Published in the *Proc. 8th International Conference on Case-Based Reasoning*.

On Similarity Measures based on a Refinement Lattice

Santiago Ontañón[1] and Enric Plaza[2]

[1] CCL, Cognitive Computing Lab Georgia Institute of Technology,
Atlanta, GA 303322/0280, santi@cc.gatech.edu
[2] IIIA, Artificial Intelligence Research Institute
CSIC, Spanish Council for Scientific Research
Campus UAB, 08193 Bellaterra, Catalonia (Spain), enric@iiia.csic.es

Abstract. Retrieval of structured cases using similarity has been studied in CBR but there has been less activity on defining similarity on description logics (DL). In this paper we present an approach that allows us to present two similarity measures for feature logics, a subfamily of DLs, based on the concept of *refinement lattice*. The first one is based on computing the anti-unification (AU) of two cases to assess the amount of shared information. The second measure decomposes the cases into a set of independent *properties*, and then assesses how many of these properties are shared between the two cases. Moreover, we show that the defined measures are applicable to any representation language for which a refinement lattice can be defined. We empirically evaluate our measures comparing them to other measures in the literature in a variety of relational data sets showing very good results.

1 Introduction

Knowledge intensive case-based reasoning (CBR) has traditionally used structured representation of cases and in the recent past it has moved more close to ontology engineering and knowledge representation formalisms like description logics. Retrieval of structured cases using similarity has been studied in CBR (see section 6) but there has been less activity on defining similarity on description logics (DL) for CBR. Part of the problem is that one can define a variety DLs: should we define a different similarity measure for each DL?

In this paper we present an approach that allows us to present two similarity measures for feature logics [8], a subfamily of DLs, based on the concept of *refinement lattice*. The concept of refinement lattice is taken from the generalization space notion of inductive learning [16], and as such is general: it is the lattice generated by a collection of refinement operators that relate two generalizations. Since any specific DL formalism can be, in principle, equipped with its own refinement operators (that induce a refinement lattice) the two similarity measures we present here can also be applied to them.

Specifically, we present two similarity measures based on a refinement lattice for feature logics. The first one computes the anti-unification (AU) of two cases,

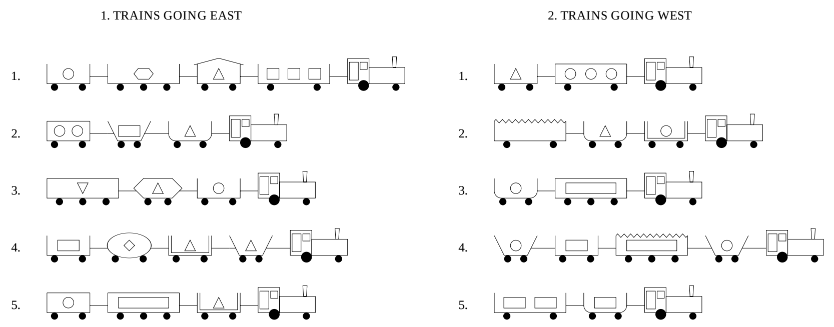


Fig. 1. Trains data set as introduced by Michalski [15].

representing all the information common to these two cases; then it assesses how much information is contained in the AU with respect to the total amount of information in the two cases. We will call it *AU-based similarity* (S_λ). The second measure is called *property-based similarity* (S_π); S_π decomposes the cases into a set of independent *properties*, and then assesses how many of these properties are shared between the two cases.

The remainder of this paper is organized as follows. In Section 2 we will briefly introduce some notions of relational machine learning required to define our measures. Sections 3 and 4 present the anti-unification-based measure and the property-based measure respectively. In Section 5 we describe our empirical evaluation of the measures, comparing them to other measures in the literature in a variety of relational data sets. Section 6 presents related work on relational similarity measures. Section 7 summarizes the contributions of this paper and outlines future lines of research.

2 A refinement lattice for feature logics

Feature logics [8] (also called feature terms, feature structures or Ψ-terms) are a generalization of first-order terms that have been introduced in theoretical computer science in order to formalize object-centered capabilities of declarative languages. In this paper we use a concrete formalization (that may differ from that of [8] or [3]), used in the language NOOS [1].

As an example, consider the apparently simple *trains* data set introduced by Michalski [15], and shown in Figure 1. The original task is to find the rule that discriminates from east-bound and west-bound trains. Notice, however, that not all the trains have the same number of cars, and that, in principle, a train can have an unbounded number of cars. Thus, it is unclear how to represent this data using a feature vector without losing information. Using a relational representation, we can just represent each car as a term, and define that a train is a set of cars, without restricting the number of cars of the train or the

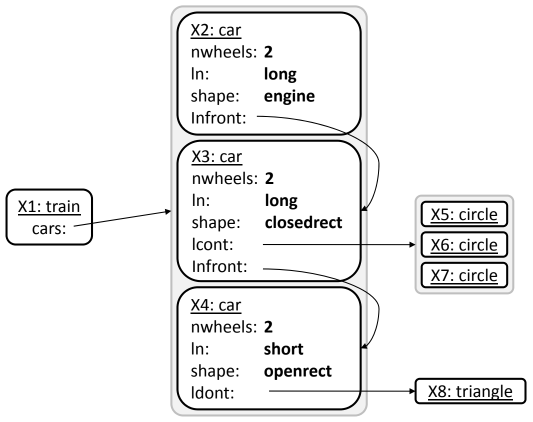


Fig. 2. A train represented using the feature terms representation formalism.

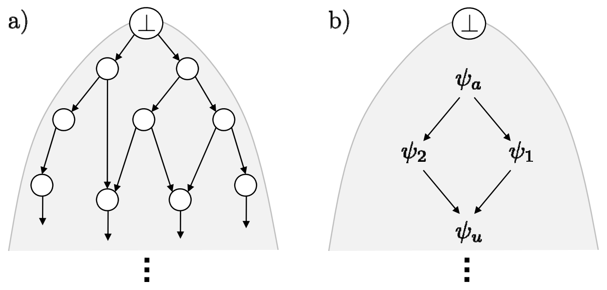


Fig. 3. a) Example of a *refinement lattice* defined by the subsumption relation, where each node represents a term, and the most general node is $\perp$. b) Example of two terms ψ_1 and ψ_2 in the refinement lattice with their unification ψ_u and anti-unification, ψ_a.

complexity of the load each train is carrying. For instance, Figure 2 represents the first west-bound train using feature terms.

Feature terms can be defined by its *signature*: $\Sigma = \langle \mathcal{S}, \mathcal{F}, \leq, \mathcal{V} \rangle$. Where $\mathcal{S}$ is a set of sorts symbols (including $\perp$, which represents both the most general sort and term), $\leq$ is a partial order among the sorts in $\mathcal{S}$ (representing the *is-a* relation common to object oriented languages)), $\mathcal{F}$ is a set of feature symbols, and $\mathcal{V}$ is a set of variable names. We can define a feature term ψ as:

$$\psi ::= X : s[f_1 \doteq \Psi_1, ..., f_n \doteq \Psi_n]$$

where ψ points to the *root* variable X (that we will note as $root(\psi)$), $X \in \mathcal{V}$, $s \in \mathcal{S}$, $f_i \in \mathcal{F}$, and Ψ_i might be either another feature term ψ_i, an already defined variable $Y \in \mathcal{V}$ or a set of feature terms $\{\psi_1, ..., \psi_m\}$. Finally, we also consider the basic data types (numbers and symbols) to be feature terms.

The basic operation between feature terms is *subsumption*, formally defined for feature terms in [1]. We say that a term ψ_1 subsumes another term ψ_2 when

ψ_1 is more general than ψ_2 and we denote that as $\psi_1 \sqsubseteq \psi_2$. The subsumption relation allows us to structure the space of possible feature terms in a semi lattice, that we will call the *refinement lattice*, where the root node is $\perp$, also called *any*. Figure 3.a shows an illustration of the refinement lattice, where each node represents a term, and arrows represent subsumption. Another interpretation of subsumption is that if a term ψ_1 subsumes another term ψ_2, all the information in ψ_1 is also contained in ψ_2. Typically, such space is only considered relevant for inductive learners since it defines the search space for hypothesis [17]. However, in this paper we are going to make use of the structure in such space in order to define similarity measures.

One way to build the refinement lattice is by defining *refinement operators*. A refinement operator ρ maps a feature term to a set of feature terms that are either generalizations of specializations depending if it is a *specialization* refinement operator or a *generalization* refinement operator. Refinement operators for subsets of first order logic have been defined in the literature [16, 21]. Such refinement operators can be used in the definition of inductive systems that systematically or heuristically explore the hypothesis space.

Given the subsumption relation, and any two terms ψ_1 and ψ_2, we can define the *anti-unification* of two terms as the *least general generalization* [19]. The anti-unification of two terms is relevant for defining similarity measures, since it contains all the information that is common to both ψ_1 and ψ_2, thus, it encapsulates in a single description all that is common to two given terms. Moreover, depending on the representation language being used, it might not be unique (it is not in the case of feature terms). A complementary operation to the anti-unification is that of *unification*, which is the *most general specialization* of a given set of terms. Figure 3.b graphically illustrates both concepts. Notice that both unification and anti-unification are operations over the refinement lattice: anti-unification corresponds to find the most specific common "parent", where as unification corresponds to find the most general common "descendant".

A fast algorithm to compute one of the anti-unifications of a set of terms T can be informally defined using a systematic search process over the refinement lattice in the following way. The search starts by having an initial candidate to be the anti-unification $c_0 = \perp$. At each step t of the algorithm, we will generate specialization refinements of the current candidate c_t. If any of those refinements subsumes all the terms in T, then that term will be taken as c_{t+1}. When in one cycle t none of the refinements subsume all of the terms in T, we will know that c_t is an anti-unification of T. Notice that this algorithm only finds one anti-unification out of all the possible ones. Moreover, the specialization refinement operator used for this algorithm must be complete, i.e. it has to be able to generate all the immediate successors of any term in the refinement lattice. It is also interesting to know the number of iterations required to find the anti-unification of two terms (since the larger the number of steps, the larger the anti-unification, and thus the more information shared among the terms).

Notice that the previous algorithm for computing the anti-unification, although defined here for feature terms, is independent of the representation language used as long as a suitable refinement operator and subsumption operation are available. However, for completeness, next section very quickly presents a refinement operator for feature terms that can be proven to be complete (although the proof is not included in this paper for the sake of space).

2.1 Refinement Operators for Feature Terms

The *specialization refinement operation* $\rho(\psi)$ for feature terms will be defined by five simpler refinement operators: $\rho(\psi) = \rho[s](\psi) \cup \rho[f](\psi) \cup \rho[v](\psi) \cup \rho[e](\psi) \cup \rho[c](\psi)$, as follows:

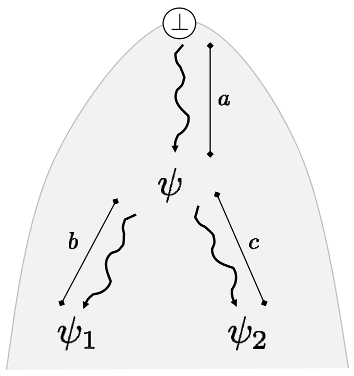
1. $\rho[s]$ generates all the possible specializations by specializing sorts in a term.
2. $\rho[f]$ generates specializations by taking each undefined feature in a term and adding them a variable with the most general sort that feature can take.
3. $\rho[v]$ generates specializations by adding "variable equalities", i.e. for any two variables X, Y in a feature term that can be unified, this operator will generate refinements where $X = Y$.
4. $\rho[e]$ generates all the possible specializations by expanding any set in a feature term (including converting single values into a set of two values). The value added is the most general value allowed in that set.
5. $\rho[c]$ generates refinements by replacing variables by constants.

Notice that in general there are an infinite number of possible refinements of a feature term (just imagine that we have a variable representing a real number, the $\rho[c]$ operator can refine that term by substituting the variable by any concrete real number). In order to make the operator tractable, it is possible to define an alternative definition $\rho(\psi, O)$, where O is a set of feature terms, and only terms that subsume at least a term in O are generated. This makes the number of refinements generated always finite (e.g. the set of constants to substitute variables for can be taken from the set of constants used in O).

Given this refinement operator, it is easy to define the opposite operator $\gamma(\psi)$. The purpose of defining refinement operators is to navigate through the refinement graph. Intuitively, $\rho(\psi)$ is an operator that maps a term to its immediate successors in the refinement lattice, and $\gamma(\psi)$ is an operator that maps a term to its immediate ancestors in the lattice.

3 Anti-Unification-based Similarity

The anti-unification of two feature terms ψ_1 and ψ_2 naturally introduces a similarity measure between any two terms. The anti-unification of two terms contains all the shared information of two terms. Thus, based on that, the *anti-unification based similarity* (S_λ) can de defined as: the ratio of shared information divided by the total amount of information. If two terms are very similar, the amount of common information will be very similar to the total information contained in both terms, and thus the similarity will approach 1.



$$S_\lambda(\psi_1, \psi_2) = \frac{a}{a+b+c}$$

Fig. 4. Illustration of the anti-unification based similarity, between two feature terms ψ_1, and ψ_2, whose anti-unification is ψ.

We need a way to count the amount of information contained in a feature term. Using the refinement lattice, we can define the amount of information in a feature term ψ as the distance in the refinement lattice from ψ to the object $\perp$ (the most general feature term). In other words, the number of times that a refinement operator has to be applied to $\perp$ to generate the feature term ψ. Figure 4 illustrates this idea, where a is the number of refinement steps from $\perp$ to the anti-unification of ψ_1 and ψ_2, b is the number of refinement steps from the anti-unification ψ to ψ_1, and c is the number of refinement steps from the anti-unification to ψ_2. Thus, we can define the similarity as:

$$S_\lambda(\psi_1, \psi_2) = \frac{a}{a+b+c}$$

Notice that by using the anti-unification algorithm outlined in Section 2, it is easy to compute a as the number of iterations required to compute the anti-unification. Moreover, in order to compute b, the same algorithm can be used, but using the anti-unification as a starting point, and using $T = \{\psi_1\}$ (c can be computed analogously).

The resulting similarity is a simple measure that can be used to compare any two cases represented using feature terms. This measure has, however, two main issues. First of all is its computational complexity. Although computing the anti-unification of two terms requires (using the algorithm mentioned in Section 2) a linear number of calls to the subsumption operator in function of the size of the terms, the subsumption operation might have an exponential complexity depending on the representation language used. Thus, in domains where the cases in the case base are large structures, this similarity measure might not be feasible. A second problem is that this similarity measure considers each refinement in the refinement lattice as equally important, it is like an edit distance where each operation has the same weight. Weights could be defined for each refinement operation, but it is not obvious how to generate them automatically. Next section presents another similarity measure which addresses these two problems.

4 Property-based Similarity

To address the problems introduced by the anti-unification-based similarity, we developed the *property-based similarity*. In our framework, A *property* is some condition that a term might satisfy or not. For example, in the trains domain introduced before, a property might be that "a train has at least 3 cars", and some trains might satisfy it and some might not. The main idea of the property-based similarity is to count, out of the set of properties that two cases satisfy, how many do they share.

In a feature-value representation it is easy to define the set of properties that a case satisfies: the set of features by which it is defined. However, in a complex relational representation such as feature terms, it is not obvious. In our framework, we will define a property as a pattern ψ_1, and given a term ψ_2, we say that ψ_2 satisfies the property if: $\psi_1 \sqsubseteq \psi_2$. Therefore, the set of properties that a term satisfies is the set of all the patterns that subsume it. Notice that that set might be very large (or even infinite). Therefore, we will rely again in the notion of refinement operators to define the set of properties that a term satisfies. Each time a refinement operator is applied to a term to make it more specific, information is added to the term, and thus the term "gains a new property". If we take the path in the refinement lattice from $\perp$ to a particular term ψ, each one of the refinement operators in that path defines a property, for which an appropriate pattern can be constructed as explained below.

4.1 An Illustrative Example

Before formally explaining the process of constructing the property patterns, let us illustrate it with an example. Imagine that we have a description ψ of a train, as shown on the top of Figure 5. The train contains two cars, one of them is a long engine, and the other one is a short open rectangle car with two circles on it. Moreover, we know that the engine is in front of the open rectangle car.

If we compute the path in the refinement lattice required to reach $\perp$ from ψ by using the $\gamma(\psi)$ generalization refinement operator, we will see that we need 17 refinements to reach it. Each one of those 17 generalization refinements removes a piece of information from the term, and thus "removes a property". For instance, let's say that the first generalization takes the value *long* of the feature *ln* in the car represented by variable *X2* and generalizes it to a variable of type *length*. The property that the train has lost is that one of the cars is *long*. Thus, the first property ψ_1 can be generated, as shown in Figure 5. The next generalization might generalize the value *engine* to a more general value *shape*. Leading to the second property ψ_2, that states that one of the cars of the train has shape *engine*. This process can go on until we reach $\perp$. Figure 5 shows all the different properties that will get created in the process. Notice that there are only 14 properties in this example, but 17 refinement steps. This is because some of the properties generated result in the same pattern, and thus we have removed duplicates. Once we have the set of properties, we can use them to approximate

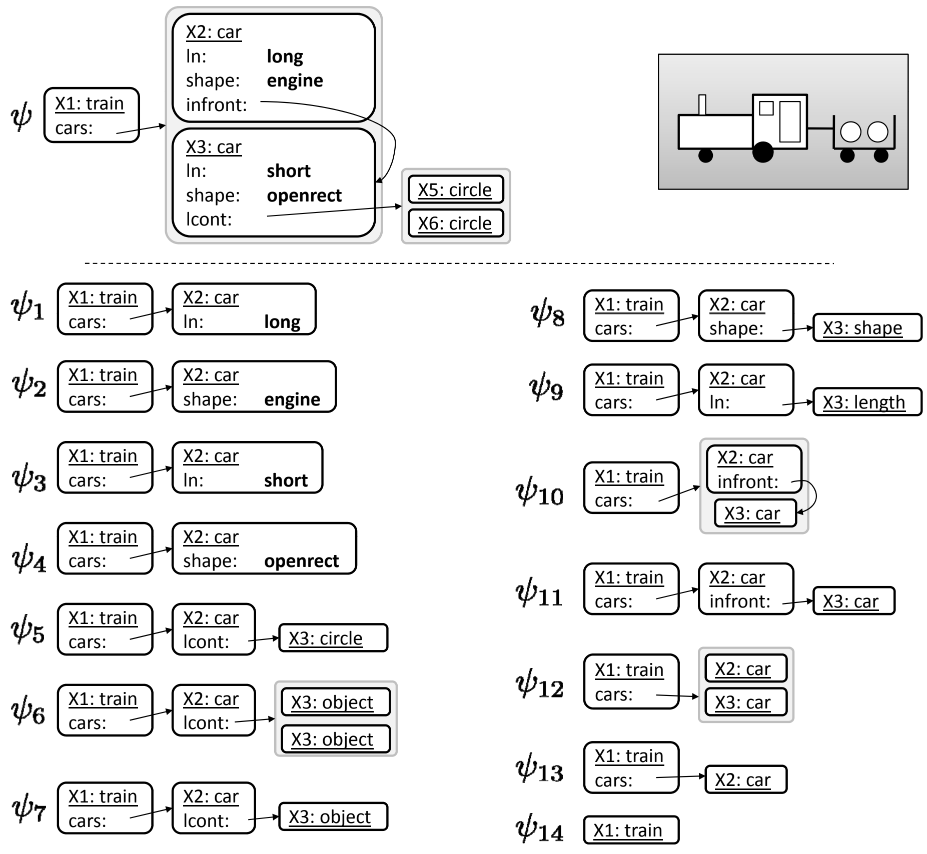

ψ
X1: train
cars:
X2: car
ln: long
shape: engine
infront:
X3: car
ln: short
shape: openrect
lcont:
X5: circle
X6: circle
ψ1 X1: train cars: X2: car ln: long
ψ2 X1: train cars: X2: car shape: engine
ψ3 X1: train cars: X2: car ln: short
ψ4 X1: train cars: X2: car shape: openrect
ψ5 X1: train cars: X2: car lcont: X3: circle
ψ6 X1: train cars: X2: car lcont: X3: object X3: object
ψ7 X1: train cars: X2: car lcont: X3: object
ψ8 X1: train cars: X2: car shape: X3: shape
ψ9 X1: train cars: X2: car ln: X3: length
ψ10 X1: train cars: X2: car infront: X3: car
ψ11 X1: train cars: X2: car infront: X3: car
ψ12 X1: train cars: X2: car X3: car
ψ13 X1: train cars: X2: car
ψ14 X1: train

Fig. 5. A simple train represented as a feature term, and all the properties that can be extracted from it.

the amount of shared information in between two cases by counting how many properties do they share.

At this point we can already see that a property associated with a generalization refinement captures exactly that piece of information that was removed from the term when generalizing. The intuition is that if we compute the unification of all the properties generated we should obtain the original term. In the case of feature terms, since unification is not unique, we can only say that one of the possible unifications of all the properties results in the original object. Therefore, the intuitive definition of a property pattern is that a pattern associated with a generalization refinement should be the smallest feature term that if unified with the generalization allows us to reconstruct the original term. Although computing such patterns can be also done in a domain independent way using the refinement lattice, next section presents a fast way to compute them when feature terms are used as the representation language.

4.2 Constructing the Properties

Given a term ψ, it is possible to use the generalization refinement operator $\gamma(\psi)$ to generalize the term step by step until $\perp$ is reached and construct properties along the way. Notice that in order to generate the generalization refinements, subsumption is not required, and thus the process of generalizing a term until $\perp$ is reached is not computationally expensive[3]. Notice that computing the shortest path from ψ to $\perp$ might be computationally expensive, but for our similarity purposes, it is enough with finding one path (not necessarily the shortest).

Each refinement will generate a property. The generalization operator $\gamma(\psi)$ manipulates a set of variables in a feature term in order to construct the generalizations. For instance, it might "change the sort s of a variable X to a more general sort s'", or "remove an element X from the feature f of another variable Y". In order to generate the pattern that corresponds to a property. It is necessary to obtain the minimum set V of variables that constitute a path from the root of a feature term to all the variables that the generalization operator manipulated. For example, in the example shown in Figure 5, a generalization step might take the value *engine* of the feature *cshape* of variable $X2$ and generalize it by changing it to Y : *shape*. For simplicity, *engine* did not have any variable name associated with it in the figure, but let us assume that its variable name is X_4 Notice that the set of variables manipulated are $\{X_2, X_4\}$. Since the root variable is X_1, the minimum set is $V = \{X_1, X_2, X_4\}$.

Once V has been computed, we have to compute which is the minimum set of features that each one of the variables in V require. For instance, X_1 requires *cars* (since it's the only way to reach X_2), and X_2 requires *cshape*. These variables and features will be the ones appearing in the pattern associated with the property.

4.3 Property-based Similarity Definition

Given a set of properties P (that can be generate by extracting them from all the cases in the case base and from the problem at hand), the first step is to compute a weight w_i for each property $p_i \in P$. Since each property divides the set of cases in two subsets, those which satisfy them and those which don't, a simple measure such as Quinlan's *Information Gain* [20] can be used to compute feature weights, where the weight of each feature is directly the normalized information gain (which is the method used in our experiments to compute weights). Let us define as $P(\psi)$ the set of properties that a particular term ψ satisfies, the similarity between two terms can be computed as:

$$S_\pi(\psi_1, \psi_2) = \frac{\sum_{p_i \in P(\psi_1) \cap P(\psi_2)} w_i}{\sum_{p_i \in P(\psi_1) \cup P(\psi_2)} w_i}$$

[3] One consideration has to be made when using feature terms: It is possible to construct infinite generalization chains when terms have cycles. However, by carefully selecting which generalizations to generate (basically, forbidding generalizations that increase the number of variables in a term, which are never necessary to reach $\perp$), it can be proved that this problem can be completely avoided.

In other words, it is the sum of the weights of those properties shared by the two terms, divided by the sum of the weights of all the properties that at least one of the term satisfies. Notice, moreover, that even if the definition of the measure involves subsumption, it is actually subsumption between a property and a full term, and subsumption when one of the terms is a property is not an expensive operation. Section 5 shows comparison in execution time between both measures, showing that the property-based measure is very efficient.

One of the main advantages of having a list of properties is that a weight can be assigned for each property. Thus, once we have two terms that we want to compare and we have extracted a set of properties, we can use information theoretical measures such as *Information Gain* [20], or the *RLDM distance* [9] to automatically assign a weight to each property.

Another interesting fact about properties, is that, under certain assumptions, if we compute the *unification* of all the properties that define a term, we obtain the original term[4]. In the same way, if we compute the unification of all the shared properties among a set of terms, we obtain the anti-unification of that set of terms. For that reason, if there is a single path in the refinement lattice from $\perp$ to the anti-unification, S_λ should provide the exact same results as S_π (if uniform weights are used for the properties).

Another advantage of the property-based similarity is that its computational requirements are lower, as we will see in our experimental results section. For data sets with complex cases, computation of the anti-unification of two terms might be prohibitive, however, properties can still be extracted. The only downside of the property-based similarity, is that the anti-unification between two terms is not explicitly computed. However, it can be computed by unifying all the shared properties (at an additional computational cost). A formal evaluation of the computational complexity of both similarity measures is subject to our future work. An explicit anti-unification can be used for explanation purposes [18], as well as for adaptation purposes (since it makes explicit the similarities between a problem and the retrieved case) [7].

Finally, notice that since each property is a pattern, any other pattern generation method can be used. For instance, any relational inductive learning method that could learn descriptions that distinguish among cases in the case base could be used to generate additional patterns.

5 Experimental Results

In order to evaluate our similarity measures, we used three different data sets: *sponges*, *trains*, and *kinship*. Trains is the data set shown in Figure 1, as presented by Michalski [15]. Kinship is a small but complex relational data set consisting of two families, each one with 12 members (thus 24 persons in total), proposed originally by [11], and used to evaluate several relational learning algorithms. The goal is to learn family relations. In our experiments the target relation to

[4] This only happens when there are no sets in the term.

	S_λ		S_π		SHAUD		RIBL	
	1-NN	*3-NN*	*1-NN*	*3-NN*	*1-NN*	*3-NN*	*1-NN*	*3-NN*
Sponges-280	95.00	94.29	**96.43**	**96.43**	95.71	95.00	91.67	91.67
Sponges-503	89.66	88.27	**92.25**	90.46	88.27	87.08	88.93	86.43
Trains-10	50.00	60.00	60.00	**70.00**	40.00	30.00	50.00	**70.00**
Kinship-24	**100.00**	91.67	**100.00**	75.00	-	-	83.33	83.33

Table 1. Classification accuracy in percentage measured using a leave one out method for different similarity measures.

learn was "uncle". The representation is purely relational, and each family is a graph (there are 4 positive examples and 20 negative examples). Finally, the sponges data set is a relational data set composed of 503 sponges belonging to 8 different solution classes. For the sponges data set, we report results both using the complete data set as well as using a subset of it (consisting of 280 sponges and 3 solution classes). We used the trains and uncle data sets as examples of data sets that are highly relational and where the value of features is not as important as the structure of the terms, and the sponges data set is a complex relational data set where both structure and feature values are important.

Table 1 shows the classification accuracy for several similarity measures in the data sets used for our evaluation. We report results for the two similarity measures presented in this paper, as well as two other relational similarity metrics for comparison purposes. For each similarity metric we measured classification accuracy using both a nearest neighbor as well as a 3-nearest neighbor by means of a leave-one-out method. We used SHAUD [2] and RIBL [10] (explained in detail in the next section) to compare our measures. SHAUD is a relational similarity metric defined for feature terms that has been shown to obtain very good results in complex relational data sets, and RIBL is a well known similarity measure for first order logic (FOL). RIBL requires examples to be represented in FOL and not as feature terms, but feature terms can be actually converted to FOL predicates without losing information. We used such conversion to evaluate RIBL. Moreover, RIBL and SHAUD require to know the ranges of each numeric feature before hand in order to compute similarity. We used the minimum and maximum values observed in the data set to define such ranges. Finally, RIBL requires a maximum depth parameter which was set to 10 in our experiments (large enough, since the deepest of the data sets is the Kinship data set where depth 5 is enough to capture each example). Finally, SHAUD only works for acyclic graphs, and thus could not be applied to the Kinship data set.

The first thing that we can observe in Table 1 is that the property-based similarity, S_π achieves the highest classification accuracy in all data sets. In the Kinship data set, the only important thing is the structure. SHAUD cannot handle it since cases are cyclic graphs, and RIBL concludes that all cases have similarity 0, since they have no values in any feature (there are no numerical or symbolic values in any of the terms in Kinhip, only a graph relating each member of the family to each other). Notice that RIBL achieves an accuracy of 83.33% only because it always predicts "negative", and there are only 4 positive examples out of 24. Both S_λ and S_π are able to capture the structure of the cases, and achieve an accuracy of 100.00%. Trains is an apparently simple but complicated data set, since there are lots of features in each train, but only two are key to determine the class. S_λ does not compute weights for any of the differences it finds, so it cannot distinguish from differences that matter from the ones that do not matter. S_π and RIBL perform the best in this data set.

Finally, in the sponges data set S_π achieves the best results. SHAUD and S_λ achieve also good results but not as good, and finally RIBL gets the lowest accuracy. The problem for RIBL is that it does not exploit completely the information in the sort hierarchy, and that is important in this data set. Moreover, we would like to remark that RIBL can accept weights in both predicates and attributes, but there is no simple way to compute them directly (like with the properties in S_π), and thus we used uniform weights. Thus, the results reported here for RIBL might be suboptimal, although weights won't be able to help at all in the uncle data set. Finally, we would like to note that the accuracy achieved by S_λ is the highest reported to date in the sponges data set.

In terms of execution time, S_λ takes 13.98 seconds per problem in the Sponges 503 data set, S_π takes 1.54 (including the time to learn the weights), SHAUD takes 4.09 seconds per problem, and RIBL is the fastest with 1.05 seconds per problem. Those times correspond to computing 502 similarities (since there are 503 examples in that data set). Time differences are similar for other data sets. We see that S_λ and SHAUD are the slowest since they require computing the anti-unification, and RIBL is the fastest. S_π is also very fast, since extracting properties does not rely on anti-unification.

We can conclude that S_π is the most balanced similarity overall, achieving the highest classification accuracy in most data sets while being computationally efficient. Moreover, both S_λ and S_π are conceptually very simple, and it is easy to understand what is being measured, whereas in more complex measures such as SHAUD and RIBL, it is hard to conceptually understand what exactly is being measured. Comparing S_λ to S_π, S_λ has the advantage of computing an explicit symbolic similarity and of being conceptually very simple, however it is computationally expensive. S_π on the other hand is computationally less expensive and it is more accurate but has the disadvantages of not computing an explicit symbolic similarity term and of being conceptually more complicated (it requires the property generation step).

6 Related Work

Hutchinson [12] presented a distance metric based on the anti-unification of two terms. Given the anti-unification of two terms, Hutchinson measures the size of a variable substitution required to unify the anti-unification with each of the terms. The distance becomes the addition of the size of the two substitution required (for each one of the two terms we are comparing). This measure is very related to our anti-unification-based measure, but it fails to take into account some of the information, since it only counts the number of variable substitutions. For example, substituting a term *number* by *integer* or substituting it by the number

45, will count as a single substitution in Hutchinson's formalism, however, in our measure changing *number* to *integer* counts as one refinement, where as *number* to 45 requires two refinements. Thus, our measure is a more fine-grained one than the one presented by Hutchinson.

Borgida, Walsh and Hirsh [6] differentiate three generic classes of similarity measures for description logics. Our two measures fall into two of their categories. S_λ is what they call an *information-content based model*, and S_π is a *feature-based model*. They already point out that the main problem of feature-based models is identifying what constitutes a feature (a property). In this paper we have given a particular answer to that question based on refinement operators.

RIBL (Relational Instance-Based Learning) was presented by Emde and Wettschereck [10] as an approach to apply lazy learning techniques based on the nearest neighbor algorithm using first-order logic as the representation formalism. The similarity measure of RIBL uses the intuition that the similarity among two terms is the average of the similarity of the value of their features (calling this function recursively if the values are terms, thus being better suited for acyclic graphs). Moreover, they define special similarity measures if the values are numeric or symbolic. Compared to our anti-unification-based measures, RIBL has the strong point of handling naturally numerical values. However, our similarity measures are more general in the sense that we do not make any assumption about the representation language being used, but only rely on the existence of a subsumption operation and refinement operators. The fact that terms are trees, graphs or lists is irrelevant to our similarity measures. Moreover, because of the recursive way that RIBL computes similarity, values deep in the tree are bound to have less importance in the computation, where as in our property-based measure, it is left to the weight computation heuristic to decide which properties are important and which ones are not. An earlier similarity measure related to RIBL was that of Bisson [5].

An extension of the RIBL similarity measure was presented by Horváth et al [22] in order to let RIBL handle lists and terms. The extension consists of a specialized routine that uses an edit-distance to compute similarities among lists and terms added to the basic similarity measure of RIBL. The downside of the similarity metric of RIBL (including this improvement) is that specialized measures have to be defined for different type of data, where as our similarity measures can handle any kind of data uniformly.

Another approach to similarity among structured terms is that of Bergmann and Stahl [4]. They present a similarity metric specific for object oriented representations based on the concepts of *intra-class similarity* (measuring similarity among all the common features of two objects) and *inter-class similarity* (providing a maximum similarity given to object classes). The similarity is defined in a recursive way, thus limiting the approach to tree representations.

SHAUD, presented by Armengol and Plaza [2] is another similarity metric related to RIBL but designed for feature terms. SHAUD also assumes that the terms are acyclic graphs, and in the same way as RIBL and Bergmann and Stalh's it can handle numerical values in a natural way by using specialized similarity measures for different data types. Another benefit of our similarity measures with respect to RIBL and SHAUD is that it can handle comparisons among generalizations (i.e. terms that have unbound variables). Hutchinson distance can handle generalizations by using a language change representation trick mapping variables to constants, and Bergmann and Stahl define some special cases to handle this situation. Notice that this is because both similarity measures presented in this paper do not make any assumptions about the data other than assuming a subsumption relation and refinement operators.

Concerning the applicability of our measures to other formalisms, other authors have proposed refinement operators for different subsets of first-order logics or other description logics, such as Laag and Nienhuys-Cheng [14] or Shapiro [21]. Thus, making our similarity measures applicable to those representation formalisms. Moreover, feature terms can represent naturally object oriented data, making our approach applicable to those representations.

Finally, extracting properties of a term is related to the *propositionalization* operation that can map relational terms to flat feature vectors, see [13] for an overview.

7 Conclusions

In this paper we have presented two similarity measures for relational cases that can be used for case-based reasoning systems with complex case representations. Both similarity measures have been presented and evaluated for the feature-term representation formalism, but can be easily applied to other representation formalisms by defining an appropriate subsumption relation and refinement operators. Moreover, we have evaluated our measures with several relational data-sets showing very good results.

Compared to other similarity measures, our measures have the advantage of being independent on the representational formalism of the cases (they can work with flat feature vectors, trees, graphs, or any other if adequate refinement operators are available). The down side of the measures presented is that, due to their generality, might be computationally more expensive than other ad-hoc similarity measures, and that due to their symbolic nature, they cannot naturally handle proper comparisons among real numbers.

As part of our future work, we plan to formally evaluate the computational complexity of the measures and study ways to incorporate natural comparisons for real-number valued data and evaluate the similarity for other representation formalisms. Other interesting lines of future work are the combination of inductive learning techniques for generating more informative patterns for a property-based similarity, and the use of the symbolic similarity and dissimilarity terms that can be computed by unifying the shared and not shared properties among two cases for different purposes such as explanation generation and adaptation.

Acknowledgements Support for this work came from the project MID-CBR TIN2006-15140-C03-01.

References

[1] Josep Lluís Arcos. *The Noos representation language*. PhD thesis, Universitat Politècnica de Catalunya, 1997.

[2] Eva Armengol and Enric Plaza. Relational case-based reasoning for carcinogenic activity prediction. *Artif. Intell. Rev.*, 20(1-2):121–141, 2003.

[3] H. At-Kaci and A. Podelski. Towards a meaning of life. Technical Report 11, Digital Research Laboratory, 1992.

[4] R. Bergmann and A. Stahl. Similarity measures for object-oriented case representations. In *Proc. European Workshop on Case-Based Reasoning, EWCBR-98*, Lecture Notes in Artificial Intelligence, pages 8–13. Springer Verlag, 1998.

[5] Gilles Bisson. Learing in fol with a similarity measure. In *Proceedings of AAAI 1992*, pages 82–87, 1992.

[6] Alexander Borgida, Thomas Walsh, and Haym Hirsh. Towards measuring similarity in description logics. In Ian Horrocks, Ulrike Sattler, and Frank Wolter, editors, *Proceedings of the 2005 International Workshop on Description Logics (DL2005), July 26-28, 2005, Edinburgh, Scotland, UK*, volume 147 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.

[7] K Börner. Structural similarity as a guidance in case-based design. In *Topics in Case-Based Reasoning: EWCBR'94*, pages 197–208, 1994.

[8] Bob Carpenter. Typed feature structures: an extension of first-order terms. In V. Saraswat and K. Ueda, editors, *Proceedings of the International Symposium on Logic Programming*, pages 187–201, San Diego, 1991.

[9] R. López De Mántaras. A distance-based attribute selection measure for decision tree induction. *Mach. Learn.*, 6(1):81–92, 1991.

[10] W. Emde and D. Wettschereck. Relational instance based learning. In Lorenza Saitta, editor, *Machine Learning - Proceedings 13th International Conference on Machine Learning*, pages 122 – 130. Morgan Kaufmann Publishers, 1996.

[11] G.E. Hinton. Learning distributed representations of concepts. In *Proceedings of CogSci*, 1986.

[12] Alan Hutchinson. Metrics on terms and clauses. In *ECML '97: Proceedings of the 9th European Conference on Machine Learning*, pages 138–145, London, UK, 1997. Springer-Verlag.

[13] Stefan Kramer, Nada Lavrač, and Peter Flach. Propositionalization approaches to relational data mining. pages 262–286, 2000.

[14] P.R.J. van der Laag and S.-H. Nienhuys-Cheng. Subsumption and refinement in model inference. Technical report, 1992.

[15] J. Larson and R. S. Michalski. Inductive inference of vl decision rules. *SIGART Bull.*, (63):38–44, 1977.

[16] Nada Lavrač and Sašo Džeroski. *Inductive Logic Programming. Techniques and Applications*. Ellis Horwood, 1994.

[17] Tom Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, 1982.

[18] Enric Plaza, Eva Armengol, and Santiago Ontañón. The explanatory power of symbolic similarity in case-based reasoning. *Artif. Intell. Rev.*, 24(2):145–161, 2005.

[19] Gordon D Plotkin. A note on inductive generalization. In *Machine Intelligence*, number 5. 1970.

[20] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[21] E. Y. Shapiro. Inductive inference of theories from facts. Technical Report 624, Department of Computer Science, Yale University, 1981.

[22] Horváth T., Wrobel S., and Bohnebeck U. Relational instance-based learning with lists and terms. *Machine Learning*, 43(1-2):53–80, 2001.