AAMAS 2010 TORONTO

Toronto, Canada
May 10 - 14, 2010

THE NINTH INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS

Workshop 25:

Optimisation in Multi-agent Systems

Nick Jennings
Sarvapali Ramchurn
Alessandro Farinelli
Juan A. Rodríguez-Aguilar
Alex Rogers
(Editors)

Workshop Organization

Organizing Committee

Program Committee

Table of Contents

Optimal defender allocation for massive security games: A branch and price approach

Manish Jain, Erim Kardeş, Christopher Kiekintveld, Fernando Ordóñez,
Milind Tambe
University of Southern California,
Los Angeles, CA 90089
{manish.jain,kardes,kiekintv,fordon,tambe}@usc.edu

ABSTRACT

Algorithms to solve security games, an important class of Stackelberg games, have seen successful real-world deployment by LAX police and the Federal Air Marshal Service. These algorithms provide randomized schedules to optimally allocate limited security resources for infrastructure protection. Unfortunately, these state-of-the-art algorithms fail to scale-up or to provide a correct solution for massive security games with arbitrary scheduling constraints. This paper provides ASPEN, a branch-and-price algorithm to overcome this limitation based on two key contributions: (i) A column-generation approach that exploits an innovative compact network flow representation, avoiding a combinatorial explosion of schedule allocations; (ii) A branch-and-bound approach with novel upper-bound generation via a fast algorithm for solving under-constrained security games. ASPEN is the first known method for efficiently solving real-world-sized security games with arbitrary schedules. This work contributes to a very new area of work that applies techniques used in large-scale optimization to game-theoretic problems—an exciting new avenue with the potential to greatly expand the reach of game theory.

Categories and Subject Descriptors

I.2.11 [**Computing Methodologies**]: Artificial Intelligence — Distributed Artificial Intelligence - Intelligent Agents

General Terms

Algorithms, Security, Arbitrary Scheduling Problem, Performance

Keywords

Game Theory, Stackelberg Games, Algorithms, Uncertainty, Security, Randomization, Column Generation, Branch and Price, ASPEN, SPARS

1. INTRODUCTION

Algorithms for attacker-defender Stackelberg games, resulting in randomized schedules for deploying limited security resources at airports, subways, ports, and other critical infrastructure have garnered significant research interest [13, 9]. Indeed, two important deployed security applications are using such algorithms: ARMOR and IRIS. ARMOR has been in use for over two years by Los Angeles International Airport police to generate canine-patrol and vehicle-checkpoint schedules [15, 14]. IRIS was recently deployed by the Federal Air Marshals Service (FAMS) to create flight schedules for air marshals [16][1]. These applications are fueled by efficient algorithms that enable scale-up [13, 5, 2, 7] in the input games, with the latest significant scale-up achieved in ERASER-C, the algorithm used in IRIS [9].

Unfortunately, current state of the art algorithms for Stackelberg games are inadequate for larger security scheduling applications [14]. For example, given that US carriers fly over 27,000 domestic and 2,000 international flights daily, generating randomized flight schedules for the limited air marshals resources is a massive scheduling challenge. IRIS addresses an important part of this space — the international sector — but only considers schedules of size two (one departure and one return flight). However, algorithms like ERASER-C in IRIS fail to scale-up and/or to provide a correct solution when air marshals are allowed to fly tours of more than two flights (common in the domestic sector) [9]. The culprit is the exponential explosion in the defender's strategy space in such games caused by the arbitrary size and structure of possible security schedules, and the concomitant combinatorial allocations of security resources to schedules. Indeed, as shown by Korzhyk et al. [10], the problem can be solved in polynomial time only if the schedules are of size 0 or 1, or if there is exactly one resource type for a schedule size of 2, and is NP-hard in general.

Motivated by FAMS and other applications with complex scheduling constraints, including transportation networks and border patrols, this paper presents algorithms for SPARS (*Security Problems with ARbitrary Schedules*) — where there are no special restrictions on the possible schedules. Our main contribution is ASPEN (Accelerated SPars ENgine), a new algorithm for SPARS solving massive Stackelberg security games with arbitrary scheduling constraints. ASPEN is based on the branch and price framework used to solve very large mixed-integer programs, and provides two novel contributions. First, ASPEN uses column generation to avoid representing the full (exponential) strategy space for the defender. To this end, we provide a novel decomposition of the security scheduling problem (SPARS) into a master problem and a network flow subproblem that can be used to generate feasible defender strategies as needed. Second, ASPEN uses a novel branch-and-bound method for searching the space of attacker strategies, achieving significant performance improvements by integrating branching criteria and bounds using the ORIGAMI algorithm [9]. Additionally, we apply column generation to improve the scope of correct application of previous methods such as ERASER-C. We evaluate ASPEN empirically on problems motivated by the FAMS domain, illustrating that this is the first known method for efficiently solving real-world-sized security games with arbitrary schedules.

2. SPARS

[1]FAMS deploys armed air marshals on US commercial aircraft to deter and defeat terrorist/criminal attempts to gain control of the aircraft.

We adopt the model of security games from Kiekintveld et. al. [9], though our work here will focus on the most general type of scheduling problem allowed in this framework. A security game is a two-player game between a defender and an attacker. The attacker's pure strategy space $\mathcal{A}$ is the set of targets T that could be attacked, $T = \{t_1, t_2, \ldots, t_n\}$. The corresponding mixed strategy $\mathbf{a} = \langle a_i \rangle$ is a vector where a_i represents the probability of the adversary attacking t_i. The defender allocates resources of different types $\lambda \in \Lambda$ to protect targets, with the number of available resources given by $R = \{r_1, r_2, \ldots, r_{|\Lambda|}\}$. Each resource can be assigned to a *schedule* covering multiple targets, $s \subseteq T$, so the set of all legal schedules $S \subseteq \mathcal{P}(T)$, where $\mathcal{P}(T)$ represents the power set of T. There is a set of legal schedules for each λ, $S_\lambda \subseteq S$.

The defender's pure strategies are the set of *joint schedules* that assign each resource to at most one schedule. Additionally, we assume that a target may be covered by at most 1 resource in a joint schedule (though this can be generalized). A joint schedule $\mathbf{j}$ can be represented by the vector $\mathbf{P_j} = \langle P_{jt} \rangle \in \{0,1\}^n$ where P_{jt} represents whether or not target t is covered in joint schedule $\mathbf{j}$. We define a mapping M from $\mathbf{j}$ to $\mathbf{P_j}$ as: $M(\mathbf{j}) = \langle P_{jt} \rangle$, where $P_{jt} = 1$ if $t \in \bigcup_{s \in \mathbf{j}} s$; $P_{jt} = 0$ otherwise. The set of all feasible joint schedules is denoted by $\mathbf{J}$. The defender's mixed strategy $\mathbf{x}$ specifies the probabilities of playing each $\mathbf{j} \in \mathbf{J}$, where each individual probability is denoted by x_j. Let $\mathbf{c} = \langle c_t \rangle$ be the vector of coverage probabilities corresponding to $\mathbf{x}$, where $c_t = \sum_{\mathbf{j} \in \mathbf{J}} P_{jt} x_j$ is the marginal probability of covering t.

Payoffs depend on the target attacked and whether or not a defender resource is covering the target. $U_d^c(t)$ denotes the defender's utility if t is attacked when it is covered by a resource of any type. If t is not covered, the defender gets $U_d^u(t)$. Likewise, the attacker's utilities are denoted by $U_a^c(t)$ and $U_a^u(t)$. We assume adding coverage to target t is strictly better for the defender and worse for the attacker: $U_d^c(t) > U_d^u(t)$ and $U_a^c(t) < U_a^u(t)$, however, not necessarily zero-sum. For a strategy profile $\langle \mathbf{c}, \mathbf{a} \rangle$, the expected utilities for the defender and attacker are given by:

$$U_d(\mathbf{c}, \mathbf{a}) = \sum_{t \in T} a_t \left(c_t U_d^c(t) + (1 - c_t) U_d^u(t) \right) \quad (1)$$

$$U_a(\mathbf{c}, \mathbf{a}) = \sum_{t \in T} a_t \left(c_t U_a^c(t) + (1 - c_t) U_a^u(t) \right) \quad (2)$$

We adopt a Stackelberg model in which the defender acts first and the attacker chooses a strategy after observing the defender's mixed strategy. Stackelberg games are common in security domains where attackers can surveil the defender strategy [13]. The standard solution concept is Strong Stackelberg Equilibrium (SSE) [11, 4, 17, 9], in which the leader selects an optimal mixed strategy based on the assumption that the follower will choose an optimal response, breaking ties in favor of the leader.[2] There always exists an optimal pure-strategy response for the attacker, so we restrict our attention to this set in the rest of the paper. The formal definition of a Stackelberg equilibrium is given in Definition 1. The problem of finding Stackelberg equilibria in security games has been shown to be in polynomial time only if the schedule size is 0 or 1, or if the schedule size is 2 and there is exactly one resource type; it is NP-hard in all other settings [10].

DEFINITION 1. *A pair of strategies $\langle \mathbf{C}, g \rangle$ forms a* Strong Stackelberg Equilibrium *(SSE) if they satisfy the following:*

1. *The leader (defender) plays a best-response:* $U_d(\mathbf{c}, g(\mathbf{c})) \geq U_d(\mathbf{c}', g(\mathbf{c}'))$, *for all* $\mathbf{c}'$.
2. *The follower (attacker) plays a best-response:* $U_a(\mathbf{c}, g(\mathbf{c})) \geq U_a(\mathbf{c}, g'(\mathbf{c}))$, *for all* $\mathbf{c}, g'$.
3. *The follower breaks ties optimally for the leader:* $U_d(\mathbf{c}, g(\mathbf{c})) \geq U_d(\mathbf{c}, \tau(\mathbf{c}))$, *for all* $\mathbf{c}$, *where* $\tau(\mathbf{c})$ *is the set of follower best-responses to* $\mathbf{c}$.

Example: Consider a FAMS game with 5 targets (flights), $T = \{t_1, \ldots, t_5\}$, and three marshals of the same type, $r_1 = 3$. Let the set of feasible schedules be as follows:

$$S_1 = \{\{t_1, t_2\}, \{t_2, t_3\}, \{t_3, t_4\}, \{t_4, t_5\}, \{t_1, t_5\}\}$$

Thus, in this example, the set of targets can be thought of as being arranged in a pentagon, where each of the five links corresponds to a schedule. The set of feasible joint schedules is shown below, where column $\mathbf{J}_1$ represents the joint schedule $\{\{t_1, t_2\}, \{t_3, t_4\}\}$. Thus, since targets t_1, t_2, t_3 and t_4 are covered by $\mathbf{J}_1$, $\mathbf{P}_{\mathbf{J}_1}$ has one's in the corresponding entries and 0 corresponding to t_5.

$$\mathbf{P} = \begin{array}{c} \\ t_1: \\ t_2: \\ t_3: \\ t_4: \\ t_5: \end{array} \begin{array}{c} \begin{array}{ccccc} \mathbf{J}_1 & \mathbf{J}_2 & \mathbf{J}_3 & \mathbf{J}_4 & \mathbf{J}_5 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

Each joint schedule in $\mathbf{J}$ assigns only 2 air marshals in this example, since no more than 1 FAM is allowed on any flight. Thus, the third air marshal will remain unused. Suppose all of the targets have identical payoffs $U_d^c(t) = 1$, $U_d^u(t) = -5$, $U_a^c(t) = -1$ and $U_a^u(t) = 5$. In this case, the optimal strategy for the defender randomizes uniformly across the joint schedules, $\mathbf{x} = \langle .2, .2, .2, .2, .2 \rangle$, resulting in coverage $\mathbf{c} = \langle .8, .8, .8, .8, .8 \rangle$. All pure strategies have equal payoffs for the attacker, given this coverage vector.

3. ASPEN SOLUTION APPROACH AND RELATED WORK

The ERASER-C mixed-integer linear program [9] is the most recent algorithm developed for larger and more complex Stackelberg security games [5]. Whereas previous work has focused on patrolling arbitrary topologies using Stackelberg games [2, 13], it has typically focused on a single defender. In contrast, ASPEN and ERASER-C focus on games with large numbers of defenders of different types, handling the combinatorial explosion in the defender's joint schedules. Unfortunately, as the authors note, ERASER-C may fail to generate a correct solution in cases where arbitrary schedules with more than two flights (i.e., multi-city tours) are allowed in the input, or when the set of flights cannot be partitioned into distinct sets for departure and arrival flights. For instance, ERASER-C incorrectly outputs the coverage vector $\mathbf{c} = \langle 1, 1, 1, 1, 1 \rangle$ for the example above (no legal joint schedule can cover more than 4 targets, so it is not possible to cover all targets with probability 1). ERASER-C avoids enumerating joint schedules to gain efficiency, but loses the ability to correctly model arbitrary schedules. Furthermore, ERASER-C only outputs a coverage vector $\mathbf{c}$ and not the distribution $\mathbf{x}$ over joint schedules $\mathbf{J}$ necessary to implement the coverage in practice (though for some restricted cases it is possible to construct $\mathbf{x}$ from $\mathbf{c}$ in polynomial time using the Birkhoff-von Neumann Theorem [10]). New algorithms are needed to solve general SPARS problems that ERASER-C cannot handle.

[2]This tie-breaking rule is counter-intuitive, but the defender can make this response strictly optimal for the attacker by playing a strategy an infinitesimal ϵ away from the SSE strategy.

SPARS problems can be formulated as mixed-integer programs in which adversary strategies are represented by integer variables $\mathbf{a}$ with $a_t = 1$ if target t is attacked and 0 otherwise. Two key computational challenges arise in this formulation. First, the space of possible strategies (joint schedules) for the defender suffers from combinatorial explosion: a FAMS problem with 100 flights, schedules with 3 flights, and 10 air marshals has up to 100,000 schedules and $\binom{100000}{10}$ joint schedules. Second, integer variables are a well-known challenge for optimization. Branch and Price [1] is a framework for solving very large optimization problems that combines branch and bound search with column generation to mitigate both of these problems. Column generation [6] can be viewed as a "double oracle" algorithm [12], and is used to avoid explicitly enumerating all the variables in a large problem (in our problem, these variables represent the joint schedules). This method operates on joint schedules (and not marginal probabilities, like ERASER-C), so it is able to handle arbitrary scheduling constraints directly.

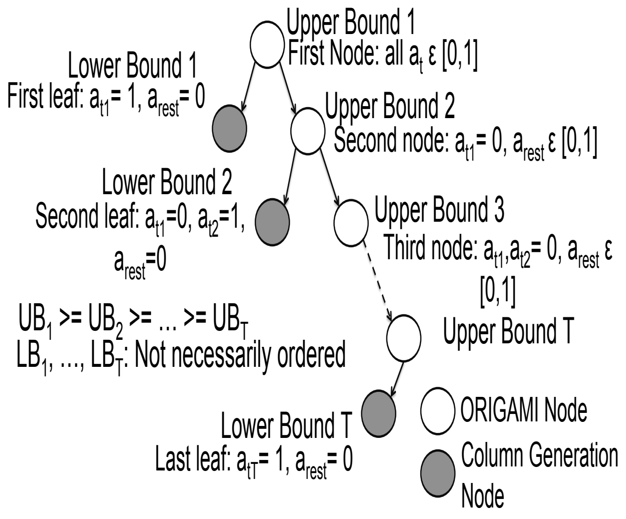

Upper Bound 1
First Node: all $a_t \in [0,1]$
Lower Bound 1
First leaf: $a_{t1}= 1$, $a_{rest}= 0$
Upper Bound 2
Second node: $a_{t1}= 0$, $a_{rest} \in [0,1]$
Lower Bound 2
Second leaf: $a_{t1}=0$, $a_{t2}=1$, $a_{rest}=0$
Upper Bound 3
Third node: $a_{t1}, a_{t2}= 0$, $a_{rest} \in [0,1]$
$UB_1 >= UB_2 >= ... >= UB_T$
$LB_1, ..., LB_T$: Not necessarily ordered
Upper Bound T
Lower Bound T
Last leaf: $a_{tT}= 1$, $a_{rest}= 0$
ORIGAMI Node
Column Generation Node

Figure 1: Working of Branch and Price

An example of branch and price for our problem is shown in Figure 1, with the root representing the original problem. Branches to the left (gray nodes) set exactly one variable t_i in $\mathbf{a}$ to 1 and the rest to zero, resulting in a linear program that gives a lower bound on the overall solution quality. Branches to the right fix variable t_i to zero, leaving the remaining variables unconstrained. An upper bound on solution quality computed for each white node can be used to terminate execution without exploring all of the possible integer assignments. Solving the linear programs in each gray node normally requires enumerating all joint schedules for the defender. Column generation (i.e., pricing) is a technique that avoids this by iteratively solving a restricted *master problem*, which includes only a small subset of the variables, and a *slave problem*, that identifies new variables to include in the master problem to improve the solution.

Unfortunately, branch and price and column generation are not "out of the box approaches" and have only recently begun to be applied in game-theoretic settings [8]. We introduce a novel master-slave decomposition to facilitate column generation for SPARS, including a network flow formulation of the slave problem. We also show experimentally that conventional linear programming relaxations used for branch and bound perform poorly in this domain, and we replace them with novel techniques based on fast algorithms for security games without scheduling constraints.

4. ASPEN COLUMN GENERATION

The linear programs at each leaf in Figure 1 are decomposed into into *master* and *slave* problems for column generation (see Algorithm 1). The master solves for the defender strategy $\mathbf{x}$, given a restricted set of columns (i.e., joint schedules) $\mathbf{P}$. The objective function for the slave is updated based on the solution of the master, and the slave is solved to identify the best new column to add to the master problem, using *reduced costs* (explained later). If no column can improve the solution the algorithm terminates.

Algorithm 1 Column generation employed at each leaf
```
1. Initialize $P$
2. Solve Master Problem
3. Calculate reduced cost coefficients from solution
4. Update objective of slave problem with coefficients
5. Solve Slave Problem
if Optimal solution obtained then
    6. Return $(\mathbf{x}, \mathbf{P})$
else
    7. Extract new column and add to $\mathbf{P}$
    8. Repeat from Step 2
```

4.1 Master Problem

The master problem (Equations 3 to 8) solves for the probability vector $\mathbf{x}$ that maximizes the defender reward (Table 1 describes the notation). This master problem operates directly on columns of $\mathbf{P}$, and the coverage vector $\mathbf{c}$ is computed from these columns as $\mathbf{Px}$. Constraints 4–6 enforce the SSE conditions that the players choose mutual best-responses defined in Definition 1, mirroring similar constraints in ERASER-C. The defender expected payoff (Equation 1) for target t is given by the t^{th} component of the column vector $\mathbf{DPx} + \mathbf{U}_d^u$ and denoted $(\mathbf{DPx} + \mathbf{U}_d^u)_t$. Thus, Equation 4 coupled with Equation 3 corresponds to condition 1, the leader's best response, of Definition 1. Similarly, the attacker payoff for target t is given by $(\mathbf{APx} + \mathbf{U}_a^u)_t$. Constraints 4 and 5 are active only for the single target t^* attacked ($a_{t^*} = 1$). This target must be the adversary's best-response, due to Constraint 6, thus corresponding to condition 2, in Definition 1. The follower will break ties optimally for the leader in this formulation of the master problem, since the formulation will choose that target t^* which is the follower's best response and maximizes the defender payoff.

$$\begin{aligned}
\max \quad & d && (3)\\
\text{s.t.} \quad \mathbf{d} - \; & \mathbf{DPx} - \mathbf{U}_d^u \leq (\mathbf{1} - \mathbf{a})M && (4)\\
\mathbf{k} - \; & \mathbf{APx} - \mathbf{U}_a^u \leq (\mathbf{1} - \mathbf{a})M && (5)\\
& \mathbf{APx} + \mathbf{U}_a^u \leq \mathbf{k} && (6)\\
& \sum_{j \in J} x_j = 1 && (7)\\
& \mathbf{x}, \mathbf{a} \geq 0 && (8)
\end{aligned}$$

4.2 Slave Problem

The purpose of the slave problem is to find the best column to add to the current columns in $\mathbf{P}$. This is done using *reduced cost*, which captures the total change in the defender payoff if a candidate column is added to $\mathbf{P}$. The candidate column with minimum reduced cost improves the objective value the most [3]. The reduced cost $\bar{c}_j$ of variable x_j, associated with column $\mathbf{P_j}$, is given

Table 1: Notation

Variable	Definition	Dimension
$\mathbf{J}$	Joint Schedules	$\|J\|$
T	Targets	$\|T\|$
$\mathbf{P}$	Mapping between T and J	$\|T\| \times \|J\|$
$\mathbf{x}$	Probability Distribution over J	$\|J\| \times 1$
$\mathbf{a}$	Attack vector	$\|T\| \times 1$
d	Defender Reward	-
k	Adversary Reward	-
$\mathbf{d}$	Column vector of d	$\|T\| \times 1$
$\mathbf{k}$	Column vector of k	$\|T\| \times 1$
$\mathbf{D}$	Diag. matrix of $U_d^c(t) - U_d^u(t)$	$\|T\| \times \|T\|$
$\mathbf{A}$	Diag. matrix of $U_a^c(t) - U_a^u(t)$	$\|T\| \times \|T\|$
$\mathbf{U}_d^u$	Vector of values $U_d^u(t)$	$\|T\| \times 1$
$\mathbf{U}_a^u$	Vector of values $U_a^u(t)$	$\|T\| \times 1$
M	Huge Positive constant	-

in Equation 9, where $\mathbf{w}, \mathbf{y}, \mathbf{z}$ and h are dual variables of master constraints 4, 5, 6 and 7 respectively. The dual variable measures the influence of the associated constraint on the objective, defender payoff, and can be calculated using standard techniques [3].

$$\bar{c}_j = \mathbf{w}^T(\mathbf{DP_j}) + \mathbf{y}^T(\mathbf{AP_j}) - \mathbf{z}^T(\mathbf{AP_j}) - h \qquad (9)$$

An inefficient approach would be to iterate through all of the columns and calculate each reduced cost to identify the best column to add. Instead, we formulate a minimum cost network flow (MCNF) problem that efficiently finds the optimal column. Feasible flows in the network map to feasible joint schedules in the SPARS problem, so the scheduling constraints are captured by this formulation. For a SPARS instance we construct the MCNF graph G as follows.

A source node source$_\lambda$ with supply r_λ is created for each defender type $\lambda \in \Lambda$. A single sink node has demand $\sum_{\lambda \in \Lambda} r_\lambda$. Targets in schedule s for resource λ are represented by pairs of nodes $(a_{s_\lambda,t}, b_{s_\lambda,t})$ with a connecting link (so each target corresponds to many nodes in the graph). For every schedule $s_\lambda \in S_\lambda$ we add the following path from the source to the sink:

$$\langle \text{source}_\lambda, a_{s,t_{i_1}}, b_{s,t_{i_1}}, a_{s,t_{i_2}}, \dots, b_{s,t_{i_L}}, \text{sink}\rangle$$

The capacities on all links are set to 1, and the default costs to 0. A dummy flow with infinite capacity is added to represent the possibility that some resources are unassigned. The number of resources assigned to t in a column $\mathbf{P_j}$ is computed as follows:

$$\text{assigned}(t) = \sum_{s \in S} \text{flow}[\text{link}(a_{s,t}, b_{s,t})]$$

Constraints are added to G so that the number of scheduled resources on each target is not greater than 1. Therefore, the added constraints can be written down as follows:

$$\text{assigned}(t) \leq 1 \quad \forall\, t \in T.$$

A partial graph G for our earlier example is shown in Figure 2, showing paths for 3 of the 5 schedules. The paths correspond to schedules $\{t_1, t_2\}$, $\{t_2, t_3\}$ and $\{t_1, t_5\}$. The supply and demand are both 3, corresponding to the number of available FAMS. Double-bordered boxes mark the flows used to compute assigned(t_1) and assigned(t_2). Every joint schedule corresponds to a feasible flow in G. For example, the joint schedule $\{\{t_2, t_3\}, \{t_1, t_5\}\}$ has a flow of 1 unit each through the paths corresponding to schedules $\{t_2, t_3\}$ and $\{t_1, t_5\}$, and a flow of 1 through the dummy. Similarly, any feasible flow through the graph G corresponds to a feasible joint schedule, since all resource constraints are satisfied.

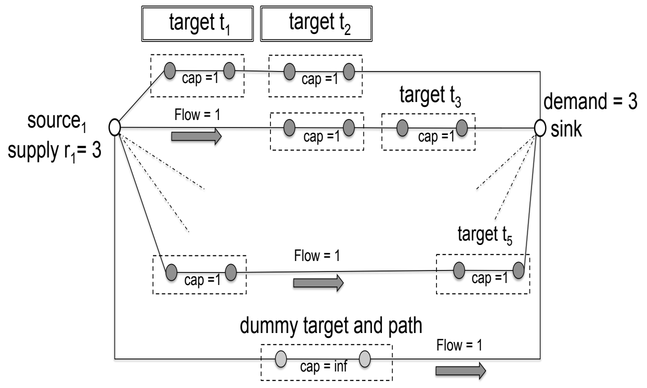


Figure 2: Example Network Graph

It remains to define link costs such that the cost of a flow is the reduced cost for the joint schedule. We decompose $\bar{c}_j$ into a sum of cost coefficients per target, $\hat{c}_t$, so that $\hat{c}_t$ can be placed on links $(a_{s,t}, b_{s,t})$ for all targets t. $\hat{c}_t$ is defined as follows:

$$\begin{aligned} \hat{c}_t &= w_t.D_t + y_t.A_t - z_t.A_t && (10)\\ D_t &= U_d^c(t) - U_d^u(t) && (11)\\ A_t &= U_a^c(t) - U_a^u(t) && (12) \end{aligned}$$

where w_t, y_t and z_t are t^{th} components of $\mathbf{w}, \mathbf{y}$ and $\mathbf{z}$. The overall objective given below for the MCNF problem sums the contributions of the reduced cost from each individual flow and subtracts the dual variable h. If this is non-negative, no column can improve the master solution, otherwise the optimal column (identified by the flow) is added to the master and the process iterates.

$$\min_{\text{flow}} \sum_{(a_{s,t}, b_{s,t})} \hat{c}_t.\text{flow}[(a_{s,t}, b_{s,t})] - h$$

To recap, the entire column generation algorithm employed at each node of the branch and bound tree shown in Figure 1 is given by Algorithm 1 and was described in this section. If the minimum reduced cost obtained at an iteration is non-negative, it indicates that the optimal solution for the current leaf in the tree has been achieved. Otherwise, a new column $\mathbf{P}_j$ is obtained by setting P_{jt} = assigned(t) for all $t \in T$, and the master problem is re-solved.

5. IMPROVING BRANCHING AND BOUNDS

ASPEN uses branch and bound to search over the space of possible attacker strategies. A standard technique in branch and price is to use LP relaxation, i.e. allow the integer variables to take on arbitrary values, to give an optimistic bound on the objective value of the original MIP for each internal node. Unfortunately, our experimental results show that this generic method is ineffective in our domain. We introduce ORIGAMI-S, a novel branch and bound heuristic for SPARS based on ORIGAMI [9], which is an efficient solution method for security games without scheduling constraints and heterogeneous resources. We use ORIGAMI-S to solve a relaxed version of SPARS, and integrate this in ASPEN to give bounds and select branches.

$$
\begin{aligned}
\min \quad & k && (13)\\
\mathbf{U}_a(\mathbf{c}) = \quad & \mathbf{A}\mathbf{c} + \mathbf{U}_a^u && (14)\\
\mathbf{0} \leq \quad & \mathbf{k} - \mathbf{U}_a(\mathbf{c}) && (15)\\
\mathbf{k} - \mathbf{U}_a(\mathbf{c}) \leq \quad & (\mathbf{1} - \mathbf{q}) \cdot M && (16)\\
c_t = \quad & \textstyle\sum_{s \in S} \tilde{c}_{t,s} \quad \forall t \in T && (17)\\
\sum_{s \in S_\lambda} \tilde{c}_{T_\lambda(s),s} \leq \quad & r_\lambda \quad \forall \lambda \in \Lambda && (18)\\
\sum_{t \in T} c_t \leq \quad & L \cdot \textstyle\sum_{\lambda \in \Lambda} r_\lambda && (19)\\
\mathbf{c} \leq \quad & \mathbf{q} && (20)\\
\mathbf{q} \in \quad & \{0, 1\} && (21)\\
\mathbf{c}, c_{t,s} \in \quad & [0, 1] \quad \forall t \in T, s \in S && (22)
\end{aligned}
$$

The ORIGAMI-S model is given in Equations 13–22. It minimizes the attacker's maximum payoff (Equations 13–16). The vector $\mathbf{q}$ represents the *attack set*, and is 1 for every target that gives the attacker maximal expected payoff (Equation 16). The remaining nontrivial constraints restrict the coverage probabilities. ORIGAMI-S defines a set of probabilities $\tilde{c}_{t,s}$ that represent the coverage of each target t in each schedule $s \in S_\lambda$. The total coverage c_t of target t is the sum of coverage on t across individual schedules (Equation 17). We define a set T_λ which contains one target from each schedule $s \in S_\lambda$. The total coverage assigned by resource type λ is upper-bounded by r_λ (Equation 18), analogous to the constraint that the total flow from a source in a network flow graph cannot be greater than the available supply. Total coverage is also bounded by multiplying the number of resources by the *maximum* size of any schedule (L) in Equation 19. The defender can never benefit by assigning coverage to nodes outside of the attack set, so these are constrained to 0 (Equation 20).

ORIGAMI-S is solved once at the beginning of ASPEN, and targets in the attack set are sorted by expected defender reward. The maximum value is an initial upper bound on the defender reward. The first leaf node that ASPEN evaluates corresponds to this maximum valued target (i.e, setting its attack value to 1), and a solution is found using column generation. This solution is a lower bound of the optimal solution, and the algorithm stops if this lower bound meets the ORIGAMI-S upper bound. Otherwise, a new upper bound from the ORIGAMI-S solution is obtained by choosing the second-highest defender payoff from targets in the attack set, and ASPEN evaluates the corresponding leaf node. This process continues until the upper bound is met, or the available nodes in the search tree are exhausted.

THEOREM 1. *The defender payoff, computed by ORIGAMI-S, is an upper bound on the defender's payoff for the corresponding SPARS problem. For any target not in the attack set of ORIGAMI-S, the restricted SPARS problem in which this target is attacked is infeasible.*

Proof Sketch: ORIGAMI and ORIGAMI-S both minimize the maximum attacker payoff over a set of feasible coverage vectors. If there are no scheduling constraints, minimizing the maximum attacker payoff also maximizes the defender's reward [9]. Briefly, the objective of both ORIGAMI and ORIGAMI-S is to maximize the size of the attack set, such that the coverage probability of each target in the attack set is also maximal. Both of these weakly improve the defender's payoff because adding coverage to a target is strictly better for the defender and worse for the adversary.

ORIGAMI-S makes optimistic assumptions about the amount of coverage probability the defender can allocate by taking the maximum that could be achieved by *any* legal joint schedule and allowing it to be distributed arbitrarily across the targets, ignoring the scheduling constraints. To see this, consider the marginal probabilities $\mathbf{c}^*$ of any legal defender strategy for SPARS. There is at least one feasible coverage strategy for ORIGAMI that gives the same payoff for the defender. Constraints 17 and 22 are satisfied by $\mathbf{c}^*$, because they are also constraints of SPARS. Each variable $\tilde{c}_{T_\lambda(s),s}$ in the set defined for Constraint 18 belongs to a single schedule associated with resource type λ, and at most r_λ of these can be selected in any feasible joint schedule, so this constraint must also hold for $\mathbf{c}^*$. Constraint 19 must be satisfied because it assumes that each available resource covers the largest possible schedule, so it generally allows excess coverage probability to be assigned. Finally, constraint 20 may be violated by $\mathbf{c}^*$ for some target t. However, the coverage vector with coverage identical to $\mathbf{c}^*$ for all targets in the ORIGAMI-S attack set and 0 coverage outside the attack set has identical payoffs (since these targets are never attacked).

6. COLUMN GENERATION ON MARGINALS FOR JOINT PATROLLING SCHEDULES

ERASER-C [9] is the algorithm that is used in IRIS and addresses the FAMS security game. However, it only generates the vector $\mathbf{c}$ (*marginals*), which specify the probability of each flight being covered by a federal air marshal. Joint schedules $\mathbf{P}$ and the distribution of over joint schedules $\mathbf{x}$ assigning every federal air marshal to a flight schedule is not provided. While the use of marginals in ERASER-C did provide an exponential scale-up over DOBSS [13], the best prior algorithm, the actual joint schedules still need to be generated from these marginals. In the section, we propose the use of the same column generation approach described in Section 4 to generate these joint schedules from the marginals output by ERASER-C. Thus, the objective of this section is to find joint schedules $\mathbf{P}$ and probability distribution $\mathbf{x}$ such that the obtained coverage vector is given by the output of ERASER-C, and can be formulated as:

$$
\begin{aligned}
\min_x \quad & ||\mathbf{P}\mathbf{x} - \mathbf{c}||_1 \\
& \textstyle\sum_j x_j = 1 \\
& x_j \geq 0
\end{aligned} \qquad (23)
$$

Here, $||\mathbf{P}\mathbf{x} - \mathbf{c}||_1$ refers to the L_1 norm of $||\mathbf{P}\mathbf{x} - \mathbf{c}||_1$, or the sum of the absolute differences between each corresponding term of $\mathbf{P}\mathbf{x}$ and $\mathbf{c}$. L_1 is chosen so as to keep the objective function linear.

The formulation of the associated master problem in this case is given from Equation 24 to Equation 28. The absolute distance between corresponding terms of $\mathbf{P}\mathbf{x}$ and $\mathbf{c}$ is calculated in Equations 25 and 26, and is minimized in the objective as given in Equation 24.

$$
\begin{aligned}
& \min_{x,\gamma} \sum_{t \in T} \gamma_t && (24)\\
s.t. \quad & \mathbf{P}\mathbf{x} - \gamma \leq \mathbf{c} && (25)\\
& \mathbf{P}\mathbf{x} + \gamma \geq \mathbf{c} && (26)\\
& \sum_{t \in T} x_t = 1 && (27)\\
& x \geq 0 && (28)
\end{aligned}
$$

The slave problem in this case is the same as the one used before, where the reduced cost of a joint schedule is:

$$
\begin{aligned}
\bar{c}_j \quad = \quad & -(\mathbf{w}_1 + \mathbf{w}_2)^T \mathbf{P_j} - \sigma && (29)\\
& && (30)
\end{aligned}
$$

where $\mathbf{w}_1, \mathbf{w}_2, \sigma$ are the optimal dual variables of the current master problem associated constraints 25, 26, and 27. Again, the reduced cost $\bar{c}_j$ can be decomposed into reduced cost coefficients per target $\hat{c}_t$, which can be computed using Equation 31.

$$\hat{c}_t = -(w_{1t} + w_{2t}) \tag{31}$$

7. EXPERIMENTAL RESULTS

We evaluate our algorithms on randomly generated instances of the scheduling problem, and provide two sets of results. We first compare the runtime results for ASPEN with regular Branch and Prices and column generation on the output of ERASER-C. We then provide runtime results for ASPEN when the problem size is scaled.

7.1 Comparison on FAMS domain

We compare the runtime performance of ASPEN, branch and price without the ORIGAMI-S heuristic (BnP) and ERASER-C. For this experiment we generate random instances of FAMS problems [9] with schedules of size two, with one departure flight and one arrival flight drawn from disjoint sets, so the set of feasible schedules form a bipartite graph with nodes as flights (for correct operation of ERASER-C). We vary the number of targets, defender resources, and schedules.

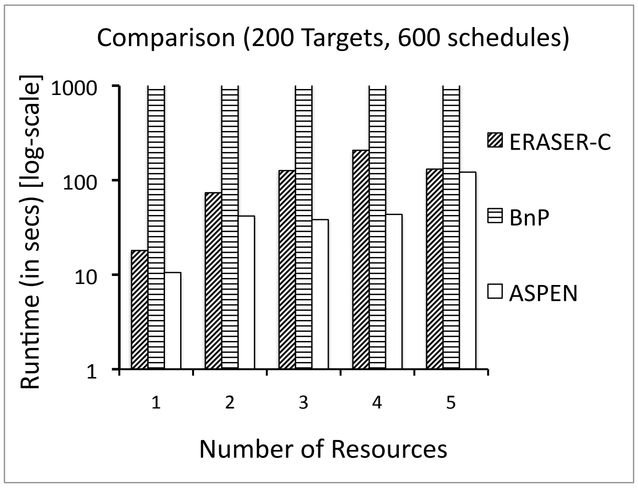


Figure 3: Runtime Comparison Changing Number of Resources

As mentioned earlier, ERASER-C outputs only the coverage vector $\mathbf{c}$. Obtaining the probability distribution $\mathbf{x}$ over joint schedules $\mathbf{J}$ from ERASER-C, specifically when the problem contains heterogeneous resources is a non-trivial challenge given the large numbers of joint schedules. This distribution over joint schedules was obtained using column generation on marginals as described in Section 6. In the following, when we refer to ERASER-C runtimes, they include the time for this column generation in order to allow a head-to-head comparison with ASPEN.

All experiments were based on 15 sample games, and problem instances that took longer than 30 minutes to run were terminated. Results varying the number of defender resources are shown in Figure 3. The y-axis shows the runtime in seconds on the *logarithmic scale*. The x-axis shows the number of resources. ASPEN is the fastest of the three algorithms. The effectiveness of the ORIGAMI-S bounds and branching are clear in the comparison with standard BnP method. The improvement over ERASER-C was an unexpected trend, and can be attributed to the number of columns generated by the two approaches, as shown in Table 2. In fact, ASPEN was 6 times faster than ERASER-C in some instances. Yet it is not the precise amount of speedup, but the fact that ASPEN is extremely competitive with ERASER-C in this specialized domain that is key; for ASPEN can correctly solve a far more general set of security games (SPARS) as we report next.

Table 2: Number of columns: 200 **targets,** 600 **schedules**

Resources	ASPEN	ERASER-C	BnP (max. 30 mins)
10	126	204	1532
20	214	308	1679
30	263	314	1976
40	227	508	1510
50	327	426	1393

We observe similar results in the second and third data sets presented in Figures 4 and 5. Figure 4 shows the results when the number of schedules is changed, where as Figure 5 shows the results when the number of targets is varied. The y-axis in both figures shows the runtime in seconds in a logarithmic scale. The x-axis shows the number of schedules and targets respectively. For example, the average runtime required by ERASER-C when there are 1000 schedules, 200 targets and 10 resources is 29.26 seconds for ERASER-C, 5.34 seconds for ASPEN and the simulation was terminated after 30 minutes for Branch and Price.

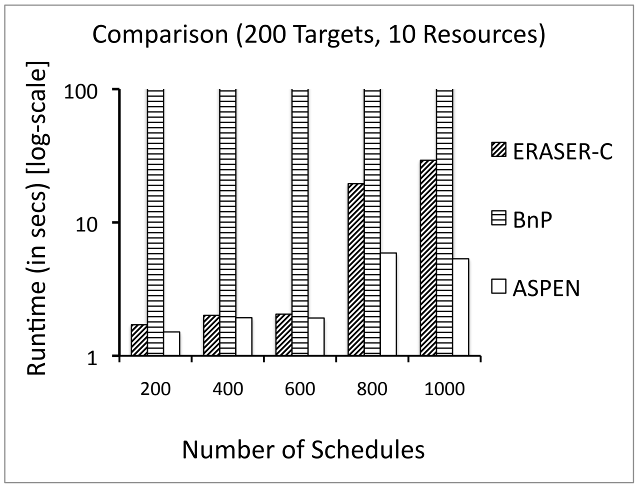
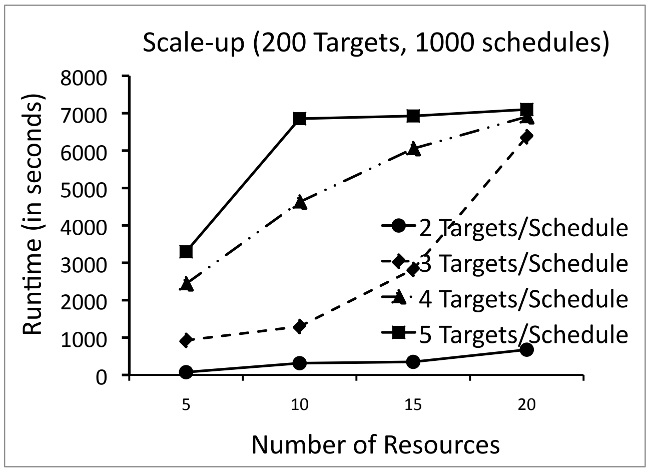


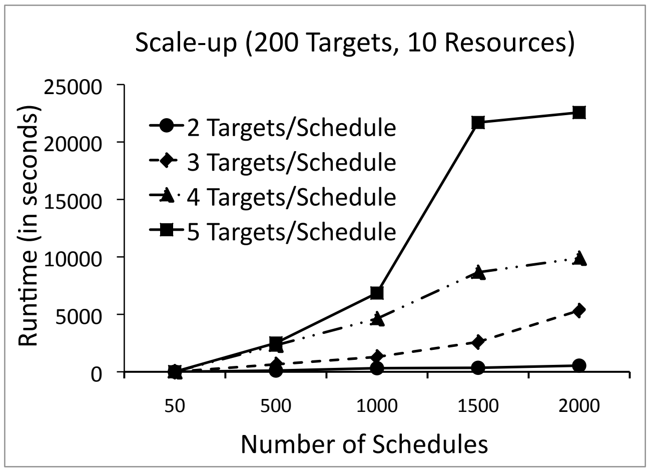
Figure 4: Runtime Comparison Changing Resources and Schedules

7.2 ASPEN on Large SPARS Instances

We also evaluate the performance of ASPEN on arbitrary scheduling problems as the size of the problem is varied to include very large instances. No comparisons could be made because ERASER-C does not handle arbitrary schedules and the only correct algorithms known, DOBSS [13] and BnP, do not scale to these problem sizes. Since ERASER-C does not handle arbitrary schedules, Branch and Price does not scale to these problem sizes (as shown in the previous section) and the only known algorithm (DOBSS [13]) that correctly solves this class of games requires an exponential representation size, ASPEN is the only algorithm capable of solving these instances. We vary the number of resources, schedules, and



(a) Resources



(b) Schedules

Figure 6: Runtime Scale-up Changing Number of Resources and Schedules is varied

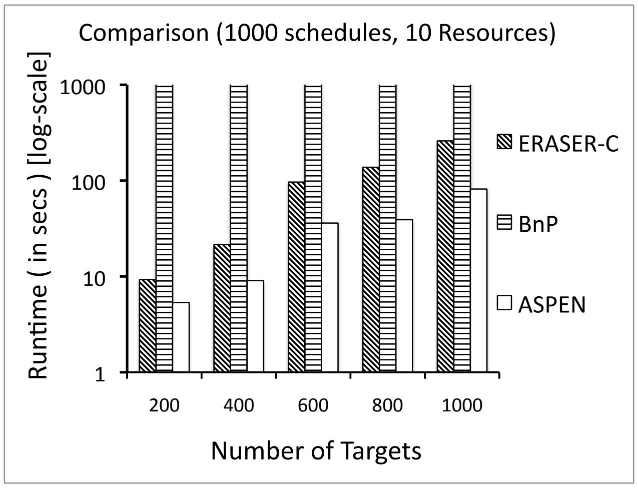


Figure 5: Runtime Comparison Changing Number of Targets

targets as before. In addition, we vary the number of targets per schedule for each of the three cases to test more complex scheduling problems.

Table 3: Number of columns: 200 **targets,** 1000 **schedules**

Resources	3 Tar. / sch.	4 Tar. / sch.	5 Tar. / sch.
5	456	518	658
10	510	733	941
15	649	920	1092
20	937	1114	1124

Figure 6(a) shows the runtime results with 1000 feasible schedules and 200 targets, averaged over 10 samples. The x-axis shows the number of resources, and the y-axis shows the runtime in seconds. Each line represents a different number of schedules per target. The number of joint schedules in these instances can be as large as 10^{23} ($\binom{1000}{10} \approx 2.6 \times 10^{23}$). Interestingly, the runtime does not increase much when the number of resources is increased from 10 to 20 when there are 5 targets per schedules. Column 4 of Table 3 illustrates that the key reason for constant runtime is that the average number of generated columns remains similar.

The graph also shows that increasing the complexity of schedules (i.e., the number of targets per schedule) increases the runtime. This happens because the complexity of the slave problem increases when the complexity of schedules is increased, in turn leading to the generation of more columns before the optimal solution is attained. This leads to the increase in runtime with the increase in complexity of schedules. This can also be seen in Table 3 when looking across a row. For example, the average number of columns required when the number of resources is 5 is $157, 456, 518$ and 658 when there are $2, 3, 4$ and 5 targets per schedule.

Similar trends are obtained in the other two sets of experiments as well. Figure 6(b) shows the runtime results when the number of schedules is increased, whereas Figure 7 shows the results when the number of targets is varied. The y-axes in both the cases shows the runtime in seconds, whereas the x-axis shows the number of schedules and targets respectively.

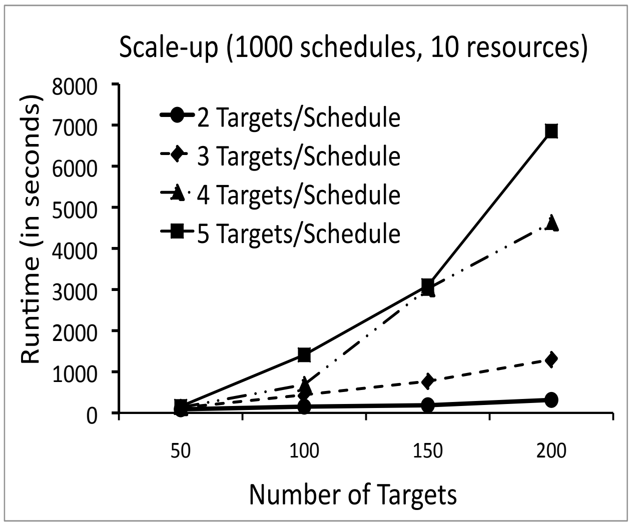


Figure 7: Runtime Scale-up Changing Number of Targets

8. CONCLUSIONS

We present a branch and price method, ASPEN, for solving large-scale Stackelberg security games with arbitrary constraints, including important real-world applications such as FAMS scheduling. ASPEN incorporates several novel contributions, including a decomposition of SPARS to enable column generation and the integration of ORIGAMI-S to substantially speed up the branch and bound search. Experimental results show that ASPEN is competitive with ERASER-C for the restricted class of games where ERASER-C is applicable. More importantly, ASPEN solves far more general instances of scheduling problems where ERASER-C and other existing techniques fail. ASPEN is also substantially faster than a standard implementation of branch and price for this domain. This work contributes to a very new area of work that applies techniques used in large-scale optimization to game-theoretic problems—an exciting new avenue with the potential to greatly expand the reach of game theory.

9. REFERENCES

[1] C. Barnhart, E. Johnson, G. Nemhauser, M. Savelsbergh, and P. Vance. Branch and price: Column generation for solving huge integer programs. In *Operations Research*, volume 46, pages 316–329, 1994.

[2] N. Basilico, N. Gatti, and F. Amigoni. Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In *AAMAS*, pages 500–503, 2009.

[3] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1994.

[4] M. Breton, A. Alg, and A. Haurie. Sequential Stackelberg equilibria in two-person games. *Optimization Theory and Applications*, 59(1):71–97, 1988.

[5] V. Conitzer and T. Sandholm. Computing the optimal strategy to commit to. In *ACM EC-06*, pages 82–90, 2006.

[6] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. In *Operations Research*, volume 8, pages 101–111, 1960.

[7] N. Gatti. Game theoretical insights in strategic patrolling: Model and algorithm in normal-form. In *ECAI-08*, pages 403–407, 2008.

[8] E. Halvorson, V. Conitzer, and R. Parr. Multi-step multi-sensor hider-seeker games. In *IJCAI*, pages 336–341, 2009.

[9] C. Kiekintveld, M. Jain, J. Tsai, J. Pita, M. Tambe, and F. Ordonez. Computing optimal randomized resource allocations for massive security games. In *AAMAS*, pages 689–696, 2009.

[10] D. Korzhyk, V. Conitzer, and R. Parr. Complexity of computing optimal stackelberg strategies in security resource allocation games. *Technical Report, 2010.* Department of Computer Science, Duke University. http://www.cs.duke.edu/∼dima/security_games_bvn.pdf.

[11] G. Leitmann. On generalized Stackelberg strategies. *Optimization Theory and Applications*, 26(4):637–643, 1978.

[12] H. McMahan, G. Gordon, and A. Blum. Planning in the presence of cost functions controlled by an adversary. In *International Conference on Machine Learning (ICML)*, pages 97–104, 2003.

[13] P. Paruchuri, J. P. Pearce, J. Marecki, M. Tambe, F. Ordonez, and S. Kraus. Playing games with security: An efficient exact algorithm for Bayesian Stackelberg games. In *AAMAS-08*, pages 895–902, 2008.

[14] J. Pita, H. Bellamane, M. Jain, C. Kiekintveld, J. Tsai, F. Ordóñez, and M. Tambe. Security applications: Lessons of real-world deployment. In *SIGECOM Issue 8.2*, December 2009.

[15] J. Pita, M. Jain, C. Western, C. Portway, M. Tambe, F. Ordonez, S. Kraus, and P. Parachuri. Deployed ARMOR protection: The application of a game-theoretic model for security at the Los Angeles International Airport. In *AAMAS-08 (Industry Track)*, pages 125–132, 2008.

[16] J. Tsai, S. Rathi, C. Kiekintveld, F. Ordóñez, and M. Tambe. IRIS a tool for strategic security allocation in transportation networks. In *AAMAS-09 (Industry Track)*, 2009.

[17] B. von Stengel and S. Zamir. Leadership with commitment to mixed strategies. Technical Report LSE-CDAM-2004-01, CDAM Research Report, 2004.

Game-Theoretic Allocation of Security Forces in a City

Jason Tsai, Zhengyu Yin, Jun-young Kwak, David Kempe, Christopher Kiekintveld, Milind Tambe

University of Southern California, Los Angeles, CA 90089
{jasontts, zhengyuy, junyounk, dkempe, kiekintv, tambe} @usc.edu

ABSTRACT

Law enforcement agencies frequently must allocate limited resources to protect targets embedded in a network, such as important buildings in a city road network. Since intelligent attackers may observe and exploit patterns in the allocation, it is crucial that the allocations be randomized. We cast this problem as an attacker-defender Stackelberg game: the defender's goal is to obtain an optimal mixed strategy for allocating resources. The defender's strategy space is exponential in the number of resources, and the attacker's exponential in the network size. Existing algorithms are therefore useless for all but the smallest networks.

We present a solution approach based on two key ideas: (i) a polynomial-sized game model obtained via an approximation of the strategy space, solved efficiently using a linear program; (ii) two efficient techniques that map solutions from the approximate game to the original, with proofs of correctness under certain assumptions. We present in-depth experimental results, including an evaluation on part of the Mumbai road network.

Categories and Subject Descriptors

I.2.11 [**Computing Methodologies**]: Artificial Intelligence—*Distributed Artificial Intelligence - Intelligent Agents*

General Terms

Algorithms, Performance, Experimentation, Security, Theory

Keywords

Game Theory, Stackelberg Games, Algorithms, Uncertainty, Security, Randomization, Patrolling, Risk Analysis

1. INTRODUCTION

Protecting targets against potential attacks is an important problem for security forces worldwide. The general setting is as follows: An attacker assigns different values to reaching (and damaging or destroying) one of multiple targets. A defender wants to allocate resources (such as patrol cars or canine units) to capture the attacker before he reaches a target. In many of these situations, the domain has structure that is naturally modeled as a graph. For example, city maps can be modeled with intersections as nodes and roads as edges, where nodes are targets for attackers. In order to prevent attacks, security forces can schedule checkpoints on edges (e.g., roads) to detect intruders. For instance, in response to the devastating terrorist attacks in 2008 [3], Mumbai police deploy randomized checkpoints as one countermeasure to prevent future attacks [1]. The strategy for placing these checkpoints must necessarily be decided in advance of attack attempts, should account for targets of differing importance, and should anticipate an intelligent adversary who can observe the strategy prior to attacking.

In light of these requirements, game-theoretic approaches have been developed to assist in generating randomized security strategies in several real-world domains, including applications in use by the Los Angeles International Airport [14] and the Federal Air Marshals Service [15]. To account for the attacker's ability to observe deployment patterns, these methods model the problem as a Stackelberg game and solve for an optimal probability distribution over the possible deployments to ensure unpredictability. Novel solvers for classes of security games have recently been developed [2, 13, 4]. However, these solvers take time at least polynomial in the number of actions of both players. In our setting, every path from an entry point to a target is an attacker action, and every set of r or fewer edges is a defender action. (r is the maximum number of checkpoints.) Since the attacker's actions grow exponentially with the size of the network, and the defender's actions grow exponentially with r, existing methods quickly become too slow when applied to large real-world domains.

In this work, we develop an efficient procedure for generating checkpoint deployments based on two key ideas: (i) a polynomial-sized approximation of the strategy space solved using a linear program; (ii) two efficient sampling techniques to map solutions back to the original space. To avoid the exponential strategy space over all possible combinations of checkpoint placements (the joint distribution), our methods operate on the marginal probabilities of edges, i.e., the total probability of placing a checkpoint on an edge. Our linear program, RANGER, upper-bounds the capture probabilities along paths by the sum of marginal probabilities.

Our sampling algorithms efficiently generate joint distributions in the original problem space from RANGER's solution of marginal probabilities. We prove that under certain conditions, the actual capture probabilities of our algorithms match the upper bounds of RANGER, and thus necessarily give *optimal* payoff. *Radius Sampling* generates optimal joint distributions if certain conditions on the marginal distribution are met. *Comb Sampling* generates distributions which are optimal against an *approximating attacker* who calculates the expected value of an attack by summing the marginal probabilities on the path.

In addition to our theoretical results, we test our methods empirically. First, we evaluate the quality of RANGER against an optimal solution technique, DOBSS, to verify the accuracy of RANGER's approximation. Then, we evaluate the sampling procedures by testing against an *exact attacker* who plays a best response to the defender's true joint distribution. We also apply our methods to a game model of the city of Mumbai and the targets attacked in 2008.

2. RELATED WORK

Aside from the literature on Stackelberg games for security, our approach is also based on insights from network interdiction [17, 16, 9, 5]. These are the special case of our model when there is a single target, or — equivalently — all targets have identical values. For such games, Washburn and Wood (1995) give an algorithm finding optimal strategies for both players based on Min-Cut computations. However, different target values can cause their algorithm to perform arbitrarily poorly, as we see in our experiments.

Two additional lines of work are somewhat related. Mavronicolas et al. (2008) define and analyze a network security game where each attacker can attack any node of the network, and the defender chooses a path to patrol to capture as many attackers as possible. Because the attacker is not restricted to paths, the types of results for this game are different from ours, and the focus in [12, 11] is on understanding the impact of selfish behavior by defenders rather than optimal strategies. Hider-seeker games [2, 8] are also studied on graphs, but here, the attacker's goal is only to evade capture, not to reach any particular target.

3. PROBLEM DESCRIPTION

A graph-based security game models an attacker and a defender who take actions on a graph $G = (V, E)$, with $n = |V|$ nodes and $m = |E|$ edges. The attacker starts at one of the source nodes $s \in S \subseteq V$ *of his choosing* and travels along a path in an attempt to reach one of the targets $t \in T \subseteq V$. The attacker's pure strategies are thus all s-t paths P, denoted by $\mathcal{B}$, from some source s to some target t. The defender tries to capture the attacker before he reaches a target, by placing up to r resources on *edges* of the graph. The defender's pure strategies are subsets of r or fewer edges; we denote the set of all such sets by $\mathcal{L}$. Assuming that the defender plays $L \in \mathcal{L}$ and the attacker $P \in \mathcal{B}$, the attacker is captured whenever $P \cap L \neq \emptyset$, and succeeds in his attack when $P \cap L = \emptyset$.

Unsuccessful attacks always have a payoff of c for the defender, while successful ones have a penalty of $D(t)$. We make the natural restriction that $D(t) \leq c$. We also assume that the game is zero-sum, meaning that the attacker's payoff for a successful attack on target t is $-D(t)$, and $-c$ for an unsuccessful one. We stress here that targets may have vastly different payoffs associated with them, unlike in [16]. This distinction is crucial to model real security domains, and thus to bridge the gap between theory and practice.

Since the attacker can choose which source to enter from, for our analysis, we merge all sources into a *single source* without loss of generality. More formally, we reform the graph so that all original source-incident edges are incident to the new source. While this operation obviously changes the graph, it does so only in a way that does not impact the game: no rational attacker would ever include multiple sources in his path, and therefore, a defender will never select an edge between two sources. Those are the only edges that disappear from the problem. Thus, to simplify presentation and analysis, we will assume that the attacker always enters from a unique known source s.

In a world of increasingly sophisticated and determined attackers, a good defender strategy must take into account the fact that the attacker will observe and exploit patterns in the defender's behavior. Thus, the game is naturally modeled as a Stackelberg game, an approach also taken (for the same reasons) in past work in security settings [7, 10]. The defender is modeled as the *leader* and moves first, by selecting a mixed strategy $\lambda \in \Lambda$ that assigns a probability to each pure strategy $L \in \mathcal{L}$. The attacker is the *follower* and chooses a strategy after observing the defender's mixed strategy. There is always a pure-strategy best response for the attacker, so we restrict the attacker to pure strategies without loss of generality. Thus, the attacker's Stackelberg strategy is a function $f : \lambda \mapsto P$. For any pair of strategy profiles (λ, f), the expected rewards for the defender (R_D) and attacker (R_A) are given by:

$$\begin{aligned} R_D(\lambda, f) &= p \cdot c + (1-p) \cdot D(t) && (1) \\ R_A(\lambda, f) &= p \cdot -c + (1-p) \cdot -D(t), && (2) \end{aligned}$$

where t is the target at the end of the path specified by $f(\lambda)$, and p the probability that the attacker is captured on the path to t given the defender's strategy λ. Although the optimal defender strategy is a Stackelberg Equilibrium, since our game is zero-sum, this is equivalent to a Maximin strategy [6]. Unfortunately, as $\mathcal{L}$ has size $\Theta(m^r)$, and $\mathcal{B}$ has size exponential in n, existing methods for computing such strategies do not scale to realistic problem sizes. We therefore develop a linear program and two accompanying sampling methods to efficiently solve graph-based security games.

4. RANGER

We first introduce RANGER (Resource Allocation for Network-based Games with Efficient Representations), a linear program for finding an optimal set of *marginal* checkpoint probabilities for the defender. We denote the marginal probability associated with edge e by x_e. Formally, $x_e = \sum_{L \in \mathcal{L}, e \in L} \lambda_L$, where λ_L is the probability of the set L under λ. We denote the marginal distribution by $\vec{x} = \langle x_e \rangle$.

By reasoning over $\vec{x}$, we avoid the exponential size of the defender's space. The key insight of our approach is the following simple consequence of the Union Bound: *For any path P, the capture probability under λ is at most $\sum_{e \in P} x_e$.* We use this upper bound (the sum of x_e) as an approximation of the true capture probability in deriving RANGER. The power of our approach is that we subsequently present ways to sample joint distributions where the total capture probability *matches* this upper bound under certain conditions; this immediately implies optimality of our procedures, and retroactively justifies the approximation.

In the RANGER linear program below, x_e is the marginal probability of placing a checkpoint on edge e. The d_v are, for each vertex v, the minimum sum of checkpoint probabilities along any path from the source s to v[1]. This is enforced by the constraints (4)–(6). Constraint (7) enforces that at most r checkpoints are placed, and Constraint (3) captures the payoff for the defender.

$$\begin{aligned} &\text{Maximize } R_D, \text{ s.t.:} \\ &R_D \leq (1 - d_t) \cdot D(t) + d_t \cdot c && (3) \\ &d_s = 0 && (4) \\ &d_v \leq \min(1, d_u + x_e) \qquad \forall e = (u, v) && (5) \\ &0 \leq x_e \leq 1 \qquad\qquad \forall e \in E && (6) \\ &\sum_{e \in E} x_e \leq r && (7) \end{aligned}$$

Notice that, as specified, the d_v values do not have a lower bound. However, we can show that in an optimal solution, we can, without loss of generality, raise all d_v values to their upper bound, the shortest sum of marginals of a path to v.

THEOREM 1. *Let $(R_D^*, \vec{x}^*, \vec{d}^*)$ be an optimal solution returned by RANGER. Define $\vec{d'} = \langle d'_v \rangle$ for $v \in V$, where $d'_v = \min(1, \min_{P \in \mathcal{P}_v} \sum_{e \in P} x_e)$. $\mathcal{P}_v$ is the set of paths from s to v. Then $(R_D^*, \vec{x}^*, \vec{d'})$ is also an optimal solution.*

[1]Recall that we can assume a single source w.l.o.g.

PROOF. We will show that $(R_D^*, \vec{x}^*, \vec{d'})$ satisfies all RANGER constraints and leads to a reward greater than or equal to R_D^* and, therefore, must also be optimal.

From Constraint 5 in RANGER, for any node v, $d_v^* \leq d'_v$. By definition of $\vec{d'}$, $d'_s = 0$. In addition, consider edge $e' = (u, v)$. Let $P_u = \arg\min_{P \in \mathcal{P}_u} \sum_{e \in P} x_e$. By definition $d'_v \leq 1$. And,

$$d'_v = \min\left(1, \min_{P \in \mathcal{P}_v} \sum_{e \in P} x_e\right) \leq \min\left(1, \sum_{e \in P_u} x_e + x_{e'}\right)$$
$$\leq \min\left(1, \sum_{e \in P_u} x_e\right) + x_{e'} = d'_u + x_{e'}.$$

Hence, $d'_v \leq \min(1, d'_u + x_{e'})$. Finally, given $c \geq D(t)$ and $d'_t \geq d_t^*$,

$$\begin{aligned} R_D^* &\leq (1 - d_t^*)D(t) + (d_t^*)c \\ &\leq (1 - d'_t)D(t) + (d'_t)c. \end{aligned}$$

Thus, $(R_D^*, \vec{x}^*, \vec{d'})$ satisfies all RANGER constraints and leads to a reward greater than or equal to R_D^*. Since R_D^* is the optimal value of RANGER, $(R_D^*, \vec{x}^*, \vec{d'})$ must also be an optimal solution to RANGER. □

Thus, although RANGER may not produce these maximal d_v values, in an optimal solution each d_v can be set to exactly the shortest sum of marginals to v without loss of generality. We will define d_v as such going forward to simplify our analysis.

We verify the claim that RANGER's solution is an overestimate of an optimal solution.

THEOREM 2. *Let λ^* be the optimal strategy and R^* the corresponding defender reward. Then, $R_D^* \geq R^*$, where R_D^* is the defender reward returned by the LP.*

PROOF. Let $\vec{x^*}$ be the marginal probabilities of λ^*. Obviously, $0 \leq x_e^* \leq 1$, and $\sum_{e \in E} x_e^* = \sum_{L \in \mathcal{L}} |L| \cdot \lambda_L^* \leq r$. For each vertex v, let d_v^* be the probability (under λ^*) of capturing the intruder assuming he chooses the best path to reach v. Then $0 \leq d_v^* \leq 1$, and for each edge $e = (u, v)$, $d_v^* \leq d_u^* + x_e^*$, by the Union Bound. The attacker will choose the path to maximize his own payoff R_A^*. Because the game is zero-sum,

$$\begin{aligned} R^* = -R_A^* &= -\max_t\{(1 - d_t^*) \cdot -D(t) + d_t^* \cdot -c\} \\ &= \min_t\{(1 - d_t^*) \cdot D(t) + d_t^* \cdot c\}. \end{aligned}$$

Thus, for any target t, $R^* \leq (1 - d_t^*) \cdot D(t) + d_t^* \cdot c$. Thus, the values R^*, $\vec{d^*}$ and $\vec{x}^*$ are feasible for the LP; because RANGER finds the optimum feasible solution, we obtain that $R_D^* \geq R^*$. □

RANGER is an exponentially more compact representation of both the attacker and defender strategy spaces. This can be seen by noticing that RANGER has a polynomial number of variables with respect to n and m. Any straightforward application of prior formulations would have $\Theta(m^r)$ variables for the defender and exponentially many ($|\mathcal{B}|$) constraints for the attacker.

5. CREATING JOINT DISTRIBUTIONS

To deploy security resources, we require joint schedules, drawn from a joint distribution over $\mathcal{L}$. We develop sampling procedures that use the $\vec{x}$ computed by RANGER to generate a distribution over joint schedules. The principle behind these methods is to bring the *actual* capture probability for a target t up to the value d_t. One way to ensure this would be if no deployment ever placed two checkpoints on any s-t path to any target. More generally (and informally), it is sufficient if this "checkpoint disjointness" is ensured for "critical" s-t paths: those whose sum of marginal probabilities are close to the minimal (d_t).

Notice that placing checkpoints by independently sampling from $\vec{x}$ violates this approach with increasing frequency as r increases, and yields very suboptimal solutions. Instead, we introduce two novel sampling procedures that achieve a certain "checkpoint disjointness", under some assumptions, and are therefore *optimal*.

5.1 Radius Sampling

Radius Sampling (RS) interprets the marginal probabilities as "distances", and places a security ring around the source. In this way, it avoids sampling multiple times on "critical paths", in a way we make precise now. For any $h \geq 0$, we define the *ring of radius* h around s as $R_h := \{e = (u, v) | d_u \leq h < d_v\}$, i.e., the set of edges from a node with probability of capture at most h from s to a node with probability of capture more than h from s.

We define $\alpha := \int_0^\infty |R_h| dh$ (a normalization constant), and the density function $\phi(h) := \frac{|R_h|}{\alpha}$. Notice that

$$\alpha \;=\; \textstyle\sum_{e=(u,v)} (d_v - d_u) \;\leq\; \sum_e x_e \leq r. \qquad (8)$$

Our algorithm works as follows: Choose a radius h from $[0, \infty]$ according to the density function ϕ. Now, choose r of the edges in R_h uniformly at random (or all edges in R_h if $|R_h| \leq r$). Place checkpoints on these edges. We call the resulting set L_R. Notice that both h and L_R are random variables, and L_R is a set of at most r edges.

THEOREM 3. *If for all h, $|R_h| \geq r$ or $|R_h| = 0$, then RS produces an optimal distribution for the defender.*

Theorem 3 follows from Lemma 4 and Theorem 2 as follows: By Lemma 4, the capture probability for any s-t path is at least d_t, i.e., RANGER's value. Therefore, the defender's payoff is at least RANGER's, which by Theorem 2 is at least the payoff with the optimum mixed strategy.

LEMMA 4. *Under the assumptions of Theorem 3, let P be any s-v path and w the node maximizing d_w among all nodes on P. The capture probability along P is at least d_w.*

PROOF. We prove the lemma by induction on $|P|$, the number of edges on path P. In the base case $|P| = 0$, the only node v with a path from s is s itself, and the statement holds.

For the inductive step, let P be a path of length $\ell + 1$ and $e = (v', v)$ the last edge of P. Let $P' = P \setminus \{e\}$ be the path of length ℓ from s to v', and w' the node on P' maximizing $d_{w'}$. By Induction Hypothesis, $\text{Prob}[L_R \cap P' \neq \emptyset] \geq d_{w'}$.

We distinguish two cases. If $d_{w'} \geq d_v$, then

$$\begin{aligned} \text{Prob}[L_R \cap P \neq \emptyset] &\geq \text{Prob}[L_R \cap P' \neq \emptyset] \\ &\geq d_{w'} \geq d_v, \end{aligned}$$

implying the claim.

If $d_v > d_{w'}$, then consider the event $\mathcal{E} = [h > d_{w'}$ and $e \in L_R]$. $\mathcal{E}$ is the event when we include e in L_R and h is sufficiently large that no edge from P' can also be sampled. The probability of $\mathcal{E}$ is

$$\begin{aligned} &\int_{d_{w'}}^{d_v} \text{Prob}[e \in L_R \mid h = x]\phi(x)dx \\ = \;&\int_{d_{w'}}^{d_v} \frac{r}{|R_x|} \cdot \frac{|R_x|}{\alpha} dx = \int_{d_{w'}}^{d_v} \frac{r}{\alpha} dx \geq d_v - d_{w'}. \end{aligned}$$

Here, we substituted the definitions of the sampling process, then used that $\frac{r}{\alpha} \geq 1$ from Equation (8), and that $\text{Prob}[e \in L_R \mid h = x] = \frac{r}{|R_x|}$ using the assumption of Theorem 3.

Whenever L_R intersects P', by definition, we must have that $h \leq d_{w'}$ (because no edge $e' \in P'$ is in R_h for $h > d_{w'}$). Thus, the events $\mathcal{E}$ and $[R_h \cap P' \neq \emptyset]$ are disjoint, and

$$\begin{aligned} \text{Prob}[R_h \cap P \neq \emptyset] &\geq \text{Prob}[[R_h \cap P' \neq \emptyset] \cup \mathcal{E}] \\ &= \text{Prob}[R_h \cap P' \neq \emptyset] + \text{Prob}[\mathcal{E}] \\ &\geq d_{w'} + (d_v - d_{w'}) \\ &= d_v. \end{aligned}$$

The penultimate step used the induction hypothesis as well as the inequality $\text{Prob}[\mathcal{E}] \geq d_v - d_{w'}$ derived above. □

To implement RS, we need to find d_v values to each node, determine the edges and weights for each R_h, and sample according to the above procedure. Each step takes time polynomial in n. Thus, we have a polynomial-time procedure that *optimally* solves a graph-based security game under the conditions of Theorem 3. Previously known techniques either required time exponential in the graph size or can not provide quality guarantees. However, since we cannot guarantee performance when Radius Sampling's condition is not met, we also explore another sampling algorithm.

5.2 Comb Sampling

Now, we consider a somewhat simpler case for the defender: the attacker only observes marginal distributions and approximates the capture probability on any path by adding the probabilities. This may occur because observing the full joint probability distribution is much more time- and resource-intensive than observing the marginals. When only able to observe marginals, adding probabilities is a reasonable and conservative approximation for the attacker.

Comb Sampling (CS) is based on two ideas: (1) If the marginals of the joint distribution match RANGER's x_e values, then an attacker summing probabilities will choose the target t and a path P that is the best path for the attacker to use to reach t as calculated by RANGER. (2) If the edges on P are chosen mutually exclusively, then the capture probability on P matches that of RANGER.

Let $e_1, \ldots, e_{|P|}$ be the edges on the path P (in arbitrary order), and $e_{|P|+1}, \ldots, e_m$ the remaining edges, in arbitrary order. For each $1 \leq j \leq m$, let $X_j = \sum_{i<j} x_i$, and define the interval $I_j = [X_j, X_j + x_j)$. Because $\sum_i x_i = r$ (w.l.o.g.), the I_j form a disjoint cover of the interval $[0, r)$. We now generate a deployment, L_C, as follows: Pick a number $y \in [0, 1)$ uniformly at random, and include in L_C all edges e_j such that $y + k \in I_j$ for some integer k. In other words, include exactly the edges which "own" the intervals containing the points $y, y+1, y+2, \ldots, y+r-1$. This samples exactly r edges.

LEMMA 5. *Given a marginal distribution $\vec{x}$, CS will exactly meet all marginal probabilities, x_e.*

PROOF. Consider any edge e_j, and two cases. If $I_j \subseteq [k, k+1)$ for some k (i.e., I_j contains no integer point), then e_j is included if and only if $k + y \in I_j$, which happens with probability $|I_j| = x_j$. On the other hand, if $I_j = [X_j, k) \cup [k, X_j + x_j)$, then e_j is included if and only if $y+k-1 \in [X_j, k)$ or $y+k \in [k, X_j+x_j)$; because $x_k \leq 1$, this happens with probability $(k - X_j) + (X_j + x_j - k) = x_j$. □

Lemma 5 ensures that the attacker will follow the path predicted by RANGER. Now consider P. If $\sum_{e \in P} x_e \geq 1$, then for any y, some edge $e \in P$ will be included, so the attacker is always captured. This correctly matches the d_t value produced by RANGER, which would also be 1 by Constraint 5. Otherwise, an edge from P is included if and only if $y < \sum_{e \in P} x_e$, which happens with probability $\sum_{e \in P} x_e$, i.e., the sum of marginals on P. Combined with Lemma 5, this guarantees that RANGER's reward is achievable using CS if the defender faces an approximating attacker. The sampling time is clearly polynomial in the graph size. Thus, we have a polynomial-time procedure to optimally defend against an approximating attacker.

6. EXPERIMENTS

6.1 Quality Comparison

Our first evaluation studies the quality of solutions generated by Radius and Comb Sampling in the general case, against both exact and approximating attackers, as defined in the introduction. Neither method is guaranteed to achieve the optimal value against an exact attacker in all cases, so we are interested in whether these methods give good approximations. We compare them against DOBSS, which computes the optimal solution against an exact attacker. DOBSS may not be optimal against an approximating attacker, so we also report the quality of DOBSS against approximating attackers, labeled DOBSS Marginal. As a benchmark, we include a simple Independent Sampling (IS) strategy, wherein for each checkpoint, edge e is selected independently with probability $\frac{x_e}{r}$.

We generate random graphs that are representative of the domains where our methods are most relevant. First, we test on random geometric graphs to estimate performance for road network domains. Then we test on scale-free graphs as an analogy for subway networks. For each graph type, we generate 500 instances each of 4, 5, and 6 nodes, zero-sum games and report results in Table 1. Every graph has one source, between 1 and $|V| - 1$ targets with values from -1 to -10 for the defender when successfully attacked, and 1 to 3 checkpoints. The payoffs for capture are all 0. *Graphs were kept simple so DOBSS could solve them within a reasonable time limit.*

For each graph, we run each sampling method on the marginals produced by RANGER and calculate the *actual* capture probabilities for the joint distribution generated. Using the true capture probabilities, we select an optimal target for the attacker and compute expected payoffs based on this choice. We compare these rewards against DOBSS to evaluate the quality of our methodology.

For DOBSS Marginal, we calculate the marginal probabilities from the joint distribution given and calculate the path to each target with the least (approximate) probability of capture by summing the marginals along each path. Taking the (approximate) expected reward for attacking each target, we can determine an *approximating attacker*'s optimal action and the corresponding reward for the defender.

For Independent Sampling, the actual probability of capture is $1 - (1 - p/r)^r$, where p is the sum of RANGER's marginal probabilities on the edges on the path P. To see this, recall that since r checkpoints are placed independently, and the probability that the j^{th} checkpoint is not on P is $1 - \sum_{e \in P} x_e/r = 1 - p/r$, the probability that there is no checkpoint on P is $(1 - p/r)^r$. Thus, the probability for capture is $1 - (1 - p/r)^r$.

For Radius Sampling, we find all rings R_h and the probability for selecting each. Then, we calculate the probability of selecting edges within each ring. From these probabilities, we then obtain the marginal probabilities for each edge; when the conditions of Theorem 3 are violated, these marginal probabilities might be less than RANGER's values. Finally, we add the marginal probabilities on any path to find the actual probability of capture.

For Comb Sampling, recall that we have a joint probability distribution. For each path, we determine which joint actions place a checkpoint on the path and sum the probabilities associated with these actions. In general, this would be an exponential procedure, since there are exponentially many possible joint actions and exponentially many paths that must be calculated. However, Comb Sampling produces a joint distribution using only $O(m)$ joint actions, making experiments relatively efficient. In practice, we could randomize the order in which the x_e's are processed to create a more complex joint distribution, but experimental evaluation would become computationally infeasible.

For each of the sampling methods, we report the expected defender reward based on the *exact attacker*'s action as determined by the procedures outlined previously. The DOBSS value is simply the expected defender reward calculated by the corresponding MILP.

Table 1 shows the number of cases where a difference in quality of more than 0.001 exists between methods. Empirically, RANGER computes very good estimates of the optimal reward value, never differing from DOBSS for more than 5% of cases. Unsurprisingly, Independent Sampling frequently results in suboptimal distributions (44%–78%). Remarkably, CS attains the optimal expected reward in every single one of the 3,000 graphs tested. RS also performs very well, never differing from DOBSS for more than 11% of the games. DOBSS Marginal never differs from DOBSS or RANGER, indicating that DOBSS remains optimal against an approximating attacker (not shown). However, as runtime experiments will show, DOBSS is completely incapable of solving reasonable game sizes.

	Random Geo.			Scale-Free		
Nodes	4	5	6	4	5	6
RG > DOBSS	12	8	5	0	4	22
IS < DOBSS	220	283	280	389	347	247
CS < DOBSS	0	0	0	0	0	0
RS < DOBSS	0	0	3	0	29	53

Table 1: Results by number of cases (RG - RANGER).

6.2 Mumbai

As a real-world trial, we use our algorithms to create security policies for the southern tip of Mumbai, shown in Figure 1, which was an area of heavy terrorist activity in 2008. The region is modeled as a graph with 35 nodes and 58 edges. Attackers can potentially enter from *any* entry node, chosen based on historical and likely entry points. A total of four target nodes are chosen based on historical attacks, marked with black circles in Figure 1. These are held constant throughout testing. Payoffs are decided as in the Quality Comparison experiments.

Figure 2(a) shows the averaged defender rewards obtained across eight configurations, each with their own setup of target values and sources, with each configuration being run with checkpoints varying from 2 to 10. Figure 2(b) shows results averaged across a different set of eight configurations, each with their own setup of target values and 4 checkpoints, with each configuration being run with the number of sources increasing from 1 to 7.

DOBSS is unable to solve even the simplest case within the 20-minute limit; thus, we include only Comb Sampling and Radius Sampling's expected reward, Minimum Cut, as well as three natural defense strategies, *Uniform Random*, *Entry-Incident* and *Weighted-Target-Incident*. Minimum Cut, as introduced by [16], contracts all sources into a super-source and all targets into a super-target and finds the minimum cut on the resulting graph, uniformly random-

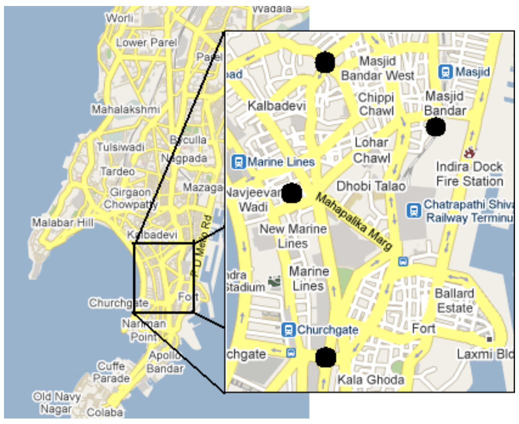


Figure 1: Target layout for southern Mumbai.

izing resources across it. This effectively ignores target value variation, but is extremely efficient. Uniform Random places checkpoints uniformly randomly across all edges in the graph. Entry-Incident places checkpoints on edges incident to entry nodes with equal probability. Weighted-Target-Incident places checkpoints on edges incident to target nodes, weighted according to their payoff. The y-axis shows the expected reward in Figure 2(a) and 2(b). The x-axis of Figure 2(a) shows the number of checkpoints allowed and the number of sources in Figure 2(b). RANGER ran in ≈ 0.2 seconds in all trials.

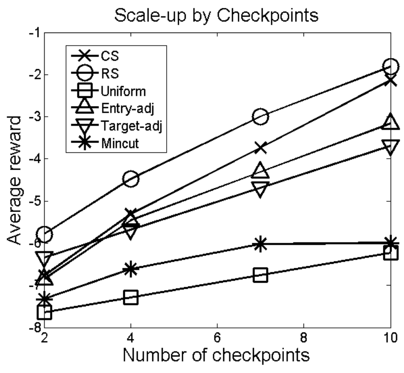
Throughout the tests, the two sampling methods developed here outperformed all others, with RS performing worse than CS when its optimality condition was violated more severely. Minimum Cut, which does not prioritize higher value targets, performs worse than the Uniform Random strategy in some situations, which sometimes happens to place more coverage on higher value targets simply because there are more roads to them.

RANGER actually exploits resources differently and to better effect, which we explore in Figure 3(a) and 3(b). Data was used from the games in Figure 2(b). Figure 3(a) shows the average number of edges with non-zero marginal probability in Entry-Incident and RANGER strategies as the number of sources increases (other methods would be a constant value). As can be seen, RANGER actually uses *fewer* edges than Entry-Incident as the number of entry points increases, but uses them to much better effect.

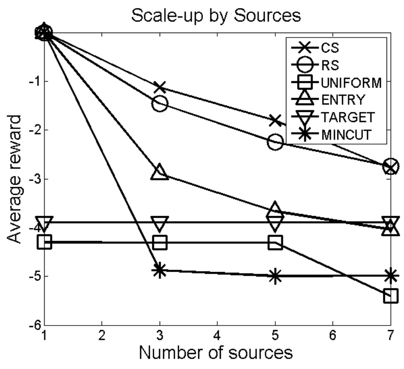
Figure 3(b) shows the standard deviation of marginal probabilities of edges. As expected, Entry-Incident remains constant at 0. RANGER's results vary from 0 (with only one source) to 0.2, which is actually a standard deviation of 20% in marginal probability on edges. Although we cannot provide guarantees on the performance of Radius or Comb Sampling against an exact attacker in general, the techniques yield non-trivial mixed strategies of high quality in practice that outperform the sensible alternatives explored here.

6.3 Runtime Comparison

We have discussed the exponentially smaller solution space RANGER, RS, and CS operate in; we now show runtime comparisons to verify our conclusions empirically. Specifically, we evaluate the runtime performance of RANGER against the fastest-known exact algorithm for solving general Bayesian Stackelberg games, DOBSS [13], as well as a faster solver for security games, ERASER [10]. DOBSS serves as a benchmark, since it provides the optimal solution against exact attackers. ERASER exploits structural proper-

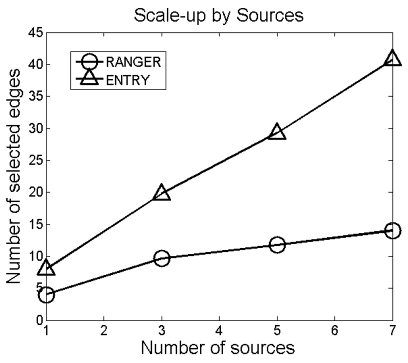


(a) Avg reward by checkpoints.

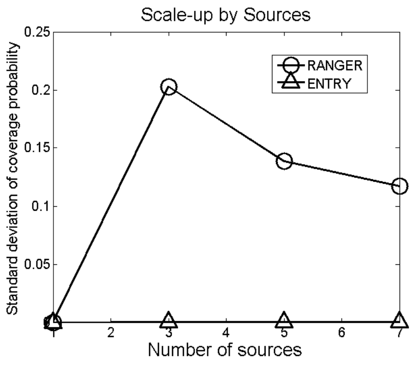


(b) Avg reward by sources.

Figure 2: Rewards in Mumbai domain.



(a) No. edges used.



(b) Standard dev. of Prob.

Figure 3: RANGER strategies in Mumbai domain.

ties that exist in many security domains to create a compact game representation. However, in order for the solution to be correct, it also requires that defender actions be independent from each other, which is not the case in our domain, since placing two checkpoints on one path will violate this assumption. Nevertheless, ERASER serves as another approximation algorithm that runs much more efficiently than DOBSS, so we compare RANGER's runtime against it.

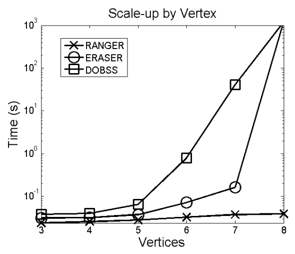
Experiments were run on quad-core Intel 3.0GHz processors with 3GB of RAM. Each approach was run 100 times on each problem, and we report the average time. All algorithms were given a maximum time of 20 minutes.

Figure 4 shows the scaling of each method's runtime with respect to n. In these experiments, we use complete graphs (3 to 8 vertices) and random geometric graphs (4 to 12 vertices), each with one source, one target (value -1 to -10 for defender), and two checkpoints. The x-axis shows the number of vertices in the graph, and the y-axis shows runtime in seconds. Each result is an average of 10 trials. Unsurprisingly, DOBSS and ERASER scale poorly and are only able to solve the problem with up to 7 vertices for complete graphs and 8 (DOBSS) and 11 (ERASER) vertices on random geometric graphs. RANGER is capable of solving games with 400 vertices within the time limit (not shown).

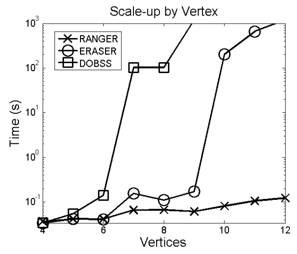
7. CONCLUSIONS

In this work, we provide three primary contributions. First, we develop a linear program, RANGER, to efficiently create optimal marginal distributions for placing checkpoints on the edges of a graph. We prove that the reward found by RANGER is an overestimate of the true optimal reward. Second, we introduce Radius Sampling, which we show produces optimal joint distributions under specific conditions. Third, we develop Comb Sampling, which we prove guarantees optimality against an approximating attacker. We complete the discussion by providing experimental verification of the high quality of our techniques on random graphs as well as a real-world domain.

The techniques introduced here can be put into direct use by city officials such as those in Mumbai, where terrorist attacks of the form we model here are very real and occur with tragic regularity. We show that RANGER strategies implemented using either Radius or Comb Sampling far outperform the simpler alternatives explored. Although we cannot know the current method used by Mumbai police, we offer an efficient, high-quality solution technique based on game-theoretic foundations and hope to transition



(a) Complete graphs.



(b) Random geometric graphs.

Figure 4: Runtimes for RANGER, ERASER, and DOBSS.

this theory into practice in the near future.

8. FUTURE RESEARCH

While RANGER and Comb/Radius Sampling can be used immediately, we see a series of open research questions that could potentially improve upon these results for real-world deployment. One direction would be the generalization to general-sum domains, since it is a subject of debate whether real-world scenarios are always best modeled as zero-sum games.

Also, perhaps one can prove NP-hardness of the problem of finding optimal defender strategies in either our restricted games or in the general-sum cases. Assuming that no efficient algorithm can handle all inputs, it would then be desirable to identify conditions under which either optimality or approximation factors can be guaranteed. Even without guarantees, other practical and efficient heuristics may be of interest to warrant deployment in practice.

Another highly pertinent extension would be to introduce a "probability of capture" for each checkpoint, instead of assuming that a checkpoint will always capturing an attacker using the edge, as we do in our work. This would require major alterations in sampling, since our sampling techniques have focused on never placing two checkpoints along a single s-t path, which may no longer be desirable. All of these directions are critical in effectively implementing game-theoretic methods in real-world security domains such as the Mumbai example presented here and we hope to pursue them to further improve existing security practices.

9. ACKNOWLEDGEMENT

This research was supported by the United States Department of Homeland Security through the Center for Risk and Economic Analysis of Terrorism Events (CREATE) under grant number 2007-ST-061-000001. Any opinions, conclusions or recommendations herein are solely those of the authors and do not necessarily reflect views of the Department of Homeland Security.

10. REFERENCES

[1] S. A. Ali. Rs 18L seized in nakabandi at Vile Parle. *Times of India*, 4 August 2009.

[2] N. Basilico, N. Gatti, and F. Amigoni. Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In *AAMAS-09*, 2009.

[3] R. Chandran and G. Beitchman. Battle for Mumbai ends, death toll rises to 195. *Times of India*, 29 November 2008.

[4] V. Conitzer and T. Sandholm. Computing the optimal strategy to commit to. In *EC '06: PRoceedings of the 7th ACM conference on Electronic commerce*, pages 82–90, New York, NY, USA, 2006. ACM.

[5] K. J. Cormican, D. P. Morton, and R. K. Wood. Stochastic network interdiction. *Oper. Res.*, 46(2):184–197, 1998.

[6] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, 1991.

[7] N. Gatti. Game theoretical insights in strategic patrolling: Model and algorithm in normal-form. In *ECAI-08*, pages 403–407, 2008.

[8] E. Halvorson, V. Conitzer, and R. Parr. Multi-step Multi-sensor Hider-Seeker Games. In *IJCAI*, 2009.

[9] E. Israeli and R. K. Wood. Shortest-path network interdiction. *Networks*, 40(2):97–111, 2002.

[10] C. Kiekintveld, M. Jain, J. Tsai, J. Pita, M. Tambe, and F. Ordóñez. Computing Optimal Randomized Resource Allocations for Massive Security Games. In *AAMAS-09*, 2009.

[11] M. Mavronicolas, V. Papadopoulou, A. Philippou, and P. Spirakis. A network game with attackers and a defender: A survey. In *ECCS*, 2006.

[12] M. Mavronicolas, V. Papadopoulou, A. Philippou, and P. Spirakis. A network game with attackers and a defender. *Algorithmica*, 51(3):315–341, 2008.

[13] P. Paruchuri, J. P. Pearce, J. Marecki, M. Tambe, F. Ordóñez, and S. Kraus. Playing games with security: An efficient exact algorithm for Bayesian Stackelberg games. In *AAMAS-08*, pages 895–902, 2008.

[14] J. Pita, M. Jain, C. Western, C. Portway, M. Tambe, F. Ordóñez, S. Kraus, and P. Parachuri. Deployed ARMOR protection: The application of a game-theoretic model for security at the Los Angeles International Airport. In *AAMAS-08 (Industry Track)*, 2008.

[15] J. Tsai, S. Rathi, C. Kiekintveld, F. Ordóñez, and M. Tambe. IRIS - A Tool for Strategic Security Allocation in Transportation Networks. In *AAMAS-09 (Industry Track)*, 2009.

[16] A. Washburn and K. Wood. Two-person zero-sum games for network interdiction. *Operations Research*, 43(2):243–251, 1995.

[17] R. K. Wood. Deterministic network interdiction. *Mathematical and Computer Modeling*, 17(2):1–18, 1993.

Randomizing Security Activities with Attacker Circumvention Strategies

James Pita
University of Southern California
Los Angeles, CA 90089

Chris Kiekintveld
University of Southern California
Los Angeles, CA 90089

Milind Tambe
University of Southern California
Los Angeles, CA 90089

Michael Scott
University of Southern California
Los Angeles, CA 90089

ABSTRACT

Game theoretic methods for making resource allocation decision in security domains have attracted growing attention from both researchers and security practitioners, including deployed applications at both the LAX airport and the Federal Air Marshals Service. We develop a new class of security games designed to model decisions faced by the Transportation Security Administration and other agencies in protecting airports, ports, and other critical infrastructure. Our model allows for a more diverse set of security activities for the defensive resources than previous work, which has generally focused on interchangeable resources that can only defend against possible attacks in one way. Here, we are concerned in particular with the possibility that adversaries can circumvent specific security activities if they are aware of common security measures. The model we propose takes this capability into account and generates more unpredictable, diverse security policies as a result—without resorting to an external value for entropy or randomness.

Solving these games is a significant computational challenge, and existing algorithms are not capable of solving realistic games. We introduce a new method that exploits common structure in these problems to reduce the size of the game representation and enable faster solution algorithm. These algorithms are able to scale to make larger games than existing solvers, as we show in our experimental results.

Categories and Subject Descriptors

I.0 [**Computing Methodologies**]: General

General Terms

Game Theory, Security

Keywords

Game Theory, Stackelberg, Security, Resource Allocation

1. INTRODUCTION

Security officials face many difficult decisions in how to provide security for critical infrastructure, high-profile events, and other potential targets of criminal or terrorist attacks. Game theory is increasingly viewed as a powerful tool for modeling these decisions, due in part to the ability of these models to account for adaptive adversaries and to identify optimal randomized strategies for security forces. This basic idea has been applied in several contexts, including autonomous robot patrolling [1, 3], scheduling checkpoints and canine patrols at the Los Angeles International Airport (LAX) [14], and scheduling Federal Air Marshals (FAMS) on flights [8]. The final two examples are real-world software systems that are deployed to make critical resource allocation decisions using game-theoretic reasoning.

Our work in this paper is motivated by the challenges of a different class of security allocations problems faced by agencies in charge of security at airports, ports, and other large physical areas. We develop a new class of game models that offer a richer model of the possible security strategies for the defender, allowing the specification of both the *area(s)* that are defended along with the *activity* that is executed by the security resource. An important aspect of this model is that it represents asymmetric knowledge between the attacker and defender. While we have a detailed understanding of the possible security policies, we have less detail about all of the possible attacker strategies—in reality, it is difficult if not impossible to predict all of the possible attack scenarios a sophisticated attacker might use. Instead, we introduce the possibility for attackers to *circumvent* specific security measures into the model, at some cost. As we show in our analysis, randomized policies are much more difficult for the attacker to plan around, increasing the value of unpredictable activities. Previous models have directly added values for "entropy" as part of the objective function; in our model, the value for randomizing among similar activities is driven by the circumvention capability of the adversary.

Another way that our model generalizes previous work is by allowing multiple resources to be assigned to the same physical area (or target), increasing the level of protection afforded to this area. In addition, the most general form of our model allows for different levels of effectiveness to be associated with different activities, so some are more likely than others to prevent attacks. This is used in particular to model the effects of executing activities such as patrols in physically adjacent areas. While the main protective effect is in the area directly being patrolled, there may be some visibility and capability to respond to incidents in nearby areas, providing a reduced level of protection for those areas as an additional benefit.

The additional richness of this model comes at a computational cost, and computing solutions to this model using existing algorithms is not feasible. In particular, the standard Stackelberg approach is capable of representing these games only by enumerating an exponential number of strategies for both the attacker and defender. While similar issues have been addressed in recent work on algorithms for the FAMS game [8], these methods cannot be directly applied here because they are not designed to handle cases where resources may carry out different activities and provide varying levels of protection. We develop a novel compact representation for this game based on identifying classes of strategies that can be treated symmetrically for the purposes of computing an optimal solution. By exploiting these symmetries we are able to solve much larger game instances than previous methods, which we demonstrate in our experimental results.

2. RELATED WORK

There is work on resource allocation for security settings that uses both game-theoretic approaches as well as more standard optimization frameworks. Our work focuses on developing more detailed models of the possible security measures than previous game-theoretic approaches. A particularly unique aspect of our model is the generic capability that attackers have to circumvent specific security measures.

There are three main areas of related work. The first apply optimization techniques to model the security domain, but do not address the strategic aspects of the problem. These methods provide a randomization strategy for the defender, but they do not take into account the fact that the adversaries can observe the defender's actions and then adjust their behavior. Examples of such approaches include [13, 15] which are based on learning, Markov Decision Processes (MDPs) and Partially Observable Markov Decision Processes(POMDPs). As part of this work, the authors model the patrolling problem with locations and varying incident rates in each of the locations and solve for optimal routes using a MDP framework. Another example is the ”Hypercube Queueing Model" [9] which is based on queueing theory and depicts the detailed spatial operation of urban police departments and emergency medical services. Such frameworks can address many of the problems we raise, including different area values and increasing uncertainty by using many possible patrol routes. However, they fail to account for the possibility that an intelligent attacker will observe and exploit patterns in the security policy. If a policy is based on the historical frequency of attacks, it is essentially a reactive policy, one that an intelligent adversary can exploit.

A second set of work uses Stackelberg games to model a variety of security domains. Game-theoretic models have been applied in a variety of security settings, such as protecting critical infrastructure [6, 11, 14]. Lawrence [18] applies Stackelberg games in the context of screening visitors entering the US. They have also been used for studying missile defense systems [4] and for studying the development of an adversary's weapon system [5]. Other recent work is on randomized security patrolling using Stackelberg games for generic ”police and robbers" scenarios [7] and perimeter patrols [1]. Our work differs from the previous work in that it allows for a more fine-grained representation of security domains. A representation that allows for different levels of protection from security measures and unique security activities that are no longer interchangeable. It also allows for adversary models that do not explicitly represent modes of attack, as in much of the previous work, but still manages to capture some of the adversary's capabilities.

The final set of related work is the application of game theoretic techniques that are not based on Stackelberg games to security applications. Security problems are increasingly studied using game-theoretic analysis, ranging from computer network security [17, 10] to terrorism [16]. Babu et al [2] have worked on modeling passenger security system at US airports using linear programming approaches, however, their objective is to classify the passengers in various groups and then screen them based on the group they belong to. Thus, although game theory has been used in security domains in the past, our work focuses on extending these domains to relax some of the previous assumptions that have been made.

3. MOTIVATING DOMAINS

Our work here is motivated by a large number of security domains where the challenge is to protect a large physical environment from attackers using limited security resources. Such domains include examples like railroad and subway systems, power generation facilities, shipping ports, and airports. In any of these domains there exist a wide variety of possible security measures that could be implemented to provide protection for the facility, including activities such as perimeter patrols, screening inbound vehicles, or verifying the credentials of employees or passengers. The organizations tasked with providing security for these domains include local law enforcement, port authorities, and the Transportation Security Administration (TSA). These organizations face the challenging problem of maximizing the protection of the critical infrastructure using a limited number of available resources.

Assigning resources is complicated by the fact that there are many different areas of a large facility where resources could be allocated. For example, in an airport there are public areas (e.g., ticketing and check in areas), boarding areas, as well as secured areas such as the aircraft runways. Some of these areas are physically distant, while others may be adjoining or accessible through other areas. In addition, these areas may have different values from a security perspective, since they have different numbers of people and some may have other important assets (e.g., aircraft or expensive machinery).

There is also a wide variety of different kinds of tasks or activities that security forces could perform. Each type of activity may be able to prevent different kinds of harmful actions or events. For example, a security activity might be screening baggage for harmful substances, patrolling the perimeter for unauthorized entrants, or verifying the identify of passengers. These activities may have different effects on protecting different areas of the facility. For example, if passengers are screened at the check-in area then this may also help protect the individual terminals since it can stop a potential threat from entering the terminal area through the check-in area. Although a security activity may protect more than one area at a time, the level of protection it provides to each area may vary. Returning to our check-in area example, although screening passengers at the check-in area may protect terminals from unwanted passengers, there may be other threats to the terminals like a worker who enters from a different area. Thus screening only helps partially protect the terminal area, and different combinations of security activities may provide additional protection against a broader range of threats. We will refer to the combination of a security activity and the area where it is performed as a security "operation."

The goal of an attacker is to find a successful strategy to attack some area of the facility. This decision depends both on the goals of the attacker and on the security measures taken by the security forces.

Areas that have less security will be less costly to attack, but they may also not be desirable depending on the overall objective of the attacker. To increase the chances of success an attacker may also try to specifically circumvent particular kinds of security measures that the attacker believes are likely to be in place. This could also take the form of choosing a particular type of attack vector that will not be detected or prevented by known security measures. However, this becomes increasingly difficult as the number and variety of security activities taking place in any given area increases.

Based on these domains we are interested in, we develop a game model to capture the salient features of these domains. Our base model is a Stackelberg game, similar to previous models discussed in Section 2. However, we extend these models to incorporate decisions about both location and the types of security activities being carried out, as well as to incorporate the possibility that attackers can circumvent specific security measures at some additional cost. We begin by introducing a basic version of the model, and then describe a compact representation of this model that allows for more scalable solution methods. We then present an extension to this model that allows greater flexibility in modeling different levels of protection for different types of security activities.

4. SECURITY GAMES WITH COMPLEX ACTIONS

We begin with a high-level description of our game model before giving a more formal definition. Our game model has two players, an attacker and a defender (i.e., the security forces). The defender is trying to prevent attacks on a large physical space—such as an airport or port facility—that can be partitioned into a number of smaller areas. To prevent attacks the defender is able to execute various actions (i.e., "operations") using security resources; these actions are associated with a specific area(s), and perform a particular kind of activity.

The attacker's goal is to successfully attack one of the areas in the facility, but to do so the attacker must also avoid any security activities being performed in the area. As the defender allocates more resources to protect one area, it becomes more difficult for the attacker to successfully attack this area. In our model, areas may have different characteristics, including the payoff each player receives if there is a successful attack on the area, or a failed attack attempt in the area. The defender's actions also have different effects on each area, providing different degrees of protection in different areas. It may also be more or less difficult for attackers to circumvent security measures in different areas.

Real-world terrorist attacks are based on careful planning and often use surveillance or other means to gather detailed information about security procedures. To model this, previous work has adopted Stackelberg game models where the defender moves first and commits to a (randomized) strategy for deploying security resources. The attacker is able to observe this strategy and plan the best possible attack, based on this knowledge. A standard solution concept for these game is a Strong Stackelberg Equilibrium (SSE) in which the defender chooses an optimal mixed strategy, assuming that the attacker will choose an optimal strategy in response. We adopt this Stackelberg framework and solution concepts for the model and algorithms presented in this paper.

In the remainder of this section we define the possible strategies for both the defender and the attacker, and then describe how payoffs are assigned for the possible outcomes of the game. We initially assume for expository purposes that each operation affects exactly one area, and that all operations are identical in how effective they are at preventing attacks. We relax both of these assumptions in Section 6.

4.1 Defender Strategies

We denote the defender by Θ, and the set of defender's pure strategies by $\sigma_\Theta \in \Sigma_\Theta$. In our model the defender is able to execute a variety of security activities called *operations*, which we denote by $O = \{o_1, \ldots, o_m\}$. Each individual operation has two components. The first is the type of activity that the operation represents, and the second is the area(s) where the activity is performed. For now, we assume that each operation affects exactly one area from the set of areas denoted by $A = \{a_1, \ldots, a_n\}$.

The defender has limited resources available for running defensive operations, and so is able to run a maximum of K operations on any day. An assignment of K resources to a set of K operations represents a single strategy $\sigma_\Theta \in \Sigma_\Theta$. For example, if there are three operations, $O = \{o_1, o_2, o_3\}$ and two resources available, one possible pure strategy for the defender is to assign these two resources to o_1 and o_3. The defender's mixed strategies $\delta_\Theta \in \Delta_\Theta$ are the possible probability distributions over Σ_Θ.

4.2 Attacker Strategies

The attacker is denoted by Ψ, and the set of pure strategies for the attacker is given by $\sigma_\Psi \in \Sigma_\Psi$. Similarly, the attacker's mixed strategies are probability distributions over the pure strategies and are denoted by $\delta_\Psi \in \Delta_\Psi$. Each pure strategy for the attacker corresponds to selecting a single area $a_i \in A$ to attack. In principle, the attacker will also choose a specific mode of attack. However, in security domains it is typically not feasible to enumerate all possible modes of attack, and attackers often develop new or modified versions of attacks that have not been seen before. This is particularly the case when security measures are known and predictable, so that attackers are able to specifically plan countermeasures to circumvent the security procedures.

Rather than try to enumerate specific attack scenarios and run the risk of failing to include important possibilities, we model the attacker's strategies at a higher level of abstraction. In addition to selecting an area to attack, the attacker chooses a subset of the possible operations that could be run in that area to avoid, or circumvent. Circumventing operations will increase the attacker's chances of success, but comes with a fixed cost that is a parameter of the model. This cost could capture a variety of different things, such as using more sophisticated technology or additional people to launch the attack, or switching to a less ideal means of attack that is less destructive. For example, if the defender is searching baggage for harmful substances (the operation), but not screening passengers, the attacker could choose to use a vest bomb as their mode of attack which would avoid the baggage screening. Formally, a pure strategy for the attacker consists of an area a_i and a subset of the operations in O to circumvent. It is only necessary for the attacker to circumvent operations that affect area a_i.

4.3 Payoff Definition

Payoffs for each player are defined over all possible joint pure-strategy outcomes: $\Omega_\Theta : \Sigma_\Psi \times \Sigma_\Theta \to \Re$ for the defender and similarly for the attacker. The payoff functions are extended to mixed strategies in the standard way by taking the expectation over pure-strategy outcomes. The first component of the payoff depends on which area the attacker chooses to attack, and whether or not the attack was successful. We define four values for each area: $V_\Theta^d(a_i)$ and $V_\Theta^a(a_i)$ for the defender and $V_\Psi^d(a_i)$ and $V_\Psi^a(a_i)$ for the attacker. Here d signifies the area being successfully defended while a signifies the area being successfully attacked so $V_\Theta^d(a_i) > V_\Theta^a(a_i)$ for the defender while $V_\Psi^d(a_i) < V_\Psi^a(a_i)$ for the attacker.

The probability of success or failure depends on both the operations the defender is running in the attacked area, and the set of operations the attacker is circumventing. We define $\lambda(a_i, \sigma_\Theta)$ to be the set of operations $o_i \in \sigma_\Theta$ that affect area a_i (which might be the empty set). For now, we assume that an attack is successful if and only if $\lambda(a_i, \sigma_\Theta) \subseteq \sigma_\Psi$. This assumes that every operation has a 100% chance of preventing the attack unless it is circumvented by the attacker.

After the attack is determined to be successful or not the payoff also depends on which operations the attacker has chosen to circumvent. We introduce this cost as a function $C(a_i, \sigma_\Psi)$ which is the cost of circumventing the set of operations chosen in σ_Ψ for the attacked area. The larger the set of operations σ_Ψ contains the larger the cost becomes, so it is more difficult to successfully attack areas that are more heavily defended. We include the area because circumventing certain operations may be easier in some areas than in others depending on factors like layout, daily activities in that area, and the number of people who are regularly present in that area. This cost is deducted from the attacker's payoff and added to the defender's overall payoff, resulting in the following overall payoffs for both players in the case of a successful attack:

$$V_\Theta^a(a_i) + C(a_i, \sigma_\Psi) \tag{1}$$

$$V_\Psi^a(a_i) - C(a_i, \sigma_\Psi) \tag{2}$$

The payoff for a failed attack is identical except for substituting V_Θ^a with V_Θ^d for the defender and the same for the attacker. To further explain the game representation we have just outlined and how payoffs are calculated in this game we will turn to a concrete example. In this example there are two areas, $A = \{a_1, a_2\}$, and four operations $O = \{o_1, o_2, o_3, o_4\}$. Here o_1 and o_2 affect only a_1, and o_3 and o_4 affect only a_2. For the follower we set $V_\Psi^a(a_1) = 5$, and $V_\Psi^d(a_1) = -1$ for the first area and $V_\Psi^a(a_2) = 10$, and $V_\Psi^d(a_2) = -5$ for the second area. For the defender we set $V_\Theta^d(a_1) = 2$, and $V_\Theta^a(a_1) = -10$, for the first area and $V_\Theta^d(a_2) = 5$, and $V_\Theta^a(a_2) = -20$ for the second area. Finally we set the costs as $C(a_1, o_1) = C(a_1, o_2) = 2$ and $C(a_2, o_3) = C(a_2, o_4) = 3$. Figure 1 shows a physical representation of this game with corresponding payoffs. In our example there will be 2 resources to assign, $K = 2$. We show the possible outcomes of this game in normal-form in Table 1.

In Table 1 the first value represents the defender's payoff and the second value represents the attacker's payoff. The attacker's actions are represented first by the area selected for the attack and then by the operations avoided. For instance, the second column represents the attacker choosing area a_1 and avoiding operation o_1 where the third column represents the attacker choosing area a_1 and avoiding operation o_2. To illustrate how these values are translated into the table lets look at the case where the defender chooses o_1, o_3 and the attacker attacks area a_1 while avoiding o_1. Since the attacker avoided all the operations we were running in that area he succeeds in his attack, thus he receives $V_\Psi^a(a_1)$ or 5 points. However, the cost to avoid o_1, or $C(a_1, o_1)$, is 2 so the attacker only receives 3 points (5 - 2). On the defender's side we go through similar logic to arrive at a payoff of -8 (-10 + 2).

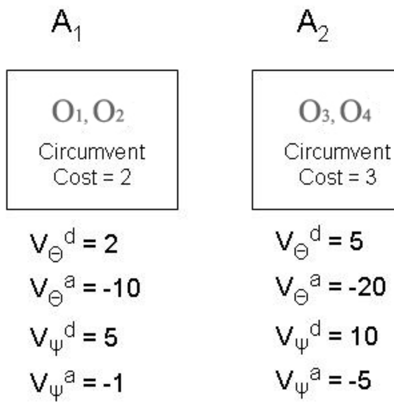


Figure 1: Game Example

	$a_1 : \emptyset$	$a_1 : o_1$	$a_1 : o_2$	$a_2 : \emptyset$	$a_2 : o_3$	$a_2 : o_4$
o_1, o_2	2, -1	4, -3	4, -3	-20, 10	-17, 7	-17, 7
o_1, o_3	2, -1	-8, 3	4, -3	5, -5	-17, 7	8, -8
o_1, o_4	2, -1	-8, 3	4, -3	5, -5	8, -8	-17, 7
o_2, o_3	2, -1	4, -3	-8, 3	5, -5	-17, 7	8, -8
o_2, o_4	2, -1	4, -3	-8, 3	5, -5	8, -8	-17, 7
o_3, o_4	-10, 5	-8, 3	-8, 3	5, -5	8, -8	8, -8

Table 1: Example payoffs for sample game

	$a_1 : \emptyset$	$a_1 : \gamma_1$	$a_2 : \emptyset$	$a_2 : \gamma_2$
γ_1, γ_1	2, -1	4, -3	-20, 10	-17, 7
γ_1, γ_2	2, -1	-2, 0	5, -5	-4.5, -5
γ_2, γ_2	-10, 5	-8, 3	5, -5	8, -8

Table 2: Example compact version

Given this setup we can construct a standard Stackelberg game. Namely, we have outlined the strategy space, Σ_Θ, for the defender to be the set of all possible combinations of K operations and the strategy space, Σ_Ψ, for the attacker to be the set of all possible circumvention strategies for each area. We have also outlined how payoffs are determined based on the strategy chosen by the defender and the attacker. Particularly, the attack fails or succeeds based on whether the attacker has circumvented the necessary operations at the area he chooses to attack and the cost of the attack is factored into both the defender's and attacker's payoffs irrespective of whether the attack succeeds or fails. Given the normal-form representation of any of these games similar to that shown in Table 1, this game can be solved using the fastest known general Stackelberg solver, DOBSS [12].

5. COMPACT REPRESENTATION

Although setting up our new problem as described is solvable using a general Stackelberg solver, it does not scale well as the size of the game increases. Both the attacker and defender strategy spaces grow combinatorially as the number of defender operations increases. We introduce a compact representation that exploits similarities in defender operations to reduce the number of strategies that must be enumerated and considered when finding an optimal solution to the game.

5.1 Exploiting Identical Operations

First, we identify operations that provide coverage to the same areas, and have the same circumvention costs; so far, all operations within a given area are identical. Let $\gamma_i \in \Gamma$ represent the sets of operations that can be grouped together because they have identical properties. The key is that each of these operations will have the same effect on the payoffs, so we can reason about them as a group and only consider the *number* of operations of each type that are selected by the defender or circumvented by the adversary. We can show that in the optimal solution, the selection probabilities and circumvention strategies take a simple form. In particular, we now argue that it is optimal for the defender to distribute probability uniformly at random across all operations within a set γ_i, so that all operations are chosen with equal probability in the solution. Given this, we only need to know how many operations are selected from each set in order to compute the expected payoffs for each player in the optimal solution.

PROPOSITION 1. *Selecting each operation $o_j \in \gamma_i$ with equal probability provides the maximum expected payoff for the defender.*

Proof Sketch: Let the vector $X = \langle x_1, x_2, \dots, x_j \rangle$ represent the total probability with which each operation associated with a given area is selected according to some mixed strategy δ. Without loss of generality, assume that this vector is sorted in descending order such that $x_1 \geq x_2 \geq \cdots \geq x_n$. The attacker strictly prefers to circumvent operations that are selected with higher probability, so the attacker will always choose to circumvent operations $x_1 \dots x_m$ for any number of circumvented operations m. Now, consider the alternative defender strategy $\hat{\delta}$ with uniform coverage probabilities $\hat{x}_1 = \hat{x}_2 = \cdots = \hat{x}_n = (\sum_{i=1}^{n} x_i)/n$. For any m operations that the attacker could circumvent, $\sum_{i=m+1}^{n} x_i \leq \sum_{i=m+1}^{n} \hat{x}_i$ because the vectors have the same sum and we have eliminated the m maximum elements of X. Therefore, the attacker succeeds no more frequently against strategy δ than $\hat{\delta}$, and the defender's expected payoff is at least as great for the uniform strategy $\hat{\delta}$. ∎

A strategy $\sigma_\Theta \in \Sigma_\Theta$ can now be represented by the number of resources assigned to each set of identical operations γ_i. For example, if there are two sets γ_1 and γ_2, and the defender has 2 available resources, the possible strategies are to assign both to γ_1, one to each set, or both to γ_2 (assuming at least two operations in each set). The original strategy space consists of all possible ways to select two operations from n possible operations, which is much larger than the compact strategy space as n grows large.

We now define define $\lambda(\gamma_i, \sigma_\Theta)$ to be number of resources assigned to γ_i in the strategy σ_Θ. We also use the notation Υ_{a_i} to represent the set of all γ_i that affect area a_i. Finally, we define Q^Θ to be the vector of resource assignments over Γ where Q_i^Θ is the number of resources assigned to γ_i.

Given that the defender strategy uniformly distributes resources among all operations $o_j \in \gamma_i$ we also know that it does not matter which specific operations the attacker chooses to circumvent from the set γ_i. For any given number of operations circumvented, the expected payoff is identical regardless of which specific operations within the set are chosen. Therefore, we can use a similar compact representation for the attacker strategy space as for the defender, reasoning only over the aggregate number of operations of each type rather than specific operations. Specifically, a strategy σ_Ψ is represented by which area the attacker chooses to attack and then by how many operations from each set γ_i the attacker circumvents. Similar to the defender, this is a much smaller strategy space than the original strategy space which enumerates all possible unique circumvention strategies. We define Q^Ψ to be the vector of the number of operations circumvented over Γ where Q_i^Ψ is the number of operations circumvented from the set γ_i.

A concrete example of this representation is presented in Table 2, for the same game shown in Table 1. In this representation there are only 3 pure strategies for the defender: assign both resources to γ_1 operations, assign one resource to γ_1 and one to γ_2, or assign both resources to γ_2. Similarly, for the attacker there are now only 2 circumvention options per area: circumvent no operations or circumvent one operation of the appropriate set γ_i. We will now explain how payoffs are calculated in this new compact version and how these payoffs map back to the full representation.

5.2 Computing Payoffs in the Compact Representation

We have defined a compact representation for both the defender and attacker strategies. It remains to describe how payoffs are calculated for combinations of these strategies, and how these payoffs reflect the payoffs in the original game. To compute the payoffs for a combination of strategies we must first calculated the probability that an attack succeeds. For any given defender strategy the defender resources allocated to each operation type (Q_i^Θ) are uniformly distributed over the operations in γ_i. In addition, the attacker will receive an identical payoff for any set of operations circumvented within γ_i. Therefore, we can select an arbitrary pure strategy from the full representation for the attacker which circumvents each fixed number of operations; we will refer to this strategy as τ.

We now describe how to compute the expected payoffs for both players for attacker strategy τ by computing the probability that the attacker will succeed against the defender strategy. Let $\xi_i \in \Xi_i$ represent the possible combinations of operations in $\gamma_i \in \Upsilon_{a_i}$, where a_i is the area attacked in τ. The attack succeeds if and only if the operations circumvented in τ are a superset of the operations in ξ. For each ξ we compute the number of times the attacker fails, f_i, by counting the occurrences where all operations in ξ are not circumvented in τ. The attacker succeeds in all other cases, denoted by w_i. The attacker's overall probability of failure taking into account all types of operations is given by $\epsilon = \Pi_{i=0}^{n} f_i/(f_i + w_i)$, and corresponding probability of success is $1 - \epsilon$. We can now compute payoffs for both defender and attacker:

$$(1-\epsilon) * V_\Theta^d(a_i) + \epsilon * V_\Theta^a(a_i) + \sum_{Q_i^\Psi \in \sigma_\Psi, o_j \in \gamma_i} C(a_i, o_j) * Q_i^\Psi \quad (3)$$

$$(1-\epsilon) * V_\Psi^d(a_i) + \epsilon * V_\Psi^a(a_i) - \sum_{Q_i^\Psi \in \sigma_\Psi, o_j \in \gamma_i} C(a_i, o_j) * Q_i^\Psi \quad (4)$$

We note that it is also possible to quickly detect situations where the attacker cannot possibly succeed because the number of operations circumvented for some type is less than the number of operations run by the defender for this type. In these cases, the above equations simplify to:

$$V_\Theta^d(a_i) + \sum_{Q_i^\Psi \in \sigma_\Psi, o_j \in \gamma_i} C(a_i, o_j) * Q_i^\Psi \tag{5}$$

$$V_\Psi^d(a_i) - \sum_{Q_i^\Psi \in \sigma_\Psi, o_j \in \gamma_i} C(a_i, o_j) * Q_i^\Psi \tag{6}$$

Looking back at Table 1 we can provide some additional insight into why this compact representation works. Notice that regardless of whether the defender chooses operation o_1 or o_2 he will receive identical payoffs. For example, if the defender chooses operation o_1 the reward value if the attacker just avoids o_1 is -8 and if the attacker just avoids o_2 it is 4. Similarly, if the defender chooses operation o_2 their reward value if the attacker just avoids o_2 is -8 and if the attacker just avoids o_1 it is 4. Given that the attacker's strategy is to optimize against the defender's strategy and that resources are split equally among o_1 and o_2, the attacker is indifferent between avoiding just o_1 or just o_2 since both yield identical payoffs.

6. EXTENSION TO MULTIPLE LEVELS OF PROTECTION

Up to this point we have assumed that each operation affects exactly one area, and that every operation is able to prevent any attack if it is not circumvented by the attacker. In this section we relax these assumptions and allow for a more general model of the effects of operations on the success or failure of an attack. We allow each operation to affect an arbitrary number of areas, and to prevent attacks in each area with a different probability. The ability to represent operations that affect different areas is useful for representing patrols in adjacent areas, or for representing security measures that may not be directly applicable to a single physical area, but has a broad effect across many different areas.

We define a function $S(a_i, o_j) \in [0 \dots 1]$ that expresses the probability that operation o_j will prevent an attack in area a_i. A value of 0 represents an operation that has no effect on a particular area, and a value of 1 represents perfect protection. As before, any operation can be circumvented by the adversary to mitigate the protective effect of the operation. The main difference in this model is that we must now consider a definition of operation types that accounts for the effectiveness of operations in different areas. Operations may only be collapsed in the compact representation if they provide identical coverage in every area. Given that restriction, we can extend Proposition 1 by a similar argument (omitted here) to show that it is optimal to randomize uniformly across operations that are identical in this respect.

Extending our model to this more comprehensive model in both the full representation and compact representation requires only a minimal change to the payoff calculations. Specifically, the total probability of successfully preventing an attack is computed by multiplying together the $1 - S(a_i, o_j)$ values for all of the operations in σ_Θ that are not circumvented in σ_Ψ for the area a_i that is attacked and then subtracting this value from 1. Specifically, the multiplication of these values, $1 - S(a_i, o_j)$, represents the chance that all operations failed to catch the adversary and the chance of success is easily determined by subtracting this value from 1. Denoting this value by Z we have the revised equations:

$$(1 - Z) * V_\Theta^d(a_i) + Z * V_\Theta^a(a_i) + \sum_{o_i \in \sigma_\Psi} C(a_i, o_i) \tag{7}$$

$$(1 - Z) * V_\Psi^d(a_i) + Z * V_\Psi^a(a_i) - \sum_{o_i \in \sigma_\Psi} C(a_i, o_i) \tag{8}$$

Computing the payoffs for the compact representation in this case requires one additional manipulation. First, we must compute the probability that each operation is circumvented for each set of identical operations γ_i, based on the attacker strategy. This probability of circumvention is factored into the computation of the overall probability of capture by scaling each $S(a_i, o_j)$ in the computation by the probability that o_j will be circumvented by the attacker. Given this scaling term, the process for computing the payoffs is the same as described previously.

7. EVALUATION

In this section we provide empirical results to demonstrate the benefits of our compact representation on scalability. The effectiveness of this representation depends primarily on the number of unique types of operations that are present in the original game; in the worst case every operation is unique in some way and in that case the compact representation is identical to the full representation. Our compact representation is most effective in cases where each operation affects relatively few areas, and the effectiveness of operations (in terms of the probability of preventing an attack) can be categorized into a small number of discrete levels of protection. This maximizes the chance that there will be identical operations which can be merged in the compact representation. In principle it would also be possible to merge similar operations with some loss of solution quality, but we defer investigation of this method for approximation to future work.

We present simulation results focusing on the computational efficiency of our methods, and particularly the benefits of the compact representation in cases where there are identical operations. All experiments are run on a system with an Intel 2 GHz processor and 1 GB of RAM. We used a publicly available linear programming package called GLPK to solve optimization problems as specified in the original DOBSS procedure. The solver was allowed to use up to 700 MB of memory during the solution process. For larger game instances, solving the problem with the full representation runs out of memory and solutions cannot be found. In the results presented below we exclude results for cases where the full representation was not able to produce a result using the allotted memory.

To test the solution methods we generate random game instances by randomly selecting payoff values and the circumvention costs for each area. For each experiment we generated 20 random game instances and averaged the results (there is little variance in the runtimes for different problem instances). We consider three different scenarios. The first scenario shown in Figure 2 has a single area, and the defender is allowed to allocate up to 5 resources to run operations. We increase the number of different operations available to protect this area along the x-axis. For the compact representation we vary the number of unique types of operations to show how this impacts the efficiency of the solution method. Results are shown for 1, 2, and 4 unique operation types, however, we only show the results in the 4 unique operation types case up to 10 operations. It is clear that more operations in this case would have taken a substantial amount of time. As shown in Figure 2, the full representation is unable to find a solution within the memory limit for games with 8 or more operations, while the compact representation is able to run up to 20 operations in less than 1 second in the ideal case where there is a single operation type.

The next scenario presents results for the case where there is an increasing number of areas, and each area has exactly 3 operations associated with it. There are 5 resources available for the defender, and each operation provides maximum protection for the area it is associated with. This implies that there is one unique operation type for every area. Examining Figure 3, we show the improvement of our compact representation over the full representation. For more than 4 areas, the full representation failed to achieve a solution within the memory bounds. For 5 areas, the compact representation runs much faster than the full representation, with a total runtime of less than 1 second versus the 177 seconds required by the full representation to find a solution for the case with only 4 areas. Even if the number of operations associated with each area is a relatively small constant our compact representation provides substantial benefits. As the number of similar operations associated with an area increases, this advantage grows (as shown in our first experiment).

Finally we consider a scenario where operations are distributed randomly across possible areas. Again, each operation is associated with a single area. The total number of operations is set similarly to the previous experiment, in that that the total number of operations is three times the number of areas. However, we randomly assign operations to areas (with each area having at least one operation) so the number is no longer uniform. Once again the defender has 5 resources available and each operation provides full protection to the area it is associated with. Looking at Figure 4, we see similar benefits for the compact representation in this case as in the previous experiment with a uniform distribution of operations.

These results show the potential benefits of the compact representation in improving scalability by exploiting similarities in the effects of some operations. We have shown that the most important factor is the number of unique types of operations that exist and how many of these operations there are. If the number of types is low the compact representation performs efficiently.

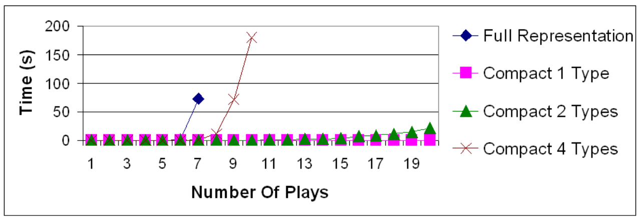


Figure 2: Runtime: Increasing number of operations with 1 target and 5 resources

8. CONCLUSION

Allocating resources to defend critical infrastructure, high profile events, and transportation systems among other things remains an important problem in many security domains. While there are a number of methods in use today for addressing this problem, one notable approach that is increasingly finding more use in security applications is that of game theory. In fact, game theory has seen successful application at the Los Angeles International Airport and for the United States Federal Air Marshals Services [8, 14].

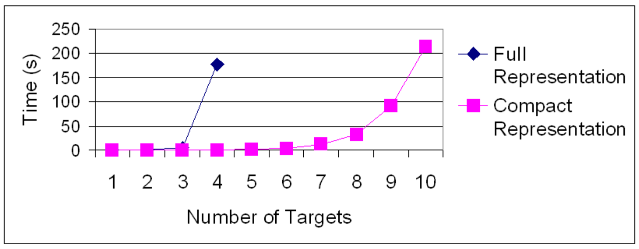


Figure 3: Runtime: Increasing areas with 5 resources and 3 operations per area

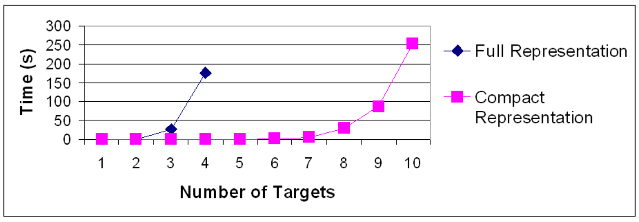


Figure 4: Runtime: Increasing areas with operations randomly distributed and 5 resources

We introduce a new form of security game that extends previous models in several important directions. First, this model includes a more fine-grained representation for the defender's strategy space, explicitly considering both the location of a security activity and the type of activity that a security resource will perform. Second, we allow for different levels of effectiveness for different security actions, including the possibility that an activity has different effects across multiple locations. Previous models have also assumed that all of the possible attack strategies for the attacker are known with certainty, which is unrealistic in real-world security problems. In particular, attackers can often adapt to circumvent specific known security measures. We extend the security game model on the attacker's side to handle this possible in a generic way, allowing attackers to circumvent specific security activities at some cost. This leads to an increased value for randomness and unpredictability in the defender's strategy, even among actions that may be similar in terms of the areas they affect. Solving this new class of games presents a computational challenge that existing solution methods are not able to handle. We address this by introducing a compact representation for these games that exploits symmetries between similar types of security activities, and provide experimental results showing the resulting improvements in runtime.

9. REFERENCES

[1] N. Agmon, V. Sadov, S. Kraus, and G. Kaminka. The impact of adversarial knowledge on adversarial planning in perimeter patrol. In *AAMAS*, 2008.

[2] L. Babu, L. Lin, and R. Batta. Passenger grouping under constant threat probability in an airport security system. *European Journal of Operational Research*, 168:633–644, 2006.

[3] N. Basilico, N. Gatti, and F. Amigoni. Leader-follower strategies for robotic patrolling in environments with arbitrary topology. In *AAMAS*, 2009.

[4] G. Brown, M. Carlyle, J. Kline, and K. Wood. A two-sided optimization for theater ballistic missile defense. *Operations Research*, 53:263–275, 2005.

[5] G. Brown, M. Carlyle, J. Royset, and K. Wood. On the

complexity of delaying an adversary's project. *The Next Wave in Computing, Optimization and Decision Technologies*, pages 3–17, 2005.

[6] G. Brown, M. Carlyle, J. Salmerón, and K. Wood. Defending critical infrastructure. *Interfaces*, 36(6):530–544, 2006.

[7] N. Gatti. Game theoretical insights in strategic patrolling: Model and algorithm in normal-form. In *ECAI*, 2008.

[8] C. Kiekintveld, M. Jain, J. Tsai, J. Pita, M. Tambe, and F. Ordóñez. Computing Optimal Randomized Resource Allocations for Massive Security Games. In *AAMAS*, 2009.

[9] R. C. Larson. A hypercube queueing model for facility location and redistricting in urban emergency services. *Computers and OR*, 1(1):67–95, 1974.

[10] K. Lye and J. Wing. Game strategies in network security. *International Journal of Information Security*, 4(1-2):71–86, 2005.

[11] X. Nie, R. Batta, C. Drury, and L. Lin. Optimal placement of suicide bomber detectors. *Military Operations Research*, 12:65–78, 2007.

[12] P. Paruchuri, J. Marecki, J. Pearce, M. Tambe, F. Ordóñez, and S. Kraus. Playing games for security: An efficient exact algorithm for solving bayesian stackelberg games. In *AAMAS*, 2008.

[13] P. Paruchuri, M. Tambe, F. Ordonez, and S. Kraus. Security in multiagent systems by policy randomization. In *AAMAS*, 2006.

[14] J. Pita, M. Jain, J. Marecki, F. Ordóñez, C. Portway, M. Tambe, C. Western, P. Paruchuri, and S. Kraus. Deployed armor protection: The application of a game theoretic model for security at the los angeles international airport. In *AAMAS*, 2008.

[15] S. Ruan, C. Meirina, F. Yu, K. R. Pattipati, and R. L. Popp. Patrolling in a stochastic environment. In *10th Intl. Command and Control Research and Tech. Symp.*, 2005.

[16] T. Sandler and D. Arce. Terrorism and game theory. *Simulation and Gaming*, 34(3):319–337, 2003.

[17] V. Srivastava, J. Neel, A. MacKenzie, R. Menon, L. Dasilva, J. Hicks, J. Reed, and R. Gilles. Using game theory to analyze wireless ad hoc networks. *IEEE Communications Surveys and Tutuorials*, 7(4), 2005.

[18] L. Wein. Homeland security: From mathematical models to policy implementation. In *Operations Research*, 2008.

Efficient, Superstabilizing Decentralised Optimisation for Dynamic Task Allocation Environments

Kathryn S. Macarthur, Alessandro Farinelli*
Sarvapali D. Ramchurn and Nicholas R. Jennings
School of Electronics and Computer Science, University of Southampton, SO17 1BJ, UK
*Department of Computer Science, University of Verona, 15 I-37134, Italy
{ksm08r, sdr, nrj}@ecs.soton.ac.uk, *alessandro.farinelli@univr.it

ABSTRACT

Decentralised optimisation is a key issue for multi-agent systems, and while many solution techniques have been developed, few provide support for dynamic environments, which change over time, such as disaster management. Given this, in this paper, we present Bounded Fast Max Sum (BFMS): a novel, dynamic, superstabilizing algorithm which provides a bounded approximate solution to certain classes of distributed constraint optimisation problems. We achieve this by eliminating dependencies in the constraint functions, according to how much impact they have on the overall solution value. In more detail, we propose iGHS, which computes a maximum spanning tree on subsections of the constraint graph, in order to reduce communication and computation overheads. Given this, we empirically evaluate BFMS, which shows that BFMS reduces communication and computation done by Bounded Max Sum by up to 99%, while obtaining 60–88% of the optimal utility.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent Systems*; G.2.2 [**Discrete Mathematics**]: Graph Theory

General Terms

Algorithms, Experimentation, Theory

Keywords

Distributed constraint optimization, disaster management

1. INTRODUCTION

Multi-agent systems have been advocated as a key solution technology for coordinating the activities of rescue forces for disaster management. Specifically, coordination in such domains can be conveniently framed as a distributed constraint optimisation problem [13]. Many solution techniques, such as ADOPT (Asynchronous Distributed OPTimisation) [10], DPOP (Distributed Pseudotree Optimization Procedure) [11] and Bounded Max Sum (BMS) [6] have been proposed to solve such optimisation problems. Now, while complete algorithms such as ADOPT and DPOP guarantee optimal solutions, they use a lot of communication and computation. On the other hand, approximate algorithms like BMS tend to incur a lower coordination overhead, and can provide bounds on the quality of the approximation they give [6].

However, very few algorithms exist that reduce redundant communication and computation when used in dynamic environments. Nonetheless, dynamism is a key issue for search and rescue in disaster management: the environment evolves over time, and new information becomes available as time progresses. For example, information about civilians to rescue from buildings may change, and new rescue agencies can arrive, at any time, to help with the rescue effort. In such applications, optimal solutions are most desirable, but may not be achievable within the available time. Thus, good quality approximate algorithms can be used, due to their smaller overheads, but only if they are able to produce good quality solutions. Now, it is also important that an algorithm can always recover from environmental change gracefully, so as not to enter an unsafe state during recovery. This is formally known as *superstabilization* [4]. In more detail, in order for an algorithm to be superstabilizing, it must be super-stabilizing [3], and a pre-defined *passage predicate* must hold at all times. More specifically, a self-stabilizing algorithm must be distributed across a number of agents, and must be able to return those agents to some legitimate state after a change in the environment. This legitimate state is defined with a *legitimacy predicate*, which, when invalidated, forces the algorithm to return the agents to a state where the predicate holds in a finite amount of time. More generally speaking, in a superstabilizing algorithm, the passage predicate must hold at all times, including whenever the legitimacy predicate does not.

To date, few algorithms are suited for use in dynamic environments, and instead would need to be re-run, incurring unneeded overheads without superstabilization guarantees. Exceptions to this include SDPOP (Superstabilizing DPOP) [12], which is superstabilizing, but requires a prohibitive amount of communication and computation on large scale scenarios. Another algorithm that partially fits our requirements is the Fast Max Sum (FMS) algorithm [13], which reduces the communicational and computational impact of a change in the environment. However, FMS is only proven to converge to an optimal solution on specific[1] problem instances, and is therefore not general enough to be applied to more realistic arbitrary disaster management environments.

Against this background, in this paper, we propose a novel algorithm for performing superstabilizing distributed constraint optimisation in dynamic environments. In more detail, our algorithm provides approximate solutions with quality guarantees for constraint graphs with arbitrary topologies[2], while reducing communication and computation. In particular, this paper advances the state of the art in the

[1]Where the underlying constraint graph contains no cycles.
[2]Even those in which the underlying graph contains cycles

following ways: first, we present iGHS, an iterative variant of the GHS algorithm (named after the authors Gallager, Humblet and Spira) [7], which computes a maximum spanning tree of the constraint graph in BMS. Second, we present Bounded Fast Max Sum (BFMS): an efficient, superstabilizing constraint optimisation algorithm, which combines iGHS with principles from FMS and BMS.

The rest of this paper is structured as follows. In Section 2 we discuss the relevant background for our work. Next, we formulate our problem in Section 3, and present our iGHS algorithm in Section 4. We present BFMS in Section 5, and empirically evaluate it in Section 6. We discuss work related to iGHS in Section 7, and, in Section 8, we conclude.

2. BACKGROUND

Here, we present the necessary background for our work. We begin with a discussion of Max Sum and two particular variants that are useful in applying Max Sum to arbitrary dynamic environments: first, Fast Max Sum, used to reduce overheads caused by recomputation in dynamic environments, and second, Bounded Max Sum, which uses approximation to allow application of Max Sum to arbitrary environments. Finally, we discuss GHS, which is a key element of Bounded Max Sum, used to preprocess the constraint graph.

2.1 Max Sum

The Max Sum algorithm belongs to the GDL (Generalised Distributive Law) framework [1], and has been shown to be a very useful technique for distributed constraint optimisation [5]. In more detail, Max Sum provides good quality approximate solutions to DCOPs (Distributed Constraint Optimisation Problems) requiring very low computation and communication. This is achieved by using message passing over a factor graph representation (see [8]) of the dependencies between agents' utilities in the global utility function. More specifically, a factor graph is a bipartite, undirected graph consisting of variable nodes and function nodes, where function nodes are connected to variable nodes they depend on.

However, while Max Sum has been proven to converge to an optimal solution on tree-structured factor graphs, it uses redundant computation in dynamic environments, and lacks optimality guarantees on cyclic graphs [14]. In the former case, Max Sum would have to be re-run after every change in the environment, and in the latter, very limited theoretical results relative to convergence and solution quality exist. As such, Fast Max Sum (FMS) [13] and Bounded Max Sum (BMS) [6] were presented in order to combat these problems: FMS reduces overheads incurred after a change in the environment, and BMS gives solutions with quality guarantees on cyclic graphs. Given this, we elaborate further on FMS and BMS in the rest of this section.

2.1.1 Fast Max Sum

Fast Max Sum (FMS) [13] provides two main improvements to Max Sum: (i) a reduction of message size, and (ii) a reduction of redundant communication and computation after a change in the graph. These improvements can be gained in certain scenarios, such as rescue scenarios, where each variable represents an agent, and each factor a target to rescue. In such cases, a variable's domain is the set of tasks that it must choose between, and each task's dependencies are the variables whose domain contains them.

Next, we explain how FMS reduces unneeded communication and computation after a change in the underlying graph: be it addition or removal of a function, or a variable. Put simply, a variable in FMS will only send a message in response to a received message if the values given in the received message are different to those it previously received from that factor. In order to do this, FMS adds storage requirements to Max Sum, requiring each variable to store their previous value, as well as the last message they received on each of their edges, compare to it, and update it, when messages are received. Then, a variable will only send a new message to a function if and only if its previous values for that node have changed. For a more detailed example of the execution of FMS, we refer the reader to [13].

Nevertheless, FMS can only produce provably optimal solutions in certain cases,[3] and is therefore not general enough for use in all environments. As such, we look at Bounded Max Sum, which provides approximate solutions but guarantees convergence and solution quality.

2.1.2 Bounded Max Sum

Bounded Max Sum (BMS) [6] produces bounded approximate solutions by eliminating cycles in the factor graph (see Section 3). More specifically, low-impact dependencies are found and removed by constructing a maximum spanning tree of the factor graph using the GHS algorithm [7] and then Max Sum is run on the resulting tree. Each node in the tree keeps track of which dependencies have been removed and, by using this information, can compute an approximation ratio of the optimal solution.

While BMS is not explicitly superstabilizing, we could make it so by introducing storage at each factor, in order to maintain information on the system state during recovery from a change in the environment. Despite this, however, BMS could still incur redundant computation and communication after such a change in the environment, as the GHS algorithm and Max Sum algorithms would need to be re-run. Now, as explained above, FMS would reduce overheads in the latter part of the algorithm, but not the former. As such, next, we detail the GHS algorithm.

2.2 GHS

The GHS algorithm [7] is a distributed technique to find the minimum spanning tree (MST) of any given weighted undirected graph, using only local communication and low computation at each node.

The basic premise of the GHS algorithm is that the nodes of a graph are formed into a number of graph fragments, which gradually join on their minimum-weight edges, in order to eventually form one large graph fragment, containing the MST of the graph. This is done through localised message passing. For a more detailed discussion of the operation of the algorithm, please refer to [7].

While the GHS algorithm is adequate for static problems, such as those BMS was designed for, when a change is made to the graph, the algorithm must be completely re-run. To avoid this, it is important to operate such an algorithm over a defined subset of the graph whenever a change occurs. To this end, we have developed a novel algorithm which we

[3]As with Max Sum, where the constraint graph contains no cycles.

present in Section 4. In order to do this, we formulate our problem in the next section.

3. PROBLEM FORMULATION

In this section, we formally describe the decentralised coordination problem that we address in this paper. We focus on a task allocation problem: specifically, an environment containing a number of agents, $\mathbf{A} = \{a_1, \ldots, a_{|A|}\}$, who must complete a number of tasks, $\mathbf{T} = \{t_1, \ldots, t_{|T|}\}$. The set of tasks which an agent $a \in \mathbf{A}$ can potentially complete is denoted $T_a \subseteq \mathbf{T}$. Similarly, the set of agents which can complete a task $t \in \mathbf{T}$ is denoted $A_t \subseteq \mathbf{A}$.

This problem can be conveniently represented with a factor graph (see [8]), which is a bipartite, undirected graph $\mathcal{FG} = \{\mathcal{N}, \mathcal{E}\}$, where $\mathcal{N}$ is the set of nodes, such that $\mathcal{N} = \mathcal{VN} \cup \mathcal{FN}$, where $\mathcal{VN}$ is a set of variable nodes, and $\mathcal{FN}$ is a set of function nodes. In addition, $\mathcal{E}$ is a set of edges, where each edge $e \in \mathcal{E}$ joins exactly one node in $\mathcal{VN}$ to exactly one node in $\mathcal{FN}$. An example factor graph formulation of our scenario is given in Figure 1.

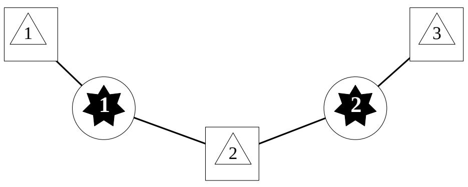


Figure 1: An example scenario containing 2 rescue agents (black stars) and 3 tasks (white triangles), formulated as a factor graph, with agents as variables (circles), and tasks as factors (squares).

In our problem, we have a set of variables, $\mathbf{x} = \{x_1, \ldots, x_m\}$, controlled by the set of agents $\mathbf{A}$. Each agent owns precisely one variable,[4] so we denote the variable belonging to agent $a \in \mathbf{A}$ as x_a. This variable represents the target of the agent: that is, the task that the agent must complete, and as such the domain of x_a is T_a.

Next, we define a set of functions, $\mathbf{F} = \{F_1, \ldots, F_n\}$, each representing a task $t \in \mathbf{T}$. A function $F_t(\mathbf{x}_t)$ is dependent on the set of variables which can potentially take the value t, or more formally, $\mathbf{x}_t = \{x_a | a \in A_t\}$. Thus, $F_t(\mathbf{x}_t)$ denotes the value for each possible assignment of the variables in $\mathbf{x}_t$.

Given this, we aim to find the state of each variable in $\mathbf{x}$ which maximises the sum of all functions in the environment (known as social welfare):

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} \sum_{t \in \mathbf{T}} F_t(\mathbf{x}_t) \tag{1}$$

Specifically, the factor graph consists of a function node $FN \in \mathcal{FN}$ for each $F \in \mathbf{F}$ and a variable node $VN \in \mathcal{VN}$ for each $x \in \mathbf{x}$. We assume that each agent $a \in \mathbf{A}$ only has control over, and knowledge of, its own local variable x_a, and thus one variable node in the factor graph. The decision as to which agent computes for shared functions has no impact on the correctness of our approach, and as such any policy can be used to decide this (e.g. the agent with the lowest ID computes for shared functions).[5] Thus, we assume that each agent may compute any number of functions in $\mathbf{F}$, but that each function $F \in \mathbf{F}$ is computed by one agent $a \in \mathbf{A}$ only. Now, as mentioned earlier, each function $F \in \mathbf{F}$ represents one task in the task space, $t \in \mathbf{T}$. Hence, each function node in the factor graph represents one task, and so, each function node $F_t(\mathbf{x}_t)$ will be connected to the variable nodes representing the variables $\mathbf{x}_t$.

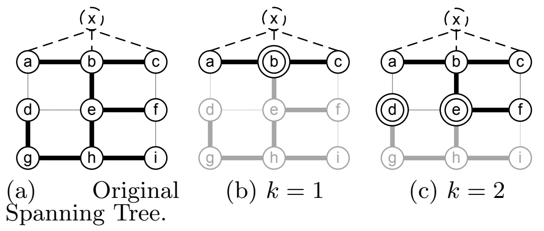
[4]This is not a limitation of our solution approach, but a feature of our reference domain.

4. IGHS

As previously mentioned, it is not ideal to completely recalculate the MST of an environment whenever the environment changes, as this is expensive in terms of communication and computation. Hence, to avoid such issues, upon certain types of graph change, we only find the spanning tree of a small part of the graph at a time. This allows us to reduce communication and computation overheads.

4.1 The Algorithm

The general idea of iGHS (iterative GHS) is to run GHS only on subgraphs of the whole problem. Specifically, given a spanning tree, the whole factor graph, and a node to add, we take a subgraph of the graph and run GHS on that in order to find a MST of this subgraph. We choose the subgraph by using a variable k, which defines the depth of nodes in the graph we consider, measured from the node to be added. This is illustrated in Figure 2, where part (a) shows the original spanning tree, with node x to be added, and parts (b) and (c) highlight the nodes which are included in subgraphs with $k = 1$, and $k = 2$, respectively.



(a) Original Spanning Tree. (b) $k = 1$ (c) $k = 2$

Figure 2: The effect of k on the size of the subgraph. Thick lines are spanning tree branches, thin lines are edges in the graph but not the spanning tree.

Finding the MST of the subgraph in order for the rest of the graph to remain an MST is a challenge. In more detail, if two or more nodes in the subgraph have spanning tree branches to nodes not in the subgraph (we call these *frontier nodes*: they are shown in Figure 2 as nodes with double outlines), then joining these two frontier nodes in the MST could cause a cycle in the spanning tree as a whole, thus invalidating the tree. We can see how this would occur by considering nodes d and e in Figure 2(c): both nodes have a spanning tree branch in the remainder of the graph, so making a spanning tree branch either on edge (a, d) or edge (d, e) would connect d and e, thus creating a cycle in the rest of the graph. In order to combat this, we identify the frontier nodes, and ensure that we only connect to *one* of these nodes, on *one* edge. We can then guarantee we have not introduced a cycle.

iGHS operates in two phases: (1) a flooding phase which determines which nodes and edges are in the subgraph to be

[5]An allocation that balances the computational load among agents might be desirable, but is beyond the scope of the current paper.

Algorithm 1 Phase 1 of the iGHS algorithm, at a node n.

```
Require: $possEdges = adj(n), lastCount = -\infty,$
    $inEdges, exEdges = \emptyset$
1: At starting node, given $k$:
2: $lastCount = k;$
3: Send $Flood(k-1)$ to all $n' \in possEdges$
4: On receipt of $Flood(d)$ on edge $j$:
5: if $d > lastCount$ then
6:    $lastCount = d; inEdges = inEdges \cup j;$
7:    $possEdges = possEdges \setminus j; exEdges = exEdges \setminus j;$
8:    if $d < 0$ then // Node is not in the subgraph
9:       Send $External$ on edge $j$, then halt.
10:   else // Node is in the subgraph
11:      Send $Flood(d-1)$ on all $e \in \{possEdges \setminus j\}$
12: if $possEdges = \emptyset$ then Send $FloodAck$ on edge $j$, then halt.
13: else Put message on end of queue.
14: On receipt of $External$ on edge $j$
15: $exEdges = exEdges \cup j; possEdges = possEdges \setminus j$
16: On receipt of $FloodAck$ on edge $j$
17: $inEdges = inEdges \cup j; exEdges = exEdges \setminus j; possEdges = possEdges \setminus j;$
```

considered, and (2) a phase based on GHS, which establishes the MST, and adds the minimum weight frontier edge (i.e., edge joining to a frontier node) to the MST.

4.2 Phase 1

Algorithm 1 gives pseudocode for the flooding phase of our algorithm. In this phase, each node identifies which of its adjacent edges in the graph are within the subgraph ($inEdges$), and which are not ($exEdges$). This is done by initially adding all of a node's adjacent edges, $adj(n)$, into $possEdges$, which denotes that they could belong to the subgraph.

In more detail, the flooding phase begins with the node to be added informing its neighbours of the value of k. More specifically, this node sends $Flood(k-1)$ messages to all of its neighbours (line 3). Now, when a node n receives a $Flood(d)$ message, it will propagate further $Flood(d-1)$ messages along edges it is unsure of the status of (line 11), unless it receives a $Flood$ message on edge j containing a value less than 0. If this happens, the node sends an $External$ message on edge j (line 9), informing the node at the other end that j is in $exEdges$ (line 15).

Now, in order for a node to classify an edge into $inEdges$, the node must have received either a $Flood$ message (line 6) or a $FloodAck$ message on that edge (line 16). $FloodAck$ messages are sent when a node has classified all of its edges into $inEdges$ or $exEdges$ (line 12), and so, we can see that they are sent from the nodes that are furthest away from the new node, back toward the node to be added. The algorithm stops when the node to be added has received a $FloodAck$ message from each of its neighbours. Thus, when the node to be added has received $FloodAck$ messages on each of its edges, we know that every node in the subgraph is aware of which of its edges are in the subgraph, and which are not, and phase 1 of the algorithm is complete.

4.3 Phase 2

We give the pseudocode for phase 2 of iGHS in Algorithm 2. For brevity, we omit the sections (lines 3, 9, 14, 20, 35) of the algorithm that are repeated from GHS and focus only on the areas in which we have made changes.

Now, we can classify nodes into frontier nodes and non-frontier nodes: if, for a node n, $exEdges \neq \emptyset$ and the previous status of at least one of the edges in $exEdges$ is $BRANCH$ (i.e., it was a branch in the spanning tree predating this computation), then n is a frontier node.

Algorithm 2 Phase 2 of the iGHS algorithm, at a node n.

```
Require: $inEdges, exEdges, frontier$
1: Procedure wakeup()
2: $bestFrontierEdge = nil; bestFrontierWeight = \infty$
3: if $frontier = false$ then Proceed as GHS wakeup()
4: On receipt of $Connect(L)$ or $Test(L, F)$ on edge $j$
5: if $frontier = true$ then // $j$ is a frontier edge.
6:    $SE(j) = REJECTED$
7:    Send $Frontier$ on $j$
8: else // $j$ is an edge in the subgraph.
9:    Proceed as GHS $Connect(L)$ or $Test(L, F)$
10: On receipt of $InstConnect(L)$ on edge $j$
11: $SE(j) = BRANCH$
12: Procedure test()
13: if there are adjacent edges in state BASIC and not in $exEdges$
    or $frontierEdges$ then
14:    Proceed as GHS test() procedure.
15: On receipt of $Frontier$ on edge $j$
16: $SE(j) = REJECTED;$
17: $frontierEdges = frontierEdges \cup j$
18: $bestFrontierEdge = \min_{e \in frontierEdges} w(e)$
19: $bestFrontierWt =$ weight of $bestFrontierEdge$
20: if $FN =$ nil then Proceed as last received GHS $Connect$
21: else test()
22: Procedure report()
23: if $findCount = 0$ and $testEdge = nil$ then
24:    $SN = FOUND$
25:    Send $Report(bestWt, bestFrontierWt)$ on $inBranch$
26: On receipt of $Report(w, fw)$ on edge $j$
27: if $j \neq inBranch$ then
28:    $findCount = findCount - 1$
29:    if $w < bestWt$ then $bestWt = w; bestEdge = j;$
30:    if $fw < bestFrontierWt$ then $bestFrontierWt = fw; bestFrontierEdge = j;$
31:    report()
32: else if $fw > bestFrontierWt$ and $w = \infty$ then
33:    ChangeFrontierRoot()
34: else
35:    Proceed as GHS $Report(w)$
36: ChangeFrontierRoot()
37: if $SE(bestFrontierEdge) = BRANCH$ then
38:    Send $ChangeFrontierRoot$ on $bestFrontierEdge$
39: else
40:    $SE(bestFrontierEdge) = BRANCH$
41:    Send $InstConnect(L)$ on $bestFrontierEdge$
```

Given this, we now go into more detail on Algorithm 2. When a node's status is $SLEEPING$, and it receives a message, it runs the `wakeup()` procedure (lines 1–3), initialising its variables, and tries to connect to one of its neighbours, if it is *not* a frontier node. When a node receives a connection attempt from its neighbour on edge j (line 4), the algorithm will proceed in one of two ways: if the node is a frontier node, then it marks j as not in the spanning tree and sends a $Frontier$ message along j (lines 5–7), and if the node is not a frontier node, then it proceeds as it would in GHS (line 8). When a node receives a $Frontier$ message along j (line 15), it marks j as not in the spanning tree (line 16), adds j to its list of frontier edges (line 17), and updates its best frontier edge and weight values (lines 18–19), before carrying on as if edge j did not exist (lines 20–21).

Now, when a node has connected on its minimum-weight edge, it tries to connect on each of its other edges, in order of weight (see lines 12–14). However, if a node has no more unclassified edges left, then it runs the procedure `report()` (line 22), and informs its parent in the tree of the lowest weight frontier edge it has (lines 23–25). When the parent receives this report (line 26), it decides which of its neighbours has the lowest-weight frontier edge, and informs its own parent (lines 27–31). However, if the report message is received by the root of the graph (line 32), then the receiving node can be sure that its best frontier edge leads to the spanning tree's best frontier edge. As such, the receiving node sends instruction to connect along the best frontier edge (line 33). Finally, if a node receives an instruction to join on its best frontier edge (line 36), it determines whether its best frontier edge points further down the tree (line 37), or is the edge to connect on (line 39), and either passes the message on (line 38), or connects on the best frontier edge (lines 41, 10–11).

4.4 Properties of iGHS

Having presented our algorithm, we now show that it is correct, and superstabilizing.

Correct. In order to prove the correctness of iGHS, we must show first, that it cannot create a cycle in the overall spanning tree of the graph, and second, that it will make a minimum spanning tree in the subgraph.

First, given that GHS is correct [7], and cannot create cycles, we know that iGHS will not make a cycle in the subgraph it runs on. Thus, it is sufficient for us to show that, in joining our MST to the spanning tree of the rest of the graph, we cannot create a cycle. This is because if a node is connected to a tree-structured graph on a single edge, then it is impossible for that node to have created a cycle in the overall graph. Now, as we are sure that we make a spanning tree across all edges but those that connect to frontier nodes, and we connect that tree to the rest of the graph on a single edge, we can guarantee that our algorithm can, indeed, not make a cycle in the graph overall. Second, we can guarantee that the spanning tree produced by running iGHS on the subgraph is minimum due to the properties of the GHS algorithm.

Superstabilizing. In order to show that iGHS is superstabilizing, we define the following predicates: *legitimacy*: when the algorithm is complete, the spanning tree produced contains no cycles, *passage*: at no time during recovery from a perturbation, are any cycles inadvertently formed. We can say that iGHS is superstabilizing with respect to these predicates: first, because GHS does not form any cycles in its operation, and second, because iGHS is correct.

Now that we have ascertained these properties, we present Bounded Fast Max Sum, which combines iGHS and Fast Max Sum in order to solve distributed constraint optimisation problems on arbitrary graphs.

5. BOUNDED FAST MAX SUM

Here, we introduce the Bounded Fast Max Sum algorithm, which consists of three main procedures running sequentially after a node is added to the graph:

1. **Efficient, superstabilizing spanning tree generation**: using iGHS (Section 4) to re-calculate the maximum spanning tree around the added node.
2. **Fast Max Sum**, to calculate the optimal assignment of variables in the spanning tree. Only the nodes who's utility changes as a result of the addition need resend their Max Sum messages through the tree: if their change in utility does not change, then the decision rules in Fast Max Sum will ensure messages are not transmitted unnecessarily.
3. **WSUM and SOLUTION propagation**, in order to calculate the quality of the solution found.

Now, while adding a node to the graph is always handled in the same manner, this is not the case for removing nodes. In more detail, we must consider both circumstances under which a (function or variable) node can be removed from the graph: first, physical disconnection, such as when a rescue agent is malfunctioning or disappears, and second, when a task has been completed. To support the first case, we introduce contingency plans. Each time a node n is certain of the status of each of its incident edges (i.e. when they have all been marked BRANCH or REJECTED), it chooses its lowest weighted, and hence, most important, BRANCH edge and informs the node at the other end (n_2) that n_2 is n's contingency. Then, if node n is removed from the graph, n_2 will instigate the iGHS algorithm, and maximise the spanning tree around itself. The second case (a task being completed) is slightly different, in that the factor node for that task will not be physically removed from the graph. Instead, the factor node acts as a 'handler' for iGHS, by setting the weight of each of its incident edges to 0 (as their utility would be 0 anyway), and instigating execution of iGHS.

In terms of how many nodes can be added and/or removed simultaneously, the iGHS algorithm imposes a restriction on both. More specifically, as iGHS has a set of values at each node which are used in the spanning tree formation, we must ensure that no one node is involved in more than one instance of iGHS at a time, so that these values are not being overwritten by multiple iGHS instances. Hence, we can say that if two or more nodes are to be added to and/or removed from the graph simultaneously, they must be of distance at least $2k + 1$ away from each other, to avoid overlap in iGHS instances. We next detail the execution of BFMS.

5.1 The Algorithm

As in BMS, each dependency link e_{ij} in the factor graph $\mathcal{FG}$ is weighted as below:

$$w_{ij} = \max_{\mathbf{x}_i \setminus j} \left[\max_{x_j} F_i(\mathbf{x}_i) - \min_{x_j} F_i(\mathbf{x}_i) \right] \tag{2}$$

This weight represents the maximum impact that variable x_i can have over function F_j. Thus, by not including link e_{ij} in the spanning tree, we say that the distance between our solution and the optimal is at most w_{ij}.

Now, once each link has been weighted, our iGHS algorithm is started (see Section 4 for more details). The outcome of the convergence of this algorithm is, initially, a maximum spanning tree.[6] Following this, after every addition to the graph, the spanning tree around the change is maximised, thus iteratively improving a section of the graph.

Next, we run Fast Max Sum on the resulting spanning tree, beginning at the leaves, and propagating up the tree, with each node waiting to receive messages from all of its children before calculating and sending its next message. These messages then propagate back down the tree from the root to the leaves in a similar manner, at which point the algorithm converges, and a variable assignment $\tilde{\mathbf{x}}$ can be computed. Functions with removed dependencies are evaluated by minimising over the removed dependencies, as in BMS.

Finally, the algorithm enters the WSUM and SOLUTION propagation phase, which is the same as that of BMS. More

[6]The maximum spanning tree can be obtained by using iGHS and negating edge weights

specifically, once the leaves have received the Fast Max Sum messages, they can compose new WSUM and SOLUTION messages. If the leaf node is a function, WSUM is the sum of the weights of its removed dependencies, and SOLUTION is $F_t(\tilde{\mathbf{x}}_t)$. If the leaf is a variable, the WSUM and SOLUTION messages are empty. When a node receives WSUM and SOLUTION from all its children, it can process them according to whether it is a function node or a variable node. If the node is a variable node, these messages are the sum of messages from its children. If the node is a function node, the messages are the sum of messages from its children, plus its own values for the removed edge weights (for WSUM) and $F_i(\tilde{\mathbf{x}}_i)$ (for SOLUTION). Once these messages reach the root, the root propagates them back down, so every node is aware of the total weight removed, W, and the solution value, $\tilde{V} = \sum_{t \in T} F_t(\tilde{\mathbf{x}}_t)$.

Now the agents have all information to compute the approximation ratio, as follows: $\rho(FG) = 1 + (\tilde{V}^m + W - \tilde{V})/\tilde{V}$ where FG is the factor graph the algorithm has been run on, $\tilde{V}^m$ is the approximate solution value, and $\tilde{V}$ is the actual solution value.Now, as the approximation ratio tends to 1, this indicates improvement in the solution quality guarantees, because this indicates that the total weight of removed edges is small. Thus, in order to help this, we wish to keep the value of W as low as possible, by only removing low-weight edges (i.e., edges that have low bearing on the overall solution). We can see that the value of k given to iGHS has a bearing on the approximation ratio, too — higher values of k optimise larger sections of the graph. This means iGHS is more likely to remove the lowest weight combination of edges.

5.2 Properties of BFMS

In order to verify that that BFMS is superstabilizing, we must first show that FMS is superstabilizing. We do this subject to the following predicates: *legitimacy*: $U(\mathbf{x}) = \max_{\mathbf{x}} \sum_{t \in \mathbf{T}} F_t(\mathbf{x}_t)$, where $U(\mathbf{x})$ is the total utility of assignment $\mathbf{x}$, and *passage*: the previous assignment of variables is maintained until a new solution has been computed.

PROPOSITION 1. *Fast Max Sum is superstabilizing on tree structured graphs, because it is self-stabilizing on tree structured graphs, and, during stabilization after a change in the graph, the previous assignment of variables is maintained until a new solution has been computed.*

PROOF. First, FMS is an extension to Max Sum, which is proven to converge to an optimal solution on tree structured graphs. Second, when a change occurs in the graph, FMS is run again, and therefore, provided the change did not introduce a cycle into the graph, FMS is guaranteed to converge to the optimal again, reaching a legitimate state within a finite amount of time. This is because FMS does not change the messages sent, it just stops duplicate messages being sent when values at some nodes have not changed as a result of the graph change. FMS is superstabilizing because it has storage at each variable node in order to maintain a previous assignment during recalculation, and so, the passage predicate always holds. □

Given this, BFMS is also superstabilizing, because FMS and iGHS (see Section 4) are. As BFMS combines these two algorithms, we can deduce that BFMS is superstabilizing.

Next, we empirically evaluate BFMS, and compare it to BMS, in order to show the improvements BFMS gives.

6. EMPIRICAL EVALUATION

In order to empirically evaluate Bounded Fast Max Sum, we ran two types of experiment, intended to measure our performance in terms of approximation quality, robustness and utility gained. We compare Bounded Fast Max Sum to Bounded Max Sum (BMS), and a greedy version of Bounded Fast Max Sum (G-BFMS), in which an added node will connect to the rest of the spanning tree on its best-weight edge.

The first experiment intends to show that our algorithm is robust to changes in the graph. In more detail, we compared BFMS with $k = 2$, $k = 3$ and $k = 4$ to BMS, and G-BFMS. More specifically, we ran these algorithms on a series of 50 randomly generated graphs, where, initially, $|A| = 5$ and $|T| = 5$. In addition, we compared results found on graphs with task nodes of degree $\delta_t = 3$, to those with agent nodes of degree $\delta_a = 3$. We used a random look-up table, drawn from a uniform distribution, as the utility function of each task in order to evaluate our algorithm in a general setting. Given this, we ran experiments where we added agents, high-weighted tasks (where, for all values, $u_t(a) \in [0.5, 1]$), and low-weighted tasks (where $u_t(a) \in [0, 0.5]$) in order to demonstrate the impact that k has on the quality of the approximation in these situations. Now, for each experiment, we first ran one of the algorithms, and alternated adding new agents and tasks, one at a time,[7] to the environment. We repeated this process 10 times, recording a number of values after each algorithm run, and calculating the mean of each value at each step, with 95% confidence intervals (which we have plotted on our graphs). In addition, we ran experiments where we added only agents to the environment, to see if adding only one type of node would show a different trend to alternating types. During these experiments, we recorded a number of values.[8] Firstly, we recorded two values from the preprocessing phase of the algorithms: namely, the mean total size of preprocessing messages sent (PTNS), and mean total preprocessing storage used (TSU). The smaller these values are, the better. In addition, we recorded some values from the message passing phase of the algorithms: mean computation units used at each node (MCU), mean total size of messages sent, in bytes (TSS), and the mean approximation ratio obtained (AR). The values of MCU and TSS should, preferably, be small, and the AR should be as close to 1 as is possible.

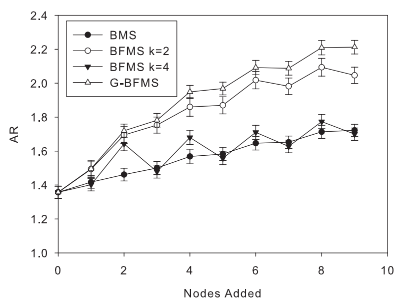
Given that FMS has been shown to outperform Max Sum in terms of MCU, TNS and TSS [13], we hypothesise:

Hypothesis: Bounded Fast Max Sum has lower MCU, and TSS than Bounded Max Sum. In addition, Bounded Fast Max Sum has lower PTNS than Bounded Max Sum.

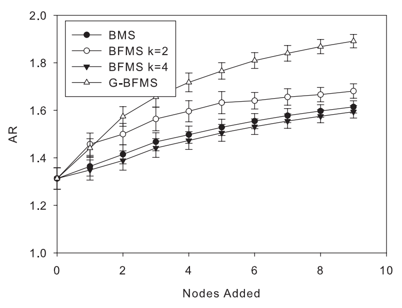
We found from our experiments that varying the degree of tasks and agents had no real effect on the performance of our algorithm. In addition, we found that using high or low utility values of added tasks made little difference to the results either, and for this reason, we present here the results for high-valued utility functions, and $\delta_a = 3$. We can see from Figure 3(a) that alternating adding agents and tasks leads to faster deterioration in the approximation ratio when tasks

[7]As mentioned in Section 5, iGHS is only guaranteed to work if simultaneously added nodes are at least $2k+1$ nodes apart. The evaluation with multiple nodes is beyond the scope of this paper.

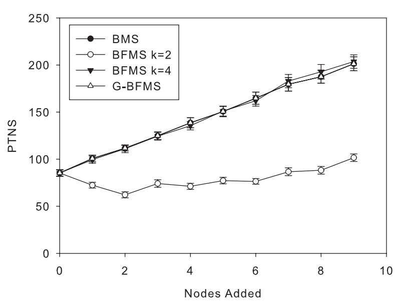
[8]Some of these are typical measures used in the DCOP community [10], approximation ratio comes from BMS [6].



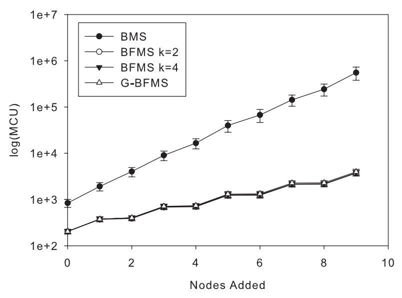
(a) Mean Approximation Ratio: alternating.



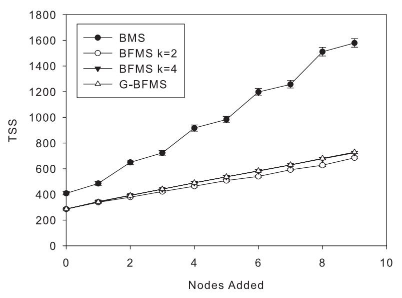
(b) Mean Approximation Ratio: adding only agents.



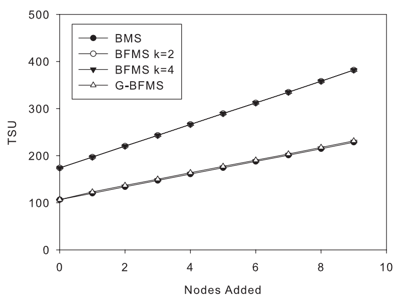
(c) Mean preprocessing messages sent.



(d) Mean computation units used.



(e) Mean total size of messages sent.



(f) Mean total storage used.

Figure 3: Experiment 1 results.

are added. This is because functions have more impact on the approximation ratio than extra variables, and thus will degrade it further when added. Given this, when we only add agents (Figure 3(b)), we see a plateau in BFMS for both values of k, after adding around 5 tasks, where G-BFMS's approximation ratio continues to degrade. Figures 3(a), (b) and (c) show the effect that the value of k has on preprocessing messages sent and approximation quality: whilst $k = 2$ requires 28–53% fewer preprocessing messages than BMS, it produces approximation ratios within 81–94% of BMS. Increasing k to 3 gains approximation ratios within 88–100% of BMS, but reduces the saving in preprocessing messages sent to only 1–7% of BMS. Hence, it can be seen that higher values of k do achieve better approximation ratios, but at the expense of sending more preprocessing messages. Now, Figure 3(d) and (e) show the marked reduction in computation and message size gained by the use of FMS: up to a maximum of 99% over BMS. Finally, Figure 3(f) shows the extra storage at each node needed by iGHS, which, whilst linear in nature, is higher than that of BMS.

It can be seen in Figures 3(c) and (f) that G-BFMS uses very little additional storage and preprocessing messages after a change in the graph. This is because the decision made is entirely local to the node to be added, and, as such, no storage is needed, and only 1 message needs to be sent to confirm the chosen edge forming part of the spanning tree.

Next, our second experiment intends to evaluate the performance of our algorithm in a real scenario, and, as such, we used our own flooding simulator (based on that used in [13]) to compare utility gleaned by BFMS (using $k = 2$ and $k = 3$), G-BFMS and BMS in comparison to using BMS with complete information about all tasks to appear (denoted BMS-OPT). To do this, we set A such that $|A| = 10$ and varied the starting set of tasks to be completed, T, in increments of 5, such that $|T| \in \{0, 5 \cdots, 30\}$. Each task was given a deadline d_t, and a workload w_t, which indicate when the task will expire, and how long it will take to complete, respectively. We also generated a list of 10 tasks which were added to the environment, one at a time, every $|T|$ timesteps: thus, when we say BMS-OPT had full information, we mean that BMS-OPT was given the set T and the complete set of tasks that would be added over time at the start of the simulation. We randomly generated 50 instances of agent and task positions for each amount of tasks, drew the deadline of each task from a uniform distribution $d_t \in U(0, 10 \times |T|)$ and drew the workload from a uniform distribution as $w_t \in U(0, \frac{10\times|T|}{2})$. We ran the three algorithms on each of the 50 instances, over $10\times|T|$ time steps, running each algorithm at each time step. We show the mean total number of tasks completed by each algorithm in Figure 4, along with 95% confidence intervals.

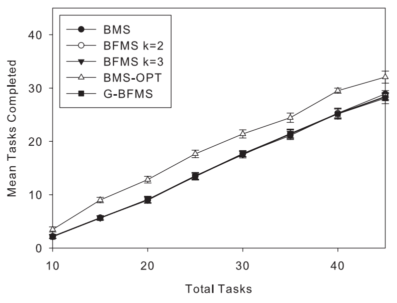


Figure 4: Mean total tasks completed.

We can see from Figure 4 that BFMS with $k = 2$ and $k = 3$ completes within 96–100% of the tasks as BMS, and that performance of BFMS with $k = 3$ is very similar to that of BMS. In addition, we found that BFMS took a much smaller amount of time to converge to a solution at every time step, compared to BMS and OPTBMS which took far longer.

7. RELATED WORK

Using the GHS algorithm [7] as a starting point, we looked into work similar to iGHS in order to locally optimise a spanning tree, without affecting the whole graph. We found some work into distributed dynamic MST algorithms: for example, Cheng et al.'s Dynamic MST protocol [2], which finds the MST of the entire graph. However, these algorithms do not suit our purpose, for we wish to reduce computation and communication done by the algorithm overall, and hence, the amount of spanning tree changed must be limited.

Given this, some work closer to ours is that of localised spanning tree calculation, which is of particular interest in wireless ad-hoc networks [9]. In such domains, low power consumption and memory limit the scope of potential algorithms to those that are localised and power-efficient. In addition, localised minimum spanning tree algorithms such as that of [9] are often constrained by bounds on node degree (i.e., the number of outgoing links from each node), and minimising the length of each links between nodes, in order to conserve precious resources. Fortunately, our domain is not so limited in terms of node degree (in fact, large node degrees would help to increase solution quality), or link length: we can assume that, if links are present in the graph of the scenario, then they are available to be chosen in the MST.

The work of Li et al. is similar to ours, in that their Incident MST and RNG Graph (IMRNG)[9] method finds the MST of nodes within two 'hops' from each node in the graph. However, the mechanism is, in a sense, locally centralised, in that each node finds out the weights of its neighbouring links through message passing, before calculating the MST locally, and informing its neighbours of the result. Our algorithm is not centralised in this way, as iGHS and FMS are both entirely distributed, in that removing a node will not cause either algorithm to collapse. In addition, the mechanism is heavily constrained in order to be compatible with ad-hoc networks, forcing node degree to be at most 6, and fixing the hop count to 2, whereas we allow k (our measure of hop count) and node degree to take any value.

8. CONCLUSIONS AND FUTURE WORK

We have presented an efficient, dynamic, superstabilizing algorithm for distributed constraint optimisation, which is able to provide approximate solutions with quality guarantees, whilst reducing redundant computation and communication in dynamic environments. Our main directions for future work are to consider how the algorithm could adapt to areas of varying communication, and to enable any number of nodes to be added and/or removed simultaneously. More specifically, we are interested to find out if dynamically varying the value of k in response to varied available bandwidth could optimise our solution quality in environments where communication capabilities can vary. Second, our algorithm is constrained in the number of nodes that can be added or removed at any one time, not allowing at simultaneous addition and/or removal of nodes which are less than $2k + 1$ from each other. Thus, we will endeavour to reduce, or possibly remove, this limit.

[9]RNG: Relative Neighbourhood Graph

Acknowledgements

This research was undertaken as part of the ALADDIN (Autonomous Learning Agents for Decentralised Data and Information Systems) Project and is jointly funded by a BAE Systems and EPSRC (Engineering and Physical Research Council) strategic partnership (EP/C 548051/1).

References

[1] S. M. Aji and R. J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.

[2] C. Cheng, I. Cimet, and S. Kumar. A protocol to maintain a minimum spanning tree in a dynamic topology. *SIGCOMM Computer Communication Review*, 18(4): 330–337, 1988.

[3] E W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11): 643–644, 1974.

[4] S Dolev and T Herman. Superstabilizing protocols for dynamic distributed systems. *Chicago Journal of Theoretical Computer Science*, 1997(4):1–40, 1997.

[5] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proc. AAMAS-08*, pages 639–646, 2008.

[6] A. Farinelli, A. Rogers, and N. Jennings. Bounded approximate decentralised coordination using the max-sum algorithm. In *DCR Workshop, IJCAI-09*, pages 46–59, July 2009.

[7] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. TOPLAS*, 5(1):66–77, 1983.

[8] F. R. Kschischang, B. J. Frey, and H. A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2):498–519, 2001.

[9] X. Li, Y. Weng, P. Wan, W. Song, and O. Frieder. Localized low-weight graph and its applications in wireless ad hoc networks. In *INFOCOM-04*, volume 1, pages 431–442, March 2004.

[10] P. J. Modi, W. S. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1-2):149–180, 2006.

[11] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *Proc. IJCAI-05*, volume 19, pages 266–271. AAAI Press, 2005.

[12] A. Petcu and B. Faltings. S-DPOP: Superstabilizing, Fault-containing Multiagent Combinatorial Optimization. In *Proc. AAAI-05*, pages 449–454, 2005.

[13] S. D. Ramchurn, A. Farinelli, K. S. Macarthur, M. Polukarov, and N Jennings. Decentralised Coordination in RobocupRescue. (to appear) *The Computer Journal*, 2010.

[14] Y. Weiss and W. T. Freeman. On the optimality of solutions of the max-product belief propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, 47(2):723–735, 2001.

Divide-and-Coordinate by Egalitarian Utilities: Turning DCOPs into Egalitarian Worlds

Meritxell Vinyals,
J. A. Rodriguez-Aguilar
Artificial Intelligence Research Institute (IIIA)
Spanish Scientific Research Council (CSIC)
Campus UAB, Bellaterra, Spain
{meritxell,jar}@iiia.csic.es

Jesus Cerquides*
WAI, Dep. Matemàtica Aplicada i Anàlisi
Universitat de Barcelona
Gran Via 585, Barcelona, Spain
cerquide@maia.ub.es

ABSTRACT

A Distributed Constraint Optimization Problem (DCOP) [7, 6] is a formal framework that can model many cooperative multi-agents domains. The *Divide-and-Coordinate* (DaC) framework [11] is one of the few general frameworks for solving DCOPs that provides bounds on solution quality for incomplete algorithms. In this paper, we formulate a novel DaC algorithm, the so-called Egalitarian Utilities Divide-and-Coordinate (EU-DaC) algorithm. The intuition behind EU-DaC is that agents would get closer to the agreement, that is to the optimal solution in DaC, when they communicate their local utilities for their decisions instead of their preferred decisions. We empirically show how this new algorithm outperforms DaCSA [11], the other DaC algorithm proposed so far, in all instances. We also show that it is very competitive when compared with bounded MGM k-optimal algorithms [5, 4], eventually outperforming them on some problem topologies. Our results also show how bounds provided by the DaC framework are much tighter than 2-optimal and 3-optimal bounds.

1. INTRODUCTION

In many cooperative multi-agents domains, such as sensor networks [13], distributed scheduling [10], and the configuration of power networks [10] a set of agents choose a set of individual actions whose rewards are dependent on the actions of other agents. A Distributed Constraint Optimization Problem (DCOP) [7, 6] is a formal framework proposed to model these cooperative networks where agents need to coordinate in a decentralized manner to find the joint action that maximize their joint reward.

Since solving a DCOP is NP-Hard [7], complete algorithms that focus on obtaining optimal solutions (e.g. ADOPT [7], OptAPO [6]) are usually unsuitable for dynamic and/or large-scale problems due to their computational and communication costs. Because of the unaffordable price of optimality, researchers have also formulated incomplete algorithms (e.g. DSA [13], DBA [13], max-sum [3]), which provide locally optimal solutions and only require a small amount of computation and local communication per agent. However, although these algorithms scale very well to large networks, they can converge to very poor solutions or even fail to converge. Another limitation is that they do not provide any quality guarantee on their solution, leaving agents with high uncertainty about the goodness of their decisions. As argued in [11, 9], quality guarantees can make a significant difference on incomplete algorithms because they allow agents to reason if it is worth investing more resources on improving their current decisions, to trade-off quality versus cost, by providing a bound on their maximum error. Some works [9, 11] have started to make headway on this direction by defining general frameworks that can provide quality guarantees over DCOP solutions even for incomplete algorithms.

On the one hand, there is the k-optimality framework [9] which defines quality guarantees for k-optimal solutions: solutions that can not be improved by changing any group of k or fewer agents decisions. The Maximum Gain Message algorithms [5, 4], namely MGM-2 and MGM-3, are DCOP approximate algorithms that converge to 2-optimal and 3-optimal solutions respectively.[1] On the other hand, Vinyals et al. [11] recently proposed the *Divide-and-Coordinate* (DaC) framework. In DaC agents iteratively divide an intractable DCOP into simpler local problems that can be individually solved by each agent and thereafter coordinate to reach an agreement over their assigments. As shown in [11], the DaC framework provides an upper bound on the quality of the optimal solution that agents can use to return per-intance quality guarantees. The Divide-and-Coordinate Sugradient Algorithm (DaCSA) [11] is a computational realization of the DaC framework, a bounded approximate DCOP algorithm in which agents coordinate by exchanging their preferred decision. However, several works [3, 12] have shown that communicating the utility of variables assignments instead of only the preferred assignments can lead to benefits in terms of solution quality.

It is this issue that we address in this paper, and to this end, we present a novel DaC algorithm, the Egalitarian Utilities Divide and Coordinate algorithm (EU-DaC). Agents running EU-DaC coordinate by exchanging the local utilities of their variables assigments with their neighbours. Concretely, this paper makes the following contributions:

*This work has been funded by projects IEA (TIN2006-15662-C02-01), Agreement Technologies (CONSOLIDER CSD2007-0022, INGENIO 2010) and EVE (TIN2009-14702-C02-01, TIN2009-14702-C02-02). Meritxell Vinyals is supported by the Spanish Ministry of Education (FPU grant AP2006-04636). JAR thanks JC2008-00337.

[1]MGM-1 is a k-optimal algorithm but no guarantees are given in the k-optimal framework for k=1.

- We formulate a novel computational realization of the DaC approach for which agents: (1) coordinate with their direct neighbours by exchanging their local utilities for shared variables assignments; and (2) update their local problems with the aim of getting closer to the utilities of their neighbours.
- We empirically evaluate the quality solutions of EU-DaC on different network topologies against state-of-the-art algorithms that provide quality guarantees (DaCSA, MGM-2 and MGM-3). Our empirical results show how EU-DaC outperforms DaCSA in all tested scenarios confirming the advantatges of communicating utilities instead of simply decisions. Moreover, it also shows that EU-DaC is competitive when compared with k-optimal algorithms (MGM-2 and MGM-3): EU-DaC solutions quality is similar to k-optimal algorithms and better on structured topologies.
- We experimentally compare the quality guarantees given by the different benchmarked algorithms: DaC quality bounds (EU-DaC and DaCSA) and k-optimal quality bounds (3-optimal for MGM-3 and 2-optimal for MGM-2). Results show that the bounds provided by EU-DaC are much tighter than k-optimal bounds, which for $k = 2$ and $k = 3$ are very loose.

This paper is structured as follows. In section 2 we give an overview of DCOPs and of the DaC framework. Next, in section 3 we describe our decentralised coordination algorithm, the EU-DaC algorithm. In section 4 we present our empirical evaluation of EU-DaC with respect to other state-of-the-art approximate algorithms with quality guarantees. Finally, we draw some conclusions and set paths to future work in section 5.

2. DCOP AND DIVIDE-AND-COORDINATE

2.1 DCOP Definition

A Constraint Optimization Problem (COP) consists of a set of variables, each one taking on a value out of a finite discrete domain. Each constraint (or relation) in this context determines the utility of every combination of values taken by the variables in its domain. The goal of a COP algorithm is to assign values to these variables so that the total utility is maximized.

Let $\mathcal{X} = \{x_1, \ldots, x_n\}$ be a set of variables over domains $\mathcal{D}_1, \ldots, \mathcal{D}_n$. A *utility relation* is a function $r : \mathcal{D}_r \rightarrow \mathbb{R}^+$ with domain variables $\{x_{i_1}, \ldots, x_{i_q}\}$ in $\mathcal{D}_r = \mathcal{D}_{i_1} \times \ldots \times \mathcal{D}_{i_q}$, that assigns a utility value to each combination of values of its domain variables. Formally, a COP is a tuple $\Phi = \langle \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ where: $\mathcal{X}$ is a set of variables; $\mathcal{D}$ is the joint domain space for all variables; and $\mathcal{R}$ is a set of utility relations. The objective function f is described as an aggregation over the set of relations. Formally:

$$f(d) = \sum_{r \in \mathcal{R}} r(d_r) \qquad (1)$$

where d is an element of the joint domain space $\mathcal{D}$ and d_r is an element of $\mathcal{D}_r$.

The goal is to assess a configuration d^* with utility f^* that maximizes the objective function in equation 1. A DCOP [7, 6] is a distributed version of a COP where: (1) variables are distributed among a set of agents $\mathcal{A}$; and (2) each agent receives knowledge about all relations that involve its variable(s). Although an agent can be in charge of one or more variables, hereafter, we assume that each agent a_i is assigned a single variable x_i. Moreover, we focus on binary DCOPs (those whose utility relations involve at most two variables). Therefore, we will refer to unary constraints involving variable $x_i \in \mathcal{X}$ as r_i, and to binary constraints involving variables $x_i, x_j \in \mathcal{X}$ as r_{ij}.

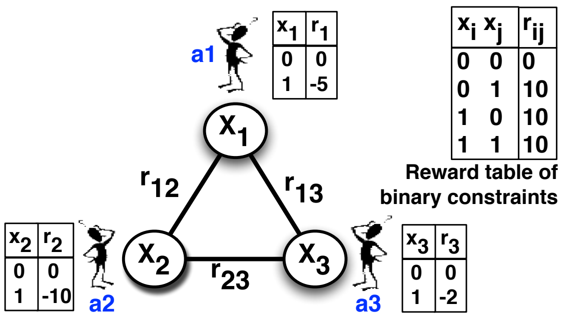

a1
a2
a3
X1
X2
X3
r12
r13
r23
x1 r1 / 0 0 / 1 -5
x2 r2 / 0 0 / 1 -10
x3 r3 / 0 0 / 1 -2
xi xj rij / 0 0 0 / 0 1 10 / 1 0 10 / 1 1 10
Reward table of binary constraints

Figure 1: Example of a DCOP constraint graph .

DCOPs are usually represented by their constraint graphs, where nodes stand for variables and edges link variables that have some direct dependency (appear together in the domain of some relation). Figure 1 shows an example of a binary DCOP in which agents choose values from $\{0, 1\}$ represented by its constraint graph. For instance, note that relation r_{12} is known by agent a_1, that controls variable x_1, and agent a_2, that controls variable x_2. In this context, the neighbours of some agent a are those that share some constraint with a. Thus, in figure 1, a_2 and a_3 are neighbours of a_1 because a_1 shares relation r_{12} with a_2 and relation r_{13} with a_3. Each relation shows its rewards in a table. Thus, agent 3 has a reward of -2 to set its variable to 1 and each pair of agents have a reward of 10 when they set at least one of their variables to 1.

2.2 Divide-and-Coordinate framework

This section defines the Divide-and-Coordinate (DaC) framework, first introduced elsewhere [11]. The DaC framework is an approach that allows agents to distributedly solve a DCOP by exploiting the concept of agreement. The key idea behind the DaC approach is the following: since solving a DCOP is NP-Hard, we can think of *dividing* this intractable problem into simpler subproblems that can be individually solved by each agent. Therefore, in the DaC framework, agents start with the so-called *divide* stage in which they distributedly break the original problem into subproblems and individually solve them. Figure 2 shows an initial division in which each agent creates its local subproblem from its local relations. If a relation is shared among multiple agents, they split the relation by dividing the rewards in equal parts. Thus, the local problem of agent a_1 is composed of its local relation over its variable x_1 and all binary relations that include its variable, namely r_{12} and r_{23} with splitted rewards (table on the left shows rewards for binary relations). Naturally, when solving individual subproblems agents may assign different values to their sharing variables getting in conflict about their values. For instance, in the example of figure 2 variable x_1 is assigned by the three agents independently getting in conflict agent a_1 and a_2 over the value of its assignment.

Thus, agents proceed to coordinate, in the so-called *coordinate* stage, by exchanging some coordination information, namely $\{\Psi\}$ about their disagreements with their neighbours. Agents subsequently employ such information to update their underlying local subproblems to create a new division, in the next *divide* step, that brings them closer to an agreement. Thus, in example of figure 2, when coordinating agents a_1 and a_2 will exchange information about their conflict over x_1 that will use to update their local subproblems.

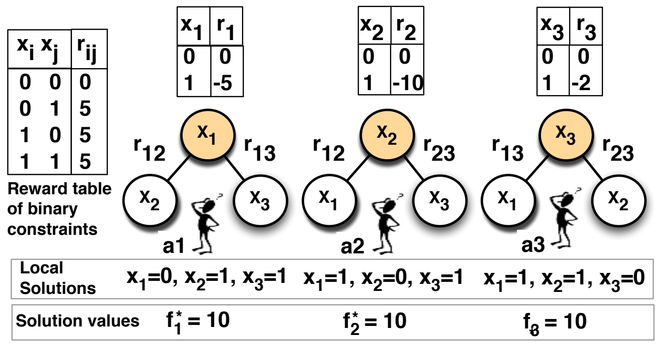


Figure 2: Subproblems for the DCOP in figure 1, divide step ($t = 0$) .

The DaC framework allows agents to distributedly provide bounded solutions for DCOPs by making use of the following two properties (described in [11]):

(**Proposition 1**) *the sum of the solutions of individual agents' subproblems is always an upper bound on the quality of the global (optimal) DCOP solution.*

(**Proposition 2**) *if all agents reach an agreement on a joint solution when optimizing their local subproblems, such a solution is the optimal one.*

Thus, to solve a DCOP by DaC, agents update their local subproblems by exchanging information with their neighbours, exploring the space of valid divisions, to find a division such that the solution of individual subproblems agree (since they know by proposition 2 that this will be the solution of the DCOP). However, even when agents do not agree on their assignments, they can provide with bounded anytime solutions by generating assignments closer to the agreement (that are expected to be better than randomly generated) bounded by the upper bound on its quality of proposition 1.

The DaC framework is an abstract approach that can result in different bounded approximate algorithms for DCOPs because the information that is exchanged among agents in the *coordinate* step and how agents use such information to update their problems in the *divide* step is not specified. The DaC framework only requires that after each agent updates each underlying problem the set of problems still are a original division of the DCOP. Thus, in [11], Vinyals et al. formulated the first DaC algorithm, the Divide-and-Coordinate Subgradient algorithm (DACSA) where agents coordinate by exchanging their preferred decisions on a formalism based on Lagrangian dual decomposition and subgradient methods.

Next, we will formulate a novel particular computational realization of the DaC approach.

3. EGALITARIAN UTILITIES DIVIDE-AND-COORDINATE

In this section we formulate the so-called Egalitarian Utilities Divide and Coordinate algorithm (EU-DaC), a novel computational realisation of the DaC approach where agents coordinate by exchanging their max-marginal utilities to set their shared decision variables to particular values. Several work in optimization [3, 12] have shown that agents lead to better solutions when they explicitly communicate their utilities for taking particular decisions than when they simply exchange their preferred decisions.

In the DaC algorithm proposed so far, DaCSA [11], agents coordinate by communicating their preferred decisions. Here we propose a new DaC algorithm, EU-DaC, that has each agent: (1) exchanges its local utilities for its shared variables with its neighbours (*coordinate* stage); and (2) updates its local problem with the aim of approaching its local utilities for its shared variables to its neighbours' utilities (*divide* stage) .

The intuition behind the EU-DaC is the following: when agents have the same utilities for setting their shared variables to particular values, they agree on their local assignments[2]. As explained in section 2.2, in the DaC framework this agreement situation also implies that they have found a DCOP solution.

In EU-DaC agents will start by exchanging their max-marginal utilities over their shared variables with their neighbours. The max-marginal utilities of an agent a_s to set some decision variables to particular values is the best utility given by its local subproblem Φ_s when restricted to assignments that satisfy this condition. More formally, the max-marginal utility of an agent a_s for setting a subset of decision variables $\mathcal{X}\rho \subseteq \mathcal{X}_s$ to some values $d_\rho \in \mathcal{D}_\rho$, namely $\mathcal{U}^s_\rho(d_\rho)$, is defined as:

$$\mathcal{U}^s_\rho(d_\rho) = \max_{d \in \mathcal{D}_{\mathcal{X}_s \setminus \mathcal{X}_\rho}} f_s(d_\rho; d) \tag{2}$$

where f_s is the local objective function of a_s. Take as example figure 3(a) that shows agents' utilities exchanged during these coordinate step given subproblems of figure 2. Observe that agent a_1 exchanges with its neighbour a_2 a message that contains its local utilities for their shared variables, namely x_1 and x_2. In the example, agent a_1 assesses its local max-marginal utilities for its variable x_1 as:

$$\mathcal{U}^1_1(0) = \max_{d \in \mathcal{D}_{23}} r_1(0) + r_{12}(0, d) + r_{13}(0, d) = 10$$
$$\mathcal{U}^1_1(1) = \max_{d \in \mathcal{D}_{23}} r_1(1) + r_{12}(1, d) + r_{13}(1, d) = 5$$

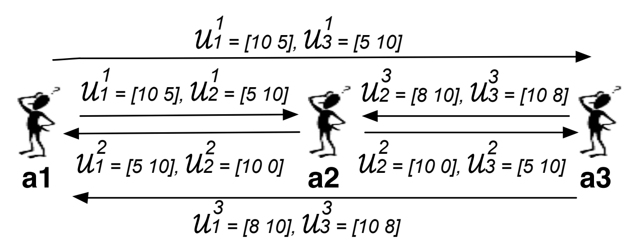
Hence, agent a_1 reports a local max-marginal utility of 10 when setting its variable x_1 to 0 and a local max-marginal utility of 5 when setting it to 1.

Once received these max-marginal utilities, agents proceed to use these information to update its local subproblems in order to get utilities closer to those reported by their neighbours. Thus, agents aim to finding a division of subproblems $\{\Phi_1, \ldots, \Phi_m\}$ such that the set of local max-marginal utilities $\{\mathcal{U}^{s=1,\ldots,m}\}$ for shared variables among the different agents are equal. Formally:

$$\mathcal{U}^i_i(k) = \mathcal{U}^v_i(k) \quad \forall x_i \in \mathcal{X} \, \forall k \in \mathcal{D}_i \, \forall x_v \in N(x_i) \tag{3}$$

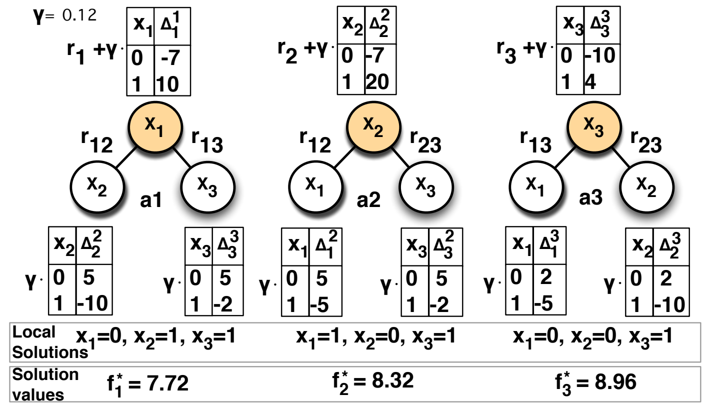
Thus, with the aim of satisfying equation 3, each agent a_s proceeds to update its max-marginal utilities for shared variables by adding the difference between its own local utilities $\{\mathcal{U}^s\}$ and the max-marginal utilities reported by each of its neighbour $a_v \in N(a_s)$, namely $\{\mathcal{U}^v\}$. Thus, each agent a_s updates its max-marginal utili-

[2]This statement is subject to having no ties in agents' local utilities: agent's local utilities to set its shared variables to particular values in their domain are all different.



U1 1 = [10 5], U3 1 = [5 10]
U1 1 = [10 5], U2 1 = [5 10]
U2 3 = [8 10], U3 3 = [10 8]
U1 2 = [5 10], U2 2 = [10 0]
U2 2 = [10 0], U3 2 = [5 10]
a1
a2
a3
U1 3 = [8 10], U3 3 = [10 8]

(a) Coordination step ($t = 0$)



(b) Divide step ($t = 1$)

Figure 3: EU-DaC execution

ties $\{\mathcal{U}^s\}$ using the following equation:

$$\{\mathcal{U}^s\} = \{\mathcal{U}^s\} + \sum_{a_v \in N(a_s)} [\{\mathcal{U}^s\} - \{\mathcal{U}^v\}] \tag{4}$$

Each agent problem Φ_s can be reparameterized in terms of a set of max-utilities over its variables, namely $\{\mathcal{U}^s\}$.[3] Concretely, in the particular case of a binary subproblem $\Phi_s = \langle \mathcal{X}^s, \mathcal{D}^s, \mathcal{R}^s \rangle$ with a tree topology (as the ones shown in figure 2) they can be represented in function of its max-marginal utilities over single and pairwise variables as follows:

$$f_s(d) = \sum_{x_i \in \mathcal{X}^s} \mathcal{U}^s_i(d) + \sum_{r^s_{ij} \in \mathcal{R}^s} \left[\mathcal{U}^s_{ij}(d) - \mathcal{U}^s_i(d) - \mathcal{U}^s_j(d)\right]$$

Thus, in the example of figure 2 agent a_1 can represent its objective function as $f_1(d_1, d_2, d_3) = \mathcal{U}^1_1(d_1) + \mathcal{U}^1_2(d_2) + \mathcal{U}^1_3(d_3) + \mathcal{U}^1_{12}(d_1, d_2) - \mathcal{U}^1_1(d_1) - \mathcal{U}^1_2(d_2) + \mathcal{U}^1_{13}(d_1, d_3) - \mathcal{U}^1_1(d_1) - \mathcal{U}^1_3(d_3)$.

Notice that, using this representation, max-marginal utilities over single variables appear at least once in each problem where the variable is included. Therefore, at each iteration t, each agent a_s updates each subproblem by adding, for each variable in its subproblem $x_i \in \mathcal{X}^s$, the (weighted) difference between its max-marginal utilities and those reported by agents with which it shares such variable. Hence, the agent objective function is updated as:

$$f^t_s(d) = f^{t-1}_s(d) + \gamma \cdot \left[\Delta^{s,t}_s + \sum_{x_i \in N_s} \Delta^{s,t}_i\right] \tag{5}$$

[3]This can be proved by the junction tree theorem[2], that states that any distribution F compiled into a junction tree can be reparameterized in function of the gains (max-marginals) of its cliques $\{\mathcal{C}\}$ and separators $\{\mathcal{S}\}$

where $\Delta^{s,t}_s$ is the coordinator parameter related to its variable x_s:

$$\Delta^{s,t}_s = \textstyle\sum_{x_i \in N_s} \left[\mathcal{U}^{i,t}_s(d) - \mathcal{U}^{s,t}_s(d)\right], \tag{6}$$

$\Delta^{s,t}_i$ is the coordinator parameter related to variable $x_i \in N(x_s)$:

$$\Delta^{s,t}_i = \mathcal{U}^{i,t}_i(d) - \mathcal{U}^{s,t}_i(d), \tag{7}$$

and $\gamma \in (0, 1]$ is a damping parameter that weighs the change over the subproblem.

Next, we describe in detail the phases that agents execute during the EU-DaC algorithm.

3.1 Algorithm description

In this section we fully describe the EU-DaC algorithm, a bounded anytime DCOP algorithm that computationally realises and interleaves the *divide* and *coordinate* stages. On the one hand, during each *divide* stage, each agent updates its local problem by adding the difference between its local max-marginals utilities with those of its neighbours according to equation 5. Then, each agent solves its updated local problem to update its preferred assignments and its assignments value. On the other hand, during the *coordinate* stage, each agent exchanges its local max-martinal utilities over single variables shared with its neighbours. In order to provide anytime solutions even in the case of disagreement, each agent generates at each iteration what is called a *candidate solution* for its variable in the exactly the same way as in DaCSA.

Algorithm 1 presents the pseudocode for EU-DaC. In what follows we describe the main stages of EU-DaC using the trace in figure 3 of a run over the DCOP in figure 1.

***Initialization* stage** (lines 1-2). At the beginning of the algorithm each agent a_i creates its local problem $\bar{\Phi}^0_i$ using its local relations. Relations shared with its neighbours (binary relations) are split in equal parts. Notice that for binary DCOPs, these are always tree-structured problems (acyclic). An example of these initial division for the DCOP of figure 1 is given in figure 2.

***Divide* stage** (lines 4-6). During a *divide* stage, each agent updates its current local problem $\bar{\Phi}^t_i$ with coordination information to subsequently solve it. Firstly, each agent a_i updates its local subproblem by using the coordination information gathered during the last *coordinate* stage, namely Ψ^t_i, using equation 5 (line 5,implemented in method $modifySubproblem$). These coordination messages contain the local max-marginal utilities of their neighbours over their shared variables. Secondly, each agent a_i solves the acyclic COP that composes its local subproblem to obtain its optimal assignments, d^*_i, its value f^*_i and its max-marginal utilities to have their individual variables in each particular state, $\{\mathcal{U}^i\}$ using the max-sum solver [3] (line 6, implemented in method $solveSubproblem$) [4]. Notice that in the very first iteration, agent do not have coordination information and therefore they solve the very initial subproblem. Figure 2 shows agents' local solutions and their values for the initial subproblems. Observe that each agent a_i prefers to set its variable x_i to 0 and the rest of variables to 1 reporting an individual utility f_i of 10. Figure 3(b) shows the same example but in the next *divide* step, where agents have coordination information to update their local subproblems. Thus, for instance, agent a_1 creates a new subproblem that is composed of

[4]Although one can use other solvers such that can solve acyclic problems performing a linear number of operations, max-sum is useful because it returns at the same time the max-marginal utilities over single variables

Algorithm 1 EU-DaC(Φ, γ)

```
Each agent $a_i$ runs:
 1: $bound \leftarrow \infty$; $\{\Psi_i^0\}, \{\Delta^{i,0}\}, solution, C_i \leftarrow \emptyset$; $bestValue \leftarrow -\infty$;
 2: $\overline{\Phi}_i^0 \leftarrow$ createSubproblem($\langle \mathcal{X}^i, \mathcal{D}^i, \mathcal{R}^i \rangle$);
 3: repeat
 4:    /* Divide stage */
 5:    $\overline{\Phi}_i^t \leftarrow$ modifySubproblem($\overline{\Phi}_i^{t-1}, \{\Delta^{i,t}\}, \gamma$);
 6:    $(d_i^{*,t}, f_i^{*,t}, \{\mathcal{U}^i\}) \leftarrow$ solveSubproblem($\overline{\Phi}_i^t$);
 7:    /* Coordinate stage */
 8:    for $x_v \in N(x_i)$ do
 9:       $\Psi_i^v \leftarrow$ makeCoordInfo($d_i^{*,t}, f_i^{*,t}, \langle \mathcal{U}_v^i, \mathcal{U}_i^i \rangle, C_i^{t-1}, \{\Psi\}$);
10:       $\Psi_v^i \leftarrow$ exchangeCoordInfo($\Psi_i^v$);
11:    end for
12:    $\{\Delta_i^{t+1}\} \leftarrow$updateCoordParams($\{\Psi_i^t\}$);
13:    $C_i^t \leftarrow$ selectCandidateSolutions($x_i, C_i^{t-1}$);
14:    if betterBoundAvailable($\{\Psi\}, bound$) then
15:       Update $bound$.
16:    end if
17:    if betterSolAvailable($\{\Psi\}, bestValue$) then
18:       Update $solution$ and $bestValue$.
19:    end if
20: until any termination condition satisfied
21: return $\langle solution, bestValue, bound \rangle$
```

its initial relations r_1. r_{12} and r_{13} along with a weighted coordination parameter Δ_i^1 (weighted by a damping factor γ) for each one of its variables $x_i \in \mathcal{X}^1$. Using a damping factor $\gamma = 0.12$, agent a_3 changes its optimal solution respect to the first iteration. Moreover all agents get a lower value for their local solution than respect to the first iteration. Notice that getting lower utilities for subproblems' solutions is a good indicator because their addition is an upper bound on the optimal solution. Thus, in the DaC framework when agents report lower solution values, their values and their solutions are closer to the optimal ones.

***Coordinate* stage**. During a *coordinate* stage, each agent exchanges coordination information with its neighbours and updates its coordination parameters trying to balance the disagreement among them. Before updating the coordination parameters, each agent a_i exchanges a message Ψ_i^v with each one of its neighbours a_v that contains its max-marginal utilities for their common variables, namely x_i and x_v (lines 8-11). Figure 3(a) shows the max-marginal utilities that are exchanged among agents during the first coordination stage in the example of figure 2. Thus, for example, agent a_1 sends to a_2 the max-marginal utility over its variable x_1, namely $[\mathcal{U}_1^1(0) = 10, \mathcal{U}_1^1(1) = 5]$, and the max-marginal utility over x_2, namely $[\mathcal{U}_2^1(0) = 5, \mathcal{U}_2^1(1) = 10]$. Next, each agent a_1 uses the max-marginal utilities received from its neighbours to update the coordination parameters $\{\Delta^i\}$ following the updates in equations 6 and 7 (line 12). Thus, in the example of figure 3(a), agent a_1 assesses the coordination parameters $\Delta_1^1(0)$ as the difference between the local utility of a_2, $\mathcal{U}_1^2(0) = 5$, and its local gain, $\mathcal{U}_1^1(0) = 10$, plus the difference between the local utility of a_3, $\mathcal{U}_1^3(0) = 8$, and its local utility: $\Delta_1^1(0) = (5 - 10) + (8 - 10) = -7$.

Also, in each *coordinate* stage, each agent a_i selects what is called a *candidate* solution for its decision variable x_i (line 13). This value, namely c_i, does not have to be the same as the preferred assignment of a_i for x_i because to generate candidate solutions agents use the coordinate information received from its neighbours in addition to their local assignment. Although agents can use different strategies to generate their candidate solutions, the intuition is to generate assignments, in presence of disagreement, as much close to the agreement as they can. For example, a typical strategy is that each agent a_i selects the solution c_i for its variable x_i that most agents agree on. Following this strategy, in the example of figure 2, each agent a_i assigns c_i to 1 as a candidate solution for its variable x_i because all neighbours assigned x_i to 1 except from a_i that set it to 0. Thus, the global candidate solution selected is $c_1, c_2, c_3 = 1$. In contrast, in the next coordination step agents a_1 and a_2 will select as candidate solutions $c_1 = c_2 = 0$ (see figure 3(b)), two of the three agents assigned x_1 and x_2 to 0. One can use different strategies simultaneously to generate the selected values. That is why we use $\mathcal{C}_i$ to note the set of candidate solutions (one for each strategy) for variable x_i. Finally, each agent communicates the candidate solution for its variable to its neighbours as a coordination information using the messages exchanged during the next coordinate stage (line 9-10).

Calculate bound and anytime solutions. In order to allow agents to return bounded anytime solutions from the optimal we need to provide agents with a protocol that allows them to distributedly evaluate their candidate solutions and assess the bound. By anytime we mean that agents in EU-DaC hold the best assignment that was generated throughout the search. However it does not mean that solutions will always increment their quality or that the optimal solution will be found if is given more time because EU-DaC can converge to a local optimum. In a distributed environment, each agent a_i only knows for each iteration its local solution f_i^* and the local value for the candidate solution $f_i(\{C\}_i)$. Thus, in the example of figure 2, agent a_1 only knows for the first iteration the value of its local solution, namely $f_1^{*,1} = 10$, and its local value for the candidate solution, namely $f_1(c_1 = 1, c_2 = 1, c_3 = 1) = 5$. Thus, agents need a protocol that allows them to distributedly assess the value of the candidate solution, defined as the sum of all local candidate solutions values, and the bound, defined as the sum of the optimal value of all subproblems. With that purpose, as in DaCSA, EU-DaC implements the protocol detailed in [14] that allows agents to calculate these aggregations of data and synchronize their bound and anytime solutions updates. This protocol requires no additional network load (it uses the coordination messages Ψ already exchanged during the coordination stage to propagate their data) and small (linear) additional space. When all agents have received the coordination information related to the aggregated data for an iteration, they use it to update the bound (lines 14-16) and the anytime solution (lines 17-19), if applies. Thus, in the example of figure 2, agents will have, after a number of propagation cycles, the value of the candidate solution for the first iteration $f(c_1 = 1, c_2 = 1, c_3 = 1) = 13$ and the value of the bound $bound = 30$. Thus, agents will update to solution $[c_1 = 1, c_2 = 1, c_3 = 1]$ with a quality guarantee that the current solution has at least $13/30 \cdot 100 = 43.33\%$ of the quality of the optimal solution. In the next iteration (see figure 3(b)), agents receives the value of the candidate solution for the second iteration $f(c_1 = 0, c_2 = 0, c_3 = 1) = 18$ and a $bound = 25$. Thus, agents update their best solution to $[c_1 = 0, c_2 = 0, c_3 = 1]$ with a quality guarantee of $18/25 \cdot 100 = 75\%$.

Termination conditions. At each iteration, each agent checks if some termination condition is satisfied. Typical termination conditions for EU-DaC are: (1) the gap between the bound and the value of the anytime solution is lower than a threshold; (2) max-marginal utilities are equal across agents (equation 3 is satisfied); or (3) the number of current iterations exceeds a maximum. Notice that unlike DaCSA, in EU-DaC agents can detect convergence even in the case when they have not found the solution. Thus, if condition (2) is satisfied it means that the max-marginal utilities are equal across agents but contains ties, so EU-DaC will not be able to improve after that point.

4. EMPIRICAL EVALUATION

In this section we provide an empirical evaluation of EU-DaC on different network topologies where agents have highly-coupled dependencies. Moreover we benchmark EU-DaC against other DCOP algorithms that can also provide quality guarantees: DaCSA [11] and bounded k-optimal Maximum-Gain-Message (MGM) algorithms [5, 4], namely MGM-2 and MGM-3 [5], by comparing their solution quality over time and the accurancy of their quality guarantees.

Firstly, we describe the different network topologies and how we generate the relations' weights in section 4.1. Next, we analyze our empirical results over these datasets in section 4.2.

4.1 Problem generation

Following [11], we perform our comparison on different network topologies where agents have highly-coupled dependencies. Thus, in our experiments we analyze three network topology alternatives:

Regular grids The constraint graphs are created following a rectangular grid where each agent is connected to its four closer neighbors.

Small-world We generate constraint graphs that show the small-world effect using the model proposed in [8]. The graphs are created by starting from a ring, where each node its connected to its two closer neighbours and adding a small number of random edges. In particular, for each node we use a probability $p = 0.3$ of adding a new edge that connects it to another random node.

Random networks The constraint graphs are created by randomly adding three links for each variable.

As in [11], we are interested in evaluating our algorithm on the presence of strong dependencies among agents. Therefore we also generate constraint values by using mixed Ising model weights [1]. Following an Ising model, the weight of each binary relation r_{ij}, is determined by first sampling a value κ_{ij} from a uniform distribution $U[-\beta, \beta]$ and then assigning

$$r_{ij}(x_i, x_j) = \begin{cases} \kappa_{ij} & x_i = x_j \\ -\kappa_{ij} & x_i \neq x_j \end{cases}$$

Note that the constraint pushes both variables to be similar when κ_{ij} is positive and forces them to be different when κ_{ij} is negative. The β parameter controls the average strength of interactions. In our experiments we set β to 1.6. The weight for each unary constraint r_i is determined by sampling κ_i from a uniform distribution $U[-0.05, 0.05]$ and then assigning $r_i(0) = \kappa_i$ and $r_i(1) = -\kappa_i$.

[5]MGM-1 is excluded of the comparative because it is a 1-optimal algorithm and no bound can be provided with $k = 1$ (see [9])

4.2 Results

Next, we provide details on the particular parameters selected for EU-DaC, MGM-{2,3}[6] and DaCSA in these experiments.

For MGM-2 and MGM-3 we set the probability q of being an offerer to .9, a value that is shown to reach the highest average solution quality on the experiments reported in [5]. Regarding DaC algorithms (EU-DaC and DaCSA), we must specify the strategy used by agents to generate configurations at each pair of *divide* and *coordinate* stages. We use the same two strategies proposed in [11]: (1) each agent assigns to its variable the value in which more agents agree on; and (2) each agent assigns to its variable the value in which more agents agree on when the remaining variables in its subproblem are given by the values selected by the candidate solution in the previous iteration. For EU-DaC we set the value of the damping parameter γ to .5. Finally for DaCSA we used the same step-size for the subgradient step as the one reported in [11], also using a constant step-size of .001 during the first steps when agents do not know any subgradient value.

4.2.1 EU-DaC solution quality

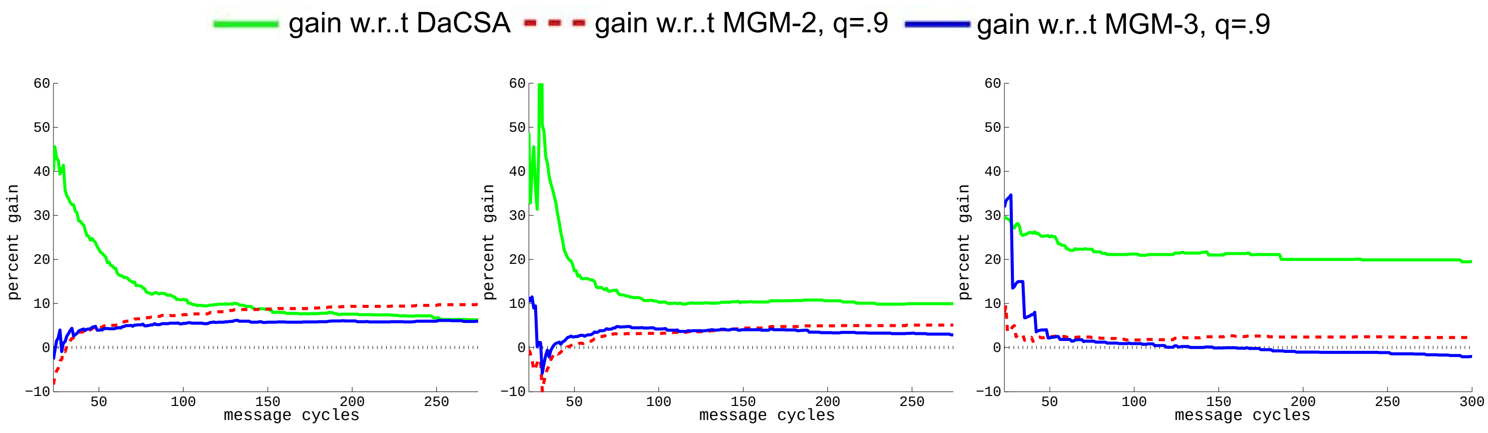
Firstly, we compare these algorithms based on the solution obtained in a number of message cycles. The number of message cycles is a commonly used measure for algorithm efficiency in the DCOP literature [9, 7, 6]. It is specially adequate to our case because all the algorithms benchmarked are low-overhead algorithms. To normalize plots, instead of using the mean of the quality of the solutions we plot the percent gain of EU-DaC respect each benchmarked algorithm. The percent gain of EU-DaC with respect to an algorithm A at iteration t is assessed as $100 \cdot (\frac{q_D - q_A}{q_A})$ where q_D is the value of the solution of EU-DaC algorithm and q_A is the value of the solution of A algorithm. Thus positive values in graphs represent positive gains of EU-DaC respect to other algorithm (higher is better).

Figures 4 (a) (b) and (c) show the results for networks of 100 agent networks on a small-world, regular grid and random topology respectively. Each graph shows the mean among 25 instances of the percent gain of EU-DaC respect to DaCSA, MGM-2 and MGM-3 when varying the number of message cycles.

First, observe that in all experimented topologies EU-DaC outperforms DaCSA, the other DaC algorithm. These results show that indeed when agents explicitly communicate their max-marginal utilities to be in a particular state instead of their decisions and try to get balanced on them they get better results closer to an agreement. Observe that while EU-DaC obtain higher gains, around $40 - 60\%$ respect to DaCSA on structured topologies (small-world and random networks, figures 4 (a)(b)) when increasing messages cycles these gains quickly reduce to around 10%. In random networks (4 (c)), however, the gains of EU-DaC respect to DaCSA are initially lower (around 30%) but remain more constant when increasing the number message cycles (around $20 - 30\%$).

Secondly, when comparing with MGM algorithms (MGM-2 and MGM-3) we observe that EU-DaC outperforms MGM-2 in all the scenarios and the same applies to MGM-3 on structured topologies. The gains of EU-DaC respect to MGM-2 and MGM-3 are lower than respect to DaCSA (around $5 - 10\%$). In random topologies EU- DaC get even negative gains respect to MGM-3 on the

[6]For MGM-2 and MGM-3 we use the code provided in *http://teamcore.usc.edu/dcop/*



(a) Small world 100 variables (b) Regular grids 100 variables (c) Random 100 variables

Figure 4: Graphs showing the percent gain of EU-DaC with respect to DaCSA, MGM-2 and MGM-3 when varying the number of message cycles over different topologies and with mixed Ising weights $\beta = 1.6$

long run. Thus, we can conclude that EU-DaC is very competitive when compared with MGM-2 and MGM-3 getting similar results and even outperforming them on some problem topologies.

4.2.2 *EU-DaC bound quality*

In this section we compare the quality guarantees of EU-DaC with those of DaCSA, MGM-2 and MGM-3 by plotting the percent bound quality of their solution when varying the number of message cycles. The percent bound quality of an algorithm A is assessed as $100 \cdot \frac{q_A}{ub_A}$ where q_A is the value of the solution of A algorithm and ub_A is an upper bound on the value of the optimal solution. Intuitively, a percent bound quality of y says that the current algorithm solution has at least a y percent of the quality of the optimal.

Before analysing the results we should make some comments over the quality guarantees provided by each algorithm. One the one hand, the quality guarantees of EU-DaC and DaCSA are those given by the DaC framework. As explained in section 2.2, DaC algorithms can explicitly calculate an upper bound on the quality of the optimal solution defined as the sum of all individual subproblems solutions in a division. Then, agents assess the percent bound quality using the value of the best evaluated candidate solution so far and the lower upper bound among all tested divisions. Hence, DaC quality guarantees are what are called *instance-per-basis* quality guarantees which depend on the specific problem instance, are known on runtime and vary along the execution of the algorithm. Furthermore, since agents are able to calculate the upper bound in an explicit manner, they can give quality guarantees for every solution generated by the algorithm.

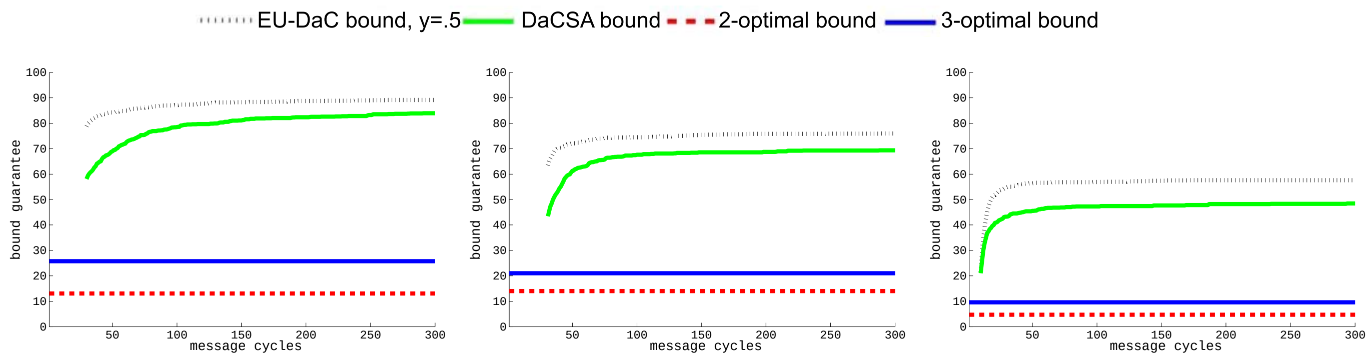
On the other hand the quality guarantees given by MGM algorithms are those of the k-optimal framework: a *worst-case* bound over k-optimal solutions. Two different quality guarantees have been formulated for k-optimal solutions [9]: (1) *general quality guarantees* that only depend on the number of variables and number of relations of the problem; and (2) *graph-based quality guarantees* that use the knowledge of the topology to obtain tighter guarantees. In our experiments, we always plot the *graph-based quality guarantees* which are assured to be better than the general ones. To assess graph-based quality guarantees agents need to solve a linear-fractional problem. That MGM algorithms quality guarantees are worst-case bounds on k-optimal solutions implies that : (a) they have a fixed quality guarantee for any of their converged solutions which can be calculated offline; and (b) they can only provide quality guarantees for solutions generated on convergence (MGM algorithms only achieve a k-optimal solution on convergence).

Figure 5 shows the mean of percent bound qualities provided by DaC algorithms (EU-DaC and DaCSA) and the graph-based guarantees of MGM-2 and MGM-3 over their converged solutions when varying the number of message cycles on the different topologies. First observe that topology influences the quality guarantee of all tested algorithms. In all cases, quality guarantees are higher on small-world topologies than on regular grids than on random topologies. Secondly, results show that the DaC bounds are significantly higher than k-optimal bounds. Moreover, the bounds provided by EU-DaC are always higher (5-10%) than those provided by DaCSA. It is not surprising since DaC quality guarantees use the quality of the best solution, which is higher for EU-DaC, to calculate the bound. For small-world topologies, EU-DaC gives a mean of percent bound of around 85% whereas those of MGM-2 and MGM-3 are around 15% and 30% respectively. For regular grids, EU-DaC gives a mean of percent bound of around 70%, a meaningful bound when compared with those of MGM-2 and MGM-3 of around 15% and 20% respectively. Finally, for random instances EU-DaC gives a mean of percent bound of around 55% whereas those of MGM-2 and MGM-3 are around 5% and 10% respectively.

Therefore, from these empirical results we can conclude that: (1) EU-DaC bounds (and in general DaC bounds) are meaningful enough to provide agents with an awareness of the quality of their solution and to trade-off quality vs resources; and (2) k-optimal guarantees for k=2 and k=3 are very loose and considerably underestimates the solution provided by MGM-2 and MGM-3 on the experiments (results on solution quality show how the value of MGM-2 and MGM-3 solutions are close to those of EU-DaC but the latter gives much higher quality guarantees).

5. CONCLUSIONS AND FUTURE WORK

Our contribution in this paper is twofold. Our first contribution is a novel DaC algorithm, the so-called Egalitarian Utilities Divide and Coordinate (EU-DaC) algorithm. The *DaC* framework [11] is one of the few general DCOP frameworks that can provide bounds on



(b) Small world 100 variables (c) Regular grids 100 variables (d) Random 100 variables

Figure 5: Graphs showing the percent bound qualities when varying the number of message cycles over different topologies and with mixed Ising weights $\beta = 1.6$

DCOP solutions on incomplete algorithms. Unlike DaCSA [11], the other DaC algorithm proposed so far, in EU-DaC agents coordinate by communicating their local max-marginal utilities for the different values of their decisions, instead of only their preferred decisions. Our empirical results show how our novel algorithm outperforms DaCSA in all experimented scenarios.

Our second contribution is to provide the first empirical comparison between the bounds provided by the DaC framework and those provided by the k-optimal framework [9]. Experiments show that DaC bounds improve the accuracy of k-optimal bounds: whereas DaC algorithms get bounds between around 55% and 85%(varying on the scenario), 2-optimal and 3-optimal bounds never go above 15% and 30% respectively in any scenario. Despite of these results, as argued in [9, 4], one advantage of k-optimal bounds, not shared by DaC bounds, is that allows to provide an offline trade-off quality versus time.[7] However, as shown in our experiments, 2-optimal and 3-optimal bounds may very loose and you may be wasting a lot of resources when using this criteria. Therefore, it is reasonable to argue that it may be better for agents to know at runtime the bounds on the maximum-error of their current solution instead of offline bounds that overestimates the error of their converged solution.

As future work we plan to study some unexplored aspects of the DaC framework to allow a broad applicability of this class of algorithms. Firstly, we would like to study the privacy aspects of the DaC framework. Although privacy aspects do not limit DaC algorithms to be applied to domains in which distribution has reasons of parallelism, communication costs and/or robustness (e.g sensor networks, traffic control or the configuration of power networks[10]) we still do not know its applicability to some domains, such as distributed scheduling [10], where privacy is the main issue. Secondly, we aim at designing versions of DaC algorithms that adapts to changes so that it can be applied to dynamic environments.

Acknowledgements. The authors would like to thank Zhengyu Yin, Manish Jain and Milind Tambe to answer our questions about k-optimality and to provide the code for MGM algorithms and graph-based k-bounds to run our experiments.

6. REFERENCES

[7]Assuming that problem structure is known beforehand (k-optimal bounds are reward independent)

[1] R. Baxter. *Exactly Solved Models in Statistical Mechanics*. Academic Press, London, 1982.

[2] R. G. Cowell, S. L. Lauritzen, A. P. David, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[3] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *AAMAS*, May 2008.

[4] H. Katagishi and J. P. Pearce. Kopt: Distributed dcop algorithm for arbitrary k-optima with monotonically increasing utility. In *Ninth DCR Workshop (CP-07)*, 2007.

[5] R. T. Maheswaran, J. P. Pearce, and M. Tambe. Distributed algorithms for dcop: A graphical-game-based approach. In *ISCA PDCS*, pages 432–439, 2004.

[6] R. Mailler and V. R. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS*, pages 438–445, 2004.

[7] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. Adopt: asynchronous distributed constraint optimization with quality guarantees. *Artif. Intell.*, 161(1-2):149–180, 2005.

[8] M. Newman and D. Watts. Renormalization group analysis of the small-world network model. *Phys. Lett. A.*, 263:341–346, 1999.

[9] J. P. Pearce and M. Tambe. Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In *IJCAI*, pages 1446–1451, 2007.

[10] A. Petcu and B. Faltings. Distributed constraint optimization applications in power networks. *International Journal of Innovations in Energy Systems and Power*, 3(1), 2008.

[11] M. Vinyals, M. Pujol, J. A. Rodriguez-Aguilar, and J. Cerquides. Divide and Coordinate: solving DCOPs by agreement. In *AAMAS*, 2010. To appear. http://www.iiia.csic.es/publications/list/author/278.

[12] M. J. Wainwright, T. Jaakkola, and A. S. Willsky. Map estimation via agreement on (hyper)trees: Message-passing and linear programming. *CoRR*, abs/cs/0508070, 2005.

[13] W. Zhang, G. Wang, Z. Xing, and L. Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artif. Intell.*, 161(1-2):55–87, 2005.

[14] R. Zivan. Anytime local search for distributed constraint optimization. In *AAMAS*, pages 1449–1452, 2008.

Real-Time agent reasoning: a temporal bounded CBR approach

Martí Navarro
Departamento de sistemas informáticos y computación
Universidad Politécnica de Valencia
mnavarro@dsic.upv.es

Dr. Vicente Botti
Departamento de sistemas informáticos y computación
Universidad Politécnica de Valencia
vbotti@dsic.upv.es

Dr. Vicente Julián
Departamento de sistemas informáticos y computación
Universidad Politécnica de Valencia
vinglada@dsic.upv.es

ABSTRACT
The main problem in the development of appropriate agent architectures in real-time environments lies in the deliberation process where it is difficult to integrate complex bounded deliberative processes for decision-making in a simple and efficient way. With this in mind, a temporal bounded deliberative case-based behaviour for agents in real-time environments is presented in this paper. More specifically, traditional CBR cycle has been modified as an anytime algorithm facilitating complex deliberative processes for agents in real-time environments.

1. INTRODUCTION
Nowadays, MAS paradigm tries to move Computation to a new level of abstraction: Computation as interaction [18] where large complex systems are viewed in terms of the services they offer, and consequently in terms of the entities or agents providing or consuming services [17]. An interaction-based vision of a system tries to model the system behaviour through the invocation of services among the entities in that system. This vision can be very complex in specific environments for current MAS technology. For example, in real-time environments the responsibility acquired by any entity or agent for the accomplishment of a required service under possibly hard or soft temporal conditions notably increases the complexity of the development of systems of this kind.

Agents which work in the above-mentioned environments, must fulfil specific restrictions, which implies the need for specific agent architectures. A *Real-Time Agent (RTA)* can be defined as an agent with temporal restrictions in, at least, one of its responsibilities (goals, services, tasks, ...) [13]. The RTA may have its interactions bounded, and this modification will affect all interaction processes in the multi-agent system where the RTA is located. The main problem with the architecture of a RTA lies in the deliberation process. This process probably needs to use AI techniques as problem-solving methods to compute more *intelligent* actions, which requires an efficient integration of high-level, deliberative processes within reactive processes. When using AI methods in real-time environments, it is necessary to provide techniques that allow their response times to be bounded. These techniques are mainly based on well-known Real-Time Artificial Intelligence System (RTAIS) techniques [10]. But, complex AI techniques are difficult to bound. Therefore, it would be interesting to integrate these complex deliberative processes for decision-making in real-time agents in a simple and efficient way. Intelligent agents may use a lot of reasoning mechanisms to achieve these capabilities. For example, planning techniques [19] or Case-Based Reasoning (CBR) techniques [2]. In the specific case of CBR, the applications of this technique for controlling some aspects of the deliberative process of agents in MAS developed for specific purposes are many [2]. The main assumption in CBR is that similar problems have similar solutions. Therefore, when a CBR system has to solve a new problem, it retrieves precedents from its case-base and adapts their solutions to fit the current situation. CBR can thus be very suitably applied in agent reasoning, where similar problems should have similar solutions. However, few of the existing approaches cope with the problem of applying CBR as a deliberative engine for agents in MAS with real-time constraints. With this in mind, this work presents a temporal bounded deliberative case-based behaviour as an *anytime* solution [6]. This approach facilitates the addition of deliberative capabilities in a real-time agent, allowing the development of agent architectures with both real-time and deliberative capabilities.

The rest of the paper is structured as follows: section 2 analyses related work; section 3 focuses on how to add a temporal bounded CBR in real-time agents; an application example and the analysis of the results obtained are shown in section 4; finally, conclusions are described in section 5.

2. BACKGROUND
Over the last few years different works have appeared in the literature that are aimed at developing mechanisms to support real-time agents. Along these line, some architectures have been proposed for real-time agents, and research in scheduling agent tasks within the architecture has been carried out. Much of the real-time agent scheduling work relies on the assumption that in order to perform a task, an agent or set of agents may have multiple ways of solving the same problem, each with varying time requirements to compute the result, and with a variation in the quality of the results produced. Typically, the more time available to solve the problem, the higher the quality of the result. This proves very useful in real-time agent scheduling because it allows for a trade-off between the quality of the result and the amount of time required in order to meet specified time constraints. Garvey et al. in [9] presents a design-to-time scheduling algorithm for incremental decision-making that provides a hierarchical abstraction of the problem solving

processes, which describe alternative methods for solving a specific goal depending on the amount of time available. This algorithm is extended in [29] to develop a more general model that can take into account any scheduling criteria, such as time, cost, and quality, and it can use uncertainty as part of the decision-making process. An example of the use of the design-to-criteria model is the DECAF architecture [11] which incorporates scheduling algorithms based on this model.

In addition, other works have tried to design new specific agent architectures for real-time environments, such as the ARTIS agent proposal [3]. The ARTIS architecture is an extension of the blackboard model, which has been adapted to work in hard real-time environments. Moreover, the ARTIS Agents can work together using the SIMBA architecture allowing for the development of multi-agent systems which work properly in social real-time domains [27]. Other agent-related works in real-time systems are: The ObjectAgent Architecture, developed by Pinceton Satellites in 2001 [28], used to control little mono-function satellites launched; the real-time multi-agent system presented by DiPippo et al. [7] providing a middleware that works as a multi-agent platform taking into account the system temporal constraints used for the communication among agents KQML with an extension where the temporal restrictions and the service quality parameter can be reflected. Finally, another example is the work of Prouskas et al. in [25] where they define time-aware agents as agents capable of operating in two temporal dimensions: agent-agent and human-agent, seamlessly combining the predictability and reliability of small-scale real-time exchanges with the fuzzy temporal requirements of large-scale human interactions.

Analysing these works one of the main problem detected is the necessity to integrate, in a bounded way, a new complex deliberation process in these kinds of agents. As mentioned in the introduction, one of the most well-known AI techniques that agents can use in order to reason is the CBR methodology. A CBR system provides agent-based systems with the ability to reason and learn autonomously from the experience of agents. The integration of CBR systems and MAS has been studied taking many different approaches. Therefore, the literature of this scientific area reports research on systems that integrate a CBR engine as a part of the system itself [14], other MAS that provide some or all of their agents with CBR capabilities, or even the development of BDI agents following a CBR methodology [4]. Some examples of multi-agent CBR systems are: the technique called *Negotiated Case Retrieval* [24], the *Federated Peer Learning* framework [23], the *Collaborative CBR (CCBR)* [15], the *Multi-CBR (MCBR)* [16], and finally, a distributed learning methodology that combines individual and cooperative learning in a MAS framework [26]. Cited above are outstanding examples of systems that join research efforts and results of both CBR and MAS together. In addition, the applications of CBR to control some aspects of the deliberative process of agents in MAS developed for specific purposes are many. Most are not intended to cope with the problem of applying CBR as deliberative engine for agents in MAS with real-time constraints. In fact, in the systems reviewed, the concept of real-time is understood as *doing things quickly*. However, in real-time multi-agent systems this concept implies always guaranteeing and meeting deadlines. In accordance with this, the employment of a temporal bounded CBR in the agent deliberation process is still an open topic. The next section presents a temporal bounded agent reasoning process based on a modification of the classic CBR cycle.

3. TEMPORAL BOUNDED CBR-BASED AGENT REASONING

The use of CBR techniques as a reasoning mechanism in real-time agents needs the adaptation of these techniques to be executed guaranteeing real-time constraints. This section explains how to develop temporal bounded CBR-based techniques in order to be integrated inside of a real-time agent architecture. This approach will enable more efficient execution time management, according to the agent's goals. The main problem is that CBR systems are highly dependent on their application domain. Therefore, designing a general CBR model that might be suitable for any type of real-time domain (hard or soft) is, to date, unattainable. In real-time environments, the CBR phases must be temporal bounded to ensure that solutions are produced on time. We therefore present some guidelines with the minimum requirements to be taken into account in order to implement a CBR method in real-time environments.

As a first step, we propose a modification of the classic CBR cycle in order to adapt it to be applied in real-time domains. The typical four CBR phases are grouped into two stages defined as: the *learning stage*, which consists of the revise and retain phases and the *deliberative stage*, which includes the retrieve and reuse phases. Both phases will have their own execution time scheduled. Therefore, the designer can choose to either assign more time to the deliberative stage or keep more time for the learning stage (and thus, design agents that are more sensitive to updates). These new CBR stages must be designed as an anytime algorithm [6], where the process is iterative and each iteration is time-bounded and may improve the final response.

In accordance with this, the operation of our Temporal Bounded CBR cycle (TB-CBR) is the following. Firstly, the main difference that can be observed between the classic CBR cycle and the TB-CBR cycle is the starting phase. Our real-time application domain and the restricted size of the case-base (as explained in the following sections) gives rise to the need to keep the case-base as up to date as possible. Commonly, recent changes in the case-base will affect the potential solution that the CBR cycle is able to provide for a current problem. Therefore, the TB-CBR cycle starts at the *learning stage*, checking if there are previous cases waiting to be revised and possibly stored in the case-base. In our model, the solutions provided at the end of the *deliberative stage* will be stored in a solution list while feedback about their utility is received. When each new CBR cycle begins, this list is accessed and while there is enough time, the learning stage of those cases whose solution feedback has been recently received is executed. If the list is empty, this process is omitted. After this, the deliberative stage is executed. Thus, the retrieval algorithm is used to search the case-base and retrieve a case that is similar to the current case (i.e. the one that characterizes the problem to be solved). Each time a similar case is found, it is sent to the reuse phase where it is transformed into a suitable solution for the current problem by using a *reuse* algorithm. Therefore, at the end of each iteration of the deliberative stage, the TB-CBR method is able to provide a solution for the problem at hand, although this solution can be improved in following iterations if the deliberative stage has enough time to perform them (as an anytime behaviour). Hence, the temporal cost of executing the algorithm (or *cognitive task*) is greater than or equal to the sum of the execution times of the learning and deliberative stages (eq. 1):

$$\begin{aligned} t_{cognitiveTask} &\geq t_{learning} + t_{deliberative} \\ t_{learning} &\geq (t_{revise} + t_{retain}) * n \\ t_{deliberative} &\geq (t_{retrieve} + t_{reuse}) * m \end{aligned} \tag{1}$$

where $t_{learning}$ and $t_{deliberative}$ are the execution time of the learning and deliberative stages; t_x is the execution time of the phase x and n and m are the number of iterations of the learning and deliberative stages respectively.

The Temporal Bounded CBR algorithm (TB-CBR) is shown in Figure 1. This algorithm can be launched when the real-time agent considers it appropriate and there is enough time to execute it. The real-time agent indicates to the TB-CBR the maximum time ($t_{cognitive}$, where $t_{cognitive} >= t_{cognitiveTask}$) that it has available to complete its execution cycle. How this time is calculated is out of the scope of this paper. There are different techniques for dynamically obtaining the slack time in systems of this kind, for instance see [8]. The time t_{max} must be divided between the learning and the deliberative stages to guarantee the execution of each stage. The *distributeTime*($t_{cognitive}$) function is in charge of completing this task. Using this function the designer must specify how the real-time agent acts in the environment. The designer can assign more time to the *learning stage* if it desires a real-time agent with greater capacity to learn. On the contrary, the function can allocate more time to the *deliberation stage*. Regardless of the type of agent, the *distributeTime* function should allow sufficient time for the *deliberative stage* to ensure a minimal answer. The anytime behaviour of the TB-CBR is achieved through the use of two loop control sequences. Through the loop associated with the learning stage is analysed a result obtained in previous execution and decides if this results must be stored in each iteration. The loop involved in the deliberative stage improves the solution obtained in each iteration. The loop condition is built using the *enoughTime* function, which determines if a new iteration is possible according to the time that the TB-CBR has to end each stage.

The first phase of the algorithm executes the *learning stage*. This stage is executed only if the real-time agent has the solutions of previous executions stored in the *solutionQueue*. The solutions are stored just after the end of the *deliberative stage*. On the other hand, the *deliberative stage* is only launched if the real-time agent has a problem to solve in the *problemQueue*. This configuration allows the agent to launch the TB-CBR in order to only learn (no solution is needed and the agent has enough time to reason about previous decisions), only deliberate (no previous solutions to consider and there is a new problem to solve) or both.

```
     Input: $T_{cognitive}$
5.1  ($T_{learning}$,$T_{deliberative}$) ⟵ distributeTime($t_{cognitive}$);
5.2  if solutionQueue $\neq \emptyset$ then
5.3      while enoughTime($t_{now}$,$t_{revise}$,$t_{retain}$,$T_{learning}$) and
         solutionQueue $\neq \emptyset$ do
5.4          {adequate ⟵ analysesResult($r$ ⟵
             pop(solutionQueue))}$^{\leq t_{revise}}$
5.5          if adequate then
5.6              {retainResult($r$)}$^{\leq t_{retain}}$
5.7          end
5.8      end
5.9  end
5.10 if problemQueue $\neq \emptyset$ then
5.11     repeat
5.12         {retrieveCases ⟵
             push(searchCase(adaptProblem(request)))}$^{\leq t_{retrieve}}$
5.13         {solution ⟵ adaptSolution(retrieveCases)}$^{\leq t_{reuse}}$
5.14         bestSolution ⟵ bestSolution(solution,bestSolution);
5.15     until !enoughTime($t_{now}$,$t_{retrieve}$,$t_{reuse}$,$T_{deliberative}$) ;
5.16     push(bestSolution,solutionQueue);
5.17     return bestSolution;
5.18 end
```

Figure 1: Temporal Bounded CBR algorithm

3.1 Data Format considerations

The design decision about the data structure of the case-base and the different algorithms that implement each CBR phase are important factors for determining the execution time of the CBR cycle. The number of cases in the case-base is another parameter that affects the temporal cost of the retrieval and retain phases. Thus, a maximum number of cases in the case-base must be defined by the designer. Note that, the temporal cost of the algorithms that implement these phases usually depends on this number. For instance, let us assume that the designer chooses a *hash table* as a data structure for the case-base. This table is a data structure that associates keys to specific values. Search is the main operation that it supports in an efficient way: it allows access to elements (e.g. phone and address) by using a *hash function* to transform a generated key (e.g. owner name or account) to a *hash number* that is used to locate the desired value. The average time to make searches in hash tables is constant and defined as O(n) in the worst case. Therefore, if the cases are stored as entries in a hash table the maximum time to look for a case depends on the number of cases in the table (i.e. O(♯cases)). Similarly, if the case-base is structured as an *auto-balanced binary tree* the search time in the worst case would be O(log n).

In any case, the retrieval and retention time can be reduced by using an indexing algorithm. These algorithms organize the case-base by selecting a specific feature (or set of features) from the cases, grouping together those cases that share the same values for these features. This reduces the search cost for similar cases (for retrieval or prior to the introduction of new cases in the case-base) to a specific set of cases with the same index as the current case [22]

3.2 Learning stage

This stage begins with the *revise phase*. During this phase, the accuracy of the final solutions obtained in previous executions of the TB-CBR cycle is checked. The algorithm only checks one solution per iteration, fixing the potential problems that it had in cases of erroneous results. The outcome of this phase is used to update the case-base. Thus, the maximum temporal cost of this phase is bounded by the worst-case execution time (WCET) of the revision algorithm:

$$t_{revise} = WCET(f_{revision}(solution)) \tag{2}$$

Note that, in order to guarantee a known maximum execution time, this check must be performed automatically by the computer without human interference. This WCET does not depend on the number of stored solutions or the number of cases in the case-base and again, is fixed by the selected algorithm. In our TB-CBR proposal, this phase is implemented by an *analysesResult* function which determines if the solution is correct or not, and if it has to be included in the case-base.

After the *revise phase*, one of the most important phases in the TB-CBR cycle begins: the retain phase. In this phase it is decided whether a checked solution should be added as a new case in the case-base. Here, maintaining the maximum size of the case-base is crucial, since the temporal cost of most retention algorithms depends on this size. If there is a case in the case-base that is similar enough to the current case, this case (its problem description and solution) is updated if necessary. On the contrary, if there is not a case that represents the problem solved, a new case is created and added to the case-base. Maintaining the maximum size of the case-base could entail removing an old case from it. This decision should be taken by the retention algorithm. Nevertheless, the maximum temporal cost that the retain phase needs to execute one iteration is its WCET.

$$t_{retain} = WCET(f_{retention}(solution, case - base)) \tag{3}$$

The *retain phase* is built by the *retainResult* function which includes the solution in the case-base if it is sufficiently significant. In order to do so, the function determines if the solution entails adding a new case in the case-base or if a similar case exists. If the solution has a similar case in the base-case, the similar case is update with the new data that the TB-CBR can extract from the solution. If the solution entails adding a new case and the case-base is full, the algorithm extracts a case from the case-base that it considers outdated or useless to make space for a new case. Otherwise, the new case is added to the case-base.

3.3 Deliberative stage

The deliberative stage begins with the *retrieve phase.* In this phase the retrieval algorithm is executed to find a case that is similar to the current problem in the case-base. In order to bound the temporal cost of the algorithm and to ensure the adequate temporal control of this phase, the execution time of the algorithm is approximated to its WCET. Thus, the temporal cost of the execution of this algorithm will never exceed the WCET. Since WCET depends on the structure of the case-base and its number of cases, the designer must calculate this WCET and use this time to estimate the time needed to execute an iteration of the retrieval algorithm.

$$t_{retrieve} = WCET(f_{retrieval}(currentCase, case - base)) \tag{4}$$

The execution of the retrieval algorithm will provide a unique case similar to the current problem (if it exists in the case-base). This result is used as input for the reuse phase. However, in the following iterations of the deliberative stage more similar cases can be retrieved with the intention of providing a more accurate solution to the problem. This functionality must be done in the proposed TB-CBR algorithm by means of the *adaptProblem* function, where the problem is adapted to the correct format, and by the *search* function, which searches for similar cases in the case-base.

After this, the *reuse phase* begins. In this phase, the selected case obtained from the retrieve phase is adapted to be used as a potential solution to the current problem. Thus, this case is stored in a list of selected cases. Each time the reuse phase is launched, the adaptation algorithm searches this list and produces a solution by adapting a single case or a set of cases to fit the context of the current problem to be solved. Therefore, the execution time of this algorithm depends on the number of cases that it is working with.

$$t_{reuse} = \begin{cases} WCET(f_{adaptation}(firstCase)) \\ f_{adaptation}(listOfCases) \end{cases} \tag{5}$$

As shown in equation 5, to guarantee that the RTA assigns enough time to execute the cognitive task, the designer must know the WCET to execute the adaptation algorithm in the first iteration (with one case). Thus, the RTA can estimate if the deliberative stage can be completed and provide at least one solution. In order to control the execution time of the adaptation algorithm in subsequent iterations, the RTA must be able to stop the execution of the algorithm if it realises that the time assigned to complete the deliberative stage will be exceeded. Then, the RTA provides the best solution from among the solutions completed in previous iterations. This solution is stored in a list of solutions to be verified in the learning stage. This phase is implemented in the TB-CBR by means of the *adaptSolution* function.

4. APPLICATION EXAMPLE

In order to evaluate the proposed algorithm, this section presents an application example of a MAS including agents with real-time constraints which incorporate a temporal bounded CBR deliberative process. The problem to solve is a part of the automated management of the internal and external mail (post mail, non-electronic) in a department plant. The system must be able to request the shipment of a letter or package from an office on one floor to another office on the same floor, as well as the reception of external mail at a collection point for subsequent distribution. Once this service has been requested, mobile robots must gather the shipment and address it to the destination. Note that each mail or package distribution must be ended before a maximum time, specified in the shipment request. Three types of agents have been defined: the Interface agent, is in charge of gathering user requests, the Floor agent, which gathers the requests and distributes the work among the available robots, and the Robot agent which is in charge of controlling a physical robot and managing the mail list that this robot must deliver. The Robot agent must satisfy critical time restrictions since the tasks that control the robot sensors/effectors have temporal constraints. Furthermore, this agent periodically sends data about its position and state to the Floor agent. This data is used by the Floor agent in order to select the most appropriate agent to send a new delivery/reception request to.

The Robot agents must incorporate temporal bounded reasoning techniques to estimate the appropriate paths to the different target positions they need to achieve. Moreover, a Robot agent receives delivery/reception requests, and it must be able to commit itself or not to such requests. In order to solve this problem we have defined a module which allows the agent to estimate whether it can commit itself. This module, called Temporal Constraint Analysis module, has been added to the Robot agent and incorporates a Temporal-Bounded CBR following the previously proposed algorithm. The next section explains in detail the definition of this module.

4.1 Temporal Constraint Analysis Module

The Temporal Constraint Analysis (TCA) module must decide if a Robot agent has enough time to perform a specific service. To carry out the decision-making process regarding whether or not to contract a commitment to perform the service, the TCA module has been enhanced with a TB-CBR according to the previously presented algorithm. This new TB-CBR copes with a typical planning problem, where a path must be built as a temporal ordering of a set of activities. Therefore, it is necessary to address the problem as a planning process where the final plan (solution) is produced by the adjustment of plans stored in the case-base. The cases of the TB-CBR module are structured as follows:

$$C =< I, F, N_t, N_s, T > \tag{6}$$

where I and F represent the coordinates of a path from the initial position I to the final position F that the robot travelled (one or several N_t times) straight ahead in the past, N_s stands for the number of times that the robot successfully completed the path within the case-based estimated time and T shows the series of time values that the robot spent to cover that route. Note that only straight routes are stored as cases, since we assume that they are the quickest way between two points. Therefore, the TB-CBR estimates the duration of new paths by means of a function $t : T \to f(T)$ computed over the temporal values of similar previous paths. The expected time T_s to perform a path that consists of a collection of known sub-paths is the aggregation of the estimated time for each one of these sub-paths:

$$T_s = \sum_{i=0}^{I} t_i \tag{7}$$

Finally, the series of observed execution times could also allow the TB-CBR to estimate a *success probability* $P(T_s)$ for a request to be performed within a specified time. This is interesting data for agents, which could use this probability to make strategic decisions about their potential commitments. Setting a *confidence factor* (CF) that represents a minimum threshold for the success probability, agents would commit themselves to fulfilling a delivery/reception mail request if:

$$\exists T_s / P(T_s) \geq CF \wedge Ts \leq deadline \tag{8}$$

Thus, agents with riskier strategies could undertake commitments with lower confidence values than more cautious agents. The following subsections present the selected case-base format and the operation of each stage.

4.1.1 Data format

	analysis-Result()	retain-Result()	search()	adapt-Solution()
Asymptotic cost	O(1)	O(n)	O(n)	O(1)
case-base size	wcet	wcet	wcet	wcet
50	1	609.20	448.83	1
60	1	665.39	511.27	1
70	1	731.59	525.36	1
80	1	864.64	687.76	1
90	1	929.12	753.20	1
100	1	1025.61	847.98	1

Table 1: Asymptotic and temporal costs analysis in the case of study (all times in nanoseconds)

A Hash table has been chosen as a case-base structure as mentioned in section 3.1. The *WHAT* tool [21] has been used in order to obtain the worst-case execution time of the different functions of the algorithm which involve access to the case-base. This tool has been adapted to be used in Real-Time Java language over SUSE Linux Enterprise Real Time 10 as a real-time operating system. The results of this analysis can be seen in Table 1. The table shows the temporal behaviour of the different functions according to the number of cases stored in the case-base. The times obtained have been used in the temporal analysis of the different parts of the algorithm (in the *enoughTime* function, for example).

4.1.2 Learning stage

As shown in the previous section, the TB-CBR starts with the learning stage (revision and retention phases). Once the Robot Agent has finished the shipment service, it sends a report to the TCA module with the coordinates of each path that it finally travelled straight ahead on, and the time that it took to do so. The TCA stores this information until the revision phase is launched. Thus, the manager can check the performance of the TB-CBR by comparing the time estimated by the algorithm and the time that the robot finally took to complete the journey. To do this, the *analysisResult* function, previously defined, has been implemented. Note that if we were in a static domain, the agent could try to perform the shipment by following the same route that ended successfully in the past. However, due to the fact that some new obstacles could be found along the route, the design decision to report the specific paths that the Robot Agent has travelled along has been taken.

The second step of the reasoning cycle considers the addition of new knowledge in the case-base of the TB-CBR. As pointed out before, the size of the case-base must be controlled and therefore, only useful cases should be added (and correspondingly, out-of-date cases must be eliminated). Therefore, decisions regarding the addition of a new case in our model is crucial. In this example, the *retainResult* function has been implemented to include a simple but effective procedure by defining a threshold α below which two points must be considered to be nearby in our shipment domain. Let us consider a new case c with coordinates (x_i^c, y_i^c) (initial point) and (x_f^c, y_f^c) (final point) to be added in the case-base. Following an *Euclidean* approach, the distance (dissimilarity) between case c and each case z of the case-base can be computed with:

$$dist(c,z) = max \quad (\sqrt{(x_i^c - x_i^z)^2 + (y_i^c - y_i^z)^2}, \sqrt{(x_f^c - x_f^z)^2 + (y_f^c - y_f^z)^2}) \tag{9}$$

So, the new case will be included in the case-base iff:

$$\forall z \in caseBase \ / \ dist(c,z) > \alpha \tag{10}$$

In this case, the new case $< (x_i^c, y_i^c),\ (x_f^c, y_f^c),\ 1,\ 1,\ time >$ will be added to the case-base ('1' values stand for this first time that the path has been successfully travelled along). Note that the addition of new cases is always conditioned to the existence of 'free space' in the case-base. Otherwise, a maintenance cycle will be triggered, deleting, for instance, those old cases that have low usage. If a similar case in the case base has been identified, the number of times that the agent has travelled the path that represents the case (N) will be increased by 1 and the time taken to travel the current path will be added to the time series of that case.

4.1.3 Deliberative stage

Firstly, the TB-CBR must adapt the problem to the case-base structure, implementing the *adaptProblem* function proposed in the general algorithm. After this, the TB-CBR, by means of the *search* function, searches its case-base to retrieve a case that represents a similar path along which the Robot Agent travelled in the past. Then, for each retrieved case, the algorithm uses a *confidence function* to compute the probability of being able to travel from an initial point to a final point in an area without diverting the agent's direction. It is assumed that the probability of the shortest paths being affected by unpredictable circumstances which could deviate the agent from its route is lower and hence, they are preferred to longer ones. In the best case, there will be a case that covers exactly, or very approximately, the same path along which the agent has to travel. Therefore, the time needed to perform the shipment can be estimated by using the time taken in the previous case. Otherwise, the route could be covered by aggregating a set of cases and estimating the global time by adding the time estimation for each sub-case. If the route can somehow be composed using the stored cases, the next confidence function will be used:

$$f_{trust}(i,j) = 1 - \frac{dist_{ij}}{maxDist} * \frac{N_s}{N_t} \text{ where } dist_{ij} \leq maxDist \tag{11}$$

where $dist_{ij}$ is the distance travelled N_t times between the points $< i,j >$, N_s represents the number of times that the robot has travelled along the path within the case-based estimated time and *maxDist* specifies the maximum distance above which the agent is unlikely to reach its objective without finding obstacles. In the worst case, the agent would never have travelled along a similar path and hence, cannot be composed using the cases stored in the case-base. If this is the case, a confidence function that takes into account the distance that separates both points will be used:

$$f_{trust}(i,j) = \begin{cases} 1 - \frac{dist_{ij}}{const_1} & \text{if } 0 \leq dist \leq dist_1 \\ 1 - const_2 * dist_{ij} & \text{if } dist_1 < dist \leq dist_2 \\ \frac{dist_{ij}}{dist_{ij}^2} & \text{if } dist_2 < dist \end{cases} \tag{12}$$

where *const1* and *const2* are normalisation parameters defined by the user, $dist_{ij}$ is the Euclidean distance between the initial and final points of the path $< i,j >$ and $dist_1$ and $dist_2$ are distance bounds that represent the thresholds that delimit near, medium and far distances from the initial point. This function computes a smoothed probability of the robot being able to travel along its path straight ahead. As the distance between the initial and final point increases, the confidence in travelling without obstacles decreases.

Once the probability of reaching the robot's objective is computed for each case, the complete route with the maximum probability of success from the starting point to the final position must be selected. This route is composed using a selection function $F(n)$ (13), which follows an $A*$ heuristic search approach [12]. The function consists of two sub-functions: $g(n)$ (14) which computes the case-based confidence of travelling from the initial point to a certain point n and $h(n)$ (15) which computes the estimated confidence level of travelling from the point n to the final point (always better than the real confidence level). Finally, the function $T(n)$ (16) checks if the Robot Agent has enough time to complete the shipment service by travelling along this specific route. Otherwise, the algorithm prunes the route. The function consists of two sub-functions: *time(n)* (17) which computes the case-based time of travelling from the initial point to a certain point n and *E(n)* (18) which computes the estimated travel time from point n to the final point. In (17) $dist_{mn}$ represents the distance between the last point m visited by the algorithm and the current point n, V_{robot} is the speed of the robot, $f_{trust}(m,n)$ corresponds to (11) or (12) (depending on the possibility of composing the route by using the cases in the case-base) and the constant $const_{trust} \in [0,10]$ shows the degree of caution of the robot agent. Bigger values of this constant stand for more cautious agents.

$$\begin{aligned} F(n) &= g(n) * h(n) && (13) \\ g(n) &= g(m) * f_{trust}(m,n) && (14) \\ h(n) &= 1 - \frac{dist_{nf}}{maxDist} \text{ where } dist \leq maxDist && (15) \\ T(n) &= time(n) + E(n) && (16) \\ time(n) &= time(m) + \frac{dist_{mn}}{V_{robot}} + \frac{const_{trust}}{f_{trust}(m,n)} && (17) \\ E(n) &= \frac{dist_{nf}}{V_{robot}} && (18) \end{aligned}$$

Finally, if the TB-CBR algorithm is able to compose the entire route with the information stored in the case-base, it returns the case-based probability of performing the shipment service on time. Otherwise, it returns the product of the probability accumulated to that moment and a pessimistic probability of travelling from the last point that could be reached, by using the cases in the case-base, to the final point of the route. Finally, in the event of all possible solutions computed by the algorithm exceeding the time assigned to fulfil the service, it returns a null probability of performing the service successfully. The *bestSolution* function is in charge of returning the solution obtained.

The probability returned by the TB-CBR algorithm will be used to determine whether the agent can commit itself to performing the delivery/reception service, or whether it should reject the service. Each agent has a *confidence value.* If the returned probability is greater than or equal to the confidence value, then the service will be accepted for execution. This *confidence value* allows different behaviours of the agent to be defined. A cautious agent will have a high confidence value and thus, will only accept those services with a high probability of fulfilling the goal. On the other hand, a fearless agent will have a low confidence value.

4.2 Tests and Results

The example has been implemented using the jART platform [20] (which is specially designed for real-time multi-agent systems) and RT-Java [1] as the programming language. Once the example was implemented several simulation tests were conducted to evaluate the proposal. A simulation prototype was implemented using a Pioneer 2 mobile robot simulation software (specifically, the Webots simulator [5]). The tests were mainly conducted to show the benefits and correct behaviour of the TB-CBR integrated on the TCA module in the Robot agent. In order to compare the obtained results, the different experiments were tested on a system where the TCA module is included or not.

It is important to note that, in the case of the Robot agent without the TCA module, the received mail orders are stored in a request queue. This queue only stores up to five pending requests. When the Robot Agent receives a request, if it has space in the queue, the Robot Agent accepts the commitment associated to the request. Otherwise the request is rejected. In each case, the mail or package distribution must be ended before a maximum time, and the robot control behaviour must guarantee the robot's integrity, which implies hard real-time constraints. In the tests carried out, we assume that all requests have the same priority. Therefore, the number of requests successfully managed by the Robot Agent is an adequate metric to verify the improvement made by the use of the TCA module. If requests have different priorities, this metric will not be correct. In this case, fulfilling tasks with high priority is more important than fulfilling a greater number of low priority tasks. The tests consisted of groups of 10 simulations with a duration of 5 minutes. The Floor Agent received between 5 and 30 requests during this time. Each experiment was repeated 100 times and results show the average value obtained.

The first set of experiments investigates the success rate of accepted commitments according to package or mail arrival frequency (Figure 2). This figure shows that using the TCA module with a confidence value of 90 % is very efficient in order to maintain the success rate close to 100%. In contrast, if the Robot Agent does not use the module, the success rate decreases as the number of requests increases. Even when the saturation of requests in the system was very high, the agent with the TCA module still had a success rate of approximately 90% independent of the confidence value. The next test analyses the behaviour of the TCA module in a Robot agent as it receives new requests by increasing the number of queries. As shown in Figure 2, the number of estimations that the TCA performs decreases as new requests are queried. This demonstrates that as the number of requests increases, the case-base learns the new information properly and hence, the number of routes that can be composed with the cases increases (and an estimation is not necessary). Figure 3, which shows the relation between the number of cases in the case-base and the percentage of estimated routes, also supports this conclusion. Finally, the

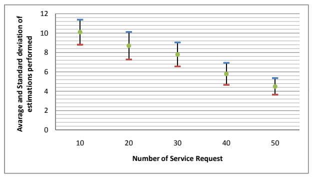


Figure 2: $\bar{x}$ and σ of estimations vs no. of service requests

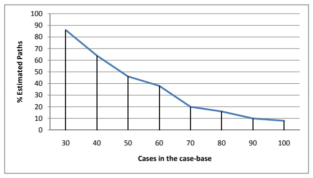


Figure 3: % of estimated paths vs no. of cases in the case-base

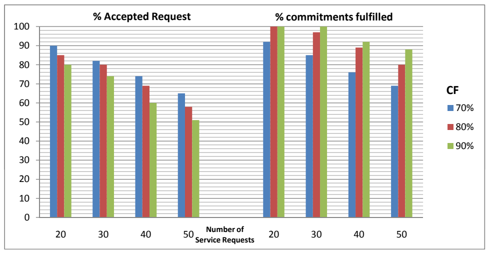


Figure 4: (a) % of accepted requests vs no. of service requests. (b) % of commitments fulfilled vs no. of service requests

percentage of distrust from which an agent can commit itself to performing a service was also checked (modifying the confidence factor values from 70%, 80% and 90%). As expected, bigger confidence percentages resulted in agents committing themselves to performing more tasks (Figure 4a). However, in such cases the percentage of services accepted and completed on time decreases, since the agent committed itself to the performance of a large amount of services (Figure 4b). Logically, when the confidence factor increases the acceptance percentage is lower. The results obtained in Figure 4a and Figure 4b have been merged in Figure 5, which shows the behaviour for each confidence factor (CF) comparing accepted requests fulfilled versus those not fulfilled. We can extract that if the agent selects a high CF, it obviously accepts fewer proposals but it can fulfil all of the proposals even with high request frequencies. On the other hand with a low CF the agent behaviour results are completely opposed. Taking these results into account, the agent

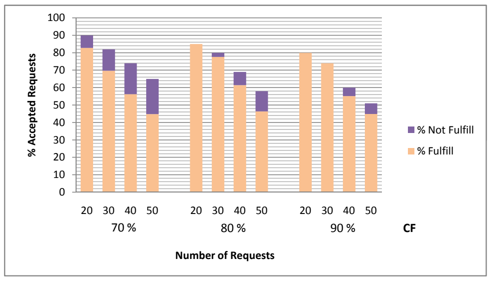


Figure 5: Analysis of the robot agent behaviour for each confidence factor

must dynamically vary its confidence factor. How it can be adapted depends on the context, mainly in the current load of the agent. With a high load the agent must increase its CF in order to avoid possible failures in its commitments.

On the contrary, the agent can decrease its *CF*, and probably accept more requests, while the system load is not excessive.

5. CONCLUSIONS

This paper has centred on the problem of the efficient integration of high-level deliberation processes with real-time behaviours in complex and dynamic environments. With this idea in mind, this paper has proposed the integration of a deliberative capacity, based on bounded case-base reasoning techniques, into a real-time agent. More specifically, the work has proposed a new temporal-bounded CBR, based in a anytime algorithm, to be integrated as a deliberative capability inside a real-time agent architecture. A multi-agent scenario with temporal bounded deliberative and reactive processes has been implemented using this approach. Specifically, an automated management simulation of internal and external mail in a department, allowing it to decide if an agent can commit itself to performing delivery/collection services without exceeding the maximum time assigned for performing these services. The results are promising for deployment within a real scenario in the near future and make the proposal very suitable for application in dynamic environments, in which learning and adaptation to constant changes is required and temporal boundaries exist.

Acknowledgments

This work is supported by TIN2009-13839-C03-01 projects of the Spanish government, PROMETEO/2008/051 project, FEDER funds and CONSOLIDER-INGENIO 2010 under grant CSD2007-00022.

6. REFERENCES

[1] Real-time specification for java. http://www.rtsj.org.

[2] A. Aamodt and E. Plaza. Case-based reasoning: foundational issues, methodological variations and system approaches. *AI Comm.*, 7(1):39–59, 1994.

[3] V. Botti and C. Carrascosa. Modelling agents in hard real-time environments. In *MAAMAW'99*, volume 1647 of *LNAI*, pages 63–76, 1999.

[4] J. Corchado and A. Pellicer. Development of CBR-BDI Agents. *International Journal of Computer Science and Applications*, 2(1):25–32, 2005.

[5] Cyberbotics. Webots. http://www.cyberbotics.com/.

[6] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proc. of the 7th National Conference on Artificial Intelligence*, pages 49–54, 1988.

[7] L. C. DiPippo, V. Fay-Wolfe, and et al. A real-time multi-agent system architecture for e-commerce applications. *ISADS*.

[8] A. Garcia-Fornes, A. Terrasa, V. Botti, and A. Crespo. Analyzing the schedulability of hard real-time artificial intelligence systems. *EAAI*, pages 369–377, 1997.

[9] A. Garvey and V. Lesser. Design-to-time real-time scheduling. *IEEE Transactions on Systems, Man and Cybernetics*, 23(6), 1993.

[10] A. Garvey and V. Lesser. A survey of research in deliberative real-time artificial intelligence. *The Journal of Real-Time Systems*, 6:317–347, 1994.

[11] J. Graham and K. Decker. Towards a distributed, environment-centered agent framework. In *Intl. Workshop ATAL-99*, 1999.

[12] P. E. Hart, N. Nilsson, and B. Raphael. A formal basic for the heuristic determination of minimum cost paths. *IEEE Transactions on SSC*, 4:100–107, 1968.

[13] V. Julian and V. Botti. Developing real-time multi-agent systems. *ICAE Journal*, 11:135–149, 2004.

[14] N. Karacapilidis and D. Papadias. Computer supported argumentation and collaborative decision-making: the HERMES system. *Information Systems*, 26(4):259–277, 2001.

[15] B. S. L. McGinty. Collaborative case-based reasoning: Applications in personalised route planing. In *4th ICCBR*, pages 362–376, 2001.

[16] D. B. Leake and R. Sooriamurthi. When two case bases are better than one: Exploiting multiple case bases, case-based reasoning research and development. In *4th ICCBR*, 2001.

[17] M. Luck and P. McBurney. Computing as interaction: Agent and agreement technologies. In *IEEE SMC conf. on DHMS*, pages 1–6, 2008.

[18] M. Luck, P. McBurney, O. Shehory, and S. Willmott. *Agent Technology: Computing as Interaction.* AgentLink, 2005.

[19] A. Martens and A. Uhrmacher. Adaptative tutoring processes and mental plans. *Intelligent Tutoring Systems*, pages 71–80, 2002.

[20] M. Navarro, V. Julián, J. Soler, and V. Botti. jART: A Real-Time Multi-Agent Platform with RT-Java. In *3rd IWPAAMS*, pages 73–82, 2004.

[21] J. Palanca and A. García-Fornes. Uso de técnicas híbridas en el cálculo del WCET. In *VIII Jornadas de Tiempo Real*, pages 243–249, 2005.

[22] D. W. Patterson, M. Galushka, and N. Rooney. Characterisation of a Novel Indexing Technique for CBR. *Artificial Intelligence Rev.*, 23:359–393, 2005.

[23] R. Plaza, J. L. Arcos, and F. Martín. Cooperative case-based reasoning. *Distributed Artificial Intelligence meets Machine Learning*, LNAI 1221:180–201, 1997.

[24] M. V. N. Prasad, V. R. Lesser, and S. Lander. Retrieval and reasoning in distributed case bases. Technical report, UMass Computer Science Report 95-27. CIIR Technical Report IC-5, 1995.

[25] K. Prouskas and J. Pitt. Towards a real-time architecture for time-aware agents. In *AAMAS '02*, pages 92–93. ACM, 2002.

[26] L.-K. Soh and C. Tsatsoulis. Reflective negotiating agents for real-time multisensor target tracking. In *IJCAI*, pages 1121–1127, 2001.

[27] J. Soler, V. Julian, M. Rebollo, C. Carrascosa, and V. Botti. Towards a real-time multi-agent system architecture. In *Workshop: Challenges in Open Agent Systems*, pages 1–11. AgentCities, 2002.

[28] D. M. Surka, M. C. Brito, and C. G. Harvey. The real-time objectagent software architecture for distributed satellite systems. In *IEEE Proceedings*, volume 6, pages 2731–2741, 2001.

[29] T. Wagner and V. Lesser. Design-to-criteria scheduling: Real-time agent control. In *AAAI Spring Symposium on RT-Systems*, pages 89–96, 2000.

Human-Agent Collaborative Optimization of Real-Time Distributed Dynamic Multi-Agent Coordination [*]

Rajiv T. Maheswaran
Univ. of Southern California
Information Sciences Institute
4676 Admiralty Way #1001
Marina Del Rey, CA, USA
maheswar@isi.edu

Craig M. Rogers
Univ. of Southern California
Information Sciences Institute
4676 Admiralty Way #1001
Marina Del Rey, CA, USA
rogers@isi.edu

Romeo Sanchez
Univ. of Southern California
Information Sciences Institute
4676 Admiralty Way #1001
Marina Del Rey, CA, USA
rsanchez@isi.edu

Pedro Szekely
Univ. of Southern California
Information Sciences Institute
Marina Del Rey, CA, USA
pszekely@isi.edu

ABSTRACT

Creating decision support systems to help people coordinate in the real world is difficult because it requires simultaneously addressing planning, scheduling, uncertainty and distribution. Generic AI approaches produce inadequate solutions because they cannot leverage the structure of domains and the intuition that end-users have for solving particular problem instances. We present a general approach where end-users can encode their intuition as guidance enabling the system to decompose large distributed problems into simpler problems that can be solved by traditional centralized AI techniques. Evaluations in field exercises with real users show that teams assisted by our multi-agent decision-support system outperform teams coordinating using radios.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms

Algorithms, Performance, Design, Experimentation

Keywords

Real-Time Dynamic Planning and Scheduling, Human-Agent Interaction, Human Guidance, Decision Support, Multi-Agent, Uncertainty, Dynamism, Coordination

1. INTRODUCTION

Teams of people need to coordinate in real-time in many dynamic and uncertain domains. Examples include disaster rescue, hospital triage, and military operations. It is possible to develop plan *a priori*, but many parts of these plans must be left unspecified because people won't know exactly what needs to be done until they are executing the plan in the field. Additionally, requirements and tasks can evolve during execution. Our work addresses a fundamental multi-agent systems endeavor of creating decision support systems that help humans perform better in these domains. The technical challenges to compute good solutions for these problems have been well documented [10, 7, 3].

Established approaches address subsets of the problem, but none have adequately addressed the full problem. Classical planning techniques can barely compute the sets of actions that each person should perform for large problems involving metric resources and cannot cope at all with uncertainty and distribution. Decision-theoretic planning addresses uncertainty, but performance degrades with increased distribution and scale. Distributed constraint optimization techniques address distribution, but do not address temporal reasoning, uncertainty or scale. In practice, it is possible to address specific domains with custom algorithms that use powerful heuristics to leverage the structures unique to that domain. These solutions are expensive to create as even these domains involve planning, uncertainty and distribution. The goal remains to develop generic approaches that produce good solutions that help human teams in many domains.

We introduce a new approach, STaC, based on the premise that people have good intuitions about how to solve problems in each domain. The idea is to enable users to encode their intuition as guidance for the system and to use this guidance to vastly simplify the problems that the system needs to address. The approach is related to heuristic planning, but differs in two important aspects. First, the goal is to capture intuition about solving specific instances of the problem rather than providing heuristics that apply to many instances in the domain. End-users rather than domain experts or developers encode heuristics for the system. Second, in STaC, the intuition is not captured by rules of what actions to take in specific situations, but rather as a decomposition of the problem into simpler problems that can be solved independently.

The model is defined by software developers, and declares the capabilities agents possess and the capabilities that relevant actions in the domain require in order to be performed. These capabilities define the vocabulary for users to express the guidance that encodes their intuition about how to solve a particular problem in-

[*]The work presented here is funded by the DARPA COORDINATORS Program under contract FA8750-05-C-0032. The U.S.Government is authorized to reproduce and distribute reports for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any of the above organizations or any person connected with them. Approved for Public Release, Distribution Unlimited.

Figure 1: Field Exercise Images from Rome, NY

stance. The key to STaC is using the model and guidance to produce sufficiently smaller task structures that can be centralized so that a single agent can determine who does what, when and where with respect to these significantly simpler task structures. This mitigates the distribution challenge and enables using auxiliary solvers based on established AI techniques which produce good solutions at a smaller scale. These smaller task structures are solved independently assuming that the human guidance has addressed any significant dependencies. STaC addresses tracking the dynamism in these task structures, the transitioning of agents assignment between these smaller task structures and the invocation of auxiliary solvers. Given that the task structures are treated independently and sufficiently small to be centralized, we call them sandbox reasoners. The sandbox reasoners required in each domain are different, so custom code must be written for each domain. However, the benefit of the approach is that sandbox reasoners are significantly simpler than the custom solvers required to produce a custom solution for a domain.

The rest of the paper is organized as follows. The next sections introduces the real-world domain where our approach was tested followed by related work. We then describe the details of the STaC approach and the particular sandbox reasoners used in our example domain. We close with evaluation results, conclusions and directions for future work.

2. FIELD EXERCISES

The field exercises were based on a simulated disaster rescue domain. The first two exercises were held in the city of Rome, NY, and the second three were in Stanton Wood Park in Herndon, VA. Images of the field exercise in Rome, NY are shown in Figure 1 and a map of the sites and road network of Stanton Wood Park are shown in Figure 2. They were organized and evaluated by independent parties contracted by the DARPA Coordinators program. The rules of the field exercise were created collaboratively by the teams building coordinator agents, the independent evaluation team, and subject matter experts. The specific instances or *scenarios* that comprised the test problems were chosen by the independent evaluation team.

Various locations were selected as *sites* and a feasible road network was constructed. If the site was *populated*, it could have injured people in either *critical* and *serious* condition. Populated sites would also have gas, power and water substations which may have been damaged. In addition, any site could have *facilities* such as a *hospital*, *clinic*, *warehouse*, *gas main station*, *power main station* and *water main station*. A team would obtain points by rescuing injured to hospitals or operational clinics (before a deadline associated with each injured person) and by repairing main stations and substations. The goal of a scenario was to accumulate as many points as possible before the scenario deadline.

The teams were composed of 8 field agents and 2 command agents. Each agent had a different set of skills. Three *specialists* in *gas*, *power* and *water* could perform *major* and *minor* repairs in their respective skill area. The *medical specialist* could load any type of injured person by themselves. The remaining four *survey specialists* could have any collection of skills involving minor repairs. The field agents could move throughout the field exercise area and perform actions. The command agents were located at a base where they helped to coordinate the activities of the team. The *Radio Team* communicated only with radios. Our *CSC Team* had ruggedized tablet computers on which our agents were loaded, in addition to radios. The tablets had cell modems and GPS.

Many outcomes were revealed during the game for which little or no likelihood information was given *a priori*, i.e., no probability distribution functions over outcomes. Teams did know the space of possible outcomes beforehand. A *survey for damage* at a main station or substation revealed the number and type of problems chosen from a set of known possible problems. A *survey for injured* at a populated site revealed the number, types and deadlines for the injured at that site. As the result of a survey, any team member might be injured, forcing them to go to an operational medical facility to recover before proceeding with any other action. A survey could also reveal that the vehicle of the agent doing the survey had failed and would require a vehicle repair before the agent could travel to any other site. While traveling, agents could encounter*road blocks* which could not be passed until fixed. Travel and repair times could vary and repairs could fail. These dynamic and uncertain events were planned parts of the exercise. In addition, the teams had to address uncertainties inherent in the environment, such as noisy radios, weather, and other activities in the public settings. Furthermore, most of these outcomes were only observable by the agent encountering the outcome.

The independent evaluation team chose the scenario from the space of possible exercises and informed the teams of the details below one day prior to the test: (1) the locations of populated sites and facilities, (2) the road network and ranges on travel times between sites, (3) a range for the total number of injured at each site, (4) the points for rescuing each type of injured, which could vary by type and site, (5) the points for repairing each substation or main station, which could vary by type and site, (6) potential problems after surveys for damage and corresponding repair options, (7) ranges on repair times, (8) likelihoods of failure for every repair activity, and (9) the skills of the survey specialist agents. The deadlines (for the scenario and injured) did not allow teams to do all possible repairs and rescues. The teams had one day to form a high-level strategy. The only element of uncertainty which could be modeled accurately with a probability density function was (8). When a team member completed a repair activity, they would call the evaluation team, which would report whether the repair was successful or a failure. The range in (3) was respected by the scenario designers, i.e., the number of injured did not fall outside the given range.

There were many rules and couplings that forced agents to coordinate. To do surveys, gas and power substations at the site had to be off, which required agents with those skills. Two agents had to

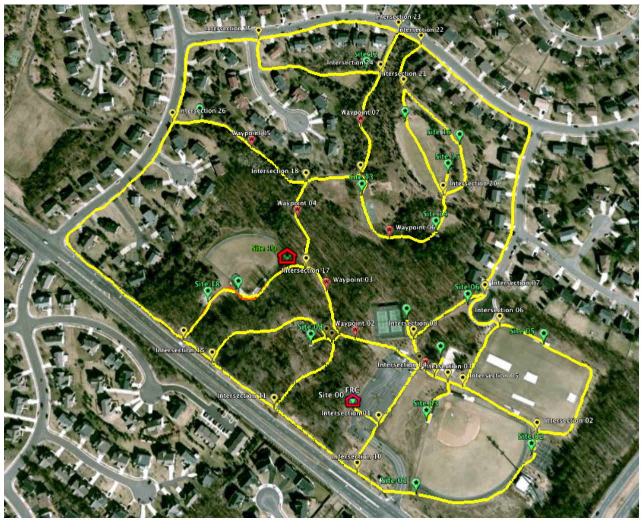


Figure 2: Stanton Woods Park, Herndon, VA

be at the same location simultaneously to load a critically injured person or repair a road block. Repair options could involve multiple tasks and require two agents with certain skills to act in synchrony or in a particular sequence. Some repair options required kits which guaranteed their success, but kits were available only at warehouses. Agents could transport at most one entity, i.e, either a repair kit or a single casualty. A substation was considered repaired only if the corresponding main station was also repaired. A clinic was not operational until all substations at the site and all corresponding main stations were repaired. These are examples of rules that, along with the dynamism and uncertainty in outcomes mentioned earlier, created challenging real-time real-world distributed coordination problems.

The goal was to see if humans operating with radios and a multi-agent decision-support system could outperform humans operating with only radios. Although the field exercises still abstracted some aspects of a real-world disaster scenario, we believe they closely approximated the challenges of helping a human team solve difficult real-world problems.

3. RELATED WORK

The STaC framework was developed during the DARPA Coordinators program. In the first two years, DARPA ran competitive evaluations on simulated scenarios, and CSC, the underlying system behind the STaC framework, won such evaluations by considerable margins against two competing approaches: an MDP-based approach [11] and an STN framework [14].

The MDP-based [11] approach addressed the infeasibility of reasoning over the joint state space by setting the circumstance set to a subset of local state space that is reachable from the current local state, unrolling the state space by doing a greedy estimation of boundary values. It biased its local reward function on the commitments made by the agents during execution. However, such approximations lose critical information, exploring state spaces that are far from good distributed solutions.

The STN framework [14] addressed temporal uncertainty by using a time interval (instead of a point) as the circumstance that denoted feasible start times for a method to be executed. The system used *constraint propagation* to update the start intervals of the agents' activities during execution. A policy modification phase was triggered if execution was forced outside the given set of intervals. One of the problems of this approach is that agents tried to maintain consistency and optimize their local schedules, losing information that was needed to timely trigger policy modifications for their schedules.

We encoded scenarios of the field exercise as planning problems using PDDL [5]. The motivation was to identify to the extent to which current automated planning technology can address complex distributed, resource-driven, and uncertain domains. Unfortunately, this proved to be extremely difficult for state-of-the-art planning systems. From the set of planning systems tried, only LPG-TD [6], and SGPLAN [4] solved a few simplified problems, after uncertainty, dynamism, non-determinism, resource-metrics, partial observability and deadlines were removed. Planners were unable to scale to more than 5 sites. LPG-TD produced solutions more efficiently but less optimally.

In general, mixed-initiative approaches where humans and software collaborate can often produce better solutions for complex problems. Mixed-initiative planning systems have been developed where users and software interact to construct plans. Users manipulate plan activities by removing or adding them during execution while minimizing the changes from a reference schedule [1, 8, 12]. However, most of these systems are centralized, so humans and systems are fully aware of the entire plan, and of the consequences of updating it. In our scenario, agents (including humans) have subjective views of the world, and any decision may trigger many unknown global effects.

Multi-agent systems for disaster domains have been studied in the context of adjustable autonomy. The idea is to improve limited human situational awareness that reduces human effectiveness in directing agent teams by providing the flexibility to allow for multiple strategies to be applied. A software prototype, DEFACTO, was presented and tested on a simulated environment under some simplifications (e.g., no bandwidth limitations, reliable communications, omnipresence) [13]. Our work also recognizes the importance of flexible frameworks to allow better human-agent interactions. The test-bed presented in this paper does not make any major simplifications, being a first step toward creating multi-agent systems for real-world problems.

4. THE STaC APPROACH

Our goal is to create a general framework for incorporating human strategic guidance. We introduce the formalism for STaC guidance and give an example from our domain. We then describe how this guidance is executed with the use of Total Capability Requirement (TCR) sets. We provide an example of a TCR set and discuss how dynamic updates enable execution of the guidance.

4.1 STaC Guidance

We make the following assumptions about a general multi-agent coordination problem. There are a set of agents N and a set of actions A. Agents have capabilities from a set of capabilities: $\Theta_n \in \Theta$. Each action is mapped to a capability, i.e., $\gamma : A \rightarrow \Theta$. An agent can perform any action for which it has the capability.

The problem is composed of a collection of tasks T. Each task $t \in T$ is associated with a set of potential actions involved in completing it: $A^t \subset A$. It is not necessary that $\{A^t\}$ be disjoint. Furthermore, for the purposes of guidance, it is not relevant how these tasks relate to the actual reward function. It is only important that the notion of tasks exists.

We can define a generic representation for human strategic guidance as follows. Guidance is an ordered set of *guidance groups*: $G = \{G_i\}$. Each guidance group G_i is associated with a *subteam* of agents $S_i \subset N$ and an ordered set of guidance elements E_i. Each *guidance element* $e_i^j \in E_i$ is composed of a task $t_i^j \in T_i$, a set of *constraints* C_i^j, and a temporal bound b_i^j. The constraints C_i^j are a collection of capability-number pairs $\{(\theta, n_\theta)\}$ where $\theta \in \Theta$ and $n_\theta \in \mathbb{Z}^*$ is a non-negative integer. The pair (θ, n_θ) indicates that each agent in the subteam can use the capability θ at most n_θ times for the task in the guidance element. The temporal bound $b_i^j \in \{0\} \cup \{<, >\} \times \mathbb{R}^+$ is another constraint that can indicate that the guidance element is only valid if the time remaining is greater or less than some number ($b_i^j = 0$ indicates no temporal constraint). Thus,

$$\begin{aligned} G = \{G_i\} = \{(S_i, E_i)\} &= \{(S_i, \{(t_i^j, C_i^j, b_i^j)\})\} \\ &= \{(S_i, \{(t_i^j, \{(\theta_i^{j,k}, n_{\theta_i^{j,k}})\}, b_i^j)\})\}. \end{aligned}$$

We refer to this as the STaC (Subteam-Task-Constraints) formalism for strategic guidance. One can now define a strategy composed of a sequence of subteams, each responsible for a collection of tasks, each of which are to be performed under some constraints. We note that since agents will traverse the elements of this guidance in order, STaCs are actually queues.

4.1.1 Field Exercise Example

We first defined a set of capabilities that were relevant to the field exercise. We also associated each capability with several capability classes for more compact expression of constraints. Below are the set of capabilities and associated classes for actions involving gas and injured, respectively. Capabilities and classes for power and water are analogous to those for gas.

```
gas_major: gas, gas_main
gas_minor: gas, gas_main
survey_gas_main: gas, gas_main, survey
survey_gas_sub: gas, survey
turn_off_gas_main: gas, gas_main, turnoffs
turn_off_gas_sub: gas, turnoffs
pickup_gas_kit: gas, pickup
dropoff_gas_kit: gas, dropoff
```

```
load_critical: critical, injured
assist_load_critical: critical, injured
survey_injured: injured, survey
generic: injured
```

Consider the STaC guidance fragment below. We see an ordered set of guidance groups, each with a subteam of agents and an ordered set of guidance elements. The `only`() operator sets the capability-number pairs for all capabilities not in the argument to zero. The `no`() operator sets the capability-number pairs for all capabilities in the argument to zero. The intent of this plan fragment is for the survey specialist to turn off services at the substations at Site 4 and Site 3, enabling other agents to work there. The gas and survey specialists go to the warehouse at Site 6, pick up gas kits, then restore the gas main station and gas substation at Site 1. The medical specialist and survey specialist are responsible for making sure they each rescue two critically injured people before rescuing all others. The gas and power specialist are then responsible for doing everything except water-related actions at Site 4, but if less than 10 minutes are left in the scenario, they switch to rescuing injured.

```
survey_specialist_1:
  ( task_site_04, [ only( turnoffs ) ], 0);
  ( task_site_03, [ only( turnoffs ) ], 0);
gas_specialist, survey_specialist_1:
  ( task_site_06, [ only( pickup_gas_kit ) ], 0),
  ( task_site_01, [ only( gas ) ], 0);
survey_specialist_1, medical_specialist:
  ( task_site_04, [ (load_critical, 2) ],0),
  ( task_site_04, [ only( injured ) ],0);
gas_specialist, power_specialist:
  ( task_site_03, [ no(water) ], >10),
  ( task_site_03, [ only( injured ) ], 0);
```

Here, the tasks chosen for each guidance element are all those associated with a particular site. This is not a requirement in the guidance formalism. For example, the second guidance group could have also been:

```
gas_specialist, survey_specialist_1:
  ( task_site_06, [ only( pickup_gas_kit ) ], 0),
  ( task_gas_main, [ ], 0),
  ( task_gas_substation_site_01, [ ], 0);
```

This would have specified a fixed ordering between repairing the main station and the substation which did not exist in the original. The expression of guidance is not necessarily unique and can be tailored to the intuition and structure that the designer finds most appropriate.

4.2 STaC Execution

While STaC guidance is compact and has intuitive meaning for a human, the agents have no semantic awareness of what it signifies beyond identifying tasks and limiting actions. This is due to the generality of the formalism. Furthermore, the guidance does not specify which actions to perform, which agents should perform them, the timing of these actions or how to react to any dynamism and uncertainty during execution. We address those challenges here.

4.2.1 Total Capability Requirement (TCR) Sets

Given the STaC formalism, one of the key decisions that every agent must make is when to transition from one task to another. A simple solution is to wait until the current guidance element is completed and then move to the next guidance element (which may involve going to the next relevant guidance group). This approach would lead to significant waste if the agent were unable to contribute to the current guidance element.

Consider the example shown in Section 4.1.1. If the gas specialist arrives at Site 1 first and discovers that all the repair options for the gas main station and gas substation can be completed by the gas specialist alone, or that there exists repair options for both the main station and the substation that can be performed by the gas specialist alone and are guaranteed to succeed, the gas capabilities of the survey specialist are not needed. It may make sense for the survey specialist to skip Site 1 and head to Site 4 to help the medical specialist rescue injured, even though the repairs at Site 1 have not been completed. It is important to determine dynamically whether the capabilities of each agent in the subteam are needed for the task being executed in the guidance element.

Total Capability Requirement (TCR) sets are a mechanism to achieve this. For every task $t \in T$, there is an associated TCR set $R^t = \{R_i^t\}$, which is a set of *requirements*. Each requirement $R_i^t = (n_i^t, Q_i^t)$ is a tuple of a *requirement number* n_i^t and *requirement type* Q_i^t. A requirement type $Q_i^t = \{q_{i,j}^t\}$ is a collection of *requirement elements*, where each requirement element is a tuple $q_{i,j}^t = (c_{i,j}^t, l_{i,j}^t, n_{i,j}^t)$ where $c_{i,j}^t$ is a capability, $l_{i,j}^t$ is a location, and $n_{i,j}^t$ is an element number. Thus, $R^t = \{R_i^t\} = \{(n_i^t, Q_i^t)\} = \{(n_i^t, \{q_{i,j}^t\})\} = \{(n_i^t, \{(c_{i,j}^t, l_{i,j}^t, n_{i,j}^t)\})\}$.

```
2:[(gas_minor, site_01, 1)]
1:[(gas_minor, site_01, 2)]
4:[(assist_load_critical, site_01, 2)]
1:[(power_minor, site_01, 1) (power_minor, site_03, 1)]
```

Consider the example above which is a possible TCR set for `task_site_01`. This indicates that there are two instances of the need for a single agent with gas minor capability, one instance of a need for two agents with gas minor capability, four instances of a need for two agents capable of loading a critically injured person and one instance of a need for having an agent with power minor capability at Site 1 at the same time that there is an agent with power minor capability at Site 3. The first requirement could occur because the gas main station has two problems, each of which could be solved with a gas minor repair. The second requirement could occur because the gas substation has one problem that requires two agents with gas minor skills to perform a synchronized repair. The third requirement could be due to the discovery of four critically injured people. The fourth requirement represents the need for *remote synchronization*: the need for two agents at two different locations at the same time. In the field exercise, some power substations required an agent at the substation and another at the main station simultaneously to turn the power substation on.

If the guidance element was:
`( task_site_01, [ only( gas ) ], 0 )`
then only the first two requirements involving the gas minor capability would be considered when deciding whether an agent should remain committed or released from the task. The TCR sets are dynamically updated such that once a skill is no longer needed, as repairs are completed or injured are loaded, the appropriate requirements are decremented or deleted.

4.2.2 *Calculating TCR Sets*

Our calculation of TCR sets can best be described in the context of our modeling specification for the field exercise scenarios. We used a hierarchical task network structure that was an extension of CTAEMS [2], which is itself a variant of TAEMS [9] developed for the DARPA Coordinators Phase 2 evaluation. The essential property was that tasks (including the root task which represented the overall reward function) were composed of subtasks iteratively until reaching a primitive task which was composed of actions. Tasks could also have non-hereditary relationships such as enablement and disablement. Every task was also associated with state aggregation functions that determined how the state of its subtasks (or child actions) affected the state of the task. An example of a template used to model power substations is shown in Figure 3. This

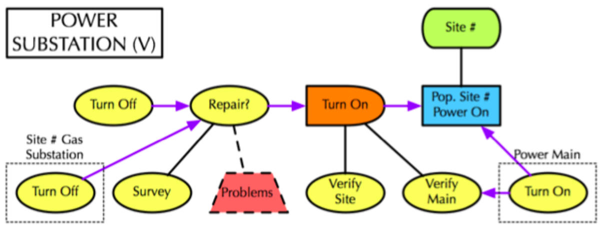


Figure 3: Model Template for Power Substation

also illustrates the issue of dynamism as the task node for *Problems* must remain without children until the power substation is surveyed for damage. Then, the appropriate problems and repair options are added dynamically to the model. It would be cumbersome and practically infeasible to express every possible combination of problems and repair options that could occur. The issues are similar when it comes to modeling the discovery of injured people.

The TCR set for a given task is calculated by applying a TCR aggregation function, chosen based on the state aggregation function associated with the task, to the TCR sets of its subtasks and enabling tasks. For example, a *sum* state aggregation function would correspond to a *union* TCR aggregation function, and a *sync* state aggregation function would correspond to a *cross-product* TCR aggregation function. Thus, TCR sets would start from actions, which are each associated capability and flow forward and up through enablement and ancestral links to form TCR sets for every task. These sets can be dynamically updated as tasks change states.

For example, once a task is completed, the TCR set can be set to null indicating that it does not require any more capabilities. This makes the TCR sets vanish as tasks are completed, allowing agents to be released as soon as possible. In order to address the dynamic nature of the model, tasks that might be expanded during execution must be marked with TCR sets that indicate reasonable upper bounds on needed capabilities. These sets are then changed to the actual TCR sets once outcomes has been observed in the environment. Having an HTN-based model helps to construct and manage TCR sets, but is not necessary. As long as there exists a non-cyclic mapping that describes the relationships of tasks to other tasks and actions, a dynamic methodology to assign TCR sets to tasks can be constructed.

4.3 Partial Centralization

STaC execution can be implemented such that a single agent is responsible for choosing all actions involved with a single task-constraint tuple of a guidance element. We create a mapping, $\omega : T \rightarrow N$, where every task has an owner. The task owner contacts agents who are responsible for related tasks and actions to subscribe to relevant state updates. When an agent reaches a particular task-constraint tuple, it cedes autonomy to the owner of that task until the task owner releases the agent from that commitment. The owner agent keeps track of the set of capabilities of all agents bound to that task as well as the TCR set of that tasks and repeatedly solves an optimization problem to find the best set of agents to keep for the current TCR set. If the solution is a set that is a strict subset of the bound agents, it can release the other agents. Our optimization problem minimized a weighted combination of the number of agents kept and their capabilities. The key insight here is that partial centralization of autonomy always occurs implicitly and thus, it is beneficial to align the metric for partial centralization with the properties of the domain where it matters.

5. SANDBOX REASONING

Once the task owner has chosen which set of agents to keep, it must then also decide, subject to the constraints in the guidance, which actions to perform and which agents should perform to accomplish the task. We call this process *sandbox reasoning* because the task owner's deliberation over what to do for a single task-constraint tuple is isolated from all actions and tasks that are not related to the task at hand. The task owner does not need to consider impact on the future or on concurrently executing tasks. It is given a collection of agents and autonomy to use them however it sees fit to accomplish the task as well as possible. The consequences of the interactions have, in principle, been considered and addressed by human strategic guidance.

In creating the agent model for a field exercise scenario, we instantiated structure (sometimes, dynamically during execution) from a small set of templates. Examples include *power substation restoration* (as shown in Figure 3) or *critically injured rescue*. Similarly, the tasks in the guidance were also be a subset of tasks that make intuitive sense to the strategy designer. In our case, the task types involved in guidance were far fewer than the templates involved in model generation. We believe that the notion of using templates for generation and more significantly for guidance is a general principle that is applicable to many domains. This belief was also reflected in the DARPA Coordinators program and its Phase 2 evaluation. The main templates needed for guidance were a repair and rescue manager. We discuss the automated reasoners in these managers below.

5.1 Repair Manager

The repair manager would take as input (1) a collection of facilities that had been damaged, (2) a set of problems for each facility, (3) a set of repair options for each problem, (4) set of agents with (5) associated capabilities and (6) times that they would be available to be scheduled for activities, The output would be a policy that yielded a collection of agent-action tuples given a simplified version of the state. While this may seem field-exercise specific, this reasoner had no semantic knowledge of the field exercise.

The problem was generalized as follows: Given a set of tasks $\{T_i\}$, where each task is a conjunction of a set of problems, $T_i = \min(\{P_j\})$, each problem is a disjunction of repairs, $P_j = \max(\{R_k\})$, and each repair is a function of actions, $R_k = \odot(\{a_l\})$ where $\odot \in \{sync, sequence, min\}$ is a collection of operators that require the elements to have synchronized start times, sequential execution, or conjunctive success in order for the repair to succeed, respectively. Each action is associated with a capability, expected duration, and probability of failure. We also have a set of agents where each have an associated set of capabilities and an availability time. This is a straightforward optimization given an appropriate objective function.

We needed a fast solution (less than five seconds) because users needed guidance from the solver after performing a survey. Our solution to was to build a policy based on a simplified representation of state. The state vector is indexed by all possible actions and takes values from the set $\{NotStarted, Succeeded, Failed\}$. The policy output given for a state is a set of agent-action pairs.

The policy is constructed by running simulation traces of the optimization problem. At every time step in the simulation, if there are idle agents and no action-agent tuples for the current state in the policy, the agents are randomly assigned to an action they can perform and marked busy for the expected duration. These assignments are then associated with the state of the simulation where executing actions are interpreted to have succeeded. The outcomes of the actions in the simulation are determined using the given probability of failure. Multiple simulation runs are used to build a single policy which receives a score based on the average idle time of all agents. Multiple policies are generated using the same mechanism and the policy with the best score is stored. To execute the policy, the task owner maps the current state of the actions to the policy state by mapping executing actions to *Succeeded* and schedule the associated action-agent tuples as the next action for the agent. If there is no agent-action tuple for the translated state or if any of the input parameters (e.g. availability times, new tasks) change, policy generation restarts from scratch. While this is a simple stochastic sampling approach, it produces reasonable policies with very limited computation requirements. Policy generation was typically bounded to five seconds. Also, while the potential state space is exponential in the number of actions, typical policies had at most about 200 states.

The key idea is that we could create a sandbox reasoner that can solve a generic context-independent problem which could be applied to many tasks in the guidance. One could, in theory, use an MDP or STN-based approach if it yielded a solution within the limits of bounded rationality in the domain at hand.

5.2 Rescue Manager

The rescue manager would similarly take as input a list of agents and a set of injured with associated types and deadlines, and output a set of agent-action pairs when agents became idle. We used a simple reactive planner with a handful of simple heuristics to determine when to wait for an agent to help load a critically injured (which requires two agents to be present simultaneously) and when to take a serious (which could be done by one agent). This also can be formulated as a generic problem consisting of a set of tasks with associated deadlines and durations and the tasks can be either a singleton action or require synchronized actions by two different agents. The rules were variations of: "If an agent is on the way and it will arrive later than the duration of the singleton action, perform the singleton action and return to the site, otherwise, wait for the agent." The variations were due to the constraints placed on the rescue task and the number of agents available to do the rescues.

The general philosophy of the STaC approach to guidance, its execution and sandbox reasoning is to create a generic framework for human strategic input to decompose a very difficult problem into smaller problems that can be solved in isolation with automated tools created to solve large classes of task structures that appear in the guidance. Our system was completely unaware of any semantics of the field exercise, and a similar approach could be used in a completely different domain.

6. EVALUATION

Figure 4 shows the scores for the three scenarios run in Herndon. For each scenario, the top, lighter bar shows the radio-team score, and the bottom, darker bar shows the score of the team using the CSC system described in this paper. The middle bar shows the results of a simulation of the scenario using a baseline version of the system with simple sandbox reasoners. In order to calibrate the simulation scores, we also ran a simulation of the complete

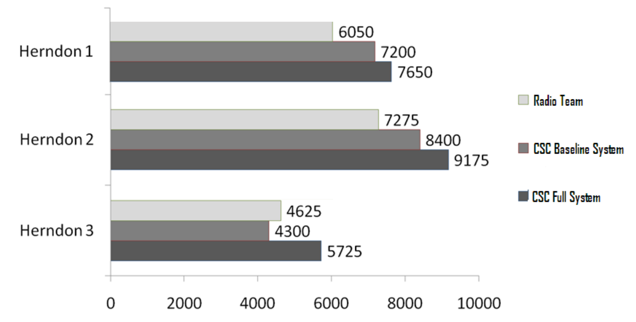


Figure 4: Herndon Evaluation

version of the system using the same dice rolls used in the physical scenario. The simulation results for the full system were within 200 points of the results obtained in the field, which suggests that if the baseline system had been used in the field, the results would also be close to those shown in the figure.

In the baseline version, the repair manager uses a random strategy where agents randomly select a repair task to perform from the set of tasks that the agent is eligible to perform according to its capabilities. The baseline rescue manager uses a greedy strategy where agents select the injured with the earliest deadline that lives long enough to arrive to the medical facility before the deadline. In the baseline rescue manager, agents don't wait for a partner that enables loading a critically injured that they would not be eligible to load otherwise.

The results show that more sophisticated sandbox reasoners always resulted in better scores: 5.8%, 8.4% and 24.8% improvements. The differences in scenarios 1 and 2 were small, and in those scenarios the baseline system also outperformed the radio team. In scenario 3, the difference is more significant. This scenario emphasized injured, and the greedy strategy used in the simple version of the system delayed rescuing the critically injured. Agents rescue seriously injured with later deadlines instead of waiting for a partner to rescue a more valuable critically injured with an earlier deadline.

Figure 5 shows simulation results that compare the effects of alternative strategies. We organized these strategies along two dimensions: the number of clinics that would be made operational (0, 1 and 2), and the number of independent subteams (1, 2 and 4). In the strategies with 1 clinic, the team repaired the clinic that was considered most useful (closest to the most valuable injured). In the scenarios with 2 and 4 teams, we specified the teams so that they could perform repairs independently. In the strategies with 0 clinics, the teams performed no repairs and rescued injured to a medical facility that was always operational and required no repairs. In the strategies with 1 and 2 clinics, the agents first repair the main stations, then the clinics and then visit the remaining sites to rescue all injured and perform all repairs according to the following algorithm. First, the sites are ordered according to the total expected number of points achievable at the site. The teams take turns picking the next most valuable site from the ordered list until the list is exhausted. The idea is to complete the most valuable sites first so that when time runs out the most valuable sites have been completed.

Figure 5 shows that in the Herndon 1 and 2 scenarios, the strategies that repair 1 or 2 clinics are competitive with the radio team, outscoring them in 11 out of the 12 strategies involved. However, in all three scenarios, the CSC strategy used in the field was significantly better than all the alternative strategies. The difference is due mainly to the use of constraints. In the alternative strategies, the agents performed all tasks at a site, whereas the strategies used in the field used constraints to prevent agents from performing tasks that we deemed not worthwhile. In addition, we used constraints on the number of injured rescued to prevent agents from rescuing all injured at a site before moving to the next site. Instead, we used longer itineraries that visited sites multiple times in a round-robin, so agents would rescue the most urgent injured first.

7. CONCLUSIONS AND FUTURE WORK

Our 18-month experience working on a system to compete against radio teams in the field exercises provided significant evidence for the benefits of our approach. Our starting point was our generic CSC system developed during the previous two years to solve generic, synthetically generated problem instances specified in CTAEMS. Even though the synthetically generated problem instances were generated according to templates that combined "typical" coordination situations, the resulting problems were not understandable by humans. In contrast, the field exercise problems are natural, and appeal to our lifetime of experience coordinating every day activities. Intuitions about space, distance, time, importance and risk all came into play, enabling teams of humans to devise a sophisticated strategy within one hour of brainstorming. It became obvious early on that the generic CSC system would not be able to produce solutions comparable to the desired sophisticated, coordinated behavior of human-produced strategies.

Our existing system had performed extremely well in Phase 2 by using our Predictability and Criticality Metrics (PCM) approach. In the PCM approach, the policy modifications that agents consider are limited to those that can be evaluated accurately through criticality metrics that capture global information. These policy modifications were simple and thus the reasoners that implemented them were simple too.

For the field exercises, we extended our approach so that policy modifications would be constrained using the guidance provided by the users. This guidance was in the form of a sequence of sites to visit. The system was left to make decisions that we believed it could evaluate accurately (e.g., how to perform repairs or rescue injured at a single site). The system relied on the TCR set criticality metric to determine how to move agents along the list of guidance elements. The approach worked well. Our users outperformed the radio team because they were able to communicate their strategy to their agents, and the system optimized the execution of the strategy, adapting it to the dynamics of the environment.

The field exercises in Rome, NY used a simpler language for specifying guidance. It had a single guidance group consisting of the entire set of agents. Also, it did not support constraints to control the capabilities within a guidance element. In that evaluation, our system remained competitive with the radio team, but lost in two out of the three scenarios. The final language for guidance was inspired by our observations of the radio-team strategies, extensive discussions with subject matter experts and extensive numbers of simulations. We noted that while the human team could not execute a strategy as well as we could, the space of strategies that they were able to engage were far more sophisticated than ours. This led

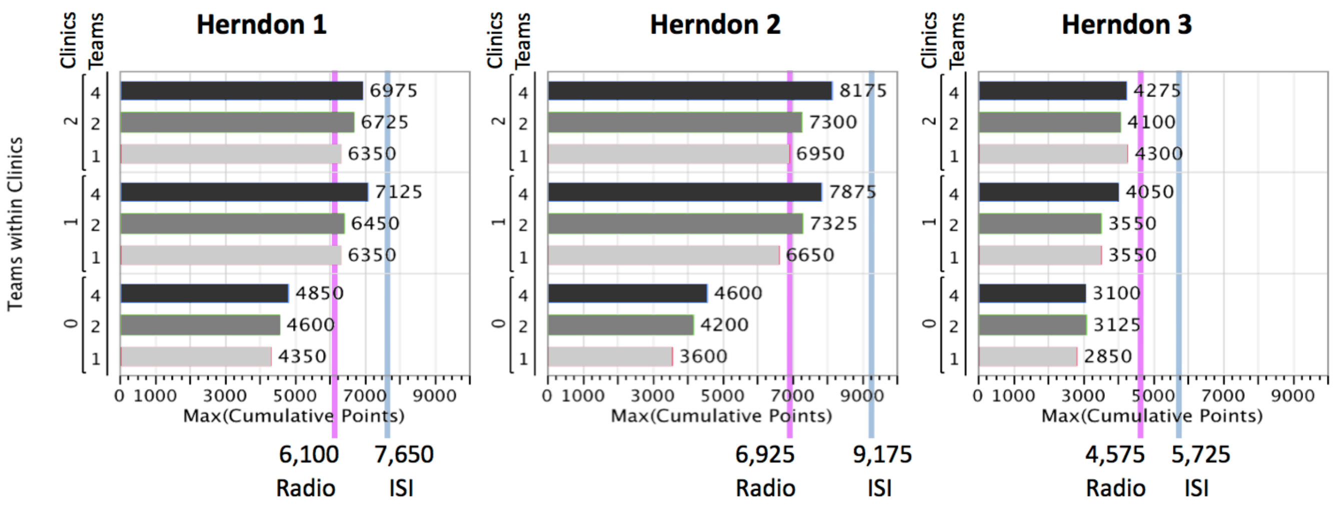


Figure 5: Baselines

to the creation of a the more sophisticated formalism for capturing human strategic guidance.

We have taken the first step towards generic coordination technology that end-users can tailor to specific problem instances. The approach was validated in one domain thanks to the extensive and expensive evaluations carried out by the DARPA Coordinators program. In the future, we hope to be able to apply this approach to other application domains. One key area that needs to be investigated is extensions to allow human users to make guidance adjustments *during* execution. There are situations where a series of outcomes either invalidates an assumption when creating the *a priori* guidance or creates an opportunity to improve on that guidance. Addressing this requires the ability for human users to quickly and easily understand and modify the guidance while it is being executed. Even more advanced steps would be evaluating and ultimately generating appropriate online guidance modifications.

8. REFERENCES

[1] M. Ai-Chang, J. Bresina, L. Charest, A. Chase, J. C. jung Hsu, A. Jonsson, B. Kanefsky, P. Morris, K. Rajan, J. Yglesias, B. G. Chafin, W. C. Dias, and P. F. Maldague. Mapgen: Mixed-initiative planning and scheduling for the mars exploration rover mission. *IEEE Intelligent Systems*, 19(1):8–12, 2004.

[2] M. Boddy, B. Horling, J. Phelps, R. P. Goldman, R. Vincent, A. C. Long, B. Kohout, and R. Maheswaran. CTAEMS language specification: Version 2.04, 2007.

[3] C. Boutilier. Multiagent systems: Challenges and opportunities for decision-theoretic planning. *AI Magazine*, 20(4):35–43, 1999.

[4] Y. Chen, B. Wah, and C.-W. Hsu. Temporal planning using subgoal partitioning and resolution in sgplan. *Journal of Artificial Intelligence Research (JAIR)*, 26(1):323–369, 2006.

[5] M. Fox and D. Long. Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research (JAIR)*, 27, 2006.

[6] A. Gerevini, A. Saetti, I. Serina, and P. Toninelli. Fast planning in domains with derived predicates: an approach based on rule-action graphs and local search. In *Proceedings of the 20th national conference on Artificial intelligence (AAAI)*, pages 1157–1162. AAAI Press, 2005.

[7] F. C. Groen, M. T. Spaan, J. R. Kok, and G. Pavlin. *Real World Multi-agent Systems: Information Sharing, Coordination and Planning*, volume 4363/2007 of *Lecture Notes in Computer Science. Logic, Language, and Computation*. Springer-Verlag, 2007.

[8] C. Hayes, A. Larson, and U. Ravinder. Weasel: A mipas system to assist in military planning. In *ICAPS05 MIPAS Workshop (WS3)*, 2005.

[9] V. Lesser, K. Decker, T. Wagner, N. Carver, A. Garvey, B. Horling, D. Neiman, R. Podorozhny, M. N. Prasad, A. Raja, R. Vincent, P. Xuan, and X. Q. Zhang. Evolution of the GPGP/TAEMS domain-independent coordination framework. *Autonomous Agents and Multi-Agent Systems*, 9(1-2):87–143, 2004.

[10] R. R. Murphy. Human-robot interaction in rescue robotics. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 34(2):138–153, 2004.

[11] D. J. Musliner, E. H. Durfee, J. Wu, D. A. Dolgov, R. P. Goldman, and M. S. Boddy. Coordinated plan management using multiagent MDPs. In *Proceedings of the 2006 AAAI Spring Symposium on Distributed Plan and Schedule Management*, March 2006.

[12] K. L. Myers, P. A. Jarvis, W. Mabry, T. Michael, and J. Wolverton. A mixed-initiative framework for robust plan sketching. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS)*, 2003.

[13] N. Schurr, J. Marecki, P. Scerri, J. Lewis, and M. Tambe. *The DEFACTO System: Coordinating human-agent teams for the future of disaster response*, chapter Programming multi-agent systems: Third international workshop. Springer Press, 2005.

[14] S. Smith, A. T. Gallagher, T. L. Zimmerman, L. Barbulescu, and Z. Rubinstein. Distributed management of flexible times schedules. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS 2007)*, Honolulu, HI, May 2007.

Market-based Risk Allocation Optimization [*]

Masahiro Ono and Brian C. Williams
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
{hiro_ono, williams}@mit.edu

ABSTRACT

This paper proposes *Market-based Iterative Risk Allocation* (*MIRA*), a new market-based decentralized optimization algorithm for multi-agent systems under stochastic uncertainty, with a focus on problems with continuous action and state space. In large coordination problems, from power grid management to multi-vehicle missions, multiple agents act collectively in order to maximize the performance of the system, while satisfying mission constraints. These optimal action plans are particularly susceptible to risk when uncertainty is introduced. We present a decentralized optimization algorithm that minimizes the system cost while ensuring that the probability of violating mission constraints is below a user-specified upper bound.

We build upon the paradigm of *risk allocation* [13], in which the planner optimizes not only the sequence of actions, but also its allocation of risk among state constraints. We extend the concept of risk allocation to multi-agent systems by highlighting risk as a resource that is traded in a computational market. The equilibrium price of risk that balances the supply and demand is found by an iterative price adjustment process called *tâtonnement* (also known as *Walrasian auction*). Our work is distinct from the classical tâtonnement approach in that we use Brent's method to provide fast guaranteed convergence to the equilibrium price. The simulation results demonstrate the efficiency and optimality of the proposed decentralized optimization algorithm.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Distributed Artificial Intelligence

General Terms

Algorithms

Keywords

Chance constrained optimal planning, Decentralized optimization, Tâtonnement, Walrasian auction, Continuous resource allocation

[*]This research is funded by The Boeing Company grant MIT-BA-GTA-1

1. INTRODUCTION

1.1 Motivation

There is an increasing need for multi-agent systems that perform optimal planning under uncertainty. An example is planning and control of power grid systems [3]. A power grid consists of a number of generators and electric transformers whose control should be carefully planned in order to maximize efficiency. A significant issue in power grid planning is the uncertainty in demand for energy by consumers. As the use of renewable energy, such as solar and wind power, become more popular, uncertainty in supply increases due to weather conditions.

Another example is the Autonomous Ocean Sampling Network (AOSN) [18], which consists of multiple automated underwater vehicles (AUVs), robotic buoys, and aerial vehicles. AOSN should maximize science gain while being exposed to external disturbances, such as tides and currents.

To deal with such problems, we developed *Market-based Iterative Risk Allocation* (MIRA), a multi-agent optimization algorithm that operates within user-specified risk bounds. The scope of this paper is a dynamic system with continuous state and action space under stochastic uncertainty.

1.2 Overview

Optimization of action sequence under uncertainty, and risk allocation.

When planning action sequence under uncertainty, there is always a risk of failure that should be avoided. However, in many cases, performance can be improved only by taking extra risk. We can reach a destination faster by driving at a faster speed and accepting a higher risk of an accident. Hannibal, a Carthaginian military commander in the third century B.C., was able to frustrate the Roman army by taking the great risk of crossing the Alps with 50,000 troops. As seen in these examples, risk and performance are in a trade-off relationship. In other words, risk is a resource that can be spent to improve the performance of the system.

Without taking any risk, nothing can be done; however, no one dares to take unlimited risk. Although the sensitivity for risk varies from person to person, everyone somehow balances risk and performance to find the optimal action sequence.

There are three main ways to formulate the trade-off problem of risk and performance; the first is to set a negative utility for failure (i.e. penalty), and maximize the expected total utility (the utilitarian approach, such as MDP [2][11][12]); the second is to set upper bound on risk and maximize performance within this bound [13][16]; the third is to set lower bound on performance and minimize risk. It is up to the user to choose which formulation to use

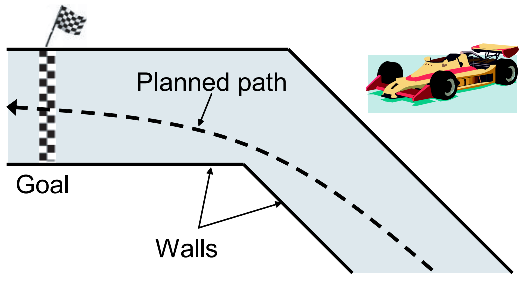


Figure 1: Risk allocation in a race car path planning scenario. A large portion of risk is allocated to the corner, since taking a risk (approaching the wall) at corner results in greater time saving than taking the same risk along straightaway.

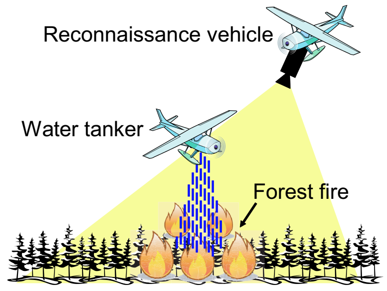


Figure 2: Risk allocation for multi-UAV fire-fighting system. The water tanker is allowed to fly low since it is allocated larger risk than the reconnaissance vehicle.

according to her needs and requirements.

Our focus is on the second approach: performance maximization with an upper-bound on risk. An example problem is to drive a car as fast as possible while limiting the probability of a crash to 0.01%. This formulation is particularly useful for optimal planning and control problems that involve high-impact low-probability risk such as loss of life.

With this formulation, [13] showed that we should optimize not only the sequence of actions but also the *risk allocation* in order to maximize the performance under a risk bound .

The example shown in Figure 1 illustrates the concept of risk allocation. A race car driver wants to plan a path to get to the goal as fast as possible. However, crashing into the wall leads to a fatal accident, so he wants to limit the probability of a crash to 0.01%. An intelligent driver would plan a path as shown in Figure 1, which runs mostly in the middle of the straightaway, but gets close to the wall at the corner. This is because taking a risk (i.e. approaching the wall) at the corner results in a greater time saving than taking the same risk along the straightaway; in other words, the utility of taking risk is greater at the corner than the straightaway. Therefore the optimal path plan allocates a large portion of risk to the corner, while allocating little to the straightaway. As illustrated by this example, *risk allocation* needs to be optimized across the constraints, in order to maximize the performance.

The optimal controller then needs to generate an optimal action sequence that abides to the allocated risk at each constraint.

Risk allocation for multi-agent system.

Past work on risk allocation [6][13][14] focused on single agent problems.

In this work we extend the concept of risk allocation to multi-agent systems. Figure 2 shows an example of a multi-agent system with two unmanned air vehicles (UAVs), whose mission is to extinguish a forest fire. A water tanker drops water while a reconnaissance vehicle monitors the fire with its sensors. The loss of either vehicle results in a failure of the mission. Two vehicles are required to extinguish the fire as efficiently as possible, while limiting the probability of mission failure to a given risk bound, say, 0.1%. The water tanker can improve efficiency by flying at a lower altitude, but it involves risk. The reconnaissance vehicle can also improve the data resolution by flying low, but the improvement of efficiency is not as great as the water tanker. In such a case a plausible plan is to allow the water tanker to take a large portion of risk by flying low, while keeping the reconnaissance vehicle at a high altitude to avoid risk. This is because the utility of taking risk (i.e. flying low) is greater for the water tanker than for the reconnaissance vehicle.

The optimal risk allocation for multi-agent systems can be found by applying the same algorithm as the single agent problems, such as [6][13][16], with extended state variable that include all agents. However, this approach requires centralized computation, which has an issue of scalability.

In this paper we propose a novel *decentralized* algorithm that finds the globally optimal risk allocation among multiple agents.

Market-based risk allocation using tâtonnement.

Our approach is to use the market-based mechanism. In a computational market each agent demands risk in order to improve its own performance. However, it cannot take risk for free; it has to purchase it from the market at a given price.

Agents are price takers. Given the price, each agent computes the optimal amount of risk to take (i.e., *demand for risk*) by solving the optimization problem where the objective function is the summation of the original cost function and the payment for the risk. The optimal action sequence and the internal risk allocation are also determined by solving the optimization problem, just as in the single-agent case described before. The demand from each agent can be seen as a function of the price of risk (*demand curve*). Typically, the higher the price is, the less each agent demands. Each agent has a different demand curve according to its sensitivity to risk. On the other hand, the supplier of the risk is the user. She supplies the fixed amount of risk by specifying the upper-bound of risk the system can take.

The price must be adjusted so that the total demand (*aggregate demand*) becomes equal to the supply. The equilibrium price is found by an iterative process called *tâtonnement* or *Walrasian auction* [17] as follows:

- Increase the price if aggregate demand exceeds supply,
- Decrease the price if supply exceeds aggregate demand,
- Repeat until supply and demand are balanced.

In classical tâtonnement, the price increment is obtained by simply multiplying the excess aggregate demand by a constant. However, the upperbound of the constant that guarantees the convergence is specific to a problem, and is hard to find. Slow convergence speed is also an issue. Our method obtains the price increment in each iteration by computing one step of Brent's method, which is a commonly-used root-finding algorithm with fast and guaranteed convergence [1].

Figure 3 gives the graphical interpretation of the market-based risk allocation in a system with two agents. Agent 1 and Agent 2 have different demand curves, since their utility of taking the same

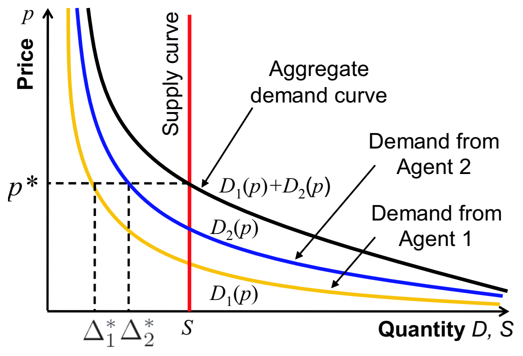
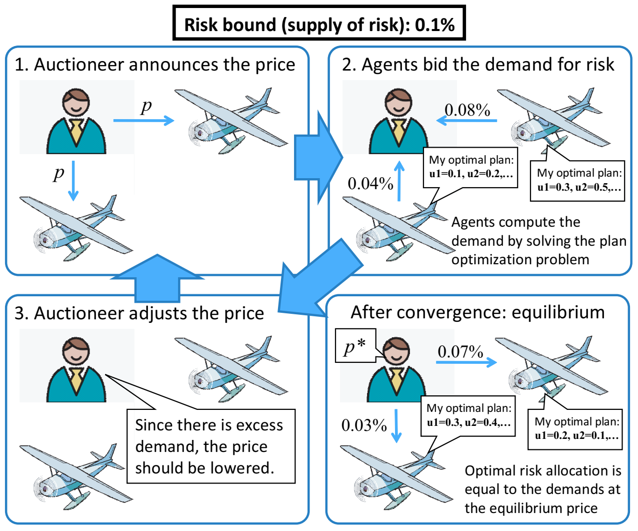


Figure 3: Market-based risk allocation in a system with two agents. Note that we followed the economics convention of placing the price on the vertical axis. The equilibrium price is $p^\star$, and the optimal risk allocation is $\Delta_1^\star = D_1(p^\star)$ for Agent 1 and $\Delta_2^\star = D_2(p^\star)$ for Agent 2.



Risk bound (supply of risk): 0.1%
1. Auctioneer announces the price
p
p
2. Agents bid the demand for risk
0.08%
0.04%
My optimal plan: u1=0.1, u2=0.2,...
My optimal plan: u1=0.3, u2=0.5,...
Agents compute the demand by solving the plan optimization problem
3. Auctioneer adjusts the price
Since there is excess demand, the price should be lowered.
After convergence: equilibrium
p^*
0.07%
0.03%
My optimal plan: u1=0.3, u2=0.4,...
My optimal plan: u1=0.2, u2=0.1,...
Optimal risk allocation is equal to the demands at the equilibrium price

Figure 4: Illustration of MIRA algorithm. Risk is allocated to agents through tâtonnement; their continuous action sequences are also optimized in the loop when computing the demand at the given price.

risk is different. The aggregate demand curve is obtained by adding the two demand curves horizontally. The supply curve is a vertical line since it is constant. The equilibrium price $p^\star$ lies at the intersection of the aggregate demand curve and the supply curve. The optimal risk allocation for the two agents corresponds to their demands at the equilibrium price ($\Delta_1^\star$ and $\Delta_2^\star$ in Figure 3).

It is proven in a later section that the performance of the entire system is maximized at the equilibrium price, although each agent only maximizes its own utility. The only information that each agent needs to communicate in each iteration is price and demand, both of which are a scalar value. These are desirable features for distributed systems.

MIRA - Decentralized optimization of risk allocation.

Our proposed algorithm, MIRA (Market-based Iterative Risk Allocation), optimizes risk allocation between agents, internal risk allocation of each agent, and action sequences of each agent concurrently.

Figure 4 illustrates the Market-based Iterative Risk Allocation (MIRA) algorithm. The tâtonnement process is repeated until it converges to the equilibrium price. Risk is not allocated until the algorithm converges. The optimal action sequence and the internal risk allocation are also obtained as the by-product of the demand optimization problem (Step 2 in the Figure 4).

Figure 5 shows how MIRA algorithm breaks down the risk for individual constrains in each agent. The risk bound for the system is given by the user (A). Risk is optimally allocated to agents through tâtonnement (B). Each agent optimizes internal risk allocation when computing the demand in each iteration of MIRA (C). At the same time, the action sequence is optimized according to the internal risk allocation (D).

1.3 Related Work

MDP-based algorithms have been mainly used to solve multi-agent planning problems under uncertainty in discrete domains [2][11][12]. M-DPFP algorithm proposed by [9] can solve problems with continuous resources. Our problem formulation is different from MDP-based approaches in that we set an upper bound on risk (*chance constraint*) and maximize performance within this bound, instead of maximizing utility. Also, our focus is on the problem with continuous state and action space.

Optimal control under uncertainty with chance constraint is in-

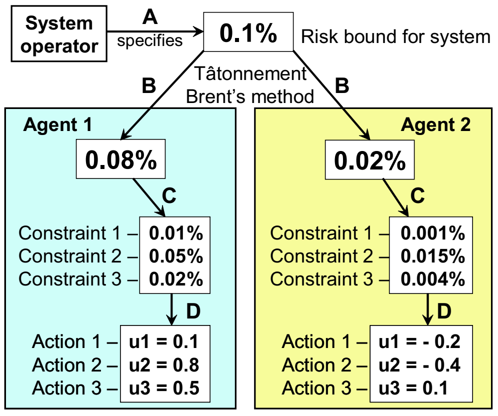


Figure 5: Distribution of risk in MIRA algorithm.

tensively researched in the robust model predictive control (RMPC) community. Due to the difficulty of handling the chance constraint analytically, past work used either a very conservative bound that resulted in large suboptimality [5][19], or a sample-based method [4] that is computationally inefficient. Based on the pioneer work by [16] that proposed the conservative approximation of chance constraint by decomposing it into multiple atomic chance constraints, [13] introduced the concept of risk allocation, and developed Iterative Risk Allocation (IRA) algorithm that can optimize risk allocation efficiently, with substantially smaller suboptimality than the past work [14].

Market-based approach has recently been recognized as an effective tool for distributed multi-agent systems in AI community. Although tâtonnement has drawn less attention than auctions, it has been successfully applied to various problems such as the distribution of heating energy in an office building[20], control of electrical power flow[7], and resource allocation in communication networks[8]. In economics, a simple linear price update rule has long been the main subject of study, but the convergence of price can only be guaranteed under a quite restrictive condition[17]. In order to substitute the linear price update rule, various root-finding methods have been employed in agent-based resource allocation algorithms, such as the bisection method[21], Newton method[22], and Broyden's method[20]. However, in general, it is difficult to guarantee quick and stable convergence to the equilibrium price. We employ Brent's method [1] to provide guaranteed convergence with a superlinear rate of convergence by exploiting the fact that a risk, which is treated as a resource in our problem formulation, is a scalar value.

2. RISK ALLOCATION FOR SINGLE-AGENT SYSTEMS

We will first briefly review the mathematical formulation of risk allocation. Our focus on this paper is a problem with continuous state space, although the concept of risk allocation can be used for discrete/hybrid systems [13].

2.1 Formulation

Optimization of action sequence under uncertainty.

We formulate an optimization problem with a chance constraint as follows:

$$\min_{\boldsymbol{u}_{1:T}} \quad J(\boldsymbol{u}_{1:T}) \tag{1}$$

$$\text{s.t.} \quad \forall_t \quad \boldsymbol{x}_{t+1} = \boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{B}\boldsymbol{u}_t + \boldsymbol{w}_t \tag{2}$$

$$\forall_t \quad \boldsymbol{u}_{\min} \leq \boldsymbol{u}_t \leq \boldsymbol{u}_{\max} \tag{3}$$

$$\Pr\left[\bigwedge_{t=0}^{T} \bigwedge_{n=1}^{N_t} g_{t,n}(\boldsymbol{x}_t) \leq 0\right] \geq 1 - \Delta \tag{4}$$

where $\boldsymbol{x}_t$ and $\boldsymbol{u}_t$ are the state vector and action (control input) vector at the tth time step, respectively. The disturbance $\boldsymbol{w}_t$ and the initial state estimation $\boldsymbol{x}_0$ have Gaussian distributions with known mean and variance. Although we focus on Gaussian-distributed uncertainty in this paper for simplicity, our algorithm can be applied to any stochastic uncertainties with quasi-concave distribution. We assume that $J(\cdot)$ and $g_{t,n}(\cdot)$ are convex functions.

A discrete-time stochastic dynamics model of the system is given as (2)(3), and state constraints are imposed as (4). Since violation of any state constraint at any time step is regarded as a mission failure, the probability of satisfying all constraints at all time steps must be more than $1 - \Delta$, where Δ is the upper bound of the probability of failure (risk bound). Given a risk bound Δ, the action sequence $\boldsymbol{u}_{1:T} := [\boldsymbol{u}_1 \cdots \boldsymbol{u}_T]^T$ that minimizes the cost J in (1) is obtained as an output by solving the optimization problem. In other words, the user can adjust the risk averseness of the system by specifying the risk bound Δ.

Decomposition of chance constraint.

The chance constraint (4) is hard to evaluate since it involves a probability defined on a multi-dimensional distribution. We decompose this constraint into multiple atomic chance constraints that only involve a single-dimensional distribution, using the following Boole's inequality:

$$\Pr\left[\bigcup_i A_i\right] \leq \sum_i \Pr\left[A_i\right] \tag{5}$$

Observe that, using Boole's inequality (5), following condition (6), together with (7), implies the original chance constraint (4).

$$\forall_{(t,n)} \quad \Pr\left[g_{t,n}(\boldsymbol{x}_t) \leq 0\right] \geq 1 - \delta_t^n \tag{6}$$

$$\sum_{t=1}^{T} \sum_{n=1}^{N_t} \delta_t^n \leq \Delta \tag{7}$$

$$\forall_{(t,n)} \quad \delta_t^n \geq 0 \tag{8}$$

Therefore, the original chance constraint (4) can be replaced with (6) and (7). Since we introduced new variables δ_t^n, the cost J in (1) needs to be optimized over δ_t^n as well as the sequence of actions $\boldsymbol{u}_{1:T}$:

$$\min_{\boldsymbol{\delta}, \boldsymbol{u}_{1:T}} \quad J(\boldsymbol{u}_{1:T}) \tag{9}$$

where $\boldsymbol{\delta} = \left[\delta_0^1 \ \delta_0^2 \ \cdots \delta_T^{N_T - 1} \ \delta_T^{N_T}\right]^T$.

We now have the revised constrained optimization problem defined by (9) with constraints (2)(3)(6)(7).

2.2 Risk allocation

The newly introduced variable $\boldsymbol{\delta}$ is the mathematical representation of *risk allocation*. In (6), each single constraint at each time step has its own risk bound δ_t^n; in other words, δ_t^n is the amount of risk allocated to the nth constraint at the tth time step. Eq.(7) states that the summation of all individual risk bound is upper-bounded by the original risk bound Δ; therefore risk is regarded as a resource with total amount Δ. In order to obtain the maximum performance (minimum cost), the risk allocation needs to be optimized (9), just as resource allocation problems.

The risk allocation optimization problem can be solved efficiently by a two-stage algorithm called Iterative Risk Allocation (IRA) [14]. Alternatively it can also be solved by a single shot optimization [6]. We employ the latter approach in this work.

3. DECENTRALIZED RISK ALLOCATION FOR MULTI-AGENT SYSTEMS

We first formulate the optimization problem under uncertainty for multi-agent systems in a centralized manner, and then derive the decentralized formulation using Karush-Kuhn-Tucker (KKT) conditions of optimality. We will then observe that economic concepts such as price, demand, and supply appear in the resulting formulation.

3.1 Formulation

Optimization of action sequence under risk for multi-agent system.

In a multi-agent system such as the UAV fire fighting system illustrated in Figure 2, the failure of one agent leads to a failure of the entire system. In a manned system, loss of one crew member is regarded as a failure of the mission. Therefore the user wants to set an upper-bound on the probability of having at least one agent fail.

With the same discussion as in the previous section, the following bound is obtained by using Boole's inequality (5):

$$\sum_{i=1}^{I} \Delta_i \leq S \tag{10}$$

where Δ_i is the upper bound on the probability of failure of the ith agent, I is the number of agents in the system, and S is the upper bound on the probability of failure of the entire system (i.e. the total amount of risk the entire system is allowed to take). Note that Δ was a given constant in the single-agent case (4)(7), but now each Δ_i is a decision variable, while S is the new given constant, which is specified by the user.

The performance (cost) of the entire system is defined as the sum of the performance (cost) of all individual agents:

$$J_{sys} = \sum_i J_i(\boldsymbol{u}_{i,1:T}) \tag{11}$$

The optimal control problem under risk for multi-agent systems is formulated as a constrained optimization problem of minimizing (11), subject to the constraints (2)(3)(6)(7) for each agent, and (10).

To simplify this formulation, we define a function[1] $J_i^\star(\Delta_i)$, which is equal to the minimized cost for the ith agent obtained by solving the constrained optimization problem for a single agent (9)(2)(3)(6)(7) given Δ_i:

$$J_i^\star(\Delta_i) = J(\boldsymbol{u}_{i,1:T}^\star)$$

where $\boldsymbol{u}_{1:T}^\star$ is the solution to the single agent optimization problem given Δ_i. An important fact is that function $J_i^\star(\Delta_i)$ is a convex function. See Appendix of [15] for the proof.

Using $J_i^\star(\Delta_i)$, the optimization problem can be rewritten in a simple form as follows:

$$\min_{\Delta_{1:I}} \quad \sum_{i=1}^{I} J_i^\star(\Delta_i) \tag{12}$$

$$\text{s.t.} \quad \sum_i \Delta_i \leq S \tag{13}$$

This formulation describes a centralized approach where the action sequences and risk allocations of all agents are planned in a single optimization problem. We will next derive the decentralized formulation using the KKT conditions of optimality.

Decentralized optimization.

We build upon the resource allocation algorithm proposed by [20], with an modification of using the KKT conditions for optimality instead of the method of Lagrange multipliers, since risk (resource) is bounded by inequality (13) in our problem formulation.

The KKT conditions of the optimization problem (12)(13) are: [2]

$$\left.\frac{dJ_i^\star}{d\Delta_i}\right|_{\Delta_i^\star} + p = 0 \tag{14}$$

$$\sum_i \Delta_i^\star \leq S \tag{13}$$

$$p \geq 0 \tag{15}$$

$$p\left(\sum_i \Delta_i^\star - S\right) = 0 \tag{16}$$

where p is the Lagrange multiplier corresponding to the constraint (13). This is the necessary and sufficient condition for optimality, since $J_i^\star(\Delta_i)$ is convex.

[1]It is often not possible, and not necessary as well, to obtain the function $J_i^\star(\Delta_i)$ in a closed form; in practice $J_i^\star(\Delta_i)$ is evaluated simply by solving the optimization problem (9)(2)(3)(6)(7), with an extra term $p\Delta_i$ added to the objective function (9).

[2]We assume the differentiability of $J_i^\star(\Delta_i)$ here; in fact, since $J_i^\star(\Delta_i)$ is a convex function, it is continuous and differentiable at all but at most countably many points in its domain; we can obtain the same result for the points where it is not differentiable by using extended KKT condition with subgradient.

Observe that (14) is also the optimality condition for the following unconstrained optimization problem:

$$\min_{\Delta_i} J_i^\star(\Delta_i) + p\Delta_i \tag{17}$$

Therefore solving the optimization problem (12) and (13) is equivalent to solving I independent optimization problems (17) with common parameter p, which is determined by (13), (15), and (16). Since (17) contains only the variables related to the ith agent, it can be solved by each agent in a decentralized manner. Each agent optimizes its internal risk allocation $\boldsymbol{\delta}$ (Figure 5-C) and action sequence $\boldsymbol{u}_{1:T}$ (Figure 5-D) by solving (17).

3.2 Economic Interpretation

The economic interpretation of these mathematical manipulations becomes clear by regarding the Lagrange multiplier p as the *price of risk*. Each agent can reduce the cost (i.e. improve the performance) by taking more risk Δ_i, but not for free. Note that a new term $p\Delta_i$ is added to the cost function (17). This is what the agent has to pay to take the amount of risk Δ_i. The agent must find the optimal amount of risk $D_i(p)$ to minimize the cost plus payment, by solving the optimization problem (17) with a given price p:

$$D_i(p) = \arg\min_{\Delta_i} J_i^\star(\Delta_i) + p\Delta_i. \tag{18}$$

In other words, $D_i(p)$ is the amount of risk the ith agent wants to take at the given price of risk p. Therefore $D_i(p)$ can be interpreted as the ith agent's *demand for risk*. On the other hand, the total amount of risk S can be interpreted as the *supply of risk*.

The optimal price $p^\star$ must satisfy the KKT conditions (13), (15), and (16), with the optimal demands at the price $\Delta_i^\star = D_i(p^\star)$. Such a price $p^\star$ is called the *equilibrium price*.

The condition (16) illustrates the relation between the equilibrium price $p^\star$, optimal demand $\Delta_i^\star$, and supply S; in the usual case where the equilibrium price is positive $p^\star > 0$, the aggregate demand $\sum_i \Delta_i^\star$ must be equal to the supply S; in a special case where the supply always exceeds the demand for all $p \geq 0$, the optimal price is zero $p^\star = 0$. If the aggregate demand always exceeds the supply for all $p \geq 0$, there is no solution that satisfies the primal feasibility condition (13), and hence the problem is infeasible. See Figure 3 for the graphical interpretation.

3.3 Global Optimality

The optimal risk allocation to each agent is equal to its demand for risk at the equilibrium price. This is the globally optimal solution since all the KKT conditions for optimality (13)-(16) are satisfied at the equilibrium price. If the supply always exceeds the demand for all $p \geq 0$, the demand at $p = 0$ is the globally optimal risk allocation.

Therefore, we must find the equilibrium price $p^\star$ that satisfies (13)(15)(16) in order to solve the optimization problem. The next section discusses how MIRA finds such an equilibrium price.

4. TÂTONNEMENT: PRICE ADJUSTMENT MECHANISM

We employ an iterative process called tâtonnement to find the equilibrium price: initialize the price with arbitrary value, and update it in each iteration according to the excess demand (supply) until the demand and supply are balanced (Figure 4).

In the real world economy the demand for a good is typically a monotonically decreasing function of price (people want more if price is less). This is also the case in our computational economy.

By differentiating (14),

$$\frac{dp}{dD_i} = -\frac{d^2 J_i^\star}{d\Delta_i^2}\bigg|_{\Delta_i = D_i} \leq 0 \tag{19}$$

The inequality comes from the fact that $J_i^\star(\Delta_i)$ is a convex function (See the Appendix of [15] for proof). Since demand is monotonically decreasing, the equilibrium price found by tâtonnement is the sole and globally optimal equilibrium.

The price is updated in each iteration (Step 3 of Figure 4). The price update rule must be carefully chosen for quick and stable convergence to the equilibrium. In the following subsections we investigate two update rules: linear increment and Brent's method [1].

4.1 Linear Price Increment

The following simple price update rule is most extensively explored in the economics literatures:

$$p_{k+1} = \max\left\{ p_k + c\left(\sum_i D_i(p_k) - S\right), 0 \right\} \tag{20}$$

where p_k is the price in the kth iteration.

With this update rule, the price is guaranteed to converge for *sufficiently small* $c > 0$, if a condition called gross substitutability is satisfied [17]. Although this update rule is commonly studied, it has three issues when computing the equilibrium. First, the condition for convergence (gross substitutability) is very restrictive. Second, the constant parameter c is specific to a problem, and it is very hard to obtain the upper bound for which the convergence is guaranteed. Third, the convergence is slow. In our case, where there is a single kind of resource (risk) exchanged in the market, gross substitutability[3] is implied by the decreasing monotonicity of the demand function. However, the other two issues still exist in our case. We solve these issues by applying Brent's method.

4.2 Brent's method

Mathematically, tâtonnement can be seen as a process of finding a root of the excess demand function: $\sum_i D_i(p) - S$.

Brent's method is a root-finding algorithm that achieves quick and guaranteed convergence, by combining three methods: the bisection method, the secant method, and inverse quadratic interpolation [1]. Another important feature of Brent's method is that it does not require the derivative of $D_i(p)$, which is very hard to obtain.

As far as we know, there is only one past work [23] that applies Brent's method to find the equilibrium price, but in economics literature; no past research has applied Brent's method to resource allocation problems.

This is probably because Brent's method can only take a scalar variable, while resource allocation algorithms typically deal with multiple resources (i.e. vector). However, in a *risk* allocation problem, risk is always a scalar value. Therefore Brent's method can efficiently and reliably find the equilibrium price of risk. It is possible to extend our algorithm, MIRA, to solve multi-resource allocation problems by using a generalization of Brent's method[10] or by decomposing the market so that each market deals with only one kind of resource[21]. However, such extensions of MIRA are out of the scope of this paper.

4.3 MIRA algorithm

Algorithm 1 shows the entire flow of the MIRA algorithm. MIRA has a distributed part and centralized part. The computation of demand (Line 5) is distributed to each agent; by solving (18), each agent obtains the optimal demand at the price given by Line 4, as well as its optimal sequence of actions ($\boldsymbol{u}_{i,1:T}$). The computation of price (Line 7) is centralized; the auctioneer collects the demands from all agents (Line 6), and updates the price using Brent's method according to the excess demand/supply. One of the agents plays the role of the auctioneer.

The computation time of the centralized part is substantially shorter than the distributed part. For example, in the fire-fighter UAV scenario with two agents (see Section 5.1), the computation time of the distributed part is 13.8 seconds while the centralized part takes only 0.046 seconds. Moreover, the number of agents does not influence the computation time of centralized part much, since Brent's method only takes the *aggregate* demand (i.e. the summation of the demands of all agents). Therefore, the centralized part of MIRA does not harm the scalability of the algorithm.

The communication requirements between agents are small; in each iteration, each agent receives a price (Line 4) and transmits its demand (Line 6), both of which are scalars.

The centralized part of the algorithm can be distributed by making all individual agents conduct the same computation of price update simultaneously. In such case the demands of all agents must be shared with all agents, while price needs to be shared only at the first iteration. Although we can remove the centralized auctioneer in this way, there is no advantage in terms of computation time.

5. SIMULATION

We implemented MIRA in Matlab. Non-linear optimization solver SNOPT is used to compute the demand $D_i(p)$, and the Matlab implementation of Brent's method (fzero) is used to find the equilibrium price $p^\star$. Simulations were conducted on a machine with Intel(R) Core(TM) i7 CPU clocked at 2.67 GHz and 8GB RAM.

5.1 Validity

We tested MIRA on the multi-UAV altitude planning problem for the fire-fighting scenario (Figure 2). Figure 6 shows the simulation result. Two vehicles fly at the constant horizontal speed, starting from $d = 0$ at altitude 0.5. The mission is to extinguish the fire at $d = 6, 7$. Both vehicles minimize the flight altitude above the fire, although the water tanker is given 100 times more penalty (cost) of flying at high altitude than the reconnaissance vehicle. Both have uncertainty in altitude, so flying at lower altitude involves more risk. The total risk must be less than 0.1%.

The optimal plan allocates 99.2% of the total risk to the water tanker, while only 0.8% to the reconnaissance vehicle. This is because the utility of taking risk (i.e. flying low) is larger for the water tanker than for the reconnaissance vehicle. As a result, the water tanker flies at a lower altitude.

Both vehicles optimize the internal risk allocation as well. For

Algorithm 1 Market-based Iterative Risk Allocation

```
1: Fix $S$;        //Total supply of risk
2: Initialize $p$;        //Price of risk
3: while $|\sum_i D_i(p) - S| > \epsilon$ and $p > 0$ do
4:     Auctioneer announces $p$;
5:     Each agent computes its demand for risk $D_i(p)$ by solving (18);
6:     Each agent submits its demand to the auctioneer;
7:     The auctioneer updates $p$ by computing one step of Brent's method;
8: end while
```

[3]In the general equilibrium theory, money is also treated as a goods; therefore, in our case, the gross substitutability is defined as the substitutability between risk and money.

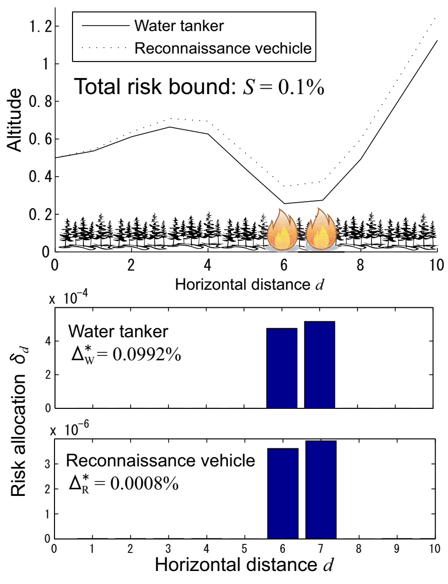


Figure 6: Simulation result of flight altitude planning problem for multi-UAV fire-fighting scenario (see Figure 2). Upper graph: Optimized altitude profile; Lower graphs: Internal risk allocation of each vehicle.

example, the water tanker takes 99.9% of the allocated risk above the fire, at $d = 6$ and 7 (the middle graph in Figure 6).

The optimal action sequence is planned according to this risk allocation; both vehicles dive before the fire, and climb as fast as possible, after they pass the fire (the top graph in Figure 6). This is because there is no benefit of conducting risky low-altitude flight before and after the fire.

These results conform with intuition. The optimality of MIRA is validated by the result that the difference in the optimized cost between MIRA and the centralized algorithm is less than 0.01%, which is accounted by numerical error.

5.2 Efficiency

In order to evaluate the efficiency of the MIRA algorithm, the computation times of the following three algorithms are compared: 1) centralized optimization, 2) decentralized optimization with the linear price update rule (classical tâtonnement), and 3) the proposed algorithm, MIRA, which is a decentralized optimization with Brent's method.

Table 1 shows the result. The three algorithms are tested with different problem sizes - two, four, and eight agents. Each algorithm is run 10 times for each problem size with randomly generated constraints. The parameter c is set so that the price converges in most problems. The average running time is shown in the table. The computation of the distributed algorithms was conducted parallelly. Communication delay is not included in the result.

The computation time of the centralized algorithm quickly grows as the problem size increases. Decentralized optimization with a linear price increment is even slower than the centralized algorithm, although the growth rate of computation time is slower.

MIRA, the proposed algorithm, outperforms the other two for all problem sizes. The advantage of MIRA becomes clearer as the problem size increases.

A counterintuitive phenomenon observed in the result is that the decentralized algorithms (MIRA and decentralized optimization with linear increment) also slow down for large problems, although not as significantly as the centralized algorithm. This is mainly because the iterations of tâtonnement must be synchronized among all agents. When each agent computes its demand for risk by solving the non-linear optimization problem, the computation time diverges from agent to agent, and from situation to situation. In each iteration of tâtonnement, all agents must wait until the slowest agent finishes computing its demand. As a result, tâtonnement process slows down for large problems, as the expected computation time of the slowest agent grows.

Table 1: Comparison of the computation time of three optimization algorithms. Values are the average of 10 runs with randomly generated constraints.

Number of agents	Computation time [sec]		
	Centralized	Decentralized (linear update)	MIRA
2	13.9	80.6	6.4
4	63.8	540.5	18.1
8	318.5	797.8	37.5

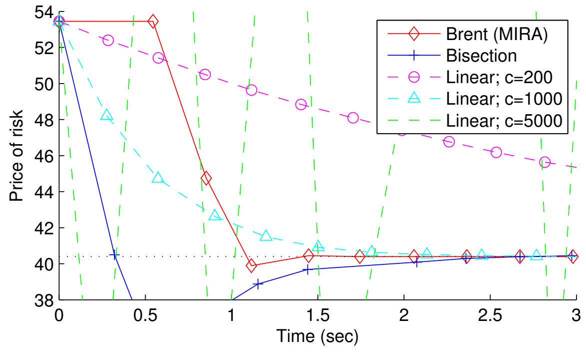


Figure 7: Convergence of different price update methods for tâtonnement

5.3 Convergence

Figure 7 shows the convergence of different price update algorithms for tâtonnement on a problem with four agents. Three algorithms are compared: Brent's method, which is employed by our proposed algorithm MIRA, the bisection method, which is used by WALRAS [21], and the linear price update. The linear price update is tested with three different settings of the parameter c in (20).

It is empirically shown from the result that increasing c makes the linear price update method faster, but it diverges when c is too large. The upperbound of c that guarantees the convergence is specific to a problem, and hard to obtain. Brent's method achieves the fastest converges among the three methods. Its convergence is guaranteed without parameter tunings.

5.4 Used Parameters

The horizontal speed of the vehicles is 1 per time step. Hence, $d = t$. The planning window is $1 \leq t \leq 10$. Other parameters are set as follows:

$$\boldsymbol{A} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \boldsymbol{B} = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix}, \boldsymbol{x}_0 = \begin{bmatrix} 0.5 \\ 0 \end{bmatrix},$$
$$u_{\min} = -0.2, u_{\max} = 0.2, \quad g_t(\boldsymbol{x}_t) = -\left[1 \;\, 0\right] \boldsymbol{x}_t + l_t$$

$\boldsymbol{w}_t$ is sampled from zero-mean Gaussian distribution with variance

$$\Sigma_w = \begin{bmatrix} 0.001 & 0 \\ 0 & 0 \end{bmatrix}.$$

l_t is the ground level at t. It is set at zero in the fire-fighter UAV scenario, and randomly generated for the evaluation of computation time. The cost functions are

$$\begin{aligned} J_W &= E\left[[100 \;\, 0]\left(\boldsymbol{x}_{6,W} + \boldsymbol{x}_{7,W}\right)\right] \\ J_R &= E\left[[1 \;\, 0]\left(\boldsymbol{x}_{6,R} + \boldsymbol{x}_{7,R}\right)\right] \end{aligned}$$

in Section 5.1 (subscript W and R indicate the water tanker and the reconnaissance vehicle respectively), and

$$J_i = E\left[\left[\begin{array}{cc} 1 & 0 \end{array}\right] \left(\sum_{t=1}^{10} \boldsymbol{x}_{t,i}\right)\right]$$

in Section 5.2 and 5.3. Note that the expectation of $\boldsymbol{x}_t$ is a function of $\boldsymbol{u}_{1:t-1}$. Therefore J is a function of $\boldsymbol{u}_{1:T}$.

6. CONCLUSION

We have developed Market-based Iterative Risk Allocation (MIRA), a multi-agent optimization algorithm that operates within user-specified risk bounds. It was built upon the concept of risk allocation. The key innovations presented in the paper include 1) extension of the concept of risk allocation to multi-agent system, 2) decentralized formulation of the multi-agent risk allocation optimization problem using market-based method, and 3) introduction of Brent's method to tâtonnement-based resource allocation algorithm. The simulation result showed that MIRA can optimize the action sequence of the multi-agent system by optimally distributing risk. It achieved substantial speed-up compared to centralized optimization approach, particularly in a large problem.

7. ACKNOWLEDGMENTS

Thanks to Michael Kerstetter, Scott Smith and the Boeing Company for their support. Hirokazu Ishise and Yuichi Yamamoto provided valuable comments on economics. Thanks also to Shuonan Dong, Cristina Wilcox, and Masako Kishida for proofreading.

8. REFERENCES

[1] K. E. Atkinson. *An Introduction to Numerical Analysis, Second Edition*. John Wiley & Sons, 1989.

[2] R. Becker, V. Lesser, and S. Zilberstein. Decentralized mdps with event-driven interactions. In *Proceedings of AAMAS-04*, 2004.

[3] K. Bell, A. I. Coles, M. Fox, D. Long, and A. J. Smith. The application of planning to power substation voltage control. In *Proceedings of ICAPS Workshop on Scheduling and Planning Applications*, 2008.

[4] L. Blackmore. A probabilistic particle control approach to optimal, robust predictive control. In *Proceedings of the AIAA Guidance, Navigation and Control Conference*, 2006.

[5] L. Blackmore, H. Li, and B. C. Williams. A probabilistic approach to optimal robust path planning with obstacles. In *Proceedings of American Control Conference*, 2006.

[6] L. Blackmore and M. Ono. Convex chance constrained predictive control without sampling. In *Proceedings of the AIAA Guidance, Navigation and Control Conference*, 2009.

[7] J. C. Jacobo, D. de Roure, and E. H. Gerding. An agent-based electrical power market. In *Proceedings of AAMAS-08: Demo Papers*, 2008.

[8] F. Kelly, A. Maulloo, and D. Tan. Rate control in communication networks: Shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49:237–252, 1998.

[9] J. Marecki and M. Tambe. Planning with continuous resources for agent teams. In *Proceedings of AAMAS-09*, 2009.

[10] J. M. Martínez. Solving nonlinear simultaneous equations with a generalization of brent's method. *BIT Numerical Mathematics*, 20:501–510, 1980.

[11] D. Musliner, E. Durfee, J. Wu, D. Dolgov, R. Goldman, and M. Boddy. Coordinated plan management using multiagent mdps. In *AAAI Spring Symposium*, 2006.

[12] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed pomdps: A synergy of distributed constraint optimization and pomdps. In *Proceedings of IJCAI-05*, 2005.

[13] M. Ono and B. C. Williams. An efficient motion planning algorithm for stochastic dynamic systems with constraints on probability of failure. In *Proceedings of AAAI-08*, 2008.

[14] M. Ono and B. C. Williams. Iterative risk allocation: A new approach to robust model predictive control with a joint chance constraint. In *Proceedings of 47th IEEE Conference on Decision and Control (CDC-08)*, 2008.

[15] M. Ono and B. C. Williams. Risk allocation for multi-agent systems using tâtonnement. Technical report, MIT Computer Science and Artificial Intelligence Laboratory, 2009.

[16] A. Prékopa. The use of discrete moment bounds in probabilistic constrained stochastic programming models. *Annals of Operations Research*, 85:21–38, 1999.

[17] J. Tuinstra. *Price Dynamics in Equilibrium Models: The Search for Equilibrium and the Emergence of Endogenous Fluctuations*. Kluwer Academic Publishers, 2000.

[18] R. M. Turner and E. H. Turner. A two-level, protocol-based approach to controlling autonomous oceanographic sampling networks. *IEEE Journal of Oceanic Engineering*, 26, 2001.

[19] D. H. van Hessem. *Stochastic inequality constrained closed-loop model predictive control with application to chemical process operation*. PhD thesis, Delft University of Technology, 2004.

[20] H. Voos. Agent-based distributed resource allocation in technical dynamic systems. In *Proceedings of IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications (DIS 2006)*, 2006.

[21] M. P. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.

[22] F. Ygge and H. Akkermans. Power load management as a computational market. In *Proceedings of Second International Conference on Multiagent Systems*, 1996.

[23] E. R. Young. Unemployment insurance and capital accumulation. *Journal of Monetary Economics*, 51:1683–1710, 2004.