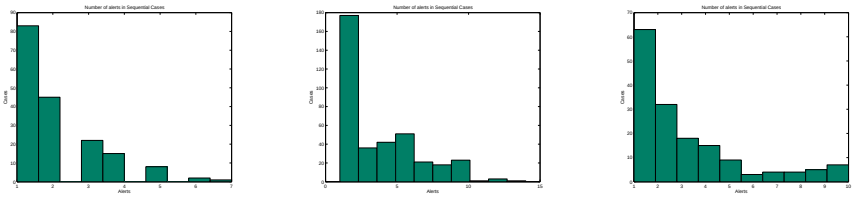
(a) (b) (c)

Fig. 6.15 Cumulative number of sequential cases found. (a) In Rustoord data-set using Ceaseless CBR with a space-based window of size 100 alerts and $\theta = 0.9$. (b) In Naxpot data-set using Ceaseless CBR with a space-based window size of 10 alerts and $\theta = 0.5$. In Huckleberry data-set using Ceaseless CBR with a space-based window of size 10 alerts and $\theta = 0.9$.

Burst situations The number of cases discovered corresponding to burst situations[11] depends exclusively on the the window model and window size as shown in

[11]Notice that the number of sequential cases that represent burst situations does not represent the number of burst occurrences but the number of distinct burst situations. For example, in Rustoord data-set using a space-based window of size 10 alerts and $\theta = 0.5$ there were 23 cases that represented burst situations. One of these cases corresponding to alert WEB-IIS cmd.exe access (sid = 1002) occurred altogether 377 times.

Table 6.8. When the window model and size were fixed the number of burst situations found did not vary. It is easy to understand that the smaller the window size, the bigger the number of distinct burst situations detected. Notice that when the size is 1 (alert-driven model) the number of burst situations detected is 0.

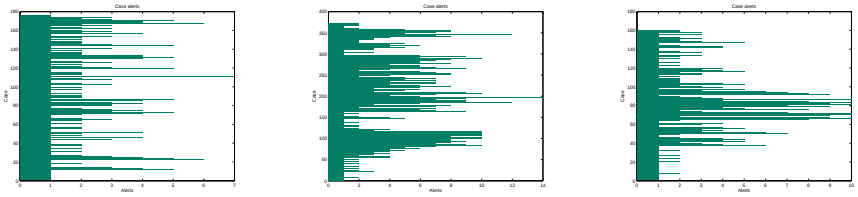


(a) (b) (c)

Fig. 6.16 Number of cases per number of alerts. (a) In Rustoord data-set using Ceaseless CBR with a space-based window of size 100 alerts and $\theta = 0.9$. In Naxpot data-set using Ceaseless CBR with a space-based window of size 10 alerts and $\theta = 0.5$. (c) In Huckleberry data-set using Ceaseless CBR with a space-based window of size 10 alerts and $\theta = 0.9$.

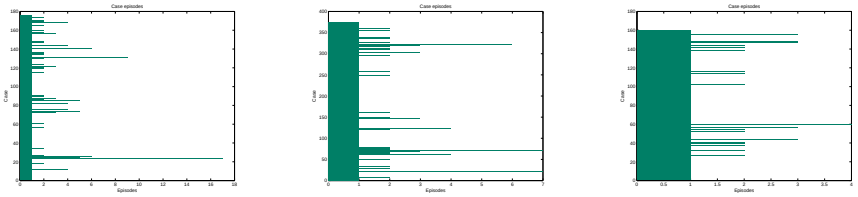
Alerts per sequential case As shown by Table 6.8 the median of alerts per case varied between 1 and 2 in both Rustoord and Huckleberry data-sets whereas in Naxpot varied between 2 and 3. Apparently, the number of alerts per case decreases with the window size and does not vary with θ. Figure 6.16(a) shows the number of alerts per case in Rustoord data-set using Ceaseless CBR with a space-based window size of 100 alerts and $\theta = 0.9$. More than 80 cases were composed of only one alert. An important number of cases, more than 40, were composed of 2 alerts. Almost 10 cases were composed of 5 alerts, and there even cases with 6 and 7 alerts. The same graphic is depicted for Naxpot and and Huckleberry datasest using Ceaseless CBR with a space-based window size of 10 alerts and $\theta = 0.5$ and $\theta = 0.9$ respectively by Figures 6.16(b) and Figures 6.16(c). Notice that there are sequential cases that were made up of up to 10 or more alerts. We have also depicted the number of alerts for each concrete sequential case. Figure 6.17(a) represents the number of alerts in each of the 176 cases in Rustoord data-set. Figure 6.17(b) and 6.17(c) do the same for the respective 373 and 160 sequential cases in Naxpot and Huckleberry.

Episodes per sequential case Another important aspect to consider is the number of episodes in each sequential case. This number is entirely related to the underlying actionable tree that will subsequently be used to represent the sequential case. A sequential case with only an episode is a serial actionable tree. The distinct episodes will determine which nodes in the representation will be serial and which ones parallel as we saw in the above examples. At first sight, it seems that the number of episodes found does not vary with the window model nor



(a) (b) (c)

Fig. 6.17 Alerts in Cases. (a) In Rustoord data-set using Ceaseless CBR with a space-based window of size of 100 alerts and $\theta = 0.9$. (b) In Naxpot data-set using Ceaseless CBR with a space-based window of size of 10 alerts and $\theta = 0.5$. (c) In Huckleberry data-set using Ceaseless CBR with a space-based window of size of 10 alerts and $\theta = 0.9$.

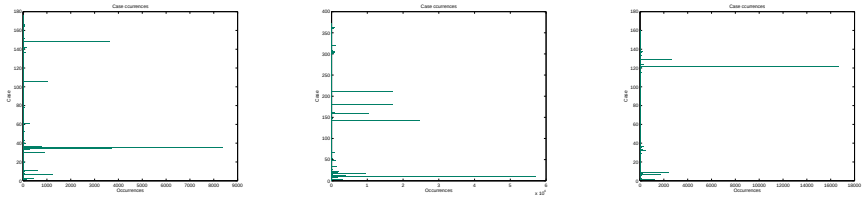


(a) (b) (c)

Fig. 6.18 Episodes in Cases. (a) In Rustoord data-set using Ceaseless CBR with a space-based window of size 100 alerts and $\theta = 0.9$. In Naxpot data-set using Ceaseless CBR with a space-based window of size 10 alerts and $\theta = 0.5$. In Huckleberry data-set using Ceaseless CBR with a space-based window of size 10 alerts and $\theta = 0.9$.

with θ. Figure 6.18(a) shows the number of episodes for each of the 176 cases discovered in Rustoord data-set. At least 8 sequential cases surpass the four episodes per sequential case. Figure 6.18(b) shows the number of episodes for each of the 373 cases in Naxpot data-set, and Figure 6.18(c) shows the number of episodes for each of the 160 sequential cases discovered in Huckleberry data-set.

Occurrences per sequential case Now, let us analyze the number of occurrences of each one the sequential cases found. That is, the number of times that the sequential case was used to prioritize the corresponding alerts. At a glance, it seems that the smaller the window size the higher the number of occurrences found and that the higher the value of θ the lower the number of occurrences per case detected. In Figure 6.19(a) we can see the number of occurrences



(a) (b) (c)

Fig. 6.19 Occurrences of cases. (a) In Rustoord data-set using Ceaseless CBR with a space-based window of size 100 alerts and $\theta = 0.9$. (b) In Naxpot data-set using Ceaseless CBR with a space-based window of size 10 alerts and $\theta = 0.5$. (c) in Huckleberry data-set using Ceaseless CBR with a space-based window of size 10 alerts and $\theta = 0.9$.

of each of the 176 cases in Rustoord. We can see, for example, that five sequential cases occurred more than a thousand times. Figures 6.19(b) and 6.19(c) data the same data for Naxpot and Huckleberry data-sets using the same configurations as above.

Adaptations The last but one column in Table 6.10 (**A**) shows the number of new sequential cases that were created adapting previous encountered sequential cases either generalizing using the hierarchy of sorts or extending the length of previous sequential cases because new alerts were found to meet the constraints expressed by the sequential case. For example, in Rustoord data-set the number of adapted sequential cases varied between 107 and 122. The percentage of adapted sequential cases over the total of sequential cases found (see Table 6.8) varied between 0 when Ceaseless CBR was configured using an alert-driven model and a 68.55% when Ceaseless CBR was configured using a space-based of size 100 alerts and $\theta = 1$. In Naxpot, the number of sequential cases adapted ranged from 87 to 167 and the percentage of adapted sequential cases ranged from a 34.12% (using a time-based window) to a 44.77% (using a space-based window). In Huckleberry, the number of sequential cases varied between 52 and 80 and the corresponding percentages between a 44.07% (using a time-based window of 3600 seconds and $\theta = 0.5$) and a 50.68% (using a space-based window of 100 alerts and $\theta = 0.5$). It is difficult to extrapolate from the data observed whether the number of adaptations depends either on the window model or θ. An important point here, which we have left for future work, would be to measure the goodness of the adaptations performed by Ceaseless CBR, separating between those adaptations that contribute to detect new attacks that are new variants of known attacks and those adaptations that are only good for dismissing false positive alerts.

Queries Remember that in our approach feedback can be provided automatically using a network model that is in turn provided by a network scanner or emulated

as in our experiments. We annotated however the number of user interactions required in each trial. They appear reflected as the number of questions (**Q**) in Table 6.10. It seems that the number of questions decreases when the evidence threshold increases while it increases when the size of the window decreases. If the cost by query is high then a large window size and high evidence threshold are recommendable.

We have seen how the window model, window size, and minimum evidence threshold affect the sequential cases discovered by Ceaseless CBR along several dimensions. Next, we will analyze the time taken by each Ceaseless CBR process.

Time Analysis

The last three columns in Table 6.9 show the mean time elapsed by each of the Ceaseless CBR processes. For example, in Rustoord data-set the mean elapsed time per retrieval varied between 0.03 seconds and 5.64 seconds. The highest value corresponded with a time-based window of 1 day (86400) seconds and the lowest with an alert-driven model. Likewise, the time elapsed by Ceaseless Reuse and Ceaseless Revise+Retain ranged respectively from almost 0 and 0.01 seconds when using an alert-driven model to 9.27 and 2.75 seconds when using a time-based window of size 1 day. In Naxpot the time taken by Ceaseless Retrieve varied between 1.84 seconds and 35.97 seconds when using a time-based window of 3600 seconds. In Huckleberry the elapsed time for all Ceaseless CBR processes (Retrive, Reuse, Revise+Retain) were the same for all trials (0.05 seconds, 0.00 seconds, and 0.07) except for when using a time-based window that the times went up to 5.15, 1.47, and 0.79 seconds respectively. These variations are due to the limiting factors that we will see below.

The total time taken by each trial is shown in the last column of Table 6.10. Ranging from 1252 seconds when using an alert-driven model in Rustoord (the smallest) data-set to 247906 seconds when using a time-based window of size 3600 seconds in Naxpot (that has almost 7 times more alerts than Rustoord). These data clearly show how the total time elapsed depends on the window model chosen. For example, in the case of Rustoord data-set there is a variation of up to 2437 seconds between an alert-driven model and a fixed time-based window of size 86400 seconds. Both tables show a number of limiting factors in the performance. Table 6.9 shows the number total of windows ($\#\mathbf{w}$), the mean of alerts per window ($\bar{\mathbf{a}}$), the mean interval time per window computed considering the inter-arrival time between the first alert and the last alert in the window (**I**), and the mean number of cases per window ($\bar{\mathbf{c}}$). The mean interval time per window gives us a rough idea of the mean reaction time that we would have if we would want to proceed on-line. Table 6.10 shows additional data such as the total number of sequential cases retrieved (**R**) by Ceaseless Retrieve, the total number of hypotheses (**H**) generated by Ceaseless Retrieve, the total number of explanations (**E**) generated by Ceaseless Reuse, the total number of queries formulated (**Q**) by Ceaseless Revise, and the total number of adapted cases (**A**).

data-set	wm	ws	θ	#w	$\bar{\mathbf{a}}$	I	$\bar{\mathbf{c}}$	Ret	Reu	Rev
Rustoord	0	100	0.1	315	99.95	30283.63	116.50	3.92	3.37	1.59
Rustoord	0	100	0.3	315	99.95	30283.63	113.70	3.68	3.09	1.54
Rustoord	0	100	0.5	315	99.95	30283.63	113.05	3.65	2.95	1.51
Rustoord	0	100	0.7	315	99.95	30283.63	114.26	3.57	2.93	1.50
Rustoord	0	100	0.9	315	99.95	30283.63	117.37	3.76	2.87	1.49
Rustoord	0	100	1	315	99.95	30283.63	105.24	3.14	2.82	1.43
Rustoord	0	50	0.5	630	49.97	14954.95	117.87	1.86	0.66	0.75
Rustoord	0	25	0.5	1260	24.99	7405.56	122.62	0.90	0.16	0.49
Rustoord	0	10	0.5	3149	10.00	2478.12	125.68	0.33	0.02	0.21
Rustoord	0	5	0.5	6297	5.00	1141.79	125.35	0.15	0.01	0.08
Rustoord	0	1	0.5	31483	1.00	1.00	65.11	0.03	0.00	0.01
Rustoord	1	3600	0.5	1137	27.69	941.53	120.19	1.23	0.67	1.02
Rustoord	1	86400	0.5	209	150.64	32226.89	105.49	5.64	9.27	2.75
Naxpot	0	10	0.5	20498	10.00	919.48	265.32	1.84	0.03	1.42
Naxpot	1	3600	0.5	5353	38.29	2450.69	171.50	35.97	1.07	9.27
Huckleberry	0	10	0.1	21989	10.00	406.85	119.35	0.05	0.00	0.07
Huckleberry	0	10	0.3	21989	10.00	406.85	117.29	0.05	0.00	0.07
Huckleberry	0	10	0.5	21989	10.00	406.85	119.35	0.05	0.00	0.07
Huckleberry	0	10	0.7	21989	10.00	406.85	122.90	0.05	0.00	0.07
Huckleberry	0	10	0.9	21989	10.00	406.85	130.96	0.05	0.00	0.07
Huckleberry	0	10	1	21989	10.00	406.85	121.19	0.05	0.00	0.07
Huckleberry	1	3600	0.5	1989	109.43	709.41	90.33	5.15	1.47	0.79

Table 6.9 Window model characteristics for CCBR. **data-set**= data-set name; **wm** = window model; **ws** = window size; θ = evidence threshold; **#w** = number of windows; $\bar{\mathbf{a}}$ = mean of alerts per window; **I** = mean interval time per window; $\bar{\mathbf{c}}$ = mean of cases per window; **Ret** = mean elapsed time per retrieval; **Reu** = mean elapsed time per reuse; **Rev** = mean elapsed time per revise & retain

data-set	wm	ws	θ	R	H	E	Q	A	T
Rustoord	0	100	0.1	4109	26053	872	69	113	2797.69
Rustoord	0	100	0.3	2682	24692	562	62	115	2618.24
Rustoord	0	100	0.5	2199	24258	437	61	112	2556.84
Rustoord	0	100	0.7	1881	24040	408	58	115	2519.09
Rustoord	0	100	0.9	1539	23872	356	59	117	2559.03
Rustoord	0	100	1	0	23670	315	50	109	2326.10
Rustoord	0	50	0.5	3347	23194	847	58	117	2058.70
Rustoord	0	25	0.5	5515	22823	1609	64	122	1948.27
Rustoord	0	10	0.5	10244	19863	3868	77	116	1754.30
Rustoord	0	5	0.5	11426	18238	7598	84	111	1460.85
Rustoord	0	1	0.5	31393	31483	31483	90	0	1252.16
Rustoord	1	3600	0.5	4271	24941	1491	78	111	3313.34
Rustoord	1	86400	0.5	1700	25789	287	58	107	3689.05
Naxpot	0	10	0.5	65783	154643	29650	206	167	67395.44
Naxpot	1	3600	0.5	29566	195586	9528	168	87	247906.52
Huckleberry	0	10	0.1	8293	33169	23289	75	68	2627.11
Huckleberry	0	10	0.3	6259	31508	22694	72	71	2588.15
Huckleberry	0	100	0.5	5025	30505	22314	72	74	2629.00
Huckleberry	0	10	0.7	5434	30913	22896	75	75	2651.00
Huckleberry	0	10	0.9	4208	29759	22523	80	80	2650.38
Huckleberry	0	10	1	0	28225	21989	79	69	2594.47
Huckleberry	1	3600	0.5	3293	39914	2299	66	52	14729.29

Table 6.10 Performance characteristics for CCBR. **data-set**= data-set name; **wm** = window model; **ws** = window size. θ = evidence threshold; **R** = number of cases retrieved; **H** = number of hypotheses generated; **E** = number of explanations generated; **Q** = number of questions formultated; **A** = number of cases adapted; **T** = total elapsed time.

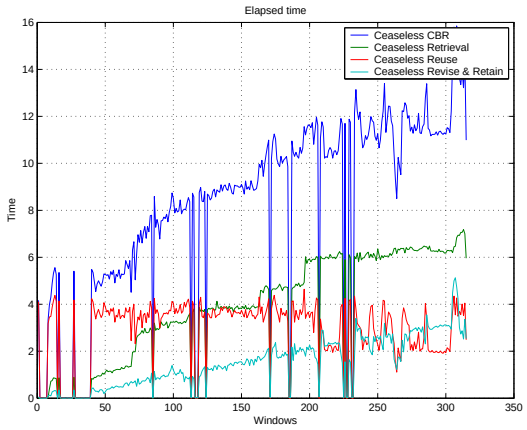


Fig. 6.20 Elapsed time by Ceaseless CBR processes in Rustoord data-set using a space-based window of size 100 alerts and $\theta = 0.9$

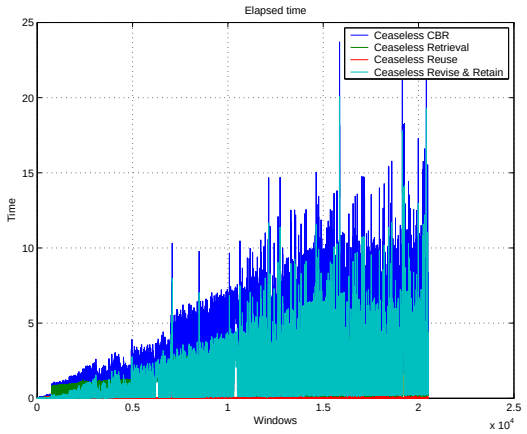


Fig. 6.21 Elapsed time by Ceaseless CBR processes in Naxpot data-set using a space-based window of size 10 alerts and $\theta = 0.5$.

A first conclusion that can be drawn form these data is that the total time elapsed decreases in function of the number of alerts per window and increases in function of θ. As a matter of fact, an upper bound on the time taken by Ceaseless Retrieve ($\mathbf{Ret}$) could be computed in terms of the number of sequential cases, the number of episodes per sequential case, the number of alerts per episode and the number of alerts per window. Likewise, an upper bound of the time elapsed by Ceaseless Reuse ($\mathbf{Reu}$) can be established using the number hypotheses generated and the mean number of alerts to explain (alerts per window in this group of experiments). An upper bound

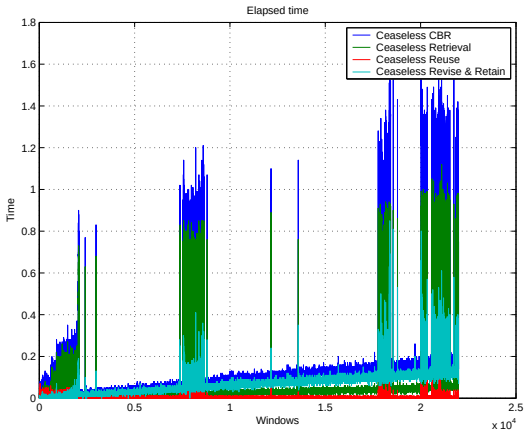


Fig. 6.22 Elapsed time by Ceaseless CBR processes in Huckleberry data-set using a space-based window of size 10 alerts and $\theta = 0.9$.

of the time used by Ceaseless Revise and Ceaseless Retain ($\mathbf{Rev}$) can be defined in terms of the mean number hypotheses per explanations, the number of queries, and the number of previous sequential cases in the case base. Figure 6.20 plots the time taken by each process against 315 windows of size 100 alerts that were used to analyzed Rustoord data-set using a minimum evidence threshold of 0.9. The time taken by Ceaseless Reuse (red line) remains practically constant since the number alerts to explain in each iteration is constant The time taken by Ceaseless Revise & Retain (light blue line) grows linearly with the size of the case-base (see Figure 6.15). However, the time taken by Ceaseless Retrieve (green line) grows more more abruptly and so does the total time taken by Ceaseless CBR (blue line) taken up to 14 seconds at the end of the analysis according with the number of sequential cases in case-base (see again Figure 6.15). Notice that each valley corresponds to a burst situation where Ceaseless CBR performed considerably fast. Figures 6.21 and 6.22 show similar plots for the other two data-sets.

All experiments described were run on a dual processor Power Mac G5 at 2GHz. We used MATLAB6p5 software package to code our algorithms.

Preemptive Ratio

Table 6.11 shows the total number of predicted alerts (**P**), the total number of successful predicted alerts or anticipated alerts (**A**), and the preemptive ratio (**PR**) (i.e., $\mathbf{PR} = \frac{\text{successes}}{\#\text{predictions}}$) for each trial. For example, in Rustoord data-set using a fixed space-based window of size 5 the preemptive ratio was 0.78. That is, about 78% of times that Ceaseless CBR predicted the occurrence of an alert it finally happened. In Naxpot the highest $\mathbf{PR}$ was only 0.43 whereas in Huckleberry went up to 0.90. We think that one of the factors that caused the poor $\mathbf{PR}$ of Ceaseless CBR in Naxpot

data-set	wm	ws	θ	P	A	PR	weekly alert load	weekly alert reduction
Rustoord	0	100	0.1	542	248	0.46	1967.69	92.35%
Rustoord	0	100	0.3	483	250	0.52	1967.69	92.36%
Rustoord	0	100	0.5	431	247	0.57	1967.69	92.41%
Rustoord	0	100	0.7	386	239	0.62	1967.69	92.39%
Rustoord	0	100	0.9	342	230	0.67	1967.69	92.40%
Rustoord	0	100	1	0	0	N/A	1967.69	91.73%
Rustoord	0	50	0.5	734	483	0.66	1967.69	93.17%
Rustoord	0	25	0.5	1240	817	0.66	1967.69	94.04%
Rustoord	0	10	0.5	2012	1518	0.75	1967.69	95.35%
Rustoord	0	5	0.5	2093	1643	0.78	1967.69	95.90%
Rustoord	0	1	0.5	0	0	N/A	1967.69	95.11%
Rustoord	1	3600	0.5	979	446	0.46	1967.69	92.48%
Rustoord	1	86400	0.5	288	143	0.50	1967.69	92.01%
Naxpot	0	10	0.5	4806	2072	0.43	5856.49	80.89%
Naxpot	1	3600	0.5	1684	342	0.20	5856.49	79.69%
Huckleberry	0	10	0.1	758	566	0.75	13742.88	93.02%
Huckleberry	0	10	0.3	575	481	0.84	13742.88	93.04%
Huckleberry	0	10	0.5	570	487	0.85	13742.88	92.92%
Huckleberry	0	10	0.7	545	458	0.84	13742.88	92.93%
Huckleberry	0	10	0.9	431	386	0.90	13742.88	93.02%
Huckleberry	0	10	1	0	0	N/A	13742.88	92.68%
Huckleberry	1	3600	0.5	388	274	0.71	13742.88	84.77%

Table 6.11 Predictive characteristics and weekly alert load reduction for CCBR. **data-set**= data-set name; **wm** = window model; **ws** = window size. θ = evidence threshold; **P** = number of predicted alerts; **A** = number of anticipated alerts; **PR**= preemptive ratio.

was variability on the number sorts (unique alerts) 331 vs 90 in Rustoord and 203 in Huckleberry. The preemptive ratio seems to increase with θ since the evidence of the corresponding actionable tree used to make the prediction is higher. The preemptive ratio also increases with small windows. Notice that the preemptive ratio is not defined when either $\theta = 1$ or when the window size is 1 (i.e., an alert-driven window model) since there is no predicted alerts at all.

Alert Load Reduction

Table 6.11 also shows for each trial the mean weekly alert load and the average weekly alert reduction percentage achieved using Ceaseless CBR. For example, in Rustoord data-set the maximum alert reduction was 95.90% that corresponded to the trial where we used a space-based window of size 5 and $\theta = 0.5$. Figures 6.23 compares original alert load with actual alert load using Ceaseless CBR with a fix window size of 100 alerts and $\theta = 0.9$. Figure 6.26 depicts alert reduction achieved along 16 weeks in Rustoord data-set using the later configuration. From this data, we can see that Ceaseless CBR achieved an alert reduction of more than 90% in 14 weeks out of 16 weeks covered by the data-set. The total alert reduction was a 95.80%. In Naxpot data-set the maximum alert reduction in the two trials shown in Table 6.11 corresponded to the space-based window model. Ceaseless CBR achieved a reduction of 80.89% of the weekly alert load that was originally of 1967 alerts per week (on average). Figure 6.24 shows the comparison between current alert load and the initial alert load in Naxpot data-set along 35 weeks of surveillance whereas Figure

6.27 shows the weekly alert load reduction. In Huckleberry data-set, we achieved an alert reduction of up to a 93.02% using a space-based window of 10 alerts and $\theta = 0.9$. Likewise, Figure 6.25 compares for Huckleberry data-set the current alert load and the initial alert load along 16 weeks. Finally, Figure 6.28 shows the alert load reduction achieved in Huckleberry.

Alert reduction seems to decrease with the increment of the window size and be practically unaffected by θ. Notice that, in all three data-sets, the configuration with the best alert reduction coincides with the best performer according to t-area. However, we think that we would need to run more experiments before a categorical conclusion can be drawn from this observation. That is, before stating that the best Ceaseless CBR performer coincides with the one achieving the highest alert reduction.

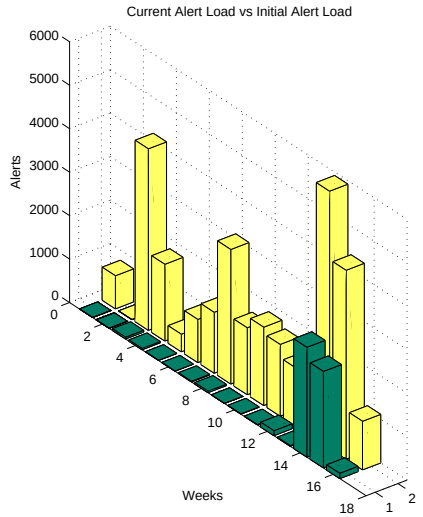


Fig. 6.23 Current alert load vs initial alert load in Rustoord data-set using Ceaseless CBR with a space-based window of size 100 alerts and $\theta = 0.9$.

These results show how automatically assessing the malignancy of alerts using Ceaseless CBR pays off by reducing significantly weekly alert load. We could say that our results outperform the results of the only work we are aware of that has tackled with alert reduction [JD02]. Julisch and Dacier achieved a maximum of 75% on average alert load reduction over a year using attribute-oriented induction (AOI) [JD02]. However, we think that a common alert database should be used to make both approaches comparable.

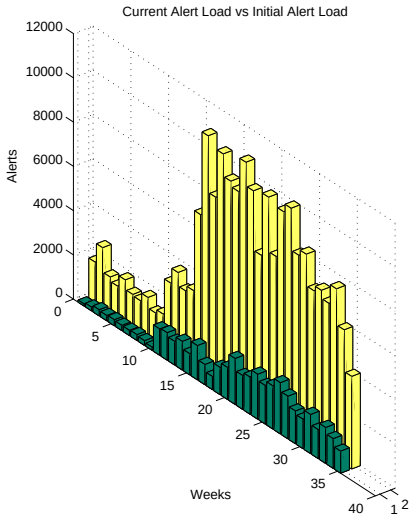


Fig. 6.24 Current alert load vs initial alert load in Naxpot data-set using Ceaseless CBR with a space-based window of size 10 alerts and $\theta = 0.5$.

Finally, let us remark that although the performance of an alert-driven approach (**wm**=0 and **ws**=1) seems, at first sight, excellent (i.e., **tpf**=1 and **fpf**=0 in Table 6.6) it has a number of disadvantages:

1. First of all, a higher number of queries (human interventions) is required, 90 were needed in Rustoord data-set as shown in Table 6.10. In practice, one query was needed for each type of alert. Notice that using a window size of 100 alerts the number of queries was reduced down to 40.

2. Second, a null preemptive ratio as seen in Table 6.11. That is, it is much more difficult to predict the occurrence of future alerts using an alert-driven model.

3. Third, the difficulty of discovering burst situations as we can see in Table 6.8.

4. The last but not least important disadvantage is the impossibility of discovering multi-stage attacks since each alert is analyzed in isolation without considering its sequential relationships with other alerts.

In pursuit of a finer-grained understanding of the influence of urgency in our approach we have carried out a second block of experiments where we tried to investigated the impact on the performance when not all alerts in the current window are considered for explanation but only the most urgent ones. For the sake of simplicity, we only show next the results corresponding to the best performers above.

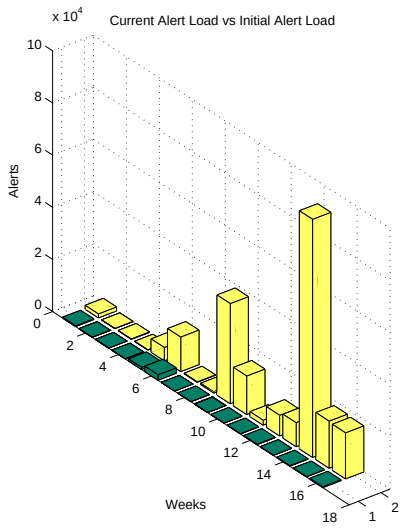


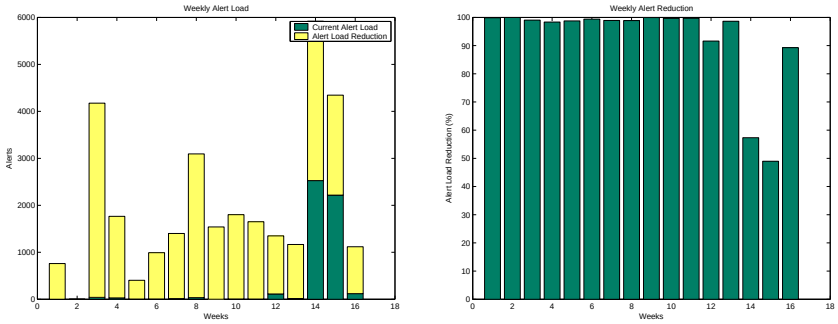
Fig. 6.25 Current alert load vs initial alert load in Huckleberry data-set using Ceaseless CBR with a space-based window of size 10 alerts and $\theta = 0.9$.

data-set	wm	ws	θ	tpf	fpf	recall	precision	prevalence
Rustoord	0	5	0.5	0.9997	0.0126	0.9997	0.9029	0.1046
Naxpot	0	10	0.5	0.9966	0.0186	0.9966	0.9303	0.1995
Naxpot	1	3600	0.5	0.9987	0.0432	0.9987	0.8520	0.1995
Huckleberry	0	10	0.9	1.0000	0.0029	1.0000	0.9712	0.0899

Table 6.12 Experimental results for UCCBR. **data-set**= data-set name; **wm** = window model; **ws** = window size; θ = evidence threshold; **tpf** = true positive fraction; **fpf** = false positive fraction.

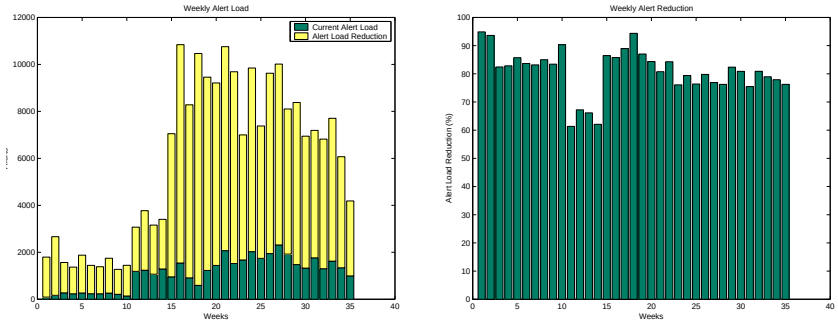
6.3.7 UCCBR

We analyzed the performance of the best configurations in the former experiments using the measure of urgency introduced in Chapter 5 and obtained the results described below. We used a deprecation factor τ of 7200 seconds (see Section 5.1.3). That is to say, after 2 hours case activations become deprecated and are filtered out from consideration and the corresponding alerts received directly the lowest priority. We used a error type weighting 1:U[1,1000] (i.e., a cost of 1 for each false positive and a cost uniformly distributed between 1 and 1000 for each false negative). Therefore each potential vulnerability simulated in our experimental setting got also associated a cost (single lost expectancy) randomly generated between 1 and 1000. The benefit of properly prioritizing a dangerous alert $B(D+|C+)$ was set to 100 whereas the benefit of not prioritizing an innocuous alert $B(D-|C-)$ was set to 1. We used a myopic iteration horizon.



(a) (b)

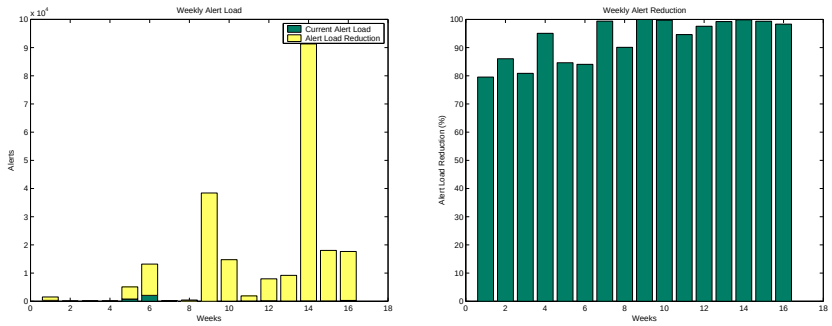
Fig. 6.26 Alert load reduction in Rustoord data-set using Ceaseless CBR with a space-based window of size 100 alerts and $\theta = 0.9$. (a) Alert load before and after Ceaseless CBR; (b) Weekly alert load reduction.



(a) (b)

Fig. 6.27 Alert load reduction in Naxpot data-set using Ceaseless CBR with a space-based window of size 10 alerts and $\theta = 0.5$. (a) Alert load before and after Ceaseless CBR; (b) Weekly alert load reduction.

Tables 6.12 and 6.13 summarize the performance results for this second block of experiments. At first view, we can see that UCCBR achieved a small improvement of t-area in each trial. Respectively, 0.0371 (0.9936 vs 0.9565), 0.0110 (0.9890 vs 0.9780), 0.091 (0.9777 vs 0.9686), and 0.013 (0.9985 vs 0.9972). This improvement was in part due to an important reduction in the number of false positives whose fraction varied from 0.0870 to 0.0126 in Rustoord data-set, from 0.0408 to 0.0186 and from 0.0597 to 0.0432 in Naxpot data-set, and from 0.0055 to 0.0029 in Huckleberry



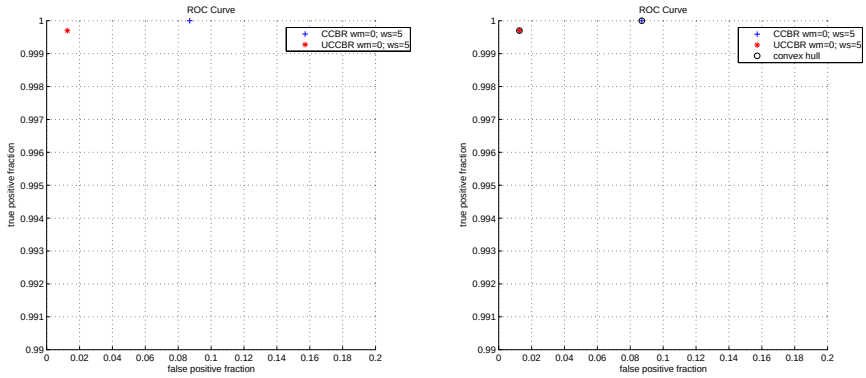
(a) (b)

Fig. 6.28 Alert load reduction in Huckleberry data-set using Ceaseless CBR with a space-based window of size 10 alerts and $\theta = 0.9$. (a) Alert load before and after Ceaseless CBR; (b) Weekly alert load reduction.

data-set	wm	ws	θ	accuracy	e-distance	f-measure	g-mean	t-area
Rustoord	0	5	0.5	0.9887	0.9911	0.9488	0.9936	0.9936
Naxpot	0	10	0.5	0.9844	0.9866	0.9623	0.9889	0.9890
Naxpot	1	3600	0.5	0.9651	0.9694	0.9195	0.9775	0.9777
Huckleberry	0	10	0.9	0.9973	0.9979	0.9854	0.9985	0.9985

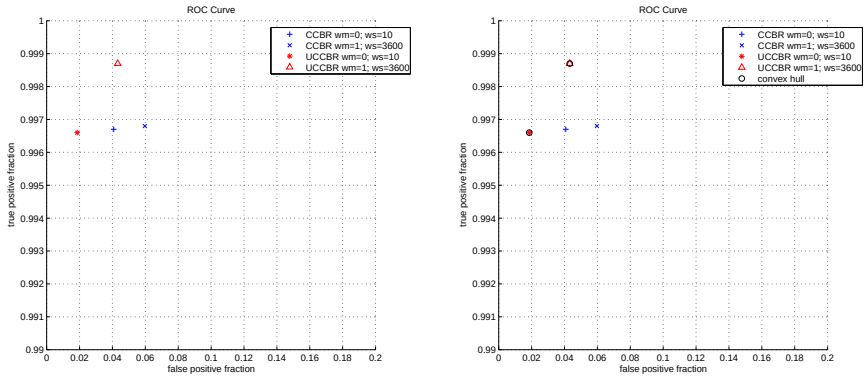
Table 6.13 Performance measures for UCCBR. **data-set**= data-set name; **wm** = window model; **ws** = window size; θ = evidence threshold;

data-set. However, the true positive fraction decreased in two of the trials. From 1 to 0.9997 in Rustoord data-set and from 0.9967 to 0.9966 in Naxpot data-set using a space-based window. Those decrements were caused by the confluence of two factors: the corresponding single lost expectancy associated to the involved sequential cases was too low and the corresponding belief was also too low. Therefore, Ceaseless CBR preferred not to prioritized them since the corresponding alerts were not considered urgent. In the other two trials, the number of true positives was kept or even improved (from 0.9967 to 0.9987 in Naxpot data-set using a time-based window). To properly claim that UCCBR outperforms CCBR in those trials we proceeded as indicated in our evaluation framework. Thus, we plotted the ROC points for both data-sets as shown in Figures 6.29(a) and 6.30(b) and their corresponding convex hulls in Figures 6.29(b) and 6.30(b). In the case of Rustoord data-set, notice that both ROC points formed part of the convex hull and therefore both are optimal under different cost conditions. In the case of Naxpot data-set, only the points representing UCCBR configurations lie in the convex hull. Thus, we only need to pick and choose between UCCBR configurations since the combination of all UCCBR configurations outperforms the combination of all CCBR configurations. We have already seen before how to use iso-performance lines to decide on this matter. As a conclusion we can say that if our sequential cases are able to properly store the damage (single lost expectancy)



(a) (b)

Fig. 6.29 (a) ROC points for several CCBR and UCCBR configurations in Rustoord data-set and (b) their corresponding convex hull.



(a) (b)

Fig. 6.30 (a) ROC points for several CCBR and UCCBR configurations in Naxpot data-set and (b) their corresponding convex hull.

caused by undesired situations then the use of a urgency measured in Ceaseless CBR is highly recommendable from the point of view of the overall performance achieved.

The sequential cases discovered using a measure of urgency did not vary so much from the results presented above except for the number of occurrences of each sequential case. This decrement was mainly due to the fact that many alerts were directly prioritized at the lowest level because they became deprecated and therefore they

data-set	wm	ws	θ	P	A	PR	weekly alert load	weekly alert reduction
Rustoord	0	50	0.5	734	613	0.83	1967.69	97.14%
Naxpot	0	10	0.5	1011	554	0.55	5856.49	89.43%
Naxpot	1	3600	0.5	746	254	0.34	5856.49	87.69%
Huckleberry	0	10	0.9	63	59	0.93	13742.88	96.29%

Table 6.14 Predictive characteristics for UCCBR. **data-set**= data-set name; **wm** = window model; **ws** = window size. θ = evidence threshold; **P** = number of predicted alerts; **A** = number of anticipated alerts; **PR**= preemptive ratio.

were not considered within the Ceaseless CBR processes anymore. We observed a noticeable increase in the time elapsed specially by Ceaseless Retrieve and Ceaseless Reuse as a consequence of the higher number of case activations considered at each iteration. In the case of Ceaseless Retrieve the time varied from hardly 10 milliseconds in Rustoord data-set up to almost a second in Naxpot data-set. In Huckleberry the difference was practically inappreciable in part due to the high number of burst situations that were directly prioritized independently of the measure of urgency. In the case of Ceaseless Reuse the differences are smaller since, in spite of the higher number of case activations, explanations are made up of a reduced number of hypotheses (only those that explain the most urgent alerts). We observed a substantial improvement in the preemptive ratio and weekly alert load reduction produced by UCCBR. Table 6.14 shows the results obtained for the above trials. Comparing Tables 6.11 and 6.14 it can be seen that UCCBR improved the preemptive ratio in Rustoord data-set in almost a 5%, from approximately 78% to 83%. In the two trials for the Naxpot data-set the improvement was of the order of 12% and 14% respectively, whereas is Huckleberry data-set it was of the order of 3%. The explanation for this higher rate of successful predictions is the fact that the measure of urgency only considered for explanation those case activations with a high degree of belief. This had two consequences. On the one side, it reduced the number of alert predicted and, on the other side, it increased the success ratio. Finally, UCCBR increased the weekly alert load in the order of the 2% in Rustoord data-set, in the order of 8% in Naxpot data-set, and in the order 3% in Huckleberry data-set. This improvement was in part due to the number of alerts that were prioritized by default because they never turned to be urgent.

6.3.8 Conclusions

Our evaluations demonstrate how a Ceaseless CBR-enhaced IDS system improves both the number of alerts that has to be managed by a SSO and the speed with which they could be triaged. As we have seen, Ceaseless CBR is able to significantly reduce the weekly alert load while keeping the number of false negatives very low and an admissible rate of false positives. Ceaseless CBR is able to keep the true positive rate over and above 99% and the false positive rate under and below 1%. This level of performance demonstrates that Ceaseless CBR can perform sufficiently for real world deployment.

Notice however that:

- on the one hand, the capability of Ceaseless CBR to 'sense' the presence of all occurring malicious attacks depends on the ability of the underlying IDS to detect suspect packets and evoke the corresponding alerts; and
- on the other hand, the capability of Ceaseless CBR to 'specify' that an attack is not present depends on either the ability of the underlying network model of knowing its own vulnerabilities or their capability of the SSO of correctly assessing the malignancy of the sequential cases proposed by Ceaseless CBR.

That is, the above rates reflect the capability of Ceaseless CBR to prioritize correctly both those alerts that really constitute a threat for the system under surveillance and those that bring no peril at all. We have also seen the capability of Ceaseless CBR to discover new sequential cases both creating them from scratch and adapting previously existing ones (discovered in former iterations). This turns out to be an additional advantage of our approach since it does not require to be provided with an initial sequential case-base. That is, Ceaseless CBR automatically acquires sequential cases [VC93] on-the-fly as the analysis of the sequence of intrusion alerts proceeds (i.e., in only one pass over the alert stream). Ceaseless CBR provides an acceptable preemptive ratio. It went up to a 78% in Rustoord data-set. We have also seen how the capability of Ceaseless CBR to predict the occurrence of future alerts can significantly be improved using a measure of urgency. In Naxpot data-set this improvement up to 14% . Moreover, we have achieved significant reductions in the weekly alert load. We got reductions up to a 95.90% in Rustoord data-set, to 80.89% in Naxpot, and to 93.02% in Huckleberry data-set. In addtion, we substantially improved those reductions using our measure of urgency in up to a 8.54% in Naxpot data-set.

Furthermore, through our experiments we have tried to determine the influence caused by distinct window models, window sizes, and minimum evidence thresholds in the distinct performance characteristics that we have used to measure the overall goodness of Ceaseless CBR. After our experimental evaluation of Ceaseless CBR we can conclude that:

- The performance of Ceaseless CBR is affected by the size of the window as well as by the value of minimum evidence threshold employed. Concretely, the smaller the window, the higher the t-area; and the higher the evidence threshold, the higher the t-area.
- The number of sequential cases discovered by Ceaseless CBR is affected by both the window size and θ. We say that the smaller the window, the bigger the number of sequential cases found; and the higher the minimum evidence threshold, the lower the number of sequential cases found. We also observed that the number of alerts per case decreases with the size of the window but seemed not to affected by θ. We also noticed that the number of episodes found does vary neither with the window model nor with θ but that the smaller the window size the higher the number of occurrences found and that the higher

the value of θ the lower the number of occurrences per case detected. Finally, the higher the evidence threshold, the lower the number of questions sent to the user; and the smaller the window, the higher the number of questions.

- The total time elapsed by Ceaseless CBR decreases in function of the number of alerts per window and increases in function of θ.
- The number of alerts correctly predicted by Ceaseless CBR (preemptive ratio) is affected by both the number of alerts per window and the minimum evidence threshold. The smaller the size of the window, the higher the preemptive ratio; and the higher the evidence threshold, the higher the preemptive ratio.
- The alert load reduction is only influenced by the window size. The higher the window size, the lower the alert reduction.
- A measure of urgency seems to contribute positively to the preemptive ratio of Ceaseless CBR. A smaller number of alerts are predicted with a higher success ratio. Urgency also seems to have a positive impact on the alert load reduction in part because many alerts never become urgent and are prioritized as non-malicious by default when become deprecated. A measure of urgency also achieves a significant reduction in the number of false positives. However, as we have seen the number of false negatives can increase and therefore depending on the specific cost conditions would be recommendable or not to use a measure of urgency. In other words, we have to use it cautiously since not always the negative impact of all possible undesired situations can be estimated in advance. However, if the damage costs are known a priori or can be estimated based on past experiences then is highly recommendable to use it to improve the overall performance of Ceaseless CBR. On the other hand, using a measure of urgency makes the Ceaseless Retrieve and Ceaseless Reuse processes longer as a consequence of the higher number of case activations considered at each iteration.

Our experimental results illustrated the efficiency and scalability of our approach across scenarios of varying difficulty. Finally, notice that while our experiments have been conducted using data stemming from a specific network IDS (i.e., Snort) our results can be extrapolated to other IDSes. Our alert model introduced in Chapter 3 guarantees the independence between our techniques and the underlying sensor used.

In this Chapter, we have provided a formal framework for the evaluation of alert triage systems that we have used to evaluate the performance of Ceaseless CBR. Moreover, we have also assessed Ceaseless CBR with respect to its ability to discover new cases, predict future alerts, and reduce weekly alert overload with varying parameters. Two remarkable results of our approach are, first, the ability to summarize the information sent to the network administrator and, second, the significant reduction on the alert overload. For example, in Rustoord experiments around 175 cases (more or less depending on the trial) summarized completely an alert stream made up of 31483 alerts and the weekly alert load was reduced up to a 97.14%. Our results show

that improvements using the Ceaseless CBR paradigm are reasonable and might find good use in practice.

7
Application

This Chapter brings the discussion down to earth, describing the architecture of a first prototype of an autonomous agent called Alba tasked with alert triage that employs the new methods proposed throughout the thesis to assist a network administrator's decision making. Initially, Section 7.1 provides a big picture of Alba architecture. The underlying network sensors as well as the rest of the IDS machinery needed to support Alba is explained in Section 7.2. Alba's architecture is sustained by two main components. First, a domain-specific language called SOID (Simple Ontology for Intrusion Detection) that facilitates the representation of the environment and contextual information in which the IDS operates and enables the use of the automated problem solving paradigms proposed. Indeed, SOID constitutes an instantiation of the formal alert model proposed in Chapter 3 on top of the Noos knowledge representation language [AP96]. SOID is described in Section 7.3. Second, an overlay management system that wraps a conventional IDS up with the knowledge and reasoning capabilities to properly evaluate the extent of threats. We explain the different knobs and switches of this system and the technology used to construct it in Section 7.4.

7.1 ALBA OVERVIEW

To demonstrate the applicability of our methods we constructed a first prototype of an agent-aided intrusion detection tool called Alba (ALert BArrage) that assists a

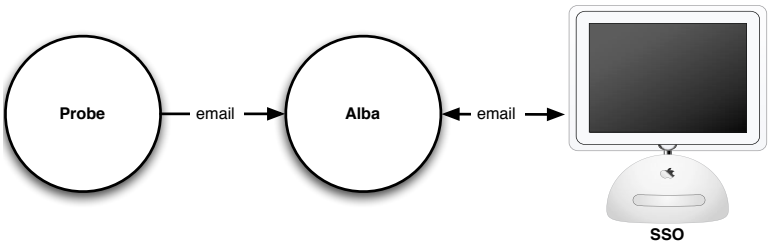


Fig. 7.1 Alba placement. Alba mediates between a conventional IDS or probe and the corresponding SSO.

network administrator's decision making reducing the burdensome output produced by a conventional intrusion detection system (IDS). [1]

As we already pointed out in Section 2.7, Alba can be catalogued as an aggregation and correlation component (ACC) [DW01, HRT03]. Alba aims at producing high-level reasoning beyond low-level probes capabilities. Alba mediates between the alarm element—such as is considered in a generic architectural model of an IDS—and the corresponding SSO (see Figure 7.1). Alba aims at performing the alert management task on behalf of her SSO. Specifically, Alba aims at reducing the number of false positives due to *innocuous* attacks and to increase the predictive power for *harmful multi-stage* attacks. Alba emits a judgment on the malignancy of each alert on an IDS alert stream. Such judgment determines whether an alert is finally notified or not to the network administrator. Additionally, Alba groups together alerts corresponding to multi-stage attacks and burst situations minimizing the number of required interactions with her SSO's. The development of Alba posed two main problems: (i) a useful level of description is required to automate a major part of the alert management task; and (ii) and efficient and robust method to rapidly interpret the alert stream is required.

We overcome the first problem introducing an ontology for intrusion detection that allowed us to comprehensibly represent a number of knowledge sources needed to properly interpret alerts [MP03e]. The purpose of this ontology is to provide a mechanism that allows a sequence of alerts (alert stream) provided by a conventional IDS not only to be readable but also understandable by a software agent. Ontologies provide a way of capturing a shared understanding of terms that can be used by humans and programs to aid in information exchange. Unfortunately, currently there is no common ontology that allows computer security incidents to be conceptualized in a standardized way. Neither does there exists a widely accepted common language for describing computer attacks. Consequently, the first and foremost step in the development of Alba was to construct an ontology that provides a comprehensive understanding of security incidents concepts. We have called that ontology SOID

[1] This tool with all its particularities should be conceived as a proof of concept and never as software ready for deployment.

that stands for Simple Ontology for Intrusion Detection. SOID establishes well-defined semantics that allow Alba to process information consistently and reason about an IDS alert stream at a higher level of abstraction, facilitating in this way the automation of the alert management task [MP03e]. As we see later on in Section 7.3, SOID constitutes an instantiation of the formal alert model proposed in Chapter 3 on top of the Noos knowledge representation language [AP96].

The second problem was solved constructing a software agent—an overlay system that wraps a conventional IDS up with the knowledge and reasoning capabilities to properly evaluate the extent of threats. We also refer to this software agent as Alba. Concretely, Alba maintains a number of models constantly updated by means of a collection of sensors and monitors and embodies Ceaseless CBR within her inner logic. Figure 7.2 sketches the architecture that provides the primitive resources that allows Alba to reason, learn, accept direction and explain itself meaningfully. In the next subsections, we provide a brief description of the three layers that conform Alba's architecture.

7.1.1 Perception Layer

The first layer provides, on the one hand, a collection of sensors (i.e., the inner IDS) strategically placed to continuously monitor and analyze every packet on the protected network, and, on the other hand, a number of scanners and monitors that enable Alba to keep several sources of knowledge (that we will introduce later on)

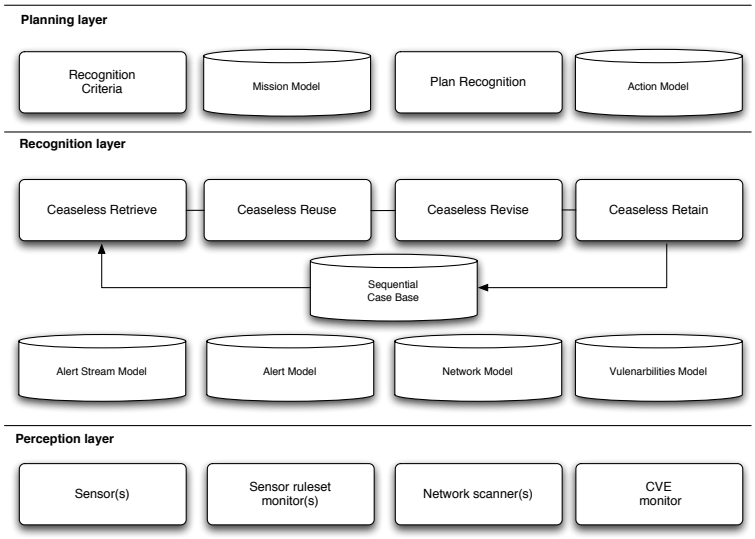


Fig. 7.2 Overview of Alba multi-layered architecture.

constantly updated. This layer allows Alba to perform additional preventive tasks such as pinpointing security weaknesses for correction.

7.1.2 Recognition Layer

The second layer provides Alba with deliberative capabilities. First, a collection of models—expressed on top of the concepts defined by SOID—allows Alba to reason about security incidents, vulnerabilities, alerts, and the protected network. Second, a sequential case base permits Alba to store sequential patterns of both malicious and innocuous activity. Those patterns are represented by means of actionable trees. Third, Alba keeps an updated profile of the alert stream (alert frequency, rareness, etc) and uses Ceaseless CBR for continuously overseeing the alert stream looking for an explanation for each group of alerts so that they can be conveniently prioritized. Alerts that have already been prioritized as well as predicted alerts are sent to the planning layer that finally decides on the convenience of notifying the SSO or initiating a proper action to repel an attack.

7.1.3 Planning Layer

The third layer provides Alba with reflective capabilities. First, a model of the network mission and costs allows Alba to make savvy judgements on the priorization of certain malicious alerts as well as to keep the number of false positives under control. Second, a plan recognition model uses priorized alerts and predicted alerts to properly anticipate the plans of a malefactor and initiate the corresponding plan of countermeasures using a collection of prespecified actions (e.g., creating a new rule in a firewall, closing a specific connection, etc). Third, Alba accepts directions from a network administrator or SSO who can provide Alba with different criteria to decide when to initiate an action instead of simply notifying. The network administrator also governs the parameters that indicate how to estimate the similarity of the current sequence of alerts with previous patterns (e.g., window model, window size, minimum evidence, etc) and establishes what constraints are of interest (e.g., IP source and IP destination) to properly prune the discovery of new sequential cases. Next, we describe the conventional IDS that feeds Alba with alerts.

7.2 THE INNER IDS

The conventional IDS that we have used is an open source architecture based on Snort as the key component [Hil01]. Other secondary components are: a MySQL database[2],

[2] dev.mysql.com

the analysis console for intrusion databases (ACID)[3], a number of PHP[4], procmail[5], and Perl scripts www.perl.org, and an Apache web server www.apache.org. We got running the same software platform into a number of Red Hat Linux boxes and into several Mac OS X's BSD-based Darwin machines. Recently a burst of new books has popularized this open source architecture for intrusion detection that we selected time ago [CBF03, Koz03, Reh03]. Those books describe its different features far better than we could ever have done here. Thus, we only provide a succinct description of its main components.

7.2.1 Snort

Snort is an open source network intrusion detection system supported by a large community[6]. Snort performs lightweight real-time traffic analysis and packet logging on IP networks [Roe99]. Snort has been compared and used together with other signature-based IDSes and has always provided excellent results [AFV02]. Basically, Snort is a signature-based IDS composed of a sniffer, a collection of preprocessors, a detection engine and a collection of postprocessors. Snort is configured using a configuration file and a collection of files containing rules (signatures) used by the detection engine. Snort sniffer acquires traffic from a network link using libpcap[7]. Then, for each network packet a number of decoder routines fill out the packet structure from link level protocols up to higher level details such as TCP or UDP ports. Packets are then processed by a number of registered preprocessors that check different features. Finally, a detection engine checks every packet against the active Snort rules as established in the configuration file. A collection of output postprocessors indicate how alerts have to be signaled. A large collection of Snort rules has been published. At the time of finishing this work there were available more than 1800 stable rules. Snort rules are classified in twenty three different classes. We have define in Noos a sort for each one of these classes. A Snort rule is composed of a Snort identification number (SID), a message, a signature and references. Moreover, for a few rules a summary, impact, more detailed information, attack scenarios, easeness of attack, false positives, false negatives and corrective actions are also provided. We developed a Perl script to automatically translate Snort rules into Noos domain knowledge. Figure 7.3 shows an example containing a couple of Snort rules.

7.2.2 Alert Database

We set up Snort to log in to a MySQL database and installed ACID (Analysis Console for Intrusion Databases) for alert browsing and management. ACID provides a web

[3] acidlab.sourceforge.net
[4] www.php.net
[5] www.procmail.org
[6] www.snort.org
[7] www.tcpdump.org

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"WEB-IIS CodeRed v2 root.exe access";
flow:to_server,established; uricontent:"/root.exe";
nocase; classtype:web-application-attack;
reference:url,www.cert.org/advisories/CA-2001-19.html; sid:1256; rev:7;)

alert udp $EXTERNAL_NET any -> $HOME_NET 1434
(msg:"MS-SQL Worm propagation attempt";
content:"|04|"; depth:1; content:"|81 F1 03 01 04 9B 81 F1 01|";
content:"sock"; content:"send";
reference:bugtraq,5310; classtype:misc-attack; reference:bugtraq,5311;
reference:url,vil.nai.com/vil/content/v_99992.htm; sid:2003; rev:2;)
```

Fig. 7.3 Example of Snort rules: WEB-IIS CodeRed v2 root.exe access (Snort signature SID=1256) and MS-SQL Worm propagation attempt Snort signature (SID=2003).

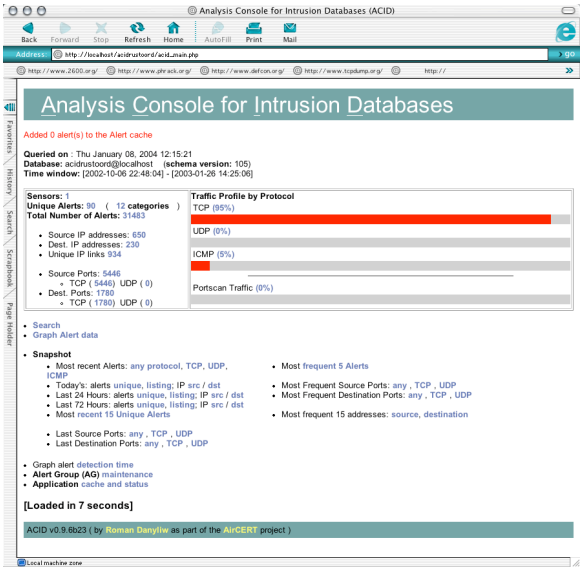


Fig. 7.4 ACID console for Rustoord dataset.

interface (a set of PHP scripts) to data stored by Snort[8]. Figures 7.4, 7.5, and 7.6 show what ACID interface looks like. That interface provides a number of features such as database searching, browsing and decoding of layer 3 and 4 logged packets, alert management, and exporting to e-mail messages.

[8]To run the experiments described in Chapter 6 we dumped the corresponding MySQL database into a comma separated value (CSV) file format that was directly imported from MATLAB.

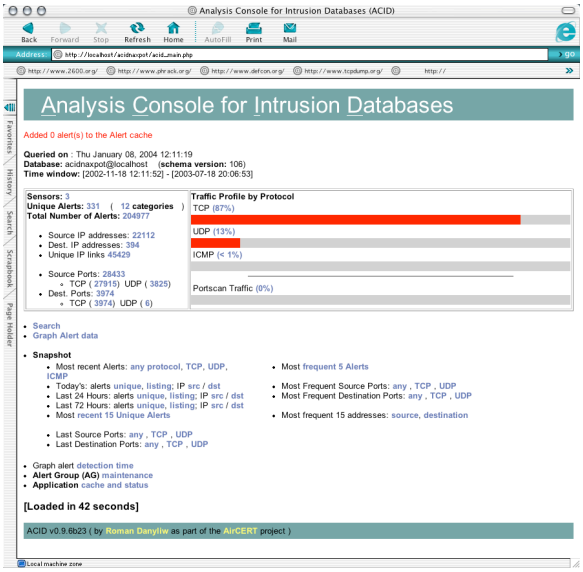


Fig. 7.5 ACID console for Naxpot dataset.

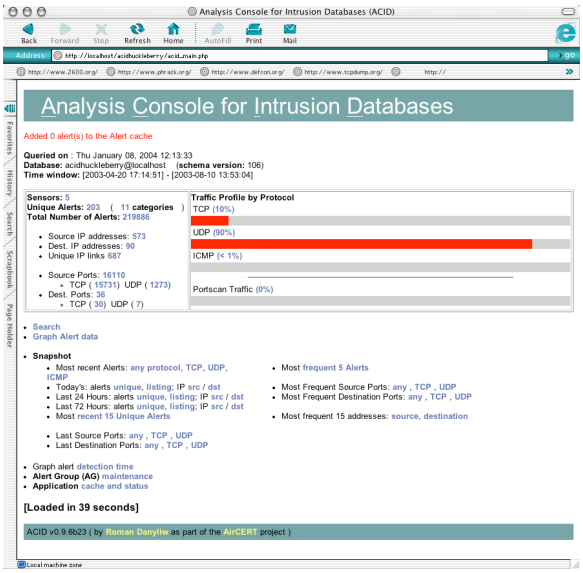


Fig. 7.6 ACID console for Huckleberry dataset.

7.2.3 Sending Alerts to Alba

We used email sending ACID feature as the mean to send alerts to Alba. A cron process periodically queries ACID database and sends an email containing new alerts to Alba. Alba uses a procmail preprocessor to handle her incoming email. This procmail preprocessor uses a Perl script to translate each alert in the body message into a Noos term (see Section 7.3.7) and forwards the new translated content to Alba. Alba checks for new emails on a regular basis. New alerts are streamed according to a pre-specified window model and sent to a Ceaseless CBR process that finally analyzes them. The whole process allows Alba to consume alerts in near real-time. Next Section describes SOID in further detail.

7.3 SOID

We refer to the term ontology as a formal specification of a vocabulary of concepts and the relationships among these concepts that provides a machine readable set of definitions that in turn create a machine understandable taxonomy of classes and subclasses and relationships between them. Several methodologies for building ontologies have been proposed [Gru93, MF95, MAN97, UG96, JBV98, MAN97]. See [Fer99] for a comparative of diverse methodologies. Nevertheless, ontological engineering[Pre97] is still immature compared to knowledge engineering and therefore a widely accepted methodology for this purpose cannot be found yet[SBG00]. To build our ontology we have followed the V-model methodology proposed by [SBG00]. This methodology was inspired by the software engineering V-process model [Oul90] and has been used for instance to build ontologies in fields such as bioinformatics [SBG00].

7.3.1 Purpose and Conceptualization

SOID aims at providing a domain-specific representation language for alert triage in intrusion detection. At a quick glance, in order to automatize the alert management task we have identified four key sources of knowledge to be conceptualized (See Figure 7.7): *networks*, *incidents*, *vulnerabilities*, and *alerts*. We built a separate ontology for each of them using the knowledge representation language Noos [AP96]. Finally, we have merged these partial ontologies in a more global ontology that we have called SOID—a Simple Ontology for Intrusion Detection [MP03e]. We did not start to build each sub-ontology from scratch. Instead, each sub-ontology was built starting from previous works such as the Network Entity Relationship Database [GHH01b], the Common Vulnerabilities and Exposures dictionary, the Common Language for Computer Security Incidents [HL98], and Snort ruleset [Roe99]. Previously, we consciously analyzed a collection of related works bearing always in mind simplicity as the driver of our approach. Next, we overview each one of the above-named key sources of knowledge.

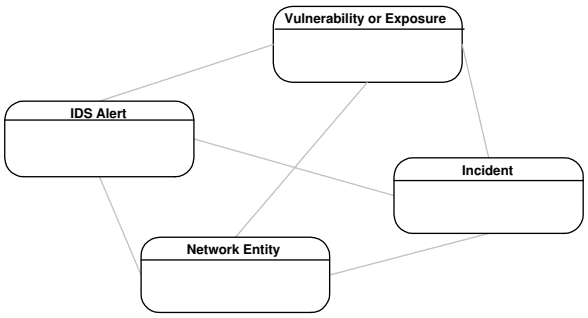


Fig. 7.7 SOID sub-ontologies: Networks, Incidents, Vulnerabilities, and Alerts

7.3.2 Networks

A network is the computer system to be protected. We have defined a set of concepts and relationships to model a network based on the Network Entity Relationship Database (NERD) proposed by Goldman et al [GHH01b]. Properly modelling the network allows the importance of each alert to be correctly assessed. That is to say, determining whether a given alert implies or not a peril for the network under protection requires a specification of the network and the implications of each alert as well. Remember the relative environment-dependency named in Section 1.2.2 and the Code-Red Worm case mentioned in Example 1. Network models based on SOID can easily be coded into Noos and automatically updated translating the reports provided by network scanners such as Nessus, Satan, or OVAL.

7.3.3 Incidents

An incident is a unauthorized use or abuse of the protected system. We have followed CLCSI [HL98] that defines an incident taxonomy based on three key concepts: *events*, *attacks* and *incidents*. An event is an *action* directed at a *target* which is intended to result in a change of state of the *target*. An attack is defined as a sequence of actions directed at a target taken by an *attacker* making use of some tool exploiting a computer or network vulnerability. Finally, an incident is defined as a set of attacks carried out by one or more attackers with one or more goals. Figure 7.9 shows some of CLCSI concepts represented in Noos.

7.3.4 Vulnerabilities

A vulnerability is a flaw in a target that could allow an unauthorized result. Knowing the vulnerabilities in our network is the main source of knowledge to automatically decide if a given alert corresponds to an innocuous attack or not. We have incorporated common vulnerabilities and exposures (CVE) dictionary provided by the MITRE corporation into our ontology. CVE provides a name and a standardized

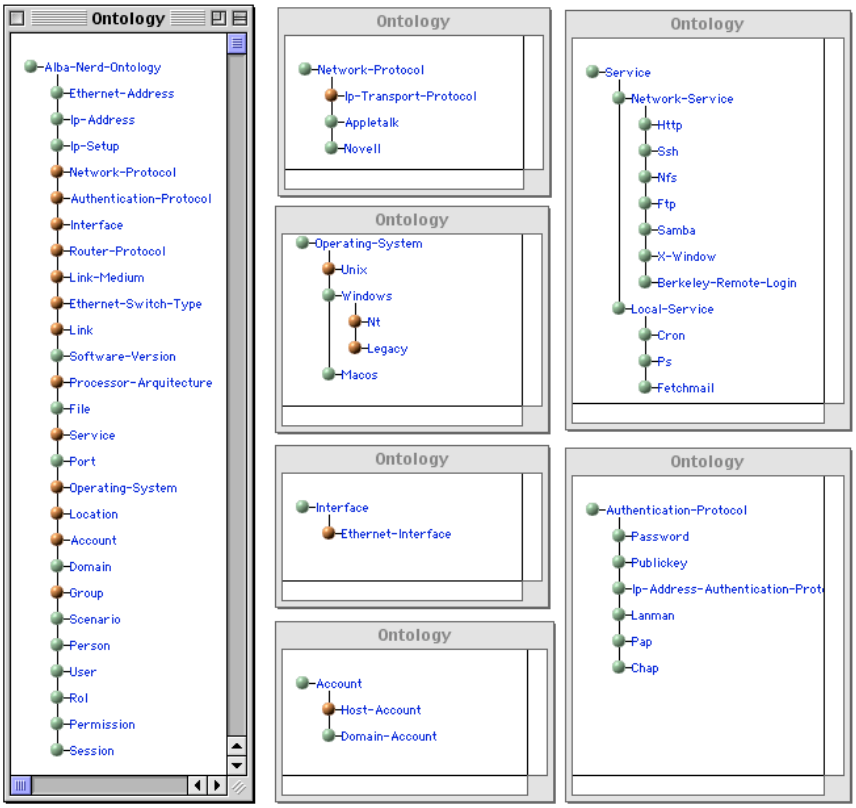


Fig. 7.8 NERD ontology represented in Noos.

description for each vulnerability or exposure. The aim of CVE is to allow disparate intrusion detection tools to interoperate and share information. Other well-known source of knowledge about vulnerabilities is ICAT, a CVE based searchable index of information on computer vulnerabilities. It provides search capability at a fine granularity and links users to vulnerability and patch information. A monitor advertises Alba of new published vulnerabilities. Alba contrasts new vulnerabilities against the network model (NERD) and pintpoints new security weaknesses for correction.

7.3.5 Alerts

We have conceptualized alerts according to the Snort ruleset [Roe99]. Snort alerts are classified in twenty three classes. Figure 7.11 shows the SOID sorts for representing them. Each alert is provided with a unique sequential identifier, time and date, sensor identifier, the signature that triggered the alert, IP and either a TCP, UDP, or ICMP header and payload. Additionally, some references to sources of informa-

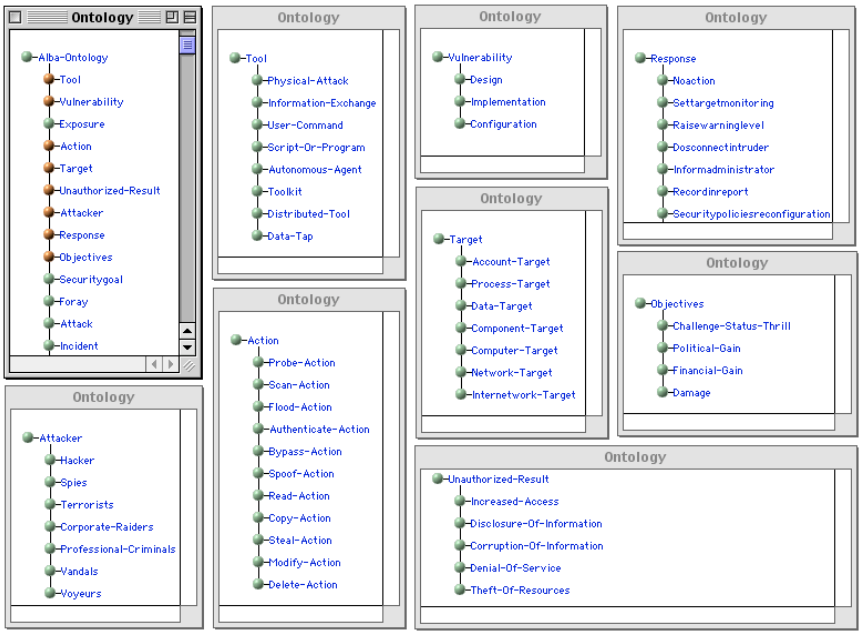


Fig. 7.9 Some of the SOID concepts represented in Noos.

```
   #(1 - 12064) [2002-11-29 18:47:22]  WEB-IIS CodeRed v2 root.exe access
   IPv4: 80.34.49.201 -> 172.26.0.4
hlen=5 TOS=0 dlen=112 ID=16914 flags=0 offset=0 TTL=118 chksum=37996
   TCP:  port=3421 -> dport: 80  flags=***AP*** seq=1955827854
ack=1657159142 off=5 res=0 win=17520 urp=0 chksum=44219
   Payload:  length = 72

   000 : 47 45 54 20 2F 73 63 72 69 70 74 73 2F 72 6F 6F   GET /scripts/roo
   010 : 74 2E 65 78 65 3F 2F 63 2B 64 69 72 20 48 54 54   t.exe?/c+dir HTT
   020 : 50 2F 31 2E 30 0D 0A 48 6F 73 74 3A 20 77 77 77   P/1.0..Host: www
   030 : 0D 0A 43 6F 6E 6E 6E 65 63 74 69 6F 6E 3A 20 63   ..Connnection: c
   040 : 6C 6F 73 65 0D 0A 0D 0A                           lose....
```

Fig. 7.10 CodeRed Worm propagation attempt.

tion about the attack are also provided. In Fig. 7.10 an alert corresponding to an attempt of propagation of the CodeRed worm is shown. This alert corresponds to the first signature shown in Figure 7.3. Figure 7.13 depicts the same alert such as it is represented by Noos.

7.3.6 Automatic, Up-to-date Model Keeping

To keep updated the several models that Alba employed, we used a collection of scripts to monitor the sources of information described above. Once one of these

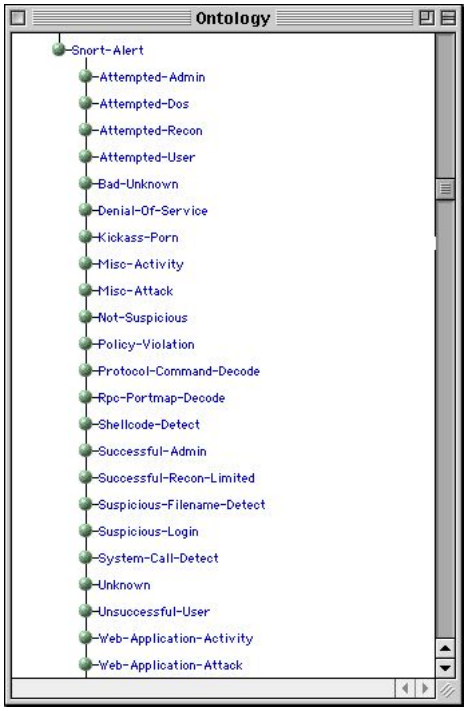


Fig. 7.11 Sorts for Snort Alerts

scripts detected a change, it automatically translated the new information into Noos. Since CVE is provided in plain text we could easily translate it into Noos using a Perl script. Likewise, we created a Perl script to automatically translate Snort rules into Noos. Thus, we were able to keep SOID updated at the same pace that new vulnerabilities and signatures were published. Figure 7.12 sketches how we automatically kept up-to-date the distinct sources of knowledge that SOID integrates.

SOID has been coded using the Noos representation language that we oveview in nex Subsection.

7.3.7 The Noos Representation Language

Noos is an object-centered knowledge representation language [AP96]. An object-centered system organizes knowledge around the unyfing abstract notion of object [AFG96b]. In an object-centered system objects are modeled using the following three basic constructs (that have received similar interpretations in different object-centered systems such as *object-oriented systems*, *description logics*, *entity-relation data models*, *frame-based systems*, or *semantic networks*) [Mac89, Rin89, AFG96b].

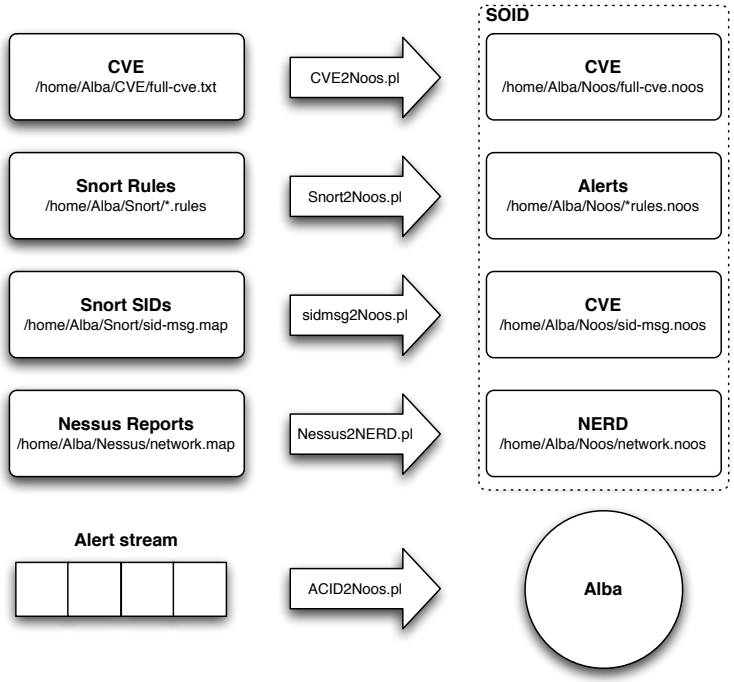


Fig. 7.12 Automatic, up-to-date SOID models keeping.

Sorts that describe the common features within a collection of terms. Sorts form a collection of partially ordered symbols. Sorts are also known as *classes*, *concepts*, *entities*, *frames*, or *nodes*.

Features that describe the relevant properties of terms. Features are also known as *link*, *properties*, *role*, *slot*, etc

Terms that describe the indivinduals of a domain. Terms are also known as *individuals* or *objects*.

Noos is useful for developing knowledge systems that integrate problem solving and learning [AMS98, Arm97]. Noos is formalized using *feature terms*. Feature terms are a generalization of first order terms and lambda terms. Feature terms constitute the Noos basic data structure and can be seen as extendable records organized in a *subsumption* hierarchy [AP96]. Feature terms are represented graphically by means of labeled directed graphs (see Figure 7.13). In Noos subsumption is defined as an informational ordering among feature terms. A feature term Ψ is subsumed by another feature term Ψ' when all information provided by Ψ' is also provided by Ψ. The three basic concepts that underpin the Noos knowledge representation language: *sorts*, *feature terms*, and *subsumption* also constitute the fundamental concepts on

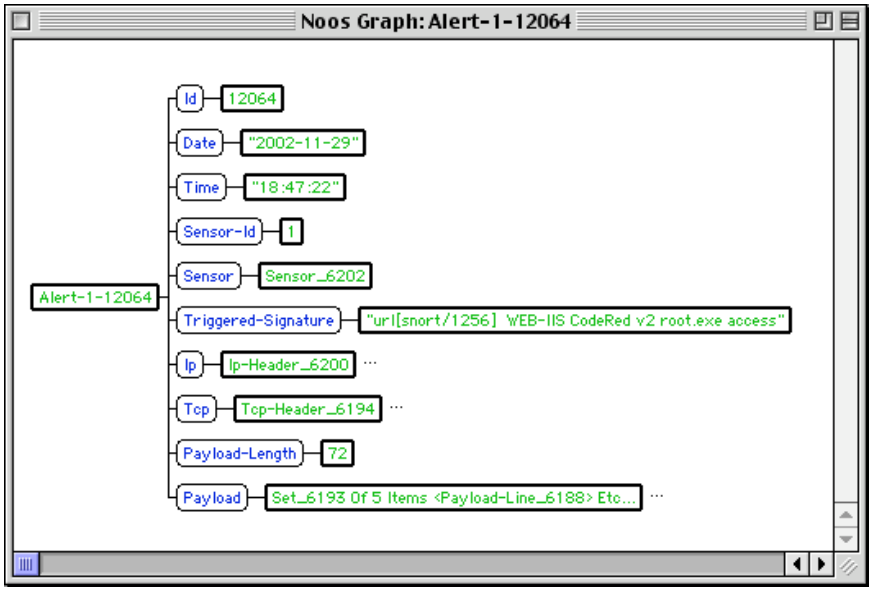


Fig. 7.13 Alert WEB-IIS CodeRed v2 root.exe access represented in Noos.

top of which we defined our alert model in Chapter 3 and sequential cases in Chapter 5. Noos not only offers representation constructs close to knowledge modeling frameworks, episodic memory and reflective capabilities but has also been upgraded with agent-programming constructs. We used these constructs to decompose Ceaseless CBR processes and to separate other concerns such as alert streaming or email sending and reading in our design such as we will see in next Section. The Noos interpreter is implemented on Common Lisp and runs on top of both Digitool Macintosh Common Lisp[9] (MCL) and the ANSI Common Lisp GNU CLisp[10] what covers a wide number of platforms.

7.4 ALBA, AN ALERT BARRAGE

Alba is an autonomous agent tasked with alert triage that employs Ceaseless CBR to assist a network administrator's decision making. Basically, the tasks that Alba addresses can be described as follows. Given an alert stream produced by a number of probes and a collection of models describing the target network under surveillance, Alba is to:

[9] www.digitool.com
[10] clisp.cons.org

1. discover new sequences of alerts that are correlated according to a number of user-defined constraints and proposed them to the user for their proper dangerousness assessment.

2. pinpoint multi-stage attacks that are undergoing, anticipating the occurrence of some of their alerts.

3. minimize the number of alerts that are really suceptible of being notified to her SSO.

The overall objective of Alba can be summarized as reducing daily alert load as much as possible while keeping both false negative and false positive ratios as low as possible minimizing in turn the number of interactions with her SSO to properly accomplish her tasks. Figure 7.14 depicts Alba general interaction scheme. Alba uses SOID to properly reason about the concrete network under protection. SOID merges and keeps up-to-date information about changes on the network protection, new published CVE vulnerabilities, and the corresponding Snort signatures. Alba receives by email sequence of alerts that have been previously stored into a MySQL database by a Snort output post-processor. Alba interacts with her SSO by email. Alba sends periodical reports on alert activity as well as prompt notifications on dangerous attacks and requests for new discovered cases supervision to her SSO. Alba receives her SSO's revisions also by email. Alba Notice that we did not aim at building an invulnerable architecture for Alba. Therefore, many issues should be addressed before deploying Alba into a mission critical scenario, which range from

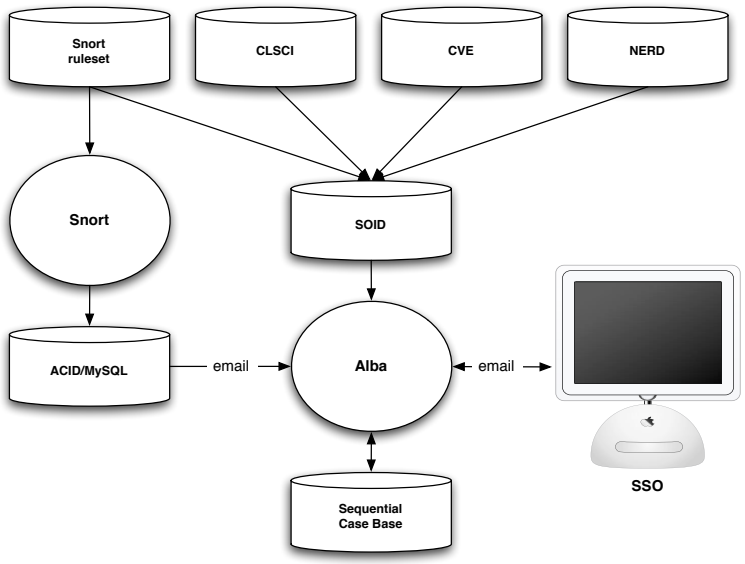


Fig. 7.14 Alba interaction model overview.

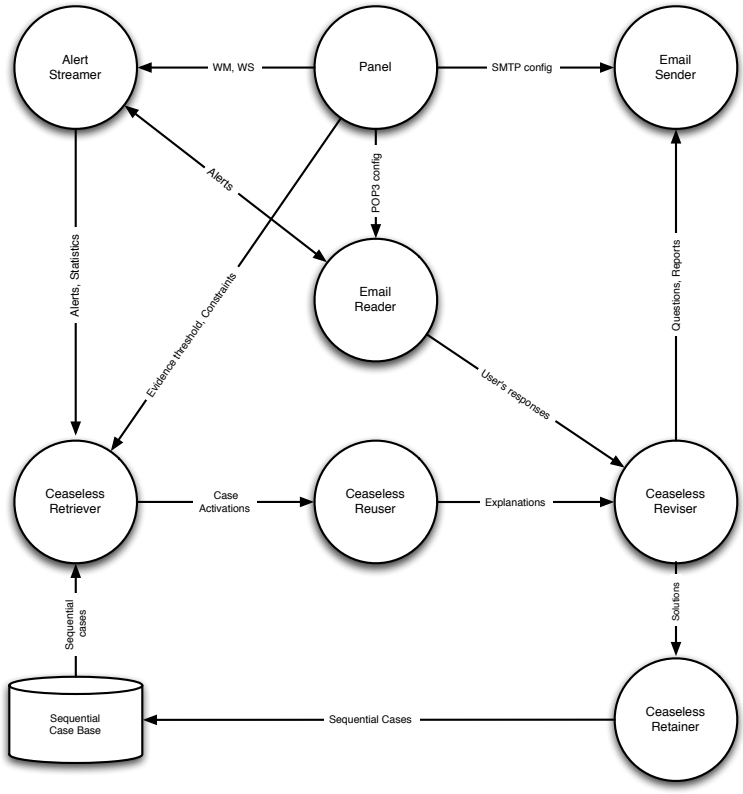


Fig. 7.15 Alba inner processes.

strengthening of core implementations to authentication and encryption of all critical interactions.

The Alba functional architecture consists of the following procesess (agents) that were implemented using Noos agent-programming constructs:

Panel provides a user-interface to Alba. Panel allows Alba to receive SSO input. Figure 7.16 shows what a simple Alba panel looks like. Alba provides a number of parameters that a SSO can use to control or adjust Alba's inner behavior such as incoming and outgoing emails accounts, window model and size, minimum evidence threshold, constraints for sequential case discovery, etc. Alba panel communicates with other processes to set their corresponding parameters.

Email Reader handles both emails with alerts sent by probes and emails from Alba SSO responding to a previously formulated question. Distinct email subjects allow Alba's Email Reader to easily forward messages to Alert Streamer or Ceaseless Reviser depending on whether the message contains new alerts or a response from the SSO.

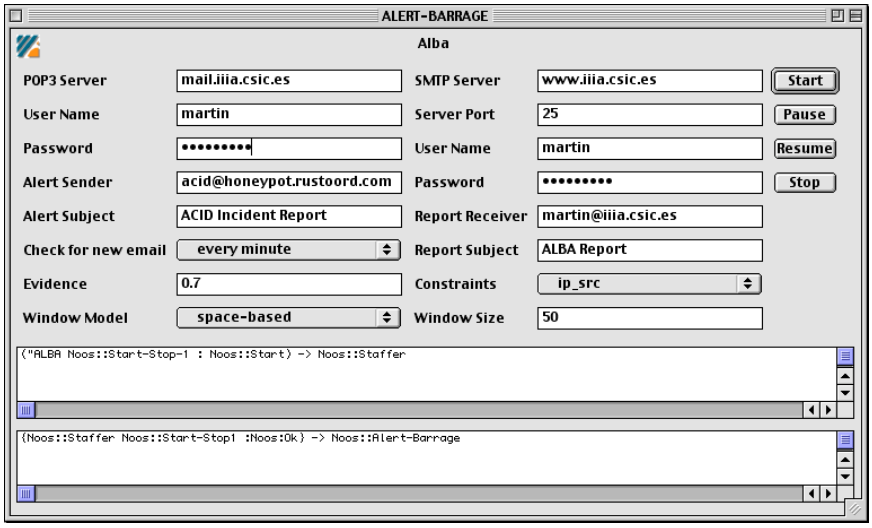


Fig. 7.16 Alba panel in Noos

Alert Streamer receives and interpret emails containing alerts from Email Reader. Alerts are received in Noos lingo since Alba uses a procmail pre-processor to conveniently translate them from ACID format such as we described above. Alert Streamer either groups alert together or divides them into small chunks according to the specified window model and window size by Alba's SSO through Alba's Panel.

Ceaseless Retriever upon reception of a new window of alerts from Alert Streamer seeks sequential cases that best match it. Ceaseless Retriever creates a number of cases of case activations to explain each and every one of the alerts in the window at hand. Cases activations that either follow the constraints indicated by the corresponding sequential cases or do not correspond to any sequential case but follow the default constraints expressed by the SSO are fused together. This process sends case activations to Ceaseless Reuser. This process and the three following ones behave on the basis of the Ceaseless CBR model explained in Chapter 5.

Ceaseless Reuser decides on which alerts should be prioritized right away and which should be prioritized later on according to the model of urgency explained in Chapter 5. This process could be also set up to always prioritize alerts as soon as they are received. The trade-off between performance and efficiency were analyzed in Chapter 6. Once Ceaseless Reuser selects which alerts to prioritize it composes a number of explanations that explain all selected alerts. The explanation with the highest degree of belief is selected and sent to Ceaseless Reviser for its revision.

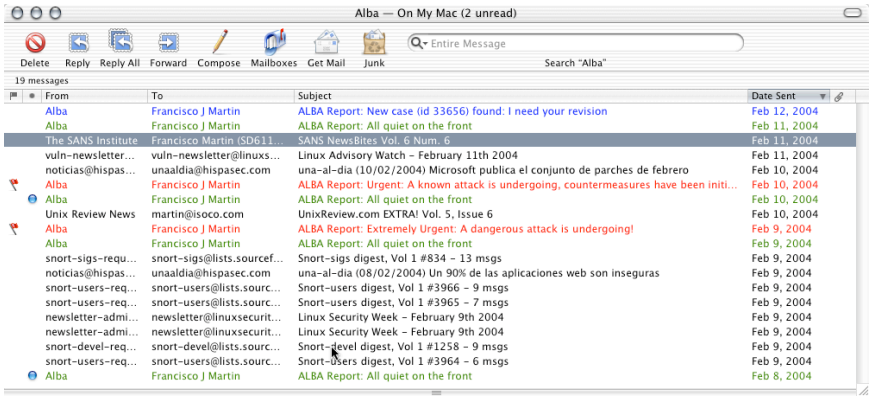


Fig. 7.17 SSO mailbox aspect.

Ceaseless Reviser receives an explanation for the current window of alerts and evaluates whether it can proceed without SSO supervision or not. A SSO can establish on what basis she wants to be informed or questioned before Ceaseless Reviser concludes on a final solution and initiates an action. For example, the SSO can indicate that all those partial explanations that correspond to innocuous attacks can be sent directly to Ceaseless Reviser without SSO intervention and supervise all new case adaptations for properly assessing their priority. The SSO can even delegate onto Ceaseless Reviser to initiate some actions upon the reception of some high-priority explanations. Further details about the concrete plan recognition model used by Alba is beyond the scope of this work. It constitutes one of the most important directions of our future work. Ceaseless Reviser communicates with Email Reader to send messages to Alba's SSO. Once a final solution has been chosen it is sent to Ceaseless Retainer who takes charge of storing it for subsequent reuse.

Ceaseless Retainer finally receives a chosen explanation from Ceaseless Reviser and conveniently updates the sequential case base. If the solution corresponds to a new case Ceaseless Reviser stores it into the sequential cases. However, if it corresponds to a previous sequential case it simply simply updates the former sequential case occurrences and statistics.

Email Sender interacts with Alba's SSO in a human readable form. Alba generates three kind of email message to its user: (i) daily reports containing those alerts of interest (whose priority is above a user-defined threshold); (ii) new discovered sequential cases for SSO revision; and (iii) emails signaling imminent peril that require an immediate response by part of Alba's SSO. Figure 7.17 shows what a daily SSO's mailbox looks like.

Summarizing we could say that Alba's main ability is to supress false alerts. Moreover, Alba helps to point out complex multi-stage attacks to security managers

and detect attacks that are still undergoing. In Chapter 6 we showed promising results regarding the alert load reduction that Ceaseless CBR can achieve while minimizing both the number of false negatives and the number of false positives.

This Chapter described a first prototype that allowed us to explore the companion machinery needed by a ceaseless case-based reasoner to solve real problems. The appropriateness of intrusion detection alert triage for evaluating the significance of Ceaseless CBR comes from the compelling challenges that an application in this domain faces. As a matter of fact, research in the area of intrusion detection systems is receiving much attention from academia, industry, and government. Since these systems pose many complex issues, there are many open problems for research and opportunities for making significant contributions. Notice, however, that the objective of the application described was to improve the efficiency of current IDSes rather than providing new substantial detection methods.

8

Conclusion and Future Work

This Chapter summarizes the contributions of the thesis, benefits of this research, limitations of the proposed approach, and speculates about profitable directions for future research. In Section 8.1 we will review the five major directions along which our research has evolved: (i) development of a representational structure of sequential cases that support part-of and temporal dependencies; (ii) specification of a dynamic sequence similarity; (iii) an efficient CBR inference that exploits sequential cases and probabilities; (iv) a validation framework for alert triage; and (v) a novel agent-aided intrusion detection tool. In Section 8.2, we succinctly describe three of the main lines for future work. Firstly, the extension of the basic components offered by our model to facilitate more elaborate problem solving tasks such adversarial plan recognition. Secondly, the evolution of our model to contemplate automated monitoring agents able to communicate and coordinate with each other, enabling in turn higher-level collaboration. That is to say, a multi-agent approach where Ceaseless CBR agents cooperate each other to solve more complex problems (e.g., problems that are inherently distributed) or to improve its individual competence and performance. Ceaseless CBR agents could share information at different levels: ranging from observational data or case activations to explanations and sequential cases. We have previously investigated techniques and mechanisms that lay the groundwork of this future line. For example, DistCBR and ColCBR [PAM96, PAM97, MPA99], conversation protocols [MPR00b], interagents [MPR98, MPR00a], etc. Thirdly, we will investigate how completely delegate problem determination tasks to the own computer system. That is to say, how to evolve our techniques to become part of autonomic problem determination tools with self-healing capabilities.

8.1 CONTRIBUTIONS

In a nutshell, the purpose of this thesis was to investigate a new Case-Based Reasoning (CBR) model for the analysis of unsegmented sequences of observational data stemming from multiple coincidental sources in complex multi-agent settings. We have called this new model Ceaseless CBR.

We argued that there are a number of real-world domains, such as intrusion detection, international event analysis, fault diagnosis, telecommunications alert management, etc, that require the on-line analysis of temporally-evolving sequences of observational data. We pointed out that most CBR systems are unable to deal with the sequential behavior of real world domains and the simultaneous occurrence of interleaved problems. Since they presuppose individualized problem descriptions with well-specified boundaries that encompass all the information needed to solve the current problem in only "one shot". That is to say, the non-coincidental sources, full-fledged problem descriptions, and individual case independency assumptions on which most CBR systems are built make impracticable their direct deployment in the aforementioned domains. In this thesis we have demonstrated that CBR can be used in real-world domains where those assumptions cannot be met. We have shown how the CBR paradigm can be extended to handle efficiently situations that are represented by in terms of an unsegmented, sparse sequence of complex objects that arrive continuously over time. We think that we have successfully answered the original question formulated in Chapter 1 and achieved our objective: To enhance the CBR paradigm to support the analysis of unsegmented sequences of observational data stemming from multiple coincidental sources. Chapter 2 placed the thesis within the larger research landscape and served to review in summary form much of the relevant material from the literature. We showed how our research advances current state-of-the-art in CBR along several dimensions. Additionally, our work has also adapted and integrated seamlessly a variety of methods from distinct fields of study, such as knowledge representation or decision theory. Nonetheless, we think that we have only given the first steps to develop a better understanding of what we have coined as *case-based sequence analysis*. Thus, we say that we have only established a first model, called Ceaseless CBR, for this fascinating undertaking. As we will see later on in this Chapter, a number of issues still remain open that will surely lead us to some further interesting research. We performed our research bearing in mind the construction of a long-lived and highly autonomous agent endowed with advanced reasoning capabilities and entrusted with mission-critical real-time decision-making tasks in *dynamic*, *imprecise*, and *adversarial* environments [MPR00a]. Concretely, we have addressed a very challenging domain, intrusion detection alert triage. We have avoided to introduce a large amount of constraints and assumptions that would have simplified our problem into a more toy-like one. We have gone beyond theoretical work and based on our new model, we have developed a first prototype of a practical application: an agent-aided intrusion detection tool called Alba (Alert Barrage) that assists a network administrator's decision making. We have also shown experimentally the performance benefits of our approach. The alert reduction achieved has a direct and

measurable impact on manpower requirements needed to handle the unmanageable output provided by current IDSes.

Our research has evolved along five major directions: (i) development of a representational structure of sequential cases that support part-of and temporal dependencies; (ii) specification of a dynamic sequence similarity; (iii) an efficient Ceaseless CBR inference that exploits sequential cases and probabilities; (iv) a validation framework for alert triage; and (v) a novel agent-aided intrusion detection tool. We list below some of the major contributions of this thesis.

8.1.1 Sequential Cases

We have addressed the conceptual and representational issues concerned with the knowledge structures that allow a case-based reasoner to analyze unsegmented sequences of complex objects that stem from automated sensors. One of the issues that we were mostly concerned with was the hierarchical structuring of sequences by means of compositional hierarchies. Our contribution in this aspect was threefold: an alert model based on feature terms; actionable trees as predictive compositional hierarchies for representing sequential cases; and SOID, a simple ontology for intrusion detection.

First, as a part of this research we have created an unified data model to represent alerts that are triggered by automated real-time systems that collect and interpret sensor data in real-time and specially intrusion detection alerts. This has been a challenging undertaking due to its inherent complexity. The alert model that we have introduced allows us to keep our techniques independent of particular alert devices. This model provides a taxonomic hierarchy that allow us to work at a higher level and use abstraction to address several issues such as the generation of abstract cases that we saw in Chapter 5. This alert model uses feature terms as the underlying formalism. This alert model was mostly described in Chapter 3 where we also presented many other definitions that form the foundations of our approach.

Second, we have introduced the notion of sequential case. A sequential case is a compositional case where additionally a temporal order is established among their parts. We have proposed to represent sequential cases by means compositional hierarchies. Particularly, we have introduced *actionable trees* to represent sequences of complex objects and reason in terms of partially observed sequences. An actionable tree is a a Multi-Rooted Acyclic Graph (MRAG) with the semantics that (i) roots symbolize alerts (observable symptom events), (ii) intermediate nodes (in the trunk and crown) embody sequences of alerts (chunks), and (iii) the arcs represent part-whole relationships as well as the likelihood of occurrence of the whole (sequence) given the part (subsequence).

We have defined two kinds of temporal relationships: serial parts and parallel parts that allow us to represent respectively totally-ordered sub-cases and sub-cases whose order do not mind since they are interchaneable. For domains where reasoning in terms of temporal intervals is essential, actionable trees could be easily extended to manage Allen's temporal logic establishing a node type for each interval type. As a

matter of fact, our serial and parallel nodes subsume the rest of interval types. We have also embodied actionable trees in a representation that facilitates predictive inference.

We used a general model of likelihood that allows several schemes (deterministic, probabilistic, temporal, etc) to model part-of strengths. This model is based on associating a semiring structure with the part-whole relationships entailed by the arcs of actionable trees. This model provided a simple mechanism for evidence propagation in compositional hierarchies. Evidence propagation in taxonomic hierarchies was succinctly introduced by Pearl [Pea88]. However, to the best of our knowledge no one has provided a detailed model for evidence propagation in compositional hierarchies. Perhaps Pfleger's thesis is the only work that we know that have pointed in that direction [Pfl02]. The level of conceptual structure that a graphical representation encodes is fundamental to determine its quality.

The conceptual structure of actionable trees is given by virtue of the homomorphism that exists between its graphical representation and the structure of the concepts that represent (i.e., computer attacks). We have been mainly concerned with compositional hierarchies that emphasize the use of decoupled substructures. That is, each substructure is assigned a local context reflecting independence from the rest of substructures at the same level in the hierarchy. Actionable trees are well-suited for representing the hierarchical structure of multi-stage attacks. At a higher level an attack can be represented by a succession of high-level steps, high-level steps can be decomposed in lower-level steps or atomic steps, each atomic step is supposed to be detected automatically by host/network sensors that originate one or more alerts.

Thus, we say that an actionable trees encompass partonomic knowledge about attacks. Reasoning in terms of partonomic knowledge is fundamental for many domains where parts can be conjugated with other to create new wholes. But this aspect is particularly interesting in our domain application where often new multi-stage attacks are created varying some parts of previous well-known attacks. From our point of view, actionable trees offer two main advantages: *local context* and *arbitrary inference*. Sub-nodes provide a local context for their parts (alerts). Alerts of a sub-node are decoupled form other alerts and sub-nodes in a composite multi-stage attack [Uta94]. Instead of simply allowing one-step forward prediction an actionable tree allows one to predict any position of a sequence given information from one side/and or another [Pfl02].

We say that actionable trees are *highly intuitive*. They are easy-to-craft and understable by human experts. We address complex domains where sometimes it is imposible for an agent to statistically/automatically learn/discover new cases, thus it is desirable that an agent might be taught by being told. This can be achieved using hand-designed knowledge structures easily understable by both an human expert and artificial agent. For instance, for an intrusion detection system it is quit difficult to discover new rare DoS attacks based on exploiting algorithmic deficiencies such as the worse case of many data structures [Mci99, CW03b] (i.e., Quicksort [Knu98] can be attacked by constructing worse case inputs that turns its complexity from $O(nlogn)$ into $O(n^2)$). Nonetheless, it is easy for a human expert to develop signatures for detecting such attacks.

It also deserves attention to point out that in some sense actionable trees addressed two main shortcomings of terminological logics or description logics: (i) the lack of representation and reasoning facilities for part-of relations; and (ii) the limitations to express uncertainty. Only few works have dealt with managing uncertainty and description logic within an unified framework [KLP97]. We see our approach as a first step to extend an object-centered representation language for knowledge modeling such as Noos with compositional hierarchies and probabistic knowledge to manage uncertainty. Actionable trees offer an admissible trade-off between knowledge expressiveness and knowledge efficiency. Other knowledge structures, such as *context-free grammars* or *bayesian networks*, are by far much more expressive than actionable trees. However, it is a price associated with the higher expressivity in terms of operationability and learnability.

Third, we have introduced a simple ontology for intrusion detection, that allows one to create a model of the system under protection so that qualified inferences can be derived [MP03e]. This ontology provides a domain-specific representation language that allows a sequence of alerts (alert stream) provided by a conventional IDS not only to be readable but also understandable by a software agent. We have identified four key sources of knowledge (networks, incidents, vulnerabilities, and alerts) to be conceptualized and built a separate ontology for each of them using the knowledge representation language Noos [AP96]. We have pieced these partial ontologies together into a more global ontology called SOID. One of the main obstacles for the rapid development of higher-levels intrusion detection applications stems from the absence of a common ontology that not only impedes to deal with computer security incidents at a higher level of abstraction but also the collaboration among different intrusion detection systems. Porras et al signaled three key challenges for the alert management task [PFV02]: the absence of widely available domain expertise, the time-consuming and expensive effort due to the large number of alerts, and the heterogeneity of the information produced by different information security devices. SOID ontology copes with such challenges establishing well-defined semantics and providing an homogeneous representation of expertise and facilitating the automation by part of Alba of the alert management task. In other words, SOID allows Alba to process information consistently and reason about an IDS alert stream at a higher level of abstraction.

8.1.2 Dynamic Similarity

We have proposed and described a similarity measure for computing the similarity of two sequences of objects that continuously adapts to data seen so far. Our dynamic sequence similarity allows us to compare the input alert stream against the sequences of alerts yielded by actionable trees. This similarity measure looks for the evidence that a sequence of alerts in the alert stream and a sequential case are derived from a common attack pattern (perhaps altered using new undetectable actions or exploiting new vulnerabilities). Our similarity measure has two outstanding characteristics: continuous adaption to data seen so far and promotion of rareness. We argued that the context and the history in which comparison occurs is fundamental to pinpoint some

aspects of interests (such as rareness) of the sequences being compared. We took a simple approach to solving this problem. We proposed a semi-global alignment that uses a time-varying subsumption scoring scheme. This scoring scheme used a taxonomic hierarchy and the frequency of occurrence to compute the score for each pair of terms in the semi-global alignment. We saw how the score is computed using the reciprocal of the *Odds* what promotes the rareness of terms being compared. There is a clear justification for this. Rare sequences of alerts can be more dangerous than frequent ones. Thus, we say that our similarity provides a mechanism to promptly advert their presence. Additionally, we have defined two operations: *abduction* and *neglection* that allow us to deal respectively with lost and spurious alerts in the alert stream. Another feature that distinguishes our approach is the fact that normalized and non-normalized versions behave differently. The non-normalized similarity computation gives preference to rare occurrences of alerts that are more indicative that something new (unknown) is undergoing. When normalized the similarity lies between 0 and 1 and gives preference to complete occurrences. That is, sequences of alerts that are completely included in the alert stream are promoted (ranked first) versus those that only occur partially. In Chapter 4, we provided a number of analyses about a collection of real-world alerts that served two main purposes: to show how our dynamic sequence similarity behaved over time and to gain some insights about the population of alerts that we dealt with.

8.1.3 Ceaseless CBR

We have introduced a new CBR model, Ceaseless CBR, that supports as input a non-segmented sequence of complex events that arrive over time corresponding to coincidental problems. A distinguishing feature of our model with respect to the mainstream CBR model is that instead of considering a sequential process composed of four tasks to solve a new problem we model those tasks to be performed coincidentally (in parallel). Moreover, in our model a CBR system is not passively awaiting the arrival of a new problem but ceaselessly active gathering evidence to complete such parallel tasks. Ceaseless CBR can be seen as a reminding-based explanation model that facilitates the generation of plausible explanations in dynamic, imprecise, and adversarial environments when situations are given by unsegmented sequences of observational data stemming from multiple coincidental sources [Lea95]. The objective of Ceaseless CBR is to to find the best explanation of the alerts under examination. In other words, it allows us to know whether an undesired situation (an attack, fault, etc) has occurred or not and if so to indicate the multiple responsible sources (if more than one intervened) or at least which ones are the most plausible.

The Ceaseless CBR model enables two types of problem solving tasks. On the one hand our model facilitates diagnostic inference. That is to say, the isolation and identification of undesired situations. Comparing the sequence of symptom events that bring the system to the current situation with past experiences and pinpointing the most probable causes of an undesired situation. An appropriate diagnostic can not be provided based only on a single event but on an aggregation function over a number of events that have to be correlated. Diagnosing what have already happened

has crucial importance to obtain insights and understand the causes that conveyed to an undesired situation and try to prevent them or similar ones from ocurring again in the future. However, predicting future events is the only way to avoid undesired situations and gain valuable time to prepare an appropriate response. Thus, on the other hand, our model enables predictive inference. That is, it allows the transformation of systems from merely reactive into proactive. Making predictions requires inferring situations from observed ones. Partially matching a sequential case allows us to engage predictive inference and determine next alerts that should occur promptly. An important particularity of compositional and sequential cases is that they enable this kind of arbitrary predictions.

However, in the dynamic, imprecise, and adversarial domains we cope with—where nothing is certain except change—predictive inference is a so tremendous difficult task that even surpass human intelligence—the only thing predictable is unpredictability. These domains are so complex that while we may be able to understand the present and explain the past, we never be able to predict the future with the same level of confidence. Even so, our model provides a high preemptive ratio as shown experimentally in Chapter 7.

Ceaseless CBR provides several capabilities for deriving inferences from a particular partial problem description that is broken up in small pieces and a sequential case base. These are basic low level constructs on top of which more elaborate reasoning methods can be developed. How to integrate these constructs with high level reasoning methods constitutes one of the main focus of our future work as we will see later on. We have also seen the capability of Ceaseless CBR to discover new sequential cases both creating them from scratch and adapting previously existing ones (discovered in former iterations). This turns out to be an additional advantage of our approach since it does not require to be provided with an initial sequential case-base. That is, Ceaseless CBR automatically acquires sequential cases on-the-fly as the analysis of the sequence of intrusion alerts proceeds [VC93].

To conclude, we think that this novel CBR model is particularly valuable for real-world domains where the monitoring of real-time processes have been automated by means of sensors but (in spite of that automation) they still require continuous human oversight. Overseeing these processes is often extremely difficult due to the huge amounts of sequential data that non-intelligent sensor produce. However, further understanding of the reasoning processes of expert security analysts, political researchers, network troubleshooters, etc to correlate events will be needed to provide more elaborate processes (such as anticipatory planning) able to initiate proper counter-measures.

8.1.4 Alert Triage Evaluation

We chose alert triage in intrusion detection as the application domain to conduct an exploratory analysis of the techniques proposed and to evaluate their performance. In Chapter 1, we enumerated the problems that this domain poses and specified in detail the problems that our Ceaseless CBR model tackles. This domain requires to efficiently and incrementally recognize hidden partially ordered plans from multiple

concurrent attackers that are represented by large alphabets of complex objects overlapped into an on-line sparse stream [GG01]. Assigning the correct priority to alerts demands the ability to recognize subtle relationships between alerts so that they can be conveniently correlated and prioritized together. An effective triage implies to be able to predict the malignancy, the innocuousness or the falsehood of alerts.

The evaluation of intrusion detection systems is specially difficult compared to other application domains for a number of reasons.

1. First of all, there is no standardized methodology. As a matter of fact, the first attempts to create a standard evaluation scenario have stirred up some controversy. As we pointed out in Section 2.8, this area needs much more research and experimentation before a framework for the evalutaion of IDSes can be widely accepted. We have introduced a formal framework for the evaluation of alert triage in Chapter 7 [MP03c]. To the best of our knowledge nobody has provided a formal method of analysis for the evaluation of this particular component of IDSes. In fact the first experimental validation of correlation systems was recently published [HRT03].

2. Second, as we also saw in Section 2.8, one of the most difficult issues when measuring the performance of an IDS is defining a meaningful measure unit. In alert triage however the unit is clear. It is "alert". Thus, we measured the performance of our techniques in terms of the number of alerts correctly classified. In Chapter 6, we discussed about the inappropriateness of accuracy as performance measure, reviewed other performance measures, and introduced *t-area* measure.

3. Third, practically there is no data-sets available for the evaluation of intrusion detection systems. Except for the polemic ones provided by MIT Lincoln Laboratory [LFG00, McH00] and the attack-intensive *DefCon capture the flag* [CAB03]. These data-sets compile traffic that includes multiple attacks but not the corresponding alerts provided by an IDS. We have collected three alert databases (Rustoord, Naxpot, and Huckleberry) in real environments along more than 15 months (in total) of continuous vigilance.

4. Four, labeling exemplars is extremely complex. Assigning the proper malignancy, innocuousness, or falsehood of alerts is a hard task given the relative environment-dependency of alerts that we mentioned in Chapter 1. Thus, to guarantee the repeatability (and diversity) of our experiments we devised a method that assigns the malignancy of each alert emulating a level of exposure of the system under surveillance.

Our evaluation framework not only allows one to select the best alert triage system but also to make practical choices when assessing different components of alert triage. We have characterized alert triage as a detection task such that for each alert in the alert stream an action is taken $\{\mathbf{notification}, \neg\mathbf{notification}\}$ in function of a previous judgement on its malignancy. ROC analysis constitutes the basis of our

framework. Thus, we tried to provide a compact explanation of ROC elements and some alternatives in Chapters 7 and 2 respectively. ROC analysis allows one to establish when a detection system has the best performance indepently of distribution of the condition of interest and the cost of misdetections. We have seen how in these circumstances choosing among certain alternatives is difficult because different measures to estimate the accuracy do not agree. We introduced the *t-area* measure for this purpose. T-area is the area of the quadrilateral formed by the segments connecting the ROC point of interest and all the singular points of the ROC space except the perfect detection system. We compute t-area using Heron's formula for calculating the area of a triangle.

Then, we have also shown that when misdetection costs are known a proper decision can be made using alternatives such as explicitly representing expected cost. Finally, we also saw that when we dealt with imprecise environments then the optimal alert triage detection system lain on the edge of the convex hull of ROC points that represented the detection system in a ROC space. The convex hull that dominated all the ROC points of a set of alert triage systems determined the group of best alert triage systems to confront imprecise environments.

The experiments conducted were designed to determine the influence of urgency in the performance of Ceaseless CBR. We have conducted two main blocks of experiments. In the first one, we did not consider urgency at all. That is, Ceaseless CBR was configured to explain all alerts in the current window right away. In the second block we analyzed the influence of urgency. We measured the efficiency of Ceaseless CBR along five dimensions: (i) performance according to the formal framework explained above; (ii) applicationÕs CPU time requirement; (iii) multi-stage correlation (i.e., new sequential cases discovered) (iv) predictiveness or preemptive ratio (i.e., number of alerts whose subsequent occurrence is properly predicted); and alert reduction (i.e., weekly alert load reduction). We have also obtained promising results after performing some preliminary experiments triaging the alerts generated by Snort standard configuration using the *capture the flag* data-set[1] [CAB03].

8.1.5 Agent-Aided Intrusion Detection

We have also investigated how human diagnostic performance can be improved by means of a software agent that makes decisions on behalf of a human diagnostician (security manager or SSO). We developed a first prototype of a cognitive assistant, Alba, that helps to reduce the burdensome output produced by current IDSes and contributes to minimize the number of false positives due to innocuous attacks and to increase the predictive power for malicious multi-stage attacks.

Ideally, the ultimate goal of secure network administration is to make the three windows (vulnerability, penetrability and compromisibility) of each possible intrusion converge into a single point in time [MP03a]. Pursuing that objective is a manpower intensive process. Moreover, the astounding growth of networks and the

[1] www.defcon.org

speed at which Internet software has been developed and released inevitably has led to an exponential growth in the number of current vulnerabilities and exposures and therefore in the complexity of network administration. Only the smart automation of network administration tasks will alleviate the ever increasing manpower needed for secure network administration.

Using current generation of IDSes, SSOs are continuously overwhelmed with a vast amount of log information and bombarded with countless alerts. The capacity to tolerate false positives of a human SSO and correctly respond to the output of current IDSes is questionable. There are those who even postulate that traditional IDS not only have failed to provide an additional layer of security but have also added complexity to the security management task. Therefore, there is a compelling need for developing a new generation of tools that help to automatize security management tasks such as the interpretation and correct diagnosis of IDSes output. The fact of the matter is that as long as the number of networked organizations proliferates and the number of computer security threats increases this need accentuates. To make the aforementioned tasks more bearable we have envisaged a new generation of intrusion detection tools under the heading of *agent-aided intrusion detection*. Some recent works can be seen as the prelude of this tendency [SZ00, CHS00, GPK99]. As we will see below, agent-aided systems (also known as agent-based systems or multi-agent systems) constituted an active area of our previous research. Alba can be seen as a first example of an agent-aided intrusion detection tool. Alba mediates between the alarm element—such as is considered in a generic architectural model of an IDS—and the corresponding SSO.

As a matter of fact, more and more artificially intelligent systems are being structured as autonomous agents operating in complex, real-world environments, where each agent has some independent capability for acting, reasoning, and learning. However, understanding the requirements of autonomous agents operating in complex, real-world environments, and developing a science of agent design grounded in this understanding is still an incipient area of research. In Chapter 7, we have described the companion machinery needed by a ceaseless case-based reasoner to solve real problems and that allowed us to explore our techniques.

The appropriateness of intrusion detection alert triage for evaluating the significance of Ceaseless CBR comes from the compelling challenges that an application in this domain faces. As a matter of fact, research in the area of intrusion detection systems is receiving much attention from academia, industry, and government. ÊSince these systems pose many complex issues, there are many open problems for research and opportunities for making significant contributions. Finally, let us to express two interrelated considerations of interest:

1. We did not aim at developing more systematic IDSes but at using existing ones more efficiently, reducing the burden that they produce in terms of false alerts and providing an early warning on dangerous multi-stage attacks. Similarly to Julisch and Dacier's recent work [JD02].

2. To improve overall system security, our techniques should form part of larger ensemble. Namely, the success of Ceaseless CBR for the diagnosis and predic-

tion of undesired situations not only depends on the proposed CBR processes but also on effectively integrating them with other supplementary techniques into a larger information security ensemble.

Although the usefulness of our approach has been shown in a concrete domain application we think that there are a number of domains that could also benefit from the techniques proposed.

8.2 FUTURE WORK

We envision three main lines of research. Firstly, the extension of the basic components offered by our model to facilitate more elaborate problem solving tasks such adversarial plan recognition. Secondly, the evolution of our model to contemplate automated monitoring agents able to communicate and coordinate each other enabling in turn high-level collaboration. Thirdly, we will investigate how completely delegate problem determination tasks to the own computer system. Next, we describe these research lines in further detail.

8.2.1 Adversarial Plan Recognition

We think that at a higher level of abstraction our Ceaseless CBR model establishes the foundations for the construction of plan recognition systems in multi-agent settings. Plan recognition is the process of deducing an agent's intentions from observed actions. Plan recognition hypothesizes the most likely goals an agent may have and tries to gather evidence that corroborate or refute them. To facilitate this task, an agent's goals are decomposed into sub-goals that correspond to plans that are easily recognizable. Part of the process of recognition implies reasoning in terms of which plans enable the consecution of others and how compounding distinct plans together to achieve composite goals. Three kinds of plan recognition can be distinguished [CPA82, GG01]:

Keyhole Plan Recognition. In this kind a recognizer agent watches an observed agent that is not aware or simply is not obstructing the observation of his actions.

Intended Plan Recognition. In this case, the observed agent cooperates facilitating the recognition. Namely, the observer actively collaborates so that the recognizer agent understands his actions.

Adversarial Plan Recognition. In this modality, the observed agent is considered hostile since he aims actively at interfering the recognition of his actions. This kind of plan recognition violates the assumptions required by the the other two kinds. In spite of their apparent applicability in domain such as military counterplanning, game-playing, or computer security, adversarial plan recognition systems have been rarely investigated [DR98, GG01, Ker03].

There exist several plan recognition approaches:

Deductive approaches use rules to capture the steps of plans. This approach was used in the very beginning of plan recognition research by Schmidt et al [SSG78].

Abductive approaches see plan recognition as a specific form of the general problem of *abduction* such as Charniak proposed [CM85, GG01]. We briefly overview abductive reasoning in Section 2.6.2.

Graph covering approaches represent plans by means of graphs. Kauz and Allen proposed to recognize plans computing the minimum set cover of the plan graphs [KA86]. Their work laid the groundwork of most of the plan recognition approaches in the literature. As we viewed respectively in Sections 2.6.1 and 2.6.2, Katzela and Schwartz designed a collection of fault localization algorithms to find the best explanation of a set of alerts [KS95]; and Peng and Reggia introduced a formal theory of diagnostic inference named *parsimonious covering theory* [PR90] both based on the same *covering* concept.

Stochastic approaches were introduced by Charniak and Goldman who proposed to carried out the abductive process involved by plan recognition by means of probabilistic inference [CG93]. Albrecht et al also proposed a bayesian model in the context of aventure games [AZN98]. Pynadath and Wellman introduced *probabilistic state-dependent grammars* (PSDG) to represent an agent's plan generation process in the context of traffic monitoring and air combat [PW00]. A problem of many plan recognition approaches is that they only consider a list of equally plausible hypotheses. That is, all hypotheses that are compatible with the events observed so far are treated all the same. However, certain domains require a mechanism that allows one to finally come to a suitable explanation. Stochastic approaches overcome this issue assigning a measure that indicates the likelihood that a given hypothesis (explanation) represents the correct plan [CG93, Bau96].

Grammatical parsing approaches model plan recognition establishing a correspondence between plans and context free grammars and then casting the plan recognition problem as parsing in a grammar [Sid85, Vil90]. The main advantage of parsing approaches stems from their efficient implementations. However, when the input corresponds to an interleaving of actions such efficiency decreases significantly. The major drawback of parsing based approaches is that plans must be totally ordered. This causes an exponential increase in the size of the plan library (since all total orderings over a partially ordered plan have to be represented) [GG01].

Plan execution The above plan recognition models are unable to satisfactorily: (i) deal with interleaving of actions stemming from multiple agents; (ii) recognize partial-ordered plans; and (iii) handle situations where actions have not been observed yet. To overcome such shortcomings Goldman et al introduced a new theory of plan recognition centered around a model of plan execution

rather than on plans as static objects or rules that describe the recognition process [GGM99]. This approach is able to handle unsegmented sequences of observational data stemming from multiple coincidental sources (sequences of observations of partially-ordered, interleaved plans). This approach is closely-related to Ceaseless CBR. We practically address the same issues (multiple agents, interleaving of actions, adversarial situations, etc). As we saw in further detail in Chapter 5 of the thesis, actionable trees behave according to Goldman et al's theory. Moreover, Geib and Goldman have also put their theory into practice in the context of intrusion detection [GG01]. They proposed to extend IDS with artificial intelligence methods for plan recognition [GG01]. They argued that plan recognition is of paramount importance for predicting the future actions of attackers and timely planning convenient responses. Plan recognition can provide current IDSes with the capabilities of predicting attacks and facilitating an early warning. They modeled adversarial plan recognition as opposed to keyhole and intended plan recognition [SSG78, CPA82]. In adversarial plan recognition, as we mentioned above, the presence of hostile agents instead of cooperative agents make much more complex recognizing other agents' intentions. They have extended their previous work on plan recognition [CG93, GGM99] and provided an algorithm that coped with the severe conditions of the network security domain [GG01].

Case-based Many plan recognition systems require complete plan libraries that have to be handcrafted by human experts beforehand [KA86]. This manual construction of plan goes unmanageable in real-word scenarios. Case-based plan recognition has two main advantages: (i) the utilization of a case-base containing previous recognized plans avoids the construction of plans starting from scratch; and (ii) the plan library can be lazily constructed from actual data as the recognizer observes other agent(s). A number of CBR approaches have dealt with plan recognition [BCD94, CKD94, KC01, Ker03, FC03a]. However, to the best of our knowledge they have addressed neither adversarial conditions nor the issues that we mention below.

Devaney and Ram have proposed a plan recognition system in a real-world domain involving multiple agents performing maneuvers in training battles [DR98]. Devaney and Ram have noted that current plan recognition systems make three major assumptions that make them unsuitable for very complex real-world domains such as training battles or intrusion detection [DR98]: (i) the observed actions stem from a single agent that acts alone rather than multiple autonomous agents that act autonomously; (ii) the observations are complete and correct instead of noisy and incomplete; (iii) the actions occur over relatively short period of times; and (iv) the agents under observation only can perform a limited number of actions. These assumptions cause that many approaches use exhaustive search or probabilistic methods that are computationally intensive [DR98]. As we saw in Chapter 1, a number of real-world domains violate all these assumptions. Ceaseless CBR is well-suited to cope with those real-world domains. In fact, the relevance of our undertaking stems in part from the small number of approaches that have explore those challenging domains before.

Our proposal not only aims at extending Ceaseless CBR to recognize an adversary's intentions but also at using CBR to dynamically construct plans that obstruct adversary's actions. That is to say, to extend Ceaseless CBR for anticipatory planning purposes.

8.2.2 Distributed Ceaseless CBR

Intelligent agents that cooperate in dynamic, imprecise, and adversarial domains require representation, inference, and learning mechanisms that currently are unavailable under a unified framework or are still very much in their infancy. We propose to address this compelling need extending our CBR model to a multi-agent scenario. That is to say, a multi-agent approach where Ceaseless CBR agents could cooperate each other to solve more complex problems (e.g., problems that are inherently distributed) or to improve its individual competence and performance. Ceaseless CBR agents could share information at different levels: ranging from observational data or case activations to explanations and sequential cases. We have previously investigated techniques and mechanisms that lay the groundwork of this future line. For example, DistCBR and ColCBR [PAM96, PAM97, MPA99], conversation protocols [MPR00b], interagents [MPR98, MPR00a], etc.

Agent-based systems provide a way of conceptualizing sophisticated software applications that face problems involving multiple and (logically and often spatially) distributed sources of knowledge. In this way, they can be thought of as computational systems composed of several agents that interact with one another to solve complex tasks beyond the capabilities of an individual agent.

The development of agent-based systems can be regarded as a process composed of two well-defined, separate stages concerning respectively:

the agents' logics from the agents' inner behavior (knowledge representation, reasoning, learning, etc.) to the agents' social behavior responsible for high-level coordination tasks (the selection, ordering, and communication of the results of the agent activities so that an agent works effectively in a group setting [Les98, Jen95]).

the agents' interactions taking place at several levels: content level, concerned with the information content communicated among agents; intentional level, expressing the intentions of agents' utterances, usually as performatives of an agent communication language (ACL), e.g., KQML, FIPA ACL, etc.; conversational level, concerned with the conventions shared between agents when exchanging utterances; transport level, concerned with mechanisms for the transport of utterances; and connection level, contemplating network protocols (TCP/IP, HTTP, etc.).

Notice that nowadays a large amount of time during the development of agent-based systems is devoted to the management of the aforementioned time-consuming, complex agents' interactions. Since most of these appear to be domain-independent, it would be desirable to devise general solutions at this development stage that can

be subsequently reused for the deployment of other multi-agent systems. In our previous work we proposed an infrastructure that eases the construction of agent-based systems by taking charge of the cumbersome interaction issues inherent to this type of systems [MPR00b, MPR00a]. Such infrastructure relied upon two fundamental elements: conversation protocols, coordination patterns that impose a set of rules on the communicative acts uttered by the agents participating in a conversation (what can be said, to whom, and when), and interagents, autonomous software agents that mediate the interaction between each agent and the agent society wherein this is situated. Very importantly, interagents employ conversation protocols for mediating conversations among agents, and so the management of conversation protocols is in fact their raison d'être.

It could be said that our former research addressed agents' social behavior and interactions whereas the research presented throughout this thesis tackles with agents' inner behavior.

8.2.3 Autonomous Problem Determination

We hope that our work contributes to the development of automated networked systems that shift problem determination and resolution from endpoints (human users) to the network itself. More and more information systems are currently being devised to be survivable (i.e., to be able to provide basic services in spite of the presence of attacks and failures and recover full services gracefully [EFL97, EFL99, LF99]). The goal of this future research will be to enable a computer network to become self-healing (self-diagnosing and self-configuring) so that the network is more efficient, more reliable and responds better to unanticipated problems. This model has also coined by the industry as autonomic computing for its resemblance with the human body's autonomic nervous system[2]. We will investigate how to evolve our techniques to become part of autonomic problem determination tools with self-healing capabilities. Specifically, Ceaseless CBR could be suitable to perform the analysis of sensed event data within an autonomic manager control loop architecture as well as a symptom service able to compile sequential cases that indicate problems and their possible causes and remedies. Automatic problem determination provides at least two clear advantages: it increases IT personnel productivity and reduces downtime and corresponding revenue losses.

This thesis has examined fundamental issues for enhancing CBR systems. Particularly, some of the issues that arise when a case-based reasoner has to solve problems that are described in terms of unsegmented sequences of events that partially reflect the state of a dynamically changing environment. The major contribution of this thesis is a new CBR model for dealing with temporally-evolving unsegmented sequences of sensor data. The most salient features of this model are a novel representation of sequential cases, a dynamic similarity between sequences, and a ceaseless CBR

[2] www.alphaworks.ibm.com/autonomic

process that uses a decision theoretic approach to establish the urgency of each alert and prioritize it in accordance. We have also provided an exploratory analysis of a real-world domain.

Ceaseless CBR is specially useful for the construction of correlation and aggregation components—software components that handle alerts triggered by automated real-time processes. Moreover, we have developed a formal framework for the evaluation of alert triage and presented a first research prototype that incorporates the techniques devised throughout the thesis. We have also compiled real-world data during almost a year and created 3 data-sets that have allowed us to test satisfactory our approach.

We think that the research provided throughout this thesis accomplish satisfactory our initial objective: to enhance the CBR paradigm to solve situations that are expressed by means of unsegmented, noisy sequences of complex events that arrive continuously over time. The knowledge structures, methods and techniques proposed contribute to wide the spectrum where the CBR paradigm can be applied. Our empirical study demonstrates the feasibility of our approach and the adequacy of CBR. Part of our future work will expand on our previous research on communication and coordination among multiple agents and cooperative case-based reasoning. We envision a cooperative framework where several Alba assistants exchange sequential cases to improve overall performance. We have already established the basis for such framework on our previous work [PAM97, MPA99].

This Chapter pinpointed the novelty and contributions of the work developed, described the cutting-edge boundaries of our research, and speculated about future research lines.

Appendix A
Data sets

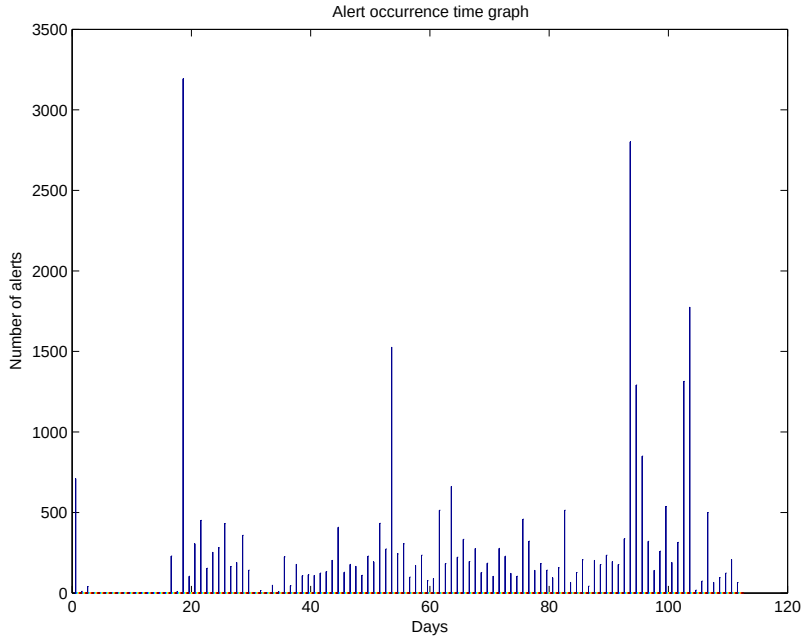
This Appendix describes the data used for experiments in the thesis. The three real-world scenarios where our data-sets where compiled are:

Rustoord Local area network composed of 5 computers (3 running Windows OS, 1 running Linux RedHat, 1 running Mac OS X) with a permanent DSL connection managed by a 3com router. We used 1 Snort sensor protected by the router port-map configuration.

Naxpot This scenario consisted of a target network with more than 200 hosts segregated into 3 sites with one local are network each. A wide area network shared between 3 cooperating offices, internal corporate network, offering multiple services to the Internet. 2 IDS sensors: one on a perimeter network that is protected by a firewall and offered services to the Internet and one before the external firewall on a direct link to the Internet.

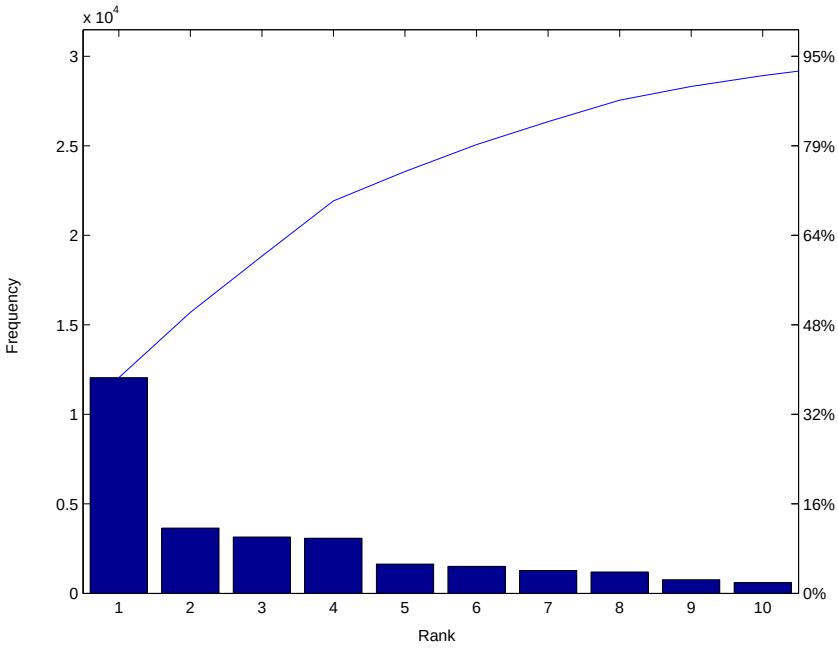
Huckleberry Local area network composed of 3 iMac computers with a permanent cable-modem connection. Originally the IDS sensor was deployed in a perimeter network protected by a firewall but due to a misconfiguration error the sensor was finally on a direct link to the Internet.

Rustoord data set characteristics



Alerts	31483
First alert	Oct 6th, 2002 at 22:48:04
Last alert	Jan 26th, 2003 at 14:25:06
Time interval	111d:15h:37m:02s
Average frequency	283.63 alerts/day
Unique alerts	90
Source IP addresses	650
Destination IP addresses	230
Unique IP links	934
TCP source ports	5446
UDP source ports	0
TCP destination ports	1780
UDP destination ports	0
TCP traffic	95%
UDP traffic	0%
ICMP traffic	5%

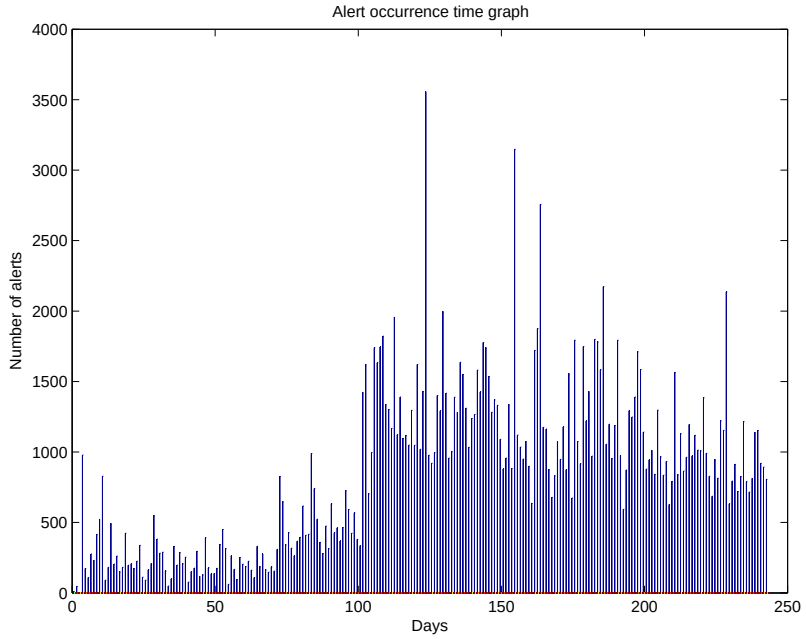
Fig. A.1 Rustoord data set characteristics



1	WEB-IIS cmd.exe access	12050	38%
2	WEB-ATTACKS netcat command attempt	3643	12%
3	WEB-CGI redirect access	3145	10%
4	spp_stream4: STEALTH ACTIVITY (nmap XMAS scan) detection	3084	10%
5	ICMP Destination Unreachable (Communication Administratively Prohibited)	1636	5%
6	WEB-ATTACKS cc command attempt	1510	5%
7	WEB-IIS CodeRed v2 root.exe access	1279	4%
8	WEB-CGI calendar access	1198	4%
9	X11 outbound client connection detected	766	2%
10	WEB-MISC /doc/ access	606	2%

Fig. A.2 Rustoord most frequent alerts

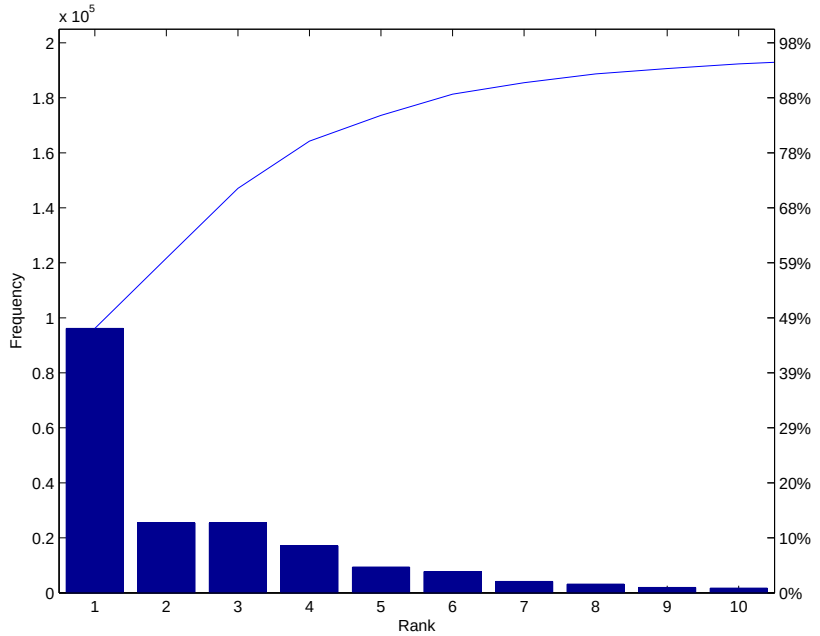
Naxpot data set characteristics



Alerts	204977
First alert	Nov 11th, 2002 at 12:11:52
Last alert	Jul 18th, 2003 at 20:06:53
Time interval	242d:07h:55m:01s
Average frequency	847 alerts/day
Unique alerts	331
Source IP addresses	22112
Destination IP addresses	394
Unique IP links	45429
TCP source ports	27915
UDP source ports	3825
TCP destination ports	3974
UDP destination ports	6
TCP traffic	87%
UDP traffic	13%
ICMP traffic	0%

Fig. A.3 Naxpot data set characteristics

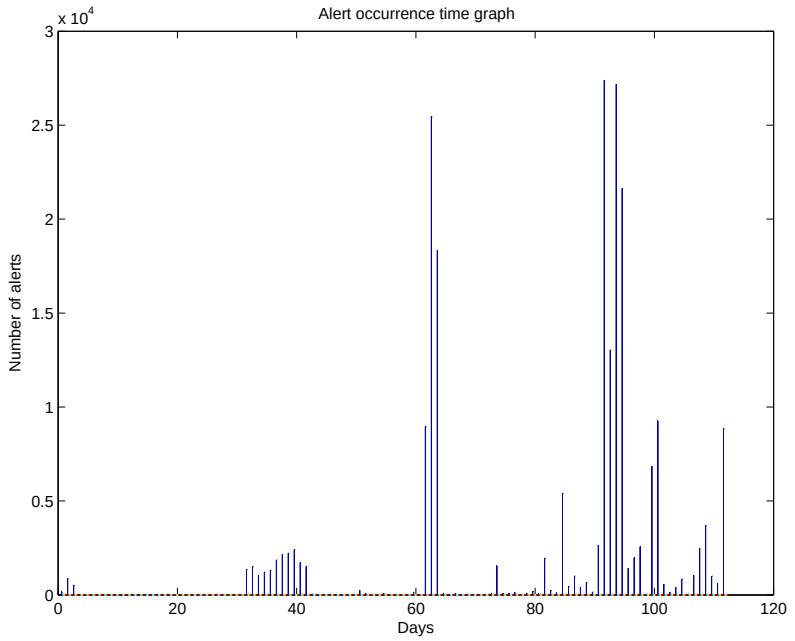
Naxpot most frequent alerts



1	WEB-MISC http directory traversal	96149	47%
2	MS-SQL Worm propagation attempt	25510	12%
3	SCAN SOCKS Proxy attempt	25437	12%
4	(spp_rpc_decode) Incomplete RPC segment	17147	8%
5	SCAN Proxy (8080) attempt	9380	5%
6	SCAN Squid Proxy attempt	7719	4%
7	(spp_rpc_decode) Fragmented RPC Records	4154	2%
8	ATTACK RESPONSES 403 Forbidden	3198	2%
9	SCAN nmap TCP	1945	1%
10	WEB-MISC intranet access	1711	1%

Fig. A.4 Naxpot most frequent alerts

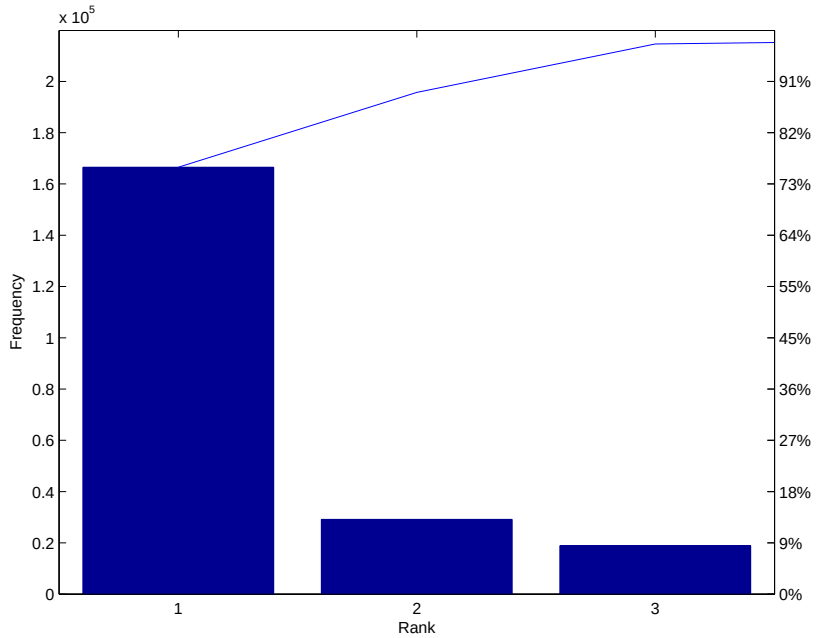
Huckleberry data set characteristics



Alerts	219886
First alert	Apr 20th, 2003 at 17:14:51
Last alert	Aug 10th, 2003 at 13:53:04
Time interval	111d:20h:38m:13s
Average frequency	1980.95 alerts/day
Unique alerts	203
Source IP addresses	573
Destination IP addresses	90
Unique IP links	687
TCP source ports	15731
UDP source ports	1273
TCP destination ports	30
UDP destination ports	7
TCP traffic	10%
UDP traffic	90%
ICMP traffic	0%

Fig. A.5 Huckleberry data set characteristics

Huckleberry most frequent alerts



1	MISC UPNP malformed advertisement	166534	75.8%
2	WARNING: Not IPv4 datagram!	29158	13.3%
3	SCAN FIN ÊÊ ÊÊ attempted-recon	18912	8.6%
4	WEB-IIS cmd.exe access	1108	0.5%
5	SNMP public access udp	1045	0.5%
6	WEB-IIS ISAPI .ida attempt	513	0.2%
7	MS-SQL Worm propagation attempt	293	0.1%
8	BAD TRAFFIC 0 ttl	254	0.1%
9	WEB-MISC /doc/ access	189	0.1%
10	WEB-IIS CodeRed v2 root.exe access	179	0.1%

Fig. A.6 Huckleberry most frequent alerts

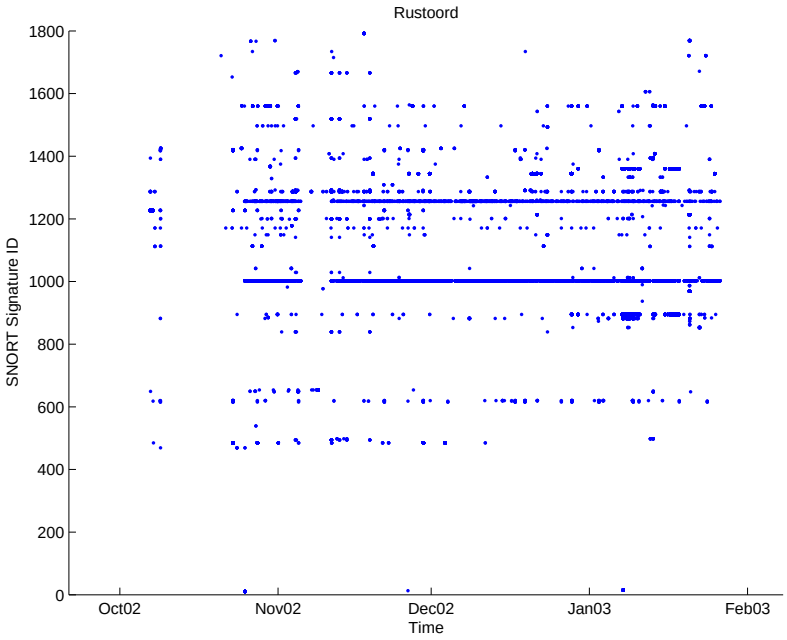


Fig. A.7 Scatter plot of alert signature identifier against time for Rustoord data-set.

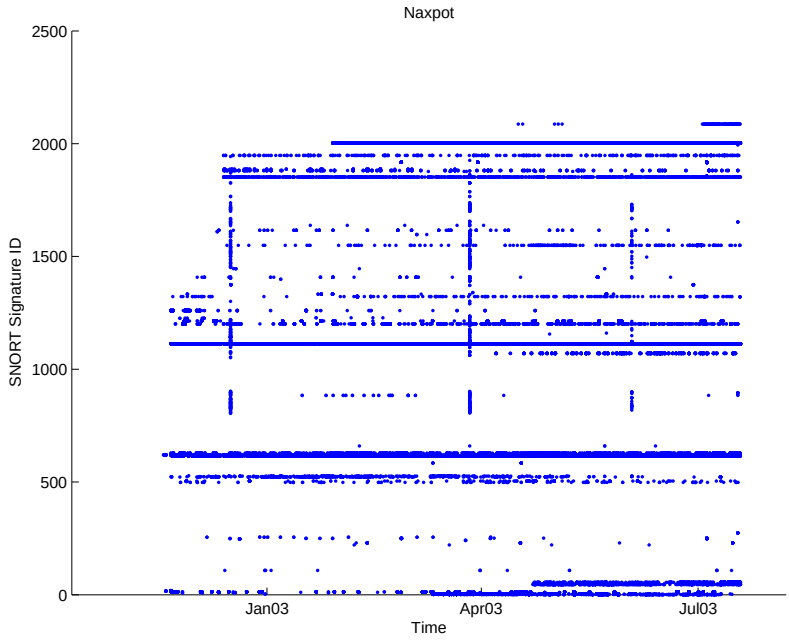


Fig. A.8 Scatter plot of alert signature identifier against time for Naxpot data-set.

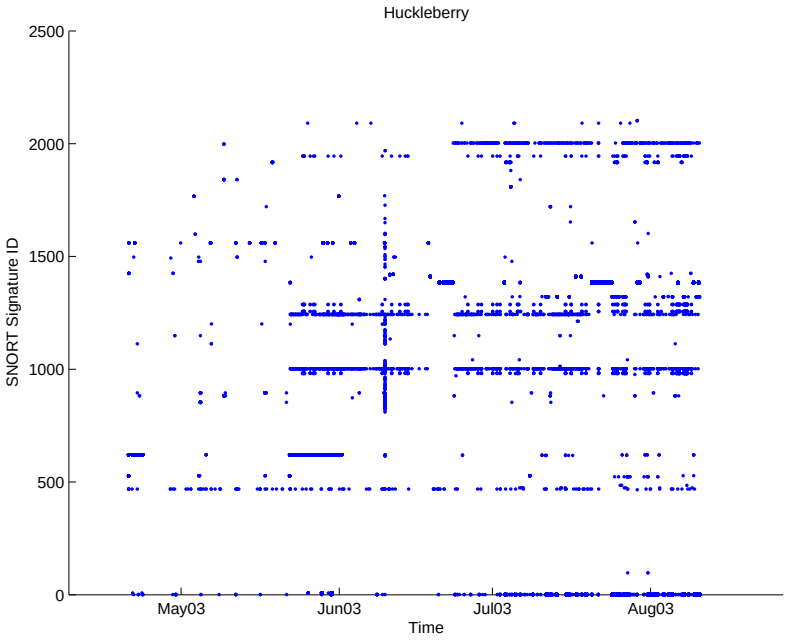


Fig. A.9 Scatter plot of alert signature identifier against time for Huckleberry data-set.

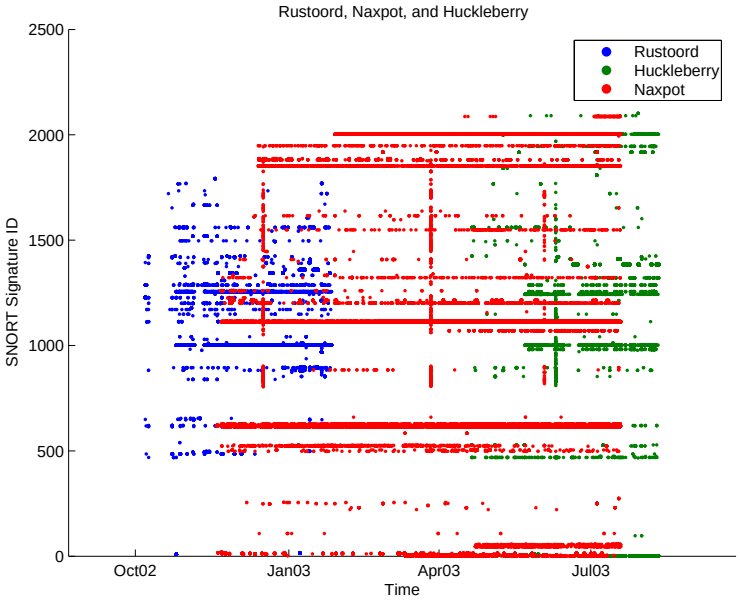


Fig. A.10 Scatter plot of alert signature identifier against time for all data-sets.

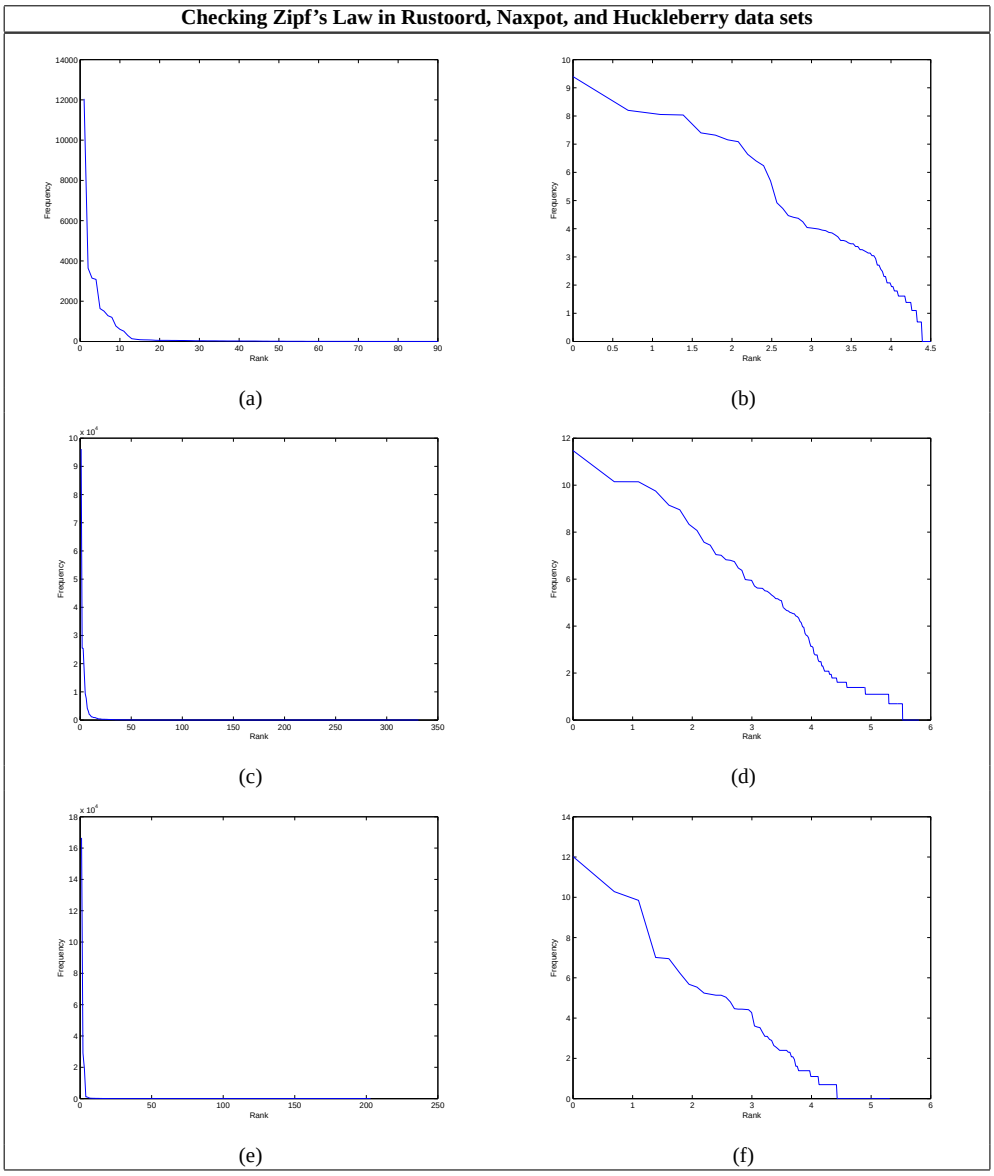


Fig. A.11 Checking Zipf's Law in Rustoord, Naxpot, and Huckleberry data sets. (a), (c), and (e) plot the alert frequency in rank order for Rustoord, Naxpot, and Huckleberry data sets respectively obtaining a charateristic hyperbolic shape for the three of them. (b), (d), and (f) show the same data plotted on logarithmic scales.

2

Acronyms

ACBARRS A Case-Based Reactive Robotic System

ACCs Aggregation and Correlation Components

ACID Analysis Console for Intrusion Databases

ACL Agent Communication Language

Alba ALert BArrage

AI Artificial Intelligence

AOI Attribute-Oriented Induction

AT Actionable Tree

ATTG Automatic Trouble Ticket Generator

AUC Area Under the Curve

BN Bayesian Network

CH Compositional Hierarchy

CBR Case-Based Reasoning

CVE Common Vulnerabilities and Exposures

DBN Dynamic Bayesian Network

DET Detection Error Tradeoff

DM Decision Maker

DoS Denial of Service

EM Expectation Maximization

FN False Negatives

FNF False Negative Fraction

FP False Positives

FPF False Positive Fraction

HMM Hidden Markov Model

HTN Hierarchical Transition Network

IAM Interactive Activation Model

ICUs Intensive-Care Units

IID Identically Independently Distributed

IDS Intrusion Detection System

IDWG Intrusion Detection Working Group

KB Knowledge-Based

MAFTIA Malicious and Accidental Fault Tolerance for Internet Applications

MAP Maximum A-Posteriori Probability

MCL Macintosh Common Lisp

MDL Minimum Description Length

MEP Most-Probable Explanation

Meta-XP Meta-Explanation Patterns

ML Machine Learning

MCI Mass Casualty Incidents

MRAG Multi-Rooted Acyclic Graph

NERD Network Entity Relationship Database

NLP Natural Language Processing

NN Neural Network

NPV Negative Predictive Value

NR Negotiation Retrieval

ORFs Open-reading Frames

PAT Predictive Actionable Tree

PCH Predictive Compositional Hierarchies

PFSM Probabilistic Finite State Machine

Poset Partially Ordered Set

PSDG Probabilistic State-Dependent Grammars

PPV Positive Predictive Value

PPS Prediction by Partial Subsumption

PR Preemptive Ratio

RL Reinforcement Learning

ROC Receiver Operating Characteristic

SINS Self-Improving Navigation System

SOID Simple Ontology for Intrusion Detection

SSO Site Security Officer

TN True Negatives

TNF True Negative Fraction

TP True Positives

TPF True Positive Fraction

TTSs Trouble Ticket Systems

UUUP Unbounded Unsegmented Unsupervised Prediction

XPs Explanation Patterns

References

AAD03. Thomas A. Alspaugh, Annie I Antón, and Laura J. Davis. “An Empirical Study of Scenario Similarity Measures.” Technical Report UCI-ISR-03-7, Institute for Software Research, 2003.

Aam91. Agnar Aamodt. *A Knowledge-Intensive Approach to Problem Solving and Sustained Learning*. PhD thesis, Norwegian Institute of Technology, Univesity of Trondheim, 1991.

Aam94. Agnar Aamodt. “Explanation-Drive Case-Based Reasoning.” In S. Wess, K.D. Althoff, and M. Richter, editors, *Topics in Case-Based Reasoning*, number 837 in Lecture Notes in Artificial Intelligence, pp. 274–288. Springer-Verlag, 1994.

Aam95. Agnar Aamodt. “Knowledge Acquisition and Learning by Experience: The Role of Case-Specific Knowledge.” In Y. Kodratoff and G. Tecuci, editors, *On Integration of Knowledge Acquisition and Machine Learning*. Academic Press, 1995.

AB95. J. Anigbogu and A. Belaid. “Hidden Markov Models in Text Recognition.” *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, **9**(6):925–958, 1995.

ABM01. David W. Aha, Len Breslow, and Hector Muñoz-Avila. “Conversational Case-Based Reasoning.” *Applied Intelligence*, **14**(1):9–32, 2001.

Abr96. J. R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

ABT99. R. Agarwal, G. Bruno, and M. Torchiano. "Modeling Complex Systems: Class models and instance models." In *Proceedings International Conference on Information Technology (CIT'99)*, 1999.

ACD01. Dominique Alessandri, Christian Cachin, Marc Dacier, Oliver Deak, Klaus Julish, Brian Randell, James Riordan, Andreas Tscharner, Andreas Wespi, and Candid Wüest. "Towards a Taxonomy of Intrusion Detection Systems and Attacks." Technical report, MAFTIA deliverable D3. Project IST-1999-11583, 2001.

ACF99. Julia Allen, Alan Christie, William Fithen, John McHugh, Jed Pickel, and Ed Stoner. "State of Practice of Intrusion Detection Technologies." Technical Report CMU/SEI-99-TR-028, CERT, 1999.

Ada00. Jean-Marc Adamo. *Data Mining for Association Rules and Sequential Patterns*. Springer-Verlag, 2000.

AFG96a. A. Artale, E. Franconi, and N. Guarino. "Open problems with part-whole relations." In *Proceedings of 1996 International Workshop on Description Logics, Boston, MA*, pp. 70–73, 1996.

AFG96b. Alessandro Artale, Enrico Franconi, Nicola Guarino, and Luca Pazzi. "Part-Whole Relations in Object-Centered Systems: An Overview." *Data Knowledge Engineering*, **20**(3):347–383, 1996.

AFV02. D. Andersson, M. Fong, and A. Valdes. "Heterogeneous Sensor Correlation: A Case Study of Live Traffic Analysis." In *IEEE Information Assurance Workshop*, 2002.

AGM03. Josep Lluís Arcos, Maarten Grachten, and Ramon López de Mántaras. "Extracting Performers' Behaviors to Annotate Cases in a CBR System for Musical Tempo Transformations." In *Fifth International Conference on Case-Based Reasoning*, pp. 20–34, 2003.

AHH93. K. Asai, S. Hayzmizu, and K. Handa. "Prediction of protein secondary structure by the hidden Markov model." *Computer applications in the biosciences*, **9**(2), 1993.

All83. James F. Allen. "Maintaining Knowledge About Temporal Intervals." *Communications of the ACM*, **26**(11):832–843, 1983.

AMB98. David W. Aha, Tucker Maney, and Leonard A. Breslow. "Supporting Dialogue Inferencing in Conversational Case-Based Reasoning." *Lecture Notes in Computer Science*, **1488**:262–266, 1998.

AMS98. Josep Lluís Arcos, Ramón López de Mántaras, and Xavier Serra. "SaxEx: A Case-Based Reasoning System for Generating Expressive Musical Performances." *Journal of New Music Research*, **27**(3), 1998.

And80. James P. Anderson. "Computer Security Threat Monitoring and Surveillance." Technical report, James P. Anderson Co, Fort Whanington, PA, USA, 1980.

AP94. Agnar Aamodt and Enric Plaza. "Case-Based Reasoning: Foundational Issues, methodological variations, and system approaches." *Artificial Intelligence Communications*, **7**(1):39–59, 1994.

AP96. Josep Lluís Arcos and Enric Plaza. "Inference and Reflection in the object-centered representation language Noos." *Journal of Future Generation Computer Systems*, **12**:173–188, 1996.

Ark89. R. C. Arkin. "Motor Schema-Based Mobile Robot Navigation." *The International Journal of Robotics Research*, **8**(4):92–112, 1989.

Arm97. Eva Armengol. *A Framework for Integrated Learning and Problem Solving*. PhD thesis, Universitat Politècnica de Catalunya, 1997.

AS95. Rakesh Agrawal and Ramakrishnan Srikant. "Mining sequential patterns." In Philip S. Yu and Arbee S. P. Chen, editors, *Eleventh International Conference on Data Engineering*, pp. 3–14, Taipei, Taiwan, 1995. IEEE Computer Society Press.

AW92. T. Acorn and S. Walden. "SMART: Support Management Automated Reasoning Technology for Compaq Customer Service." *Innovative Applications of Artificial Intelligence*, **4**, 1992.

Axe98. Stefan Axelsson. "Research in Intrusion-Detection Systems: A Survey." Technical Report TR: 98-17, Department of Computer Engineering. Chalmes University of Technology, Göteborg, Sweden, December 1998.

Axe00. Stefan Axelsson. "The Base-Rate Fallacy and the Difficulty of Intrusion Detection." *ACM Transactions on Information and System Security*, **3**(3):186–205, August 2000.

AZN98. David W. Albrecht, Ingrid Zukerman, and Anne E. Nicholson. "Bayesian Models for Keyhole Plan Recognition in an Adventure Game." *User Modeling and User-Adapted Interaction*, **8**(1-2):5–47, 1998.

BA95. L.K. Branting and D.W. Aha. "Stratified Case-Based Reasoning: Reusing Hierarchical Problem Solving Episodes." In *Proceedings of 14th International Joint Conference on Artificial Intelligence*, pp. 384–390, 1995.

Bau96. Mathias Bauer. "Justification of Plan Recognition Results." In *Proceedings of European Conference on Artificial Intelligence*, 1996.

Bay03. Valentina Bayer. *Learning Cost-Sensitive Diagnostic Policies from Data*. PhD thesis, Department of Computer Science, Oregon State University, 2003.

BBM89. Alex Borgida, Ronald J. Brachman, Deborah L. McGuinness, and Lori Alperin Resnick. "CLASSIC: A Structural Data Model for Objects." In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pp. 59–67, 1989.

BC96. I. Bichindaritz and E. Conlon. "Temporal knowledge representation and organization for case-based reasoning." In *3rd Workshop on Temporal Representation and Reasoning (TIME'96)*, 1996.

BCD94. M. Barès, D. Cañamero, J.-F. Delannoy, and Y. Kodratoff. "XPlans: Case-Based Reasoning for Plan Recognition." *Applied Artificial Intelligence*, **8**:61–643, 1994.

BCF94. A. T. Bouloutas, S. Calo, and A. Finkel. "Alarm correlation and fault identification in communication networks." *IEEE Transactions on Communications*, **42**(2/3/4), 1994.

BCF99. Lee Breslau, Pei Cao, Li Fan, Graham Philips, and Scott Shenker. "Web Caching and Zipf-like Distributions: Evidence and Implications." In *Proceedings of IEEE Infocom '99*, pp. 126–134, 1999.

Ber96. J. Bernauer. "Analysis of part-whole relation and subsumption in the medical domain." *Data & Knowledge Engineering*, **20**(3):259–286, 1996.

BET99. Giuseppe di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algortihms for the Visualization of Graphs*. Prentice Hall, 1999.

BH95. John S. Breese and David Heckerman. "Decision Theoretic Case-Based Reasoning." Technical Report MSR-TR-95-03, Microsoft Research, Advanced Technology Division, 1995.

Bil02. Jeff Bilmes. "What HMMs can do." Technical Report UWEETR-2002-0003, University of Washington, 2002.

BKS96. Mark Benson, Kristi L. Koenig, and Carl H. Schultz. " Disaster triage: START, then SAVE-A new method of dynamic triage for victims of a catastrophic earthquake." *Prehospital and Disaster Medicine*, **11**(2):117–124, 1996.

BL91. S. Brown and L. Lewis. "A Case-Based Reasoning Solution to the Problem of Redundant Resolutions of Nonconformances in Large-Scale Manufacturing." *Innovative Applications of Artificial Intelligence*, **3**, 1991.

BLB99. Leliane Nunes de Barros, Marilza Lemos, Volnys Bernal, and Jacques Wainer. "Model Based Diagnosis for Network Communication Faults."

In *Proceedings of International Workshop on Artificial Intelligence for Distributed Information Networking (AIDIN'99)*, pp. 57–62. AAAI Press, 1999.

BM88. R. Barletta and W. Mark. "Breaking Cases Into Pieces." In *AAAI-88 Case-Based Reasoning Workshop*, pp. 12–16, 1988.

BM93. J. Britanik and M. Marefat. "Distributed Case-Based Planning: Multi-Agent Cooperation for High Autonomy." In *Proceedings of the Fourth Annual Conference on AI, Simulation, and Planning in High Autonomy Systems*, Tucson, 1993.

BMM93. S. Brugnoni, R. Manione, E. Montariolo, E. Paschetta, and L. Sisto. "An Expert System for Real-Time Fault Diagnosis of the Italian telecommunications network." In *Proceeding of the 3rd International Symposium on Integrated Network Management*, pp. 617–628, 1993.

BMR97. Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. "Semiring-Based Constraint Satisfaction and Optimization." *Journal of the ACM*, **44**(2):201–236, 1997.

BMV98. R. Bergmann, H. Muñoz-Avila, M.M. Veloso, and E. Melis. "Case-based reasoning applied to planning." In M. Lenz, B. Bartsch-Sporl, H.-D. Burkhard, and S. Wess, editors, *Case-Based Reasoning Technology from Foundations to Applications*,, number 1400 in Lecture Notes on Artificial Intelligence (LNAI), pp. 169–200. Springer, 1998.

Bra83. Ronald J. Brachman. "What IS-A is and isn't: an analysis of taxonomic links in semantic networks." *IEEE Computer*, **16**(10):30–36, 1983.

BRM02. M. Brodie, I. Rish, and S. Ma. "Intelligent probing: A cost-effective approach to fault diagnosis in computer networks." *IBM Systesms Journal*, **41**(3), 2002.

BS84. B. C. Buchanan and E. H. Shortliffe. *Rule-Based Expert Systems The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley Publishing Company, 1984.

BW96. Ralph Bergmann and Wolfgang Wilke. "On the Role of Abstraction in Case-Based Reasoning." In *EWCBR*, pp. 28–43, 1996.

CAB03. Crispin Cowan, Seth Arnold, Steve Beattie, Chris Wright, and John Viega. "Defcon Capture the Flag: Defending Vulnerable Code from Intense Attack." In *DARPA Information Survivability Conference and Exposition - Volume I* , pp. 120–129, 2003.

Cal89. P. H. Callahan. "Expert Systems for AT&T Switched Network Maintenance." In E. C. Ericson, L. T. Ericson, and D. Minoli, editors, *Expert*

Systems Applications in Integrated Network Management, pp. 263–273. Artech House, 1989.

CAM02. Frédéric Cuppens, Fabien Autrel, Alexandre Miège, and Salem Benferhat. "Correlation in an Intrusion Detection Process." In *Sécurité des Communications sur Internet*, 2002.

Cap01. Olivier Cappé. "Ten years of HMMs.", March 2001.

Car86. J. G. Carbonell. "Derivational Analogy: A theory of reconstructive problem solving and expertise acquisition." In R.S. Michalski, J. G. Carbonell, and T. M. Michell, editors, *Machine Learning: An Artificial Intelligence Approach (Volume II)*, pp. 371–392. Morgan Kaufmann, 1986.

CBF03. Brian Caswell, Jay Beale, James C. Foster, and Jeremy Faircloth. *Snort 2.0 Intrusion Detection*. SYNGRESS, 2003.

CD03. D. Curry and H. Debar. "Intrusion Detection Message Exchange Format Data Model and Extensible Markup Language (XML) Document Type Definition." Internet-Draft, expired: July 31, 2003, 2003.

CG93. Eugene Charniak and Robert P. Goldman. "A Bayesian Model of Plan Recognition." *Artificial Intelligence*, **64**:53–79, 1993.

CHS00. Curtis A. Carver, John M.D. Hill, John R. Surdu, and Udo W. Pooch. "A Methodology for Using Intelligent Agents to Provide Automated Intrusion Response." In *Proc. of the IEEE WIAS*, pp. 110–116, 2000.

Chu89. G. A. Churchill. "Stochastic Models for Heterogeneous DNA Sequences." *Bulletin of Mathematical Biology*, **5**:79–94, 1989.

CKD94. D. Cañamero, Y. Kodratoff, J.-F. Delannoy, and M. Barès. "Case-Based Plan Recognition." In *Working Notes of the Workshop on Planning and Learning: On to Real Applications, AAAI-94 Fall Symposium Series*, pp. 16–22, 1994.

CKF02. Mike Y. Chen, Emre Kiciman, Eugene Fratkin, Armando Fox, and Eric Brewer. "Pinpoint: Problem Determination in Large, Dynamic Internet Services." In *Proceedings of 2002 International Conference on Dependable Systems and Networks*, pp. 595–564, 2002.

CL02. R. Coolen and H.A.M. Luijf. "Intrusion Detection: Generics and State-of-the-Art." Technical Report RTO-TR-049, North Atlantic Treaty Organisation, Research and Technology Organisation, 2002.

CM85. Eugene Charniak and D. McDermott. *Introduction to Artificial Intelligence*. Addison Wesley, 1985.

CM02. Frédéric Cuppens and Alexandre Miège. "Alert correlation in a cooperative intrusion detection framework." In *IEEE Symposium on Research in Security and Privacy*, 2002.

CMC00. U. Cortés, M. Sànchez Marrè, L. Ceccaroni, I. R-Roda, and M. Poch. "Artificial Intelligence and Environmental Decision Support Systems." *Applied Intelligence*, **13**(1):77–91, 2000.

CMG02. Joan Colomer, Joaquim Meléndez, and F. Ignacio Gamero. "Pattern recognition based on episodes and DTW: Application to diagnosis of a level control system." In *Sixteenth International Workshop on Qualitative Reasoning*, 2002.

CPA82. Philip R. Cohen, C. Raymond Perrault, and James F. Allen. "Beyond question answering." In Wendy G. Lehnert and Martin H. Ringle, editors, *Strategies for Natural Language Processing*, chapter 9, pp. 245–274. Lawrence Erlbaum Associates, 1982.

CS94. Eugene Charniak and Solomon Eyal Shimony. "Cost-Based Abduction and MAP Explanation." *Artificial Intelligence*, **66**:345–374, 1994.

Cup01. Frédéric Cuppens. "Managing Alerts in a Multi-Intrusion Detection Environment." In *17th Annual Computer Security Applications Conference (ACSAC'01)*, 2001.

CV02. Roberto Casati and Achille C. Varzi. *Parts and Places: The Structures of Spatial Representation*. The MIT Press, 2002.

CW03a. Edwin Costello and David C. Wilson. "A Case-Based Approach to Gene Finding." In *Workshop on Case-Based Reasoning in the Health Sciences. The Fifth International Conference on Case-Based Reasoning (ICCBR-03)*, pp. 19–28, 2003.

CW03b. Scott A Crosby and Dan S Wallach. "Denial of Service via Algorithmic Complexity Attacks." In *USENIX Security*, 2003.

CYA99. Christina Carrick, Qiang Yang, Irene Abi-Zeid, and Luc Lamontagne. "Activating CBR Systems through Autonomous Information Gathering." In *Case-Based Research and Development: Third International Conference on Case-Based Reasoning (ICCBR'99)*, volume 1650 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

DAN02. Jürgen Dix, Héctor Mu noz Avila, Dana Nau, and Lingling Zhang. "Planning in a multi-agent environment: theory and practice." In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pp. 944–945. ACM Press, 2002.

DCH02. Mike Dean, Dan Connolly, Frank van Harmelen, James Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea

Stein. "OWL Web Ontology Language 1.0 Reference." Technical report, W3C World-Wide Web Consortium, 2002.

DCW99. Robert Durst, Terrence Champion, Brian Witten, Eric Miller, and Luigi Spagnuolo. "Testing and evaluating computer intrusion detection systems." *Communications of the ACM*, **42**(7):53–61, 1999.

Dec98. R. Dechter. "Bucket Elimination: A Unifying Framework for Probabilistic Inference." In Michael I. Jordan, editor, *Learning in Graphical Models*, pp. 75–104. The MIT Press, 1998.

DEK98. R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis*. Cambridge University Press, 1998.

DG01. Dipankar Dasgupta and Fabio A. Gonzalez. "An Intelligent Decision Support System for Intrusion Detection and Response." In *Proceedings of International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security (MMM-ACNS)*, St.Petersburg, 2001. Springer-Verlag.

DH00. Chris Drummong and Robert C. Holte. "Explicity Representing Expected Cost:Ê An Alternative to ROC Representation." In *Sixth ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2000.

Die02. Thomas G. Dietterich. "Machine Learning for Sequential Data: A Review." In *Structural, Syntactic, and Statistical Pattern Recognition*, volume 2396 of *Lecture Notes in Computer Science*, pp. 15–30. Springer-Verlag, 2002.

DK93. Eric Domeshek and Janer Kolodner. "Using the Points of Large Cases." *AI EDAM*, **7**(2):87–96, 1993.

DKL01. Jon Doyle, Isaac Kohane, Willian Long, Howard Shrobe, and Peter Szolovits. "Event Recognition Beyond Signature and Anomaly." In *Second IEEE-SMC Information Assurance Workshop, West Point, New York, June 5-6*, 2001.

DLD02. Kristopher Daley, Ryan Larson, and Jerald Dawkins. "A Structural Framework for Modeling Multi-Stage Network Attacks." In *Proceedings of the International Conference on Parallel Processing Workshops*, 2002.

DM02a. Mayur Datar and S. Muthukrishnan. "Estimating Rarity and Similarity over Data Stream Windows." Technical report, DIMACS Technical Report 2002-21, 2002.

DM02b. Hervé Debar and Benjamin Morin. "Evaluation of the Diagnostic Capabilities of Commercial Intrusion Detection Systems." In *Proc. of the RAID 2002*, number 2516 in Lecture Notes in Computer Science, pp. 177–198. Springer, 2002.

DPV02. Carlotta Domeniconi, Chang shing Perng, Ricardo Vilalta, and Sheng Ma. "A Classification Approach for Prediction of Target Events in Temporal Sequences." In *Proceedings of the 13th European Conference on Machine Learning*, 2002.

DR98. Mark Devaney and Ashwin Ram. "Needles in a Haystack: Plan Recognition in Large Spatial Domains Involving Multiple Agents." In *Proceedings of the fifteenth National Conference on Artificial Intelligence*, pp. 942–947, 1998.

DSS00. Jon Doyle, Howard Shrobe, and Peter Szolovits. "On Widening the Scope of Attack Recognition Languages." Massachussets Institute of Technology, 2000.

DV95. Gabi Dreo and Rober Valta. "Using Master Tickets as a Storage for Problem-Solving Expertise." In *Proceedings of thee IFIP/IEEE International Symposium on Integrated Network Management*, pp. 697–708, 1995.

DW01. Hervé Debar and Andreas Wespi. "Aggregation and Correlation of Intrusion Detection Alerts." In *Proceedings of the 4th symposium on Recent Advances in Intrusion Detection (RAID 2001)*, 2001.

Dyl76. Bob Dylan. "One More Cup of Coffee (Valley Below)." In *Desire*. Sony, 1976.

EFL97. Robert J. Ellison, David A. Fisher, Richard C. Linger, Howard F. Lipson, Thomas Longstaff, and Nancy R. Mead. "Survivable Network Systems: An Emerging Discipline." Technical Report CMU/SEI-97-TR-013, CERT Coordination Center, 1997.

EFL99. Robert J. Ellison, David A. Fisher, Richard C. Linger, Howard F. Lipson, Thomas Longstaff, and Nancy R. Mead. "Survivability: Protecting Your Critical Systems." *IEEE Internet Computing*, November/December 1999.

Ega75. J. P. Egan. *Signal Detection Theory and ROC Analysis*. Academic Press, 1975.

EGG93. David Eppstein, Zvi Galil, Raffaele Giancarlo, and Giuseppe F. Italiano. "Efficient algorithms for sequence analysis." In Renato M. Capocelli, Alfredo De Santis, and Ugo Vaccaro, editors, *Sequences II: Communication, Security, and Computer Science*, pp. 225–244. Springer-Verlag, 1993. From Int. Advanced Worksh. Sequences, Positano, Italy, June 1991.

ESB96. Mansour Emaili, Rei Safavi-Naini, Bala Balachandran, and Josef Pierprzyk. "Case-Based Reasoning for Intrusion Detection." In *12th Annual Computer Security Applications Conference*, 1996.

Esk02. Eleazar Eskin. *Sparse Sequence Modeling with Applications to Computational Biology and Intrusion Detection*. PhD thesis, Columbia University, 2002.

FC03a. Michael Fagan and Pádraig Cunningham. "Case-Based Plan Recognition in Computer Games." In K. D. Ashley and D. G. Bridge, editors, *The Fifth International Conference on Case-Based Reasoning (ICCBR-03)*, number 2689 in Lecture Notes in Artificial Intelligence (LNAI), pp. 161–170. Springer, 2003.

FC03b. Florentino Fdez-Riverola and Juan M. Corchado. "Using Instance-Based Reasoning Systems for Changing Environments Forecasting." In *Workshop on Applying CBR to Time Series Prediction. The Fifth International Conference on Case-Based Reasoning*, pp. 219–230, 2003.

Fer99. Mariano Fernández-López. "Overview of Methodologies for Building Ontologies." In *Workshop on Ontologies and Problem-Solving Methods: Lessons Learned and Future Trends.*, 1999.

FFH02. Cèsar Ferri, Peter Flach, and José Hernández-Orallo. "Learning Decision Trees Using the Area Under the ROC Curve." In Claude Sammut and Achim Hoffmann, editors, *Proceedings of the 19th International Conference on Machine Learning*, pp. 139–146. Morgan Kaufmann, July 2002.

FGK99. Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. "Learning Probabilistic Relational Models." In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1300–1307, 1999.

FGS01. Philippe Flajolet, Yves Guivarc'h, Wojciech Szpankowski, , and Brigitte Vallee. "Hidden Pattern Statistics." *LNCS 2076*, pp. 152–165, 2001. CERIAS TR 2002-06.

FGS02. Alan Fern, Robert Givan, and Jeffrey Mark Siskind. "Specific-to-General Learning for Temporal Events." In *Eighteenth National Conference on Artificial Intelligence (AAAI-2002)*, 2002.

FGT91. M. Frontini, J. Griffin, and S. Towers. *A Knowledge-Based System for Fault Localization in Wide Area Networs*, pp. 510–530. North Holland, 1991.

FMW03. B. Feinstein, G. Matthews, and J. White. "The Intrusion Detection Exchange Protocol (IDXP)." Internet-Draft, expired: April 22, 2003, 2003.

Fol01. Ben Folds. "Still Fighting It." In *Rockin' the Suburbs*. EPI, 2001.

FP97. Tom Fawcett and Foster J. Provost. "Adaptive Fraud Detection." *Data Mining and Knowledge Discovery*, **1**(3):291–316, 1997.

Fra93. E. Franconi. "A treatment of plurals and plural quantification based on a theory of collections." *Minds and Machines*, **3**(4):453–474, 1993.

Fra94. Jeremy Frank. "Artificial Intelligence and Intrusion Detection: Current and Future Directions." In *17th National Computer Security Conference, National Institute of Standards and Technology, Washington,D.C.*, pp. 22–33, 1994.

Fyo97. Fyodor. "The Art of Port Scanning." *Phrack Magazine*, **7**(51), 1997.

G 73. Jr. G. D. Forney. "The Viterbi algorithm." *Proceedings of IEEE*, **61**:268–278, 1973.

G00. Peter Gärdenfors. *Conceptual Spaces*. The MIT Press, 2000.

GAD94. Ashok Goel, Khaled Ali, Michael Donnellan, Andres Gomez, and Todd Callantine. "Multistrategy Adaptive Navigational Path Planning." *IEEE Expert*, **9**(6):57–65, 1994.

Gas72. Joseph L. Gastwirth. "The Estimation of the Lorenz Curve and Gini Index." *The Review of Economics & Statistics*, **54**(3):306–316, 1972.

GD02. Dominique Gro§ and Tom Lenaerts Towards a Definition of Dynam. "Towards a Definition of Dynamical Hierarchies." In *ALifeVIII Workshop Proceedings: Modeling Dynamical Hierarchies in Artificial Life*, pp. 45–54, 2002.

Geb96. Friedrich Gebhardt. "Structure oriented Case Retrieval." In *4th German Workshop on CBR*, 1996.

Gem99. Stuart Geman. "Compositionality." Technical report, Brown University Faculty Bulletin, Spring, 1999.

GG01. Christopher W. Geib and Robert P. Goldman. "Plan Recognition in Intrusion Detection Systems." In *DARPA Information Survivability Conference and Exposition*, 2001.

GGM99. Robert P. Goldman, Christopher W. Geib, and C. A. Miller. "A New Model of Plan Recognition." In *Proceedings of the 1999 Conference on Uncertainty in Artificial Intelligence*, 1999.

GHH01a. Robert P. Goldman, Walter Heimerdinger, Steven A. Harp, Christopher W. Geib, Vicraj Thomas, and Robert L. Carter. "Information Modeling for Intrusion Report Aggregation." In *DICEX*, pp. 329–342. IEEE Computer Society, 2001.

GHH01b. Robert P. Goldman, Walter Heimerdinger, Steven A. Harp, Christopher W. Geib, Vicraj Thomas, and Robert L. Carter. "Information Modeling for Intrusion Report Aggregation." In *DICEX*. IEEE Computer Society, 2001.

GM03. Aristides Gionis and Heikki Mannila. "Finding Recurrent Sources in Sequences." In *Proceedings of RECOMB'03*, 2003.

Goo91. M. Goodman. "PRISM: A Case-Based Telex Classifier." *Innovative Applications of Artificial Intelligence*, **2**, 1991.

GPK99. V. I. Gorodetski, L. J. Popyack, I. V. Kotenko, and V. A. Skormin. "Ontology-based Multi-agent Model of Information Security System." In *7th RSFDGrC*, number 1711 in LNAI, pp. 528–532. Springer, 1999.

Gru93. T. R. Gruber. "Towards Principles for the Design of Ontologies Used for Knowledge Sharing." In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, 1993. Kluwer Academic Publishers.

Gru98. Boris Gruschke. "Integrated Event Management: Event Correlation Using Dependency Graphs." In *Proceedings of Distributed Systems Operations and Management (DSOM'98)*, 1998.

GS01. Itzhak Gilboa and David Schmeidler. *A Theory of Case-Based Decisions*. Cambridge, 2001.

GTD01. Albert Güveni, Tolga Taner, and Irini Dimitriyadis. "Evaluation of Taguchi and ROC Techniques for the Quality Assessment of Binary Decision Models in Health Care and Industrial Aplications." http://www.bme.boun.edu.tr/guvenis/taguchi.htm, 2001.

GU01. John E. Gaffney and Jacob W. Ulvila. "Evaluation of Intrusion Detectors: A Decision Theory Approach." In *The IEEE Symposium on Security and Privacy*, 2001.

Gup98. Kalyan Moy Gupta. "Knowledge-Based System For Troubleshooting Complex Equipment." *International Journal of Information and Computing Science*, **1**(1):29–41, 1998.

Gus97. Dan Gusfield. *Algortithms on Strings, Trees, and Sequences.* The Press Syndicate of the University of Cambridge, 1997.

GVP90. D. Geiger, T. S. Verma, and J. Pearl. "Identifying independence in bayesian networks." *Networks*, **20**:507–534, 1990.

Ham89. K. Hammond. *Case-Based Planning: Viewing Planning as a Memory Task.* Academic Press, 1989.

Han00. B. Hansen. "Weather Prediction Using Case-Based Reasoning and Fuzzy Set Theory." Technical report, Technical University of Nova Scotia, 2000. Master of Computer Science Thesis.

Har02. Verity Harte. *Plato on Parts and Wholes*. Oxford, 2002.

HBR94. David Heckerman, John S. Breese, and Koos Rommelse. "Troubleshooting under Uncertainty." Technical Report MSR-TR-94-07, Microsoft Research, Advanced Technology Division, 1994.

HBR95. David Heckerman, John S. Breese, and Koos Rommelse. "Decision-theoretic troubleshooting." *Communications of the ACM*, **38**(3):49–57, 1995.

Her99. Josefina López Herrera. *Time Series Prediction Using Inductive Reasoning Techniques*. PhD thesis, Technical University of Catalonia, 1999.

HGF02. Howard Hamilton, Ergun Gurak, Leah Findlater, and Wayne Olive. "Computer Science 831: Knowledge Discovery in Databases." http://www2.cs.uregina.ca/˜hamilton/courses/831/index.html, 2002.

HH02. Karim O. Hajian-Tilaki and J. A. Hanley. "Comparisson of Three Methods for Estimating the Standard Error of the Area under the Curve in ROC Analysis of Quantitative Data." *Acad Radiol*, **9**:1278–1285, 2002.

HHJ97. Karim O. Hajian-Tilaki, J. A. Hanley, L. Joseph, and J. P. Collet. "A Comparison of Parametric and nonparametric approaches to ROC analysis of Quantitative Diagnostic Tests." *Medical Decision Making*, pp. 94–102, 1997.

HHM90. B. Hubbards, T. Haley, N. McAuliffe, L. Schaefer, N. Kelem, D. Walcott, R. Feiertag, and M. Schaefer. "Computer System Intrusion Detection." Technical Report RADC-TR-90-413, Trusted Information Systems, Inc, December 1990.

Hil01. Ralf Hildebrandt. "Cain and Abel: Snort and Nmap—two sides of the same coin." *Linux Journal*, **4**:46–53, 2001.

Hin92. T. R. Hinrichs. *Problem solving in open worlds: a case study in design.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1992.

HKS95. Kathleen Hanney, Mark T. Keane, Barry Smyth, and Padraig Cunningham. "Systems, Tasks and Adaptation Knowledge: revealing some revealing dependencies." In M. Veloso and A. Aamodt, editors, *Case-Based Reasoning Research and Development*, Lecture Notes in Artificial Intelligence, LNAI 1010, pp. 461–470. Springer-Verlag, 1995.

HL98. John Howard and Thomas Longstaff. "A Common Language for Computer Security Incidents." Technical Report SAND98-8667, SNL, 1998.

HLF01. Joshua W. Haines, Richard P. Lippmann, David J. Fried, Eushiuan Tran, Steve Boswell, and Marc A. Zissman. "1999 DARPA Intrusion Detection System Evaluation: Design and Procedures." Technical report, MIT Lincoln Laboratory, 2001.

HLM90. Richard Heady, George Luger, Arthur Maccabe, , and Mark Servilla. "The architecture of a network level intrusion detection system." Technical Report CS90-20, Department of Computer Science, University of New Mexico, 1990.

HM82. J. A. Hanley and B. J. McNeil. "The meaning and use of the area under a receiver operating characteristic ROC curve." *Radiology*, **143**:29–36, 1982.

HMP02. J. L. Hellerstein, S. Ma, and C. S. Perng. "Discovering Actionable Patterns in Event Data." *IBM Systems Journal*, **41**(3):475–492, 2002.

HRT03. Joshua Haines, Dorene Kewley Ryder, Laura Tinnel, and Stephen Taylor. "Validation of Sensor Alert Correlators." *IEEE Security & Privacy*, **1**(1):46–56, 2003.

HS97. Cecil Huang and Ross Schachter. "Alarms for Monitoring: A Decision-Theoretic Framework." Technical Report SMI-97-0664, Section on Medical Informatics, Stanford University School of Medicine, 1997.

HSV99. Masum Hasan, Binary Sugla, and Ramesh Viswanathan. "A Conceptual Framework for Network Mangement Event Correlation and Filtering Systems." In *Proceedings of IEEE/IFIP International Symposium on Integrated Network Management*, 1999.

HT01a. David J. Hand and Robert J. Till. "A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems." *Machine Learning*, **45**:171–186, 2001.

HT01b. Trevor Hastie and Robert Tibshirani. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer, 2001.

Ilg93. K. Ilgun. "USTAT: A real-time intrusion detection system for Unix." In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pp. 16–28, May 1993.

Jac97. Michel Jacynski. "A Framewok for the Management of Past Experiences with Time-Extended Situations." In *Sixth ACM Conference on Information and Knowledge Management*, 1997.

Jac99. Kathlee A. Jackson. "Intrusion Detection System (IDS) Product Survey." Technical Report LA-UR-99-3883, Computer Research and Applications Group, Los Alamos National Laboratory, 1999.

Jan94. Klaus P. Jantke. "Nonstandard concepts of similarity in case-based reasoning." In Hans-Hermann Bock, Wolfgang Lenski, and Michael M. Richter, editors, *Information Systems and Data Analysis: Prospects, Foundations, and Applications, Proceedings of the 17th Annual Conference of the GfKl, Univ. of Kaiserslautern*, pp. 28–43, Kaiserslautern, 1994. Springer, Berlin.

JAS02. Martha Dorum Jaere, Agnar Aamodt, and Pal Skaalle. "Representing temporal knowledge for case-based prediction." In *Advances in case-based reasoning; 6th European Conference, ECCBR 2002*, Lecture Notes in Artificial Intelligence, LNAI 2416, pp. 174–188. Springer, 2002.

JBV98. D. Jones, T. Bench-Capon, and P. Visser. "Methodologies for Ontology Development.", 1998.

JC. Guofei Jiang and George Cybenko. "Temporal and Spatial Distributed Event Correlation for Network Security.".

JD02. Klaus Julisch and Marc Dacier. "Mining Intrusion Detection Alarms for Actionable Knowledge." In *ACM SIGKDD, Edmonton, Alberta, Canada*, 2002.

Jel97. Frederick Jelinek. *Statistical Methods for Speech Recognition*. The MIT Press, 1997.

Jen95. Nick R. Jennings. "Commitments and Conventions: The foundation of coordination in multi-agent systems." *The Knowledge Engineering Review*, **8**(3):223–250, 1995.

JJ94. John R. Josephson and Susan G. Josephson. *Abductive Inference: Computation, Philosophy, and Technology*. Cambridge, 1994.

JN98. Marko Junkkari and Marko Niinimäki. "A Path-Oriented Approach to Hierarchical Concept Structures." Technical Report A-1998-4, Department of Computer Science, University of Tamper, 1998.

JS00. Anita K. Jones and Robert S. Sielken. "Computer System Intrusion Detection: A Survey." Technical report, Department of Computer Science. University of Virginia, 2000.

JT97a. Michel Jacynski and B. Trousse. "CBR*Tools: an object oriented library for indexing cases with behavioral situation." Technical Report 3215, INRIA, 1997.

JT97b. Michel Jaczynski and Brigitte Trousse. "BROADWAY: A World Wide Web Browsing Advisor Reusing Past Navigations from a Group of Users." In *Proceedings of the 3rd UK Worksop on Case-Based Reasoning, Manchester, September*, 1997.

Jun99. Dieter Jungnickel. *Graphs, Networks, and Algorithms*. Springer, 1999.

JW93. Gabriel Jakobson and Mark . Weissman. "Alarm Correlation: Correlating Multiple Network Alarms Improves Telecommunications Network Surveillance and Fault Management." *IEEE Network*, November 1993.

KA86. Henry Kautz and James Allen. "Generalized plan recognition." In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pp. 32–37, 1986.

Kad02. Mohammed Waleed Kadous. *Temporal Classification: Extending the Classification Paradigm to Multivariate Time Series*. PhD thesis, School of

Computer Science and Engineering, The University of New South Wales, 2002.

Kas94. Ales Kass. "Tweaker: Adapting Old Explanations to New Situations." In Roger C. Schank, Alex Kass, and Christopher K. Riesbeck, editors, *Inside Case-Based Explanation*, chapter 8, pp. 263–296. Lawrence Erlbaum Associates, 1994.

KBL88. L. Kopeikina, R. Brandau, and A. Lemmon. "Case-Based Reasoning for Continuous Control." In *Proceedings of a Workshop on Case-Based Reasoning*, pp. 250–259, 1988.

KC01. Boris Kerkez and Michael T. Cox. "Incremental Case-based Keyhole Plan Recognition Using State Indices." In *Case-Based Reasoning Research and Development: Proceedings of the Fourth International Conference on Case-Based Reasoning, ICCBR-01*, 2001.

KCH04. Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. "Segmenting Time Series: A Survey and Novel Approach." In Mark Last, Abraham Kandel, and Horst Bunke, editors, *Data Mining in Time Series Databases*, volume 57 of *Series in Machine Perception and Artificial Intelligence*. World Scientific, 2004.

Ker03. Boris Kerkez. *Incremental Case-Based Plan Recognition With Incomplete Plan Libraries*. PhD thesis, Wright State University, 2003.

KH97. Andrew P. Kosoresow and Steven A. Hofmeyr. "Intrusion Detection via System Call Traces." *IEEE Software*, **14**(5):35–42, 1997.

KHM97. Miroslav Kubat, Robert Holte, and Stan Matwin. "Learning when Negative Examples Abound." In *European Conference on Machine Learning*, 1997.

KJN94. K.ÊErol, J.ÊHendler, and D.ÊS. Nau. "HTN planning: Complexity and expressivity." In *Proceedings of the National Conference on Artificial Intelligence*, 1994.

Kle99. Mika Klemettinen. *A Knowledge Discovery Methodology for Telecommunication Network Alarm Databases*. PhD thesis, Deparment of Computer Science. University of Helsinki, 1999.

KLP97. Daphne Koller, Alon Levy, and Avi Pfeffer. "P-CLASSIC: A tractable probabilistic Description Logic." In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pp. 390–397, 1997.

Knu98. Donald E. Knuth. *The Art of Computer Programming. Volume 3. Sorting and Searching. Second Edition*. Addison Wesley, 1998.

Kol88. Janet Kolodner. "Retrieving events from a case memory: a parallel implementation." In *Proceedings of a Workshop on Case-Based Reasoning, San Mateo, CA*, 1988.

Kol93. Janet Kolodner. *Case-Based Reasoning*. Morgan Kaufmann Publishers, Inc., 1993.

Koz03. Jack Koziol. *Intrusion Detection with Snort*. SAMS, 2003.

Kru83. Joseph B. Kruskal. "An Overview of Sequence Comparison." In *Time Warps, String Edits, and Macromolecules. The Theory and Practice of Sequence Comparisson*. Addison-Wesley, 1983.

KS95. Irene Katzela and Misha Schwartz. "Schemes for Faul Identification in Communication Networks." *IEEE Transactions on Networking*, **3**(6):753–764, 1995.

KS01. Mark T. Keane and Barry Smyth. "Dynamic Similarity: A Processing Perspective on Similarity." In *Similarity and Categorisation*. Oxford University Press, 2001.

Kum95. S. Kumar. *Classification adn Detection of Computer Intrusions*. PhD thesis, Department of Computer Science, Purdue University, 1995.

KW87. Johan de Kleer and Brian C. Willians. "Diagnosing Multiple Faults." *Artificial Intelligence*, **32**:97–130, 1987.

KYY95. S. Kliger, S. Yemini, Y. Yemini, D. Ohsie, and S. Stolfo. "A Coding Approach to Event Correlation." In *Proceedings of 4th International Symposium on Integrated Network Management (IFIP/IEEE)*, May 1995.

KZ03. Tomas Kocka and Nevin L. Zhang. "Effective Dimensions of Partially Observed Polytrees." In *European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, 2003.

LA01. Maxim Likhachev and Ronald C. Arkin. "Spatio-Temporal Case-Based Reasoning for Behavioral Selection." In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pp. 1627–1634, 2001.

Lam00. Patrick Lambrix. *Part-Whole Reasoning in an Object-Centered Framework*. Lecture Notes in Artificial Intelligence 1771. Springer, 2000.

Lan00. Terran Lane. *Machine Learning Techniques for the Computer Security Domain of Anomaly Detection*. PhD thesis, Purdue University, 2000.

LB99. Terran Lane and Carla E. Brodley. "Temporal Sequence Learning and Data Reduction for Anomaly Detection." *ACM Transactions on Information and System Security*, **2**(3):295–331, 1999.

Lea92. David B. Leake. *Evaluating Explanations: A Content Theory*. Lawrence Erlbaum Associates, 1992.

Lea94. David B. Leake. "Accepter: Evaluating Explanations." In Roger C. Schank, Alex Kass, and Christopher K. Riesbeck, editors, *Inside Case-Based Explanation*, chapter 6, pp. 167–206. Lawrence Erlbaum Associates, 1994.

Lea95. D.B. Leake. "Abduction, experience, and goals: A model of everyday abductive explanation." *Journal of Experimental and Theoretical Artificial Intelligence*, 1995.

Lea96. David B. Leake, editor. *Case-Based Reasoning: Experiences, Lessons, and Future Directions*. AAAI Press/The MIT Press, 1996.

Les98. Victor R. Lesser. "Reflections on the Nature of Multi-Agent Coordination and Its Implications for an Agent Architecture." *Autonomous Agents and Multi-Agent Systems*, **1**:89–111, 1998.

Lev66. V. I. Levenshtein. "Binary codes capable of correcting deletions, insertions, and reversals." *Cybernetics and Control Theory*, **10**(8):707–710, 1966.

Lew93. L. Lewis. "A Case Base Reasoning Approach to the Resolution of Faults in Communications Networks." In *Proceedings of Third International Symposium on Integrated Network Management*, 1993.

Lew95. Lundy Lewis. *Managing Computer Networks. A Case-Based Reasoning Approach*. Artech House Publishers, 1995.

LF93. A. Leinwand and K. Fang. *Network Management: A Practical Perspective*. Addison Wesley, 1993.

LF99. Howard F. Lipson and David A. Fisher. "Survivability - A New Technical and Business Perspective on Security." In *Proceedings of the 1999 New Security Paradigms Workshop*, October 1999.

LFG00. Richard Lippmann, David Fried, Isaac Graf, Joshua Haines, Kristopher Kendall, David McClung, Dan Weber, Seth Webster, Dan Wyschogrod, Robert Cunningham, and Marc Zissman. "Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation." In *Proceedings of the DARPA Information Survivability Conference and Exposition*, Los Alamitos, CA, 2000. IEEE Computer Society Press.

LG94. David D. Lewis and William A. Gale. "A sequential algorithm for training text classifiers." In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 3–12. Springer-Verlag New York, Inc., 1994.

LGW02. Tom Lenaerts, Dominique Gro§, and Richard Watson. "On the Modelling of Dynamical Hierarchies: Introduction to the Workshop WDH 2002." In *ALifeVIII Workshop Proceedings: Modeling Dynamical Hierarchies in Artificial Life*, pp. 37–44, 2002.

Lie89. J. Liebowitz, editor. *Expert System Applications to Telecommunications*. John Wiley and Sons, 1989.

LMP01. John Lafferty, Andrew McCallum, and Fernando Pereira. "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data." In *Proceedings of 18th International Conference on Machine Learning*, pp. 282–289. Morgan Kaufmann, San Francisco, CA, 2001.

LMY99. G. Liu, K. Mok, and E. J. Yang. "Composite Events for Network Event Correlation." In *Proceedings of IEEE/IFIP International Symposium on Integrated Network Management*, 1999.

LP93. Beatriz López and Enric Plaza. "Case-Based Planning for Medical Diagnosis." In *Proceeding of 7th International Symposium on Methodologies for Intelligent Systems (ISMIS-93)*, volume 689 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, 1993.

LP95. Pat Langley and Karl Pfleger. "Case-Based Acquisition of Place Knowledge." In *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 344–352, 1995.

LS97. Cyril Lattimer and Chatherine Stevens. "Some Remarks on Wholes, Parts and their Perception." *Psycoloquy*, **8**, 1997.

LS01. David B. Leake and Raja Sooriamurthi. "When Two Case Bases are Better Than One: Exploiting Multiple Case Bases." In *Case-Based Reasoning Research and Development: Proceedings of the Fourth International Conference on Case-Based Reasoning, ICCBR-01*, 2001.

LS03. David B. Leake and Raja Sooriamurthi. "Dispatching Cases Versus Merging Case-Bases: When MCBR Matters." In *Proceedings of the Sixteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS-2003)*, pp. 129–133, 2003.

Lun90. Teresa F. Lunt. "IDES: An intelligent System for Detecting Intruders." In *Proceedings of the Symposium on Computer Security, Threat and Countermeasures*, 1990.

LZ99. Sushil J. Louis and Yongmian Zhang. "A Sequential Similarity Metric for Case Injected Genetic Algorithms applied to TSPs." In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pp. 377–384, Orlando, Florida, USA, 13-17 1999. Morgan Kaufmann.

Mac89. Damian Mac Randal. "Semantic Networks." In G. A. Ringland and D. A. Duce, editors, *Approaches to Knowledge Representation*, pp. 45–80. John Wiley & Sons, New York, 1989.

MAN97. M.ÊFernandez, A.ÊGomez-Perez, and N.ÊJuristo. "METHODONTOLOGY: From Ontological Art to Ontological Engineering." In *In Workshop on Knowledge Engineering: Spring Symposium Series (AAAI'97)*, pp. 33–44, 1997.

MAN01. Hector Muñoz-Avila, David W. Aha, Dana S. Nau, Rosina Weber, Len Breslow, and Fusun Yamal. "SiN: Integrating Case-based Reasoning with Task Decomposition." In *Seventeenth International Joint Conference on Artificial Intelligence*, pp. 999–1004, 2001.

Mar89. T. D. Marques. "A symptom-driven expert system for isolating and correcting network faults." In E. C. Ericson, L. T. Ericson, and D. Minoli, editors, *Expert Systems Applications in Integrated Network Management*, pp. 251–258. Artech House, 1989.

McH00. John McHugh. "Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Liconln Laboratory." *ACM Transactions on Information and System Security*, **3**(4):262–294, November 2000.

Mci99. M. D. Mcilroy. "A Killer Adversary for Quicksort." *Software–practice and experience*, **29**(0):1–4, 1999.

MCM83. Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, editors. *Machine Learning, An Artificial Intelligence Approach*. Tioga Publishing Company, 1983.

MCV89. R. Mathonet, H. V. Cotthem, and L. Vanryckeghem. "DANTES: An Expert System for Real-Time Network Troubleshooting." In E. C. Ericson, L. T. Ericson, and D. Minoli, editors, *Expert Systems Applications in Integrated Network Management*, pp. 259–262. Artech House, 1989.

MDK97. Alvin Martin, George Doddington, Terri Kamm, Mark Ordowski, and Mark Przybocki. "The DET Curve in Assessment of Detection Task Performance." In *Proc. Eurospeech '97*, pp. 1895–1898, Rhodes, Greece, 1997.

Mei97. Dilmar Malherios Meira. *A Model for Alarm Correlation in Telecommunications Networks*. PhD thesis, Federal University of Minas Gerais, 1997.

MF95. M.ÊGruninger and M.ÊS. Fox. "Methodology for the Design and Evaluation of Ontologies." In *In IJCAI Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995.

MG01. Arthur B. Markman and Dedre Gentner. "Thinking." *Annual Reviews Psychology*, **52**:223–247, 2001.

MK96. Geoffrey McLachlan and Thriyambakam Krshnan. *The EM Algorithm and Extensions*. John Wiley & Sons, 1996.

MM01. Cédric Michel and Ludovic Mé. "ADeLe: an Attack Description Language for Knowledge-based Intrusion Detection." In *Proceedings of the 16th International Conference on Information Security (IFIP/SEC 2001)*, pp. 353–365, June 2001.

MMA90. E. W. Myers, W. Miller, S. F. Altschul, W. Gish, and D. Lipman. "Basic Local Alignment Search Tool." *Journal of Molecular Biology*, **214**, 1990.

MMD02. Benjamin Morin, Ludovic Mé, Herveé Debar, and Mireille Ducassé. "M2D2: A Formal Data Model for IDS Alert Correlation." In *Proc. of the RAID 2002*, 2002.

Mor98. Chris Mortensen. "Perceptual Cognition, Parts and Wholes." *Psycoloquy*, **9**, 1998.

MP01. Sofus A. Macskassy and Foster Provost. "Intelligent Information Triage." In *The 24th Annual International Conference on Research and Development in Information Retrieval*, 2001.

MP03a. Francisco J. Martin and Enric Plaza. "Alba: A cognitive assistant for network administration." *Artificial Intelligence Research and Development. Frontiers in Artificial Intelligence and Applications*, **100**:341–352, 2003.

MP03b. Francisco J. Martin and Enric Plaza. "Alert Triage on the ROC." Technical report, IIIA-CSIC Technical Report 2003-06, 2003. This report is an extended version of the paper published in MMM-ACNS-2003: The Second International Workshop on Mathematical Methods, Models, and Architectures for Computer Networks Security.

MP03c. Francisco J Martin and Enric Plaza. "Alert Triage on the ROC." In *Submitted*, 2003.

MP03d. Francisco J. Martin and Enric Plaza. "Case-based Sequence Analysis for Intrusion Detection." In *Workshop on Applying CBR to Time Series Prediction. The Fifth International Conference on Case-Based Reasoning*, pp. 231–241, 2003.

MP03e. Francisco J. Martin and Enric Plaza. "SOID: an Ontology for Agent-Aided Intrusion Detection." In Vasile Palade, Robert J. Howlett, and Lakhmi Jain, editors, *7th International Conference on Knowledge-based Intelligent Information & Engineering Systems*, volume 2773 of *Lecture Notes in Artificial Intelligence*, pp. 1222–1229. Springer, 2003.

MPA99. Francisco J. Martin, Enric Plaza, and Josep L. Arcos. "Knowledge and Experience Reuse through Communication among Competent (Peer) Agents." *International Journal of Software Engineering and Knowledge Engineering*, **9**(3):319–341, 1999.

MPG96. Luís Macedo, Francisco C. Pereira, Carlos Grilo, and Amílcar Cardoso. "Plans as structured networks of hierarchically and temporally related case pieces." In *Third European Workshop on Case-Based Reasoning (EWCBR'96), Lausanne, Switzerland*, 1996.

MPR98. Francisco J. Martin, Enric Plaza, Juan A. Rodriguez-Aguilar, and Jordi Sabater. "Java Interagents for Multi-Agent Systems." In *AAAI-98 Workshop on Software Tools for Developing Agents*, 1998.

MPR00a. Francisco J. Martin, Enric Plaza, and Juan A. Rodriguez. "An Infrastructure for Agent-Based Systems: an Interagent Approach." *International Journal of Intelligent Systems*, **15**(3):217–240, 2000.

MPR00b. Francisco J. Martin, Enric Plaza, and Juan A. Rodríquez-Aguilar. "Conversation Protocols: Modeling and Implementing Conversation in Agent-Based Systems." In Frank Dignum and Mark Greaves, editors, *Issues in Agent Communication*, pp. 249–263. Springer-Verlag: Heidelberg, Germany, 2000.

MR97. H. Mannila and P. Ronkainen. "Similarity of Event Sequences." In *Fourth International Workshop on Temporal Representation and Reasoning*, pp. 136–139, 1997.

MS99. Chris Manning and Hinrich Schutze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.

MS02. Kevin D. Mitnick and William L. Simon. *The Art of Deception: Controlling the Human Element of Security*. John Wiley & Sons, 2002.

MSL76. C. Metz, S. Starr, and L. Lusted. "Observer performance in detecting multiple radiographic signals: Prediction and analysis using a generalized ROC approach." *Radiology*, pp. 337–347, 1976.

MT00. C. Melchiors and L.M.R. Tarouco. "Troubleshooting Network Faults Using Past Experience." In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pp. 549–562, 2000.

MTV97. Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkano. "Discovery of Frequent Episodes in Event Sequences." Technical report, University of Helsinki, 1997.

MV03. Joaquim Meléndez and Raúl Vilcahuamán. "Refinement of Electric Load Forecasting Reusing Past Instances and Contextual Based Adaptation." In *Workshop on Applying CBR to Time Series Prediction. The Fifth International Conference on Case-Based Reasoning*, pp. 242–251, 2003.

MWA02. Ratul Mahajan, David Wetherall, and Tom Anderson. "Understanding BGP Misconfiguration." In *Proceedings of ACM SIGCOMM 2002*, 2002.

Nak94. G. Nakhaeizadeh. "Learning Prediction from Time Series: A Theoretical and Empirical Comparison of CBR with some other approaches." In *Topics in Case-Based Reasoning*, Lecture Notes in Artificial Intelligence, LNAI 837, pp. 65–76. Springer-Verlag, 1994.

NC02. Peng Ning and Yun Cui. "An Intrusion Alert Correlator Based on Prerequisites of Intrusions." Technical Report TR-2002-01, Department of Computer Science, North Carolina State University, 2002.

NCR02. Peng Ning, Yun Cui, and Douglas S. Reeves. "Analyzing Intesive Intrusion Alerts via Correlation." In Andreas Wespi, Giovanni Vigna, and Luca Deri, editors, *Proceedings of the RAID 2002*, number 2516 in Lecture Notes in Computer Science, pp. 74–94. Springer, 2002.

Ney92. H. Ney. "Stochastic Grammars and Pattern Recognition." In P. Laface and R. De Mor, editors, *Speech Recognition and Understanding: Recent Advances, Trends and Applications*, pp. 319–344. Springer, 1992.

NJW01. Peng Ning, Sushil Jajodia, and Xiaoyang Sean Wang. "Abstraction-Based Intrusion Detection In Distributed Environments." *ACM Transactions on Information and System Security*, **4**(4):407–452, November 2001.

Noe01. Steven Noel. "Development of a Cyber-Defense Ontology." Center for Secure Information Systems George Mason University, Fairfax, Virginia, 2001.

Nor99. Stephen Northcutt. *Network Intrusion Detection. An Analyst's Handbook*. New Riders, 1999.

NWY02. Steven Noel, Duminda Wijesekera, and Charles Youman. "Modern Intrusion Detection, Data Mining, and Degrees of Attack Guilt." In Daniel Barbará and Sushil Jajodia, editors, *Applications of Data Mining in Computer Security*, chapter 1, pp. 1–31. Kluwer Academic Publishes, 2002.

OB97. H. Osborne and D. Bridge. "Models of Similarity for Case-Based Reasoning." In *Proceedings of the Interdisciplinary Workshop on Similarity and Categorization*, 1997.

OK94. David Oshie and Shmuel Kliger. "Network Event Management Survey." Technical report, System Management Arts (SMARTS), 1994.

OMS03. Dirk Ourston, Sara Matzner, William Stump, and Bryan Hopkins. "Applications of Hidden Markov Models to Detecting Multi-Stage Network Attacks." In *In Proceedings of 2003 HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES*, 2003.

OP03. Santiago Ontañón and Enric Plaza. "Collaborative Case Retention Strategies for CBR Agents." In D. Bridge and K. Ashley, editors, *Advances*

in Case-Based Reasoning. Proc. ICCBR 2003, Lecture Notes in Artificial Intelligence, LNAI 2689, pp. 392–406. Springer-Verlag, 2003.

Oul90. M.A. Ould. *Strategies for Software Engineering : The Management of Risk and Quality.* John Wiley, 1990.

Owe94. Christopher C. Owens. "Retriever and Anon: Retrieving Structures From Memory." In Roger C. Schank, Alex Kass, and Christopher K. Riesbeck, editors, *Inside Case-Based Explanation*, chapter 4, pp. 89–126. Lawrence Erlbaum Associates, 1994.

PA96. Enric Plaza and Josep-Lluís Arcos. "Constructive Adaptation." In S. Craw and A. Preece, editors, *Case-based Reasoning Research and Development (ECCBR'2002)*. Springer-Verlag, 1996.

PAM96. Enric Plaza, Josep L. Arcos, and Francisco J. Martin. "Cooperation Modes among Case-Based Reasoning Agents." In *ECAI'96 Workshop on Learning in Distributed AI Systems*, 1996. Also as IIIA-CSIC Technical Report 96-10.

PAM97. Enric Plaza, Josep L. Arcos, and Francisco J. Martin. "Cooperative Case-Based Reasoning." In Gerhard Weiss, editor, *Distributed Artificial Intelligence Meets Machine Learning. Learning in Multi-agent Environments*, number 1221 in Lecture Notes in Artificial Intelligence. Springer-Verlag, 1997.

Pea88. Judea Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufman, 1988.

Pea95. Barak A. Pearlmutter. "Gradient calculation for dynamic recurrent neural networks: a survey." *IEEE Transactions on Neural Networks*, **6**(5):1212–1228, 1995.

Pei48. Charles S. Peirce. "Abduction and Induction." In J. Buchler, editor, *The Philosophy of Peirce: Selected Writings*, chapter 11. Harcourt, Brace and Company, 1948.

PF01. Foster Provost and Tom Fawcett. "Robust Classification for Imprecise Environments." *Machine Learning Journal*, **42**(3), 2001.

PFK98. Foster Provost, Tom Fawcett, and Ron Kohavi. "The Case Against Accuracy Estimation for Comparing Induction Algorithms." In *Fifteenth International Conference on Machine Learning*, 1998.

Pfl02. Karl R. Pfleger. *On-Line Learning of Predictive Compositional Hierarchies*. PhD thesis, Stanford University, 2002.

PFV02. Phillip A. Porras, Martin W. Fong, and Alfonso Valdes. "A Mission-Impact-Based Approach to INFOSEC Alarm Correlation." In *Proc. of*

the RAID 2002, number 2516 in Lecture Notes in Computer Science, pp. 95–114. Springer, 2002.

PL94. Lin Padgham and Patrick Lambris. "A Framework for Part-of Hierarchies in Terminological Logics." In Jon Doyle, Erik Sandewall, and Pietro Torasso, editors, *KR'94: Principles of Knowledge Representation and Reasoning*, pp. 485–496. Morgan Kaufmann, San Francisco, California, 1994.

Pla95. Enric Plaza. "Cases as terms: A feature term approach to the structured representation of cases." In M. Veloso and A. Aamodt, editors, *Case-Based Reasoning, ICCBR-95*, number 1010 in Lecture Notes in Artificial Intelligence, pp. 265–276. Springer-Verlag, 1995.

PLL95. M. V. Nagendra Prasad, Victor R. Lesser, and Susan E. Lander. "Retrieval and Reasoning in Distributed Case Bases." *Journal of Visual Communication and Image Representation, Special Issue on Digital Libraries*, **7**:74–87, 1995. Also as UMASS CS Technical Report 95-27.

PN97. Philip A. Porras and Peter G. Neumann. "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances." In *Proceedings of 1997 National Information Systems Security Conference*, 1997.

PNM99. G. Penido, J. M. Nogueira, and C. Machado. "An automatic fault diagnosis and correction system for telecommunications management." In *Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management*, pp. 777–791, 1999.

Por89. B. W. Porter. "Similarity Assessment: Computation vs. Representation." In *Proceedings of DARPA Case-Based Reasoning Workshop*, pp. 82–84, 1989.

Por92. P. A. Porras. *STAT: A State Transition Analysis Tool for Intrusion Detection*. PhD thesis, University of California - Santa Barbara, 1992.

PR87. Yung Peng and James A. Reggia. "A Probabilistic Causal Model for Diagnostic Problem Solving-Part I: Integrating Symbolic Causal Inference with Numeric Probabilistic Inference." In *IEEE Transactions on Systems, Mans, and Cybernetics*, 1987.

PR90. Yung Peng and James A. Reggia. *Abductive Inference Models for Diagnostic Problem-Solving*. Springer-Verlag, 1990.

Pre97. AAAI Press, editor. *Proceedings of the AAAI Spring Symposium on Ontological Engineering*, 1997.

Pta98. Thomas H. Ptacek. "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection." Technical report, Secure Networks, Inc., 1998.

PW00. D.V. Pynadath and M.P. Wellman. "Probabilistic State-Dependent Grammars for Plan Recognition." In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-2000)*, pp. 507–514, 2000.

RA96. Ashwin Ram and Jr. Anthony G. Francis. "Multi-Plan Retrieval and Adaptation in an Experience-Based Agent." In David B. Leake, editor, *Case-Based Reasoning: Experiences, Lessons, and Future Directions*, chapter 10, pp. 167–183. AAAI Press/The MIT Press, 1996.

Rab89. Lawrence R. Rabiner. "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition." *IEEE*, **77**(2):257–286, 1989.

RAM92. A. Ram, R. C. Arkin, K. Moorman, and R. J. Clark. "Case-Based Reactive Navigation: A Case-Based method for on-line Selection and Adaptation of Reactive Control Parameters in Autonomous Robotic Systems." Technical Report GIT-CC-92/57, College of Computing, Georgia Institute of Technology, 1992.

Ram94. Anshim Ram. "AQUA: Question that Drive the Explanation Process." In Roger C. Schank, Alex Kass, and Christopher K. Riesbeck, editors, *Inside Case-Based Explanation*, chapter 7, pp. 207–262. Lawrence Erlbaum Associates, 1994.

Ran01. Marcus J. Ranum. "Experiences Benchmarking Intrusion Detection Systems." Technical report, NFR Security, 2001.

RB91. R. Rimey and C. Brown. "Controlling Eye Movements with Hidden Markov Models." *International Journal of Computer Vision*, **7**(1):47–66, 1991.

Red90. Michael Redmond. "Distributed Cases for Case-Based Reasoning; Facilitating Use of Multiple Cases." In *Proceedings of AAAI-90*. AAAI Press/MIT Press, 1990.

Reh03. Rafeeq Rehman. *Intrusion Detection with SNORT: Advanced IDS Techniques Using SNORT, Apache, MySQL, PHP, and ACID*. Prentice Hall, 2003.

RH95. Isabelle Rouvellou and George W. Hart. "Automatic Alarm Correlation for Fault Identification." In *Fourteenth Annual Joint Conference of the IEEE Computer and Communication Societies*, pp. 553–561, 1995.

RHT01. Victor Raskin, Christian F. Hempelmann, Katrina E. Triezenberg, and Sergei Nirenburg. "Ontology in information security: a useful theoretical foundation and methodological tool." In *Proc Workshop on New Security Paradigms*, pp. 53–59. ACM Press, 2001.

Ric92. M. M. Richter. *Classification and Learning of Similarity Measures. Studies in Classification, Data Analysis, and Knowledge Organization*. Springer, 1992.

Rie93. M. Riese. *Model-Based Diagnosis of Communication Protocols*. PhD thesis, Swiss federal Institute of Technology at Lausanne, 1993.

Rin89. G. A. Ringland. "Structured Object Representation: Schemata and Frames." In G. A. Ringland and D. A. Duce, editors, *Approaches to Knowledge Representation*, pp. 81–100. John Wiley & Sons, New York, 1989.

RK91. Elaine Rich and Kevin Knight. *Artificial Intelligence*. McGraw-Hill, 1991.

RN95. Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

RN03. Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.

RNC93. Ashwin Ram, S. Narayanan, and Michael T. Cox. "Learning to Troubleshoot: Multistrategy Leaning of Diagnostic Knowledge for A real-world Problem-Solving Task." *Cognitive Science*, **19**:289–340, 1993.

Roe99. Martin Roesch. "Snort - Lightweight Intrusion Detection for Networks." In *Proceedings of LISA '99: 13th Systems Administration Conference Seattle, Washington, USA*, November 1999.

Ron98. Pirjo Ronkainen. "Attribute Similarity and Event Sequence Similarity in Data Mining." Technical report, University of Helsinki, 1998.

Ros98. A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1998.

Rou94. S. Rougegrez. "Similarity evaluation between observed behaviors for the prediction of processes." In *Topics in Case-Based Reasoning*, Lecture Notes in Artificial Intelligence, LNAI 837, pp. 155–166. Springer-Verlag, 1994.

RR91. M. Register and A. Rewari. "CANASTA: The Crash Analysis Troubleshooting Assistant." *Innovative Applications of Artificial Intelligence*, **3**, 1991.

RS89. Christopher K. Riesbeck and Roger C. Schank. *Inside Case-Based Reasoning*. Lawrence Erlbaum Associates, publishers, 1989.

RS93a. Ashwin Ram and Juan Carlos Santamaría. "Continuous Case-Based Reasoning." In *Proceedings of the AAAI-93 Workshop on Case-Based Reasoning*, pp. 86–93, 1993.

RS93b. Ashwin Ram and Juan Carlos Santamaría. "A Multistrategy Case-Based and Reinforcement Learning Approach to Self-Improving Reactive Control Systems for Autonomous Robotic Navigation." In R. S. Michalski and G. Tecuci, editors, *Proceedings of the Second International Workshop on Multistrategy Learning*, 1993.

RTT96. P.R. Roberge, M.A.A. Tullmin, and K.R. Trethewey. "Failure Analysis by Case-Based Reasoning." In *On-line proceeding of Intercorr-96 the 1st online corrosion conference*, 1996.

SA77. Roger Schank and Robert Abelson. *Scripts, Plans, Goals, and Understanding. An Inquiry into Human Knowledge Structures*. Lawrence Eribaum Associates, Inc, 1977.

SAB94. Gerard Salton, James Allan, and Chris Buckley. "Automatic structuring and retrieval of large text files." *Communications of the ACM*, **37**(2):97–108, 1994.

Sat95. U. Sattler. "A concept language for an engineering application with part-whole relations." In *Proceedings of the international workshop on description logics*, pp. 119–123, 1995.

Sat00. U. Sattler. "Description Logics for the Representation of Aggregated Objects." In W. Horn, editor, *Proceedings of the 14th European Conference on Artificial Intelligence*. IOS Press, Amsterdam, 2000.

SBG00. Robert Stevens, Sean Bechhofer, and Carole Goble. "Ontology-based Knowledge Representation for Bioinformatics." *Briefings in Bioinformatics*, **1**(4), Nov 2000.

SCC96. S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. "GrIDS – A Graph-based Intrusion Detection System for Large Networks." In *Proceedings of the 19th National Information Systems Security Conference*, 1996.

Sch86. Roger Schank. *Explanation Patterns: Understanding Mechanically and Creativel*. Lawrence Eribaum Associates, Inc, 1986.

Sch95. Philip A. Schrodt. "Patterns, Rules And Learning: Computational Models Of International Behavior." Technical report, University of Michigan, 1995.

Sch99. Roger C. Schank. *Dynamic Memory Revisited*. Cambridge University Press, 1999.

Sch00. Philip A. Schrodt. "Pattern Recognition of International Crises using Hidden Markov Models." In Diana Richards, editor, *Political Complexity: Nonlinear Models of Politics*, pp. 296–328. University of Michigan Pres, 2000.

SCR98. Miquel Sànchez-Marré, Ulises Cortés, Ignasi R.-Roda, and Manel Poch. "L'Eixample Distance: a New Similarity Measure for Case retrieval." In *1st Catalan Conference on Artificial Intelligence(CCIA'98). ACIA Bulletin*, volume 14-15, pp. 246–253, 1998.

SCR99. Miquel Sànchez-Marré, Ulises Cortés, Ignasi R.-Roda, and Manel Poch. "Sustainable case learning for continuos domains." *Environmental Modelling and Software*, **14**(5):349–357, 1999.

SDM00. John A. Swets, Robyn M. Dawes, and John Monahan. "Psychological Science can Improve Diagnostic Decisions." *Psychological Science in the Public Interest*, **1**(1), 2000.

SDW03. S. Sanghai, P. Domingos, and D. Weld. "Dynamic Probabilistic Relational Models." In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 2003.

SFL00. Sal Stolfo, Wei Fan, Wenke Lee, Andreas Prodromidis, and Philip K. Chan. "Cost-based Modeling for Fraud and Intrusion Dectection: Results from the JAM Project." In *Proceedings of DISCEX*, 2000.

SG01a. Ron Sun and C. Lee Giles. "Sequential Learning: From Recognition and Prediction to Sequential Decision Making." *IEEE Intelligent Systems*, pp. 2–5, July/August 2001.

SG01b. Ron Sun and G. Lee Giles. *Sequence Learning: Paradigms, Algorithms, and Applications*. Springer, 2001. Lecture Notes in Artificial Intelligence 1828.

SG03. Rainer Schmidt and Lothar Gierl. "Prognosis of Approaching Infectious Diseases." In *Workshop on Applying CBR to Time Series Prediction. The Fifth International Conference on Case-Based Reasoning*, pp. 252–259, 2003.

SGK92. K. Sycara, R. Guttal, J. Koning, S. Narasimhan, and D. Navinchandra. "CADET: a Case-based Synthesis Tool for Engineering Design." *International Journal of Expert Systems*, **4**(2), 1992.

Sha90. Jude W. Shavlik. "Case-Based Reasoning with Noisy Case Boundaries: An Application in Molecular Biology." Technical report, University of Wisconsin Computer Sciences Technical Report 988, 1990.

SHP96. R. Schmidt, B. Heindl, B. Pollwein, and L. Gierl. "Abstration of Data and Time For Multiparametric Time Course Prognoses." In I. Smith and B. Faltings, editors, *Advances in Case-Based Reasoning*, Lecture Notes in Artificial Intelligence, LNAI 1168, pp. 377–391. Springer-Verlag, 1996.

Sid85. C. L. Sidner. "Plan Parsing for Intended Response Recognition in Discourse." *Computational Inelligence*, **1**(1):1–10, 1985.

Sim87. Peter Simons. *Parts: A Study in Ontology*. Oxford, 1987.

Sim92. E. Simoudis. "Using Case-Based Retrieval for Customer Technical Support." *IEEE Expert*, **7**(5):7–13, 1992.

SK83. David Sankoff and Joseph B. Kruskal. *Time Warps, String Edits, and Macromolecules. The Theory and Practice of Sequence Comparisson*. Addison-Wesley, 1983.

SK96. Barry Smith and Mark T. Keane. "Design à la Déjà Vu: Reducing the Adaptation Overhead." In David B. Leake, editor, *Case-Based Reasoning: Experiences, Lessons, and Future Directions*, chapter 9, pp. 151–166. AAAI Press/The MIT Press, 1996.

SKR94. Roger C. Schank, Alex Kass, and Christopher K. Riesbeck. *Inside Case-Based Explanation*. Lawrence Erlbaum Associates, 1994.

SL89. Roger Schank and David B. Leake. "Creativity and Learning in a Case-Based Explainer." *Artificial Intelligence*, **40**(1–3):353–385, 1989.

SM96. Tsutomu Shimomura and John Markoff. *Takedown: The Pursuit and Capture of Kevin Mitnick, America's Most Wanted Computer Outlaw-By the Man Who Did it*. Warner Books, 1996.

Spa01. Luca Spalazzi. "A Survey on Case-Based Planning." *Artificial Intelligence Review*, **16**(1):3–36, 2001.

Spi92. M. Spivey. *The Z Notation: A Reference Manual*. Prentice-Hall, 1992.

Sri99. Ashwin Srinivasan. "Note on the Location of Optimal Classifiers in N-Dimensional ROC Space." Technical Report PG-TR-2-99, Oxford University Computing Laboratory, 1999.

SS98. S. Staniford-Chen and D. Schnackenberg. "The Common Intrusion Detection Framework (CIDF)." In *Information Survivability Workshop*, October 1998.

SS01a. M. Steinder and A. Sethi. "The present and future of event correlation: A need for end-to-end service fault localization.", 2001.

SS01b. M. Steinder and A. S. Sethi. "Non-deterministic diagnosis of end-to-end service failures in a multi-layer communication system." In *Proceedings of IEEE International Conference on Computer Communications and Networks (ICCCN)*, pp. 374–379, 2001.

SS02a. M. Steinder and A. Sethi. "Increasing robustness of fault localization through analysis of lost, spurious, and positive symptoms." In *Proceedings of IEEE INFOCOM*, 2002.

SS02b. M. Steinder and A. S. Sethi. "End-to-End Service Failure Diagnosis Using Belief Networks." In *Proceedings of Network Operations and Management Symposium (NOMS)*, pp. 374–379, 2002.

SS03. M. Steinder and A. S. Sethi. "Probabilistic Event-Driven Fault Diagnosis through Incremental Hypothesis Updating." In *Proceedings of IFIP/IEEE Symposium on Integrated Network Management*, 2003.

SSG78. C. F. Schmidt, N. S. Sridharan, and J. L. Goodson. "The plan recognition problem: An intersection of psychology and artificial intelligence." *Artificial Intelligence*, **11**:45–83, 1978.

SSH88. M. M. Sebring, E. Shellhouse, M. E. Hanna, and R. A. Whitehurst. "Expert Systems in Intrusion Detection: A Case Study." In *Proceedings of the 11th National Computer Security Conference*, pp. 74–81, October 1988.

SSR96. Juan Carlos Santamaría, R. S. Sutton, and Ashwin Ram. "Experiments with Reinforcement Learning in Problems with Continuous State and Action Spaces." Technical Report UM-CS-1996-088, Department of Computer Science, University of Massachusetts, 1996.

SSY02. D. Schwartz, S. Stoecklin, and E. Yilmaz. "A Case-Based Approach to Network Intrusion Detection." In *Fifth International Conference on Information Fusion, IF'02, Annapolis, MD, July 7-11*, pp. 1084–1089, 2002.

Str98. Matthew R. Streger. "Prehospital Triage." *Emergency Medical Services*, **1998**:21–21, 1998.

Str02. Oliver Streiter. "Abduction, Induction, and Memorizing in Corpus-based Parsing." In *Proceedings of the ESSLLI-2002 Workshop on Machine Learning Approaches in Computational Lingusistics*, 2002.

SU98. A. Seitz and A.M. Uhrmacher. "The Treatment of Time in a Case-Based Analysis of Experimental Medical Studies." In O. Herzog and A. Guenter, editors, *KI-98: Advances in Artificial Intelligence*, number 1504 in Lecture Notes in Computer Science, pp. 213–222. Springer, 1998.

Sug95. Toshiharu Sugawara. "Reusing past plans in distributed planning." In Victor Lesser, editor, *Proceedings of the First International Conference on Multi–Agent Systems*, pp. 360–367, San Francisco, CA, 1995. MIT Press.

Swe96. John A. Swets. *Signal Detection Theory and ROC Analysis in Psychology and Diagnostics. Collected Papers*. Lawrence Erlbaum Associates, 1996.

SZ00. Eugene H. Spafford and Diego Zamboni. "Intrusion detection using autonomous agents." *Computer Networks*, **34**:547–570, 2000.

TAK02. B. Taskar, P. Abbeel, and D. Koller. "Discriminative Probabilistic Models for Relational Data." In *Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI02)*, 2002.

TCW97. Costas Tsatsoulis, Qing Cheng, and Hsin-Yen Wei. "Integrating Case-Based Reasoning and Decision Thoeory." *IEEE Expert*, 1997.

TD97. Prasad Tadepalli and Thomas G. Dietterich. "Hierarchical Explanation-Based Reinforcement Learning." In *Proceedings of 14th International Conference on Machine Learning*, pp. 358–366. Morgan Kaufmann, 1997.

THC02. Sapon Tanachaiwiwat, Kai Hwang, and Yue Chen. "Adaptive Intrusion Response to Minimize Risk over Multiple Network Attacks." In *Submitted*, 2002.

TJK99. B. Trousse, M. Jaczynski, and R. Kanawati. "Using User Behavior Similarity for Recommandation Computation : The Broadway Approach." In *Proceedings of 8th international conference on human computer interaction (HCI'99)*, 1999.

TK70. A. Tversky and D. H. Krantz. "The dimensional representation and the metric structure of similarity data." *Journal of Mathematical Psychology*, **7**:572–596, 1970.

TL00. Steven J. Templeton and Karl Levitt. "A Requires/Provides Model for Computer Attacks." In *Proceedings of the New Security Paradigms Workshop 2000*, pp. 19–21, September 2000.

Tro00. B. Trousse. "Evaluation of the Prediction Capability of a User Behaviour Mining Approach For Adaptive Web Sites." In *Proceedings of the 6th RIAO Conference—Content-Based Multimedia Information Access, Paris, France*, 2000.

Tve77. A. Tversky. "Features of Similarity." *Psychological Review*, **84**:327–352, 1977.

UG96. Mike Uschold and Michael Grüninger. "Ontologies: principles, methods, and applications." *Knowledge Engineering Review*, **11**(2):93–155, 1996.

UP02. Jeffrey Undercoffer and John Pinkston. "Modeling Computer Attacks: A Target-Centric Ontology for Intrusion Detection." In *CADIP Research Symposium*, 2002.

Uta94. Joachim Utans. "Learning in Compositional Hierarchies: Inducing the Structure of Objects from Data." *Advances in Neural Information Processing Systems*, **6**, 1994.

Var96. A. Varzi. "Parts, Wholes, and Part-Whole Relations: The Prospects of Mereotopology." *Data and Knowledge Engineering. Special Issue on Modeling Parts and Wholes*, **20**(3):259–286, 1996.

VC93. Manuela M. Veloso and Jaime G. Carbonell. "Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization." *Machine Learning*, **10**:249–278, 1993.

VC98. Mark Venguerov and Padraig Cunningham. "Generalised Syntactic Pattern Recognition as a Unifying Approach in Image Analysis." In Adnan Amin,

Dov Dori, Pavel Pudil, and Herbert Freeman, editors, *Advances in Pattern Recognition, Joint IAPR International Workshops SSPR '98 and SPR '98, Sydney, NSW, Australia, August 11-13, 1998, Proceedings*, volume 1451 of *Lecture Notes in Computer Science*, pp. 913–920. Springer, 1998.

VEK00. G. Vigna, S. Eckmann, and R. Kemmerer. "Attack Languages." In *Proceedings of the IEEE Information Survivability Workshop*, 2000.

Vel92. Manuela M. Veloso. *Learning by Analogical Reasoning in General Problem Solving*. PhD thesis, School of Computer Science, Carnigie Mellon University, 1992.

Vel96. Manuela M. Veloso. "Flexible Strategy Learning: Analogical Replay of Problem Solving Episodes." In David B. Leake, editor, *Case-Based Reasoning: Experiences, Lessons, and Future Directions*, chapter 8, pp. 137–149. AAAI Press/The MIT Press, 1996.

Vig03. G. Vigna. "A Topological Characterization of TCP/IP Security." In *Proceedings of the* 12^{th} *International Symposium of Formal Methods Europe (FME '03)*, number 2805 in LNCS, pp. 914–940, Pisa, Italy, September 2003. Springer-Verlag.

Vil90. Marc Vilain. "Getting serious about parsing plans: a grammatical analysis of plan recognition." In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pp. 190–197, 1990.

VS00. Alfoson Valdes and S. Skinner. "Blue Sensors, Sensor Correlation, and Alert Fusion." In Hervé Debar, Ludovic Mé, and Shyhtsun Felix Wu, editors, *Proceedings of Recent Advances in Intrusion Detection International Workshop (RAID 2000)*, volume 1907 of *Lecture Notes in Computer Science*. Springer, 2000.

VS01. Alfonso Valdes and Keith Skinner. "Probabilistic Alert Correlation." In W. Lee, L. Mé, and A. Wespi, editors, *Proceedings of the 4th International Symposium Recent Advances in Intrusion Detection (RAID 2001), Davis, CA, USA*, number 2212 in Lecture Notes in Computer Science. Springer, 2001.

Wat97. Ian Watson. *Applying case-based reasoning: techniques for enterprise systems*. Morgan Kaufmann, 1997.

WB98. Wolfgang Wilke and Ralph Bergmann. "Techniques and Knowledge Used for Adaptation During Case-Based Problem Solving." In *IEA/AIE (Vol. 2)*, pp. 497–506, 1998.

WCH87. Morton E. Winston, Roger Chafin, and Douglas Hermann. "A taxonomy of part-whole relations." *Cognitive Science*, **11**(4):417–444, 1987.

WE02. M. Wood and M. Erlinger. “Intrusion Detection Message Exchange Requeriments.” Internet draft (Work in Progress), 2002.

WE03. M. Wood and M. Erlinger. “Intrusion Detection Message Exchange Requirements.” Internet-Draft, expired: April 22, 2003, 2003.

WF02. S. Wu and P. A. Flach. “Model selection for dynamic processes.” In M. Bohanec, B. Kašek, N. Lavrač, and D. Mladenic, editors, *ECML/PKDD’02 workshop on Integration and Collaboration Aspects of Data Mining, Decision Support and Meta-Learning*, pp. 168–173. University of Helsinki, August 2002.

WFP96. Maj. White, Eric Fisch, and Udo Pooch. “Cooperating Security Managers: A Peer-based Intrusion Detection System.” *IEEE Network*, **10**:20–23, 1996.

WG75. Roger Waters and David Gilmour. “Wish You Were Here.” In *Wish You Were Here*. EMI, 1975.

WH98. Gary M. Weiss and Haym Hirsh. “Learning to Predict Rare Events in Categorical Time-Series Data.” In *AAAI Workshop, Predicting the Future: AI Approaches to Time-Series Problems, Technical Report WS-98-07*, 1998.

Why69. Lancelot Law Whyte. “Structural Hierarchies: A Challenging Class of Physical and Biological Problems.” In Lancelot L. Whyte, Albert G. Wilson, and Donna Wilson, editors, *Hierarchical Structures*. Elsevier Publishing Company, 1969.

Woo91. W. A. Woods. “Understanding Subsumption and Taxonomy: A framework for progress.” In J. F. Sowa, editor, *Principles of Semantic Networks*. Morgan Kaufman Publishers, San Mateo, California, 1991.

WTM95. Andrew J. Weiner, David A. Thurman, and Christine M. Mitchell. “Applying Case-Based Reasoning to Aid Fault Management in Supervisory Control.” In *Proceedings of the 1995 IEEE International Conference on Systems, Man, and Cybernetics*, pp. 4213–4218, 1995.

YKM96. S. Yemini, S. Kliger, E. Mozes, Y. Yemini, and D. Ohsie. “High Speed & Robust Event Correlation.” *IEEE Communications Magazine*, May 1996.

YKP00. J. Ying, T. Kirubarajan, and A. PattersonHine. “A Hidden Markov Model-Based Algorithm for Fault Diagnosis with Partial and Imperfect Tests.” *IEEE transactions on systems, man and cybernetic*, **30**(4), 2000.

Zeh03. Farida Zehraoui. “CBR System for Sequence Prediction "CASEP".” In *Workshop on Applying CBR to Time Series Prediction. The Fifth International Conference on Case-Based Reasoning*, pp. 260–269, 2003.

ZGT02. Cliff Changchun Zou, Weibo Gong, and Don Towsley. "Code red worm propagation modeling and analysis." In *Proceedings of the 9th ACM conference on Computer and communications security*, pp. 138–147. ACM Press, 2002.

ZPW02. Xiaoliang Zhao, Dan Pei, Lan Wang, Dan Massey, Allison Mankin, S. Felix Wu, and Lixia Zhang. "Detection of Invalid Routing Announcement in the Internet." In *International Conference on Dependable Systems and Networks (DSN'02)*, 2002.

ZS03a. Y. Zhu and D. Shasha. "Efficient Elastic Burst Detection in Data Streams." In *The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2003)*, 2003.

ZS03b. Yunyue Zhu and Dennis Shasha. "Efficient Elastic Burst Detection in Data Streams." In *SIGKDD'03*, 2003.

ZXL02. Qingguo Zheng, Ke Xu, Weifeng Lv, and Shilong Ma. "Intelligent Search of Correlated Alarms from Database Containing Noise Data." In *Proceedings of the 8th IEEE/IFIP Network and Operations Management Symposium*, pp. 405–419, 2002.

ZZ02. Chengqu Zhang and Shichao Zhang. *Association Rule Mining*. Springer, 2002.