Machine Learning Journal manuscript No.
(will be inserted by the editor)

Similarity Measures over Refinement Graphs

Santiago Ontañón · Enric Plaza

Received: date / Accepted: date

Abstract Similarity assessment plays a key role in lazy learning methods such as k-nearest neighbor or case-based reasoning. In this paper we will show how refinement graphs, that were originally introduced for inductive learning, can be employed to assess and reason about similarity. We will define and analyze two similarity measures, S_λ and S_π, based on refinement graphs. The *anti-unification-based similarity*, S_λ, assesses similarity by finding the anti-unification of two instances, which is a description capturing all the information common to these two instances. The *property-based similarity*, S_π, is based on a process of disintegrating the instances into a set of *properties*, and then analyzing these property sets. Moreover these similarity measures are applicable to any representation language for which a refinement graph that satisfies the requirements we identify can be defined. Specifically, we present a refinement graph for feature terms, in which several languages of increasing expressiveness can be defined. The similarity measures are empirically evaluated on relational data sets belonging to languages of different expressiveness.

Keywords similarity measures · refinement graphs · case-based reasoning · feature terms · lazy learning

1 Introduction

Similarity assessment plays a key role in lazy learning methods such as k-nearest neighbor [14] or case-based reasoning [1], where new problems are solved typically by selecting, adapting or interpolating the solutions of the most similar training instances to the problem at hand. Similarity is also relevant for machine learning and artificial intelligence in general, since it serves as an organizing principle by which individuals classify objects, form concepts and make generalizations [43]. Similarity assessment has been widely studied for attribute-value representations. However, there has been less

Santiago Ontañón · Enric Plaza
IIIA, Artificial Intelligence Research Institute
CSIC, Spanish Council for Scientific Research
Campus UAB, 08193 Bellaterra, Catalonia (Spain),
E-mail: {santi,enric}@iiia.csic.es

activity on defining similarity for structured representations such as feature terms [3, 13,34] or description logics [9].

Structured machine learning [16] has received an increased amount of interest in the recent years for several reasons like allowing to handle complex data in a natural way (as illustrated by the success of these techniques in biomedical fields), or sophisticated forms of inference. Most recent work on similarity assessment for structured representations follows the idea of "hierarchical aggregation", in which to compute the similarity between two instances, each instance is seen as an object with a set of features and the similarity is computed as an aggregation of the similarity of the values in the features. If features have structured values, then this procedure is repeated recursively [10, 7, 11, 17, 23]. A few exceptions to that exist, such as the work of Amato et Al. [15] where they measure concept similarity as a function of the intersection of their interpretations, or the work on similarity for chemical domains where the focus is on finding common substructures among molecules [39]. However, hierarchical aggregation tends to deem those features that are closer to the root of an instance as more important than those that are deeper into the structure.

In this paper we will introduce similarity measures that do not follow the idea of hierarchical aggregation. In order to do this we will turn to existing formalizations for structured representations, which correspond to different subsets of first-order logic. Three widely used ones are Horn clauses used in inductive logic programming (ILP) [28], description logics [9] used mainly by the semantic web community, or feature terms [3, 13] proposed as a formalization of frame-based representations. In this paper we will borrow the notions of *refinement graph* and *refinement operator* from ILP and use them to define similarity measures. In particular, we will present refinement operators and a refinement graph for feature terms and use them to define similarity measures. However, thanks to the idea of the refinement graph the similarity measures introduced in this paper are, in principle, applicable to any other formalism given a refinement graph that satisfies several requirements that we will identify (a preliminary validation of the ideas presented in this paper to description logics can be found in our previous work [41]).

Specifically, we present and analyze two similarity measures, S_λ and S_π. The *anti-unification-based similarity*, S_λ, finds the anti-unification (least general generalization) of two instances, which is a description capturing all the information common to these two instances. S_λ then estimates similarity as a ratio between the common information and the total amount of information in the two instances. The second measure, the *property-based similarity* S_π, is based on a process of disintegrating the instances into a set of *properties*, and then similarity between two instances is assessed by analyzing their property sets.

Moreover, we will also show that S_π is an approximation of S_λ but with lower computational cost. In order to evaluate S_λ and S_π, we use them in nearest-neighbor algorithms upon a variety of propositional and structured data sets and compare them to other relational similarity measures. This paper extends and formalizes the core ideas informally presented in an early publication [33], defining refinement graphs for several representation languages of increasing expressiveness, generalizing the algorithms for these languages, and analyzing the behavior of S_λ and S_π within them.

The paper is organized as follows. Section 2 introduces the feature term formalism used in the rest of the paper. Section 3 presents specific refinement operators for feature terms which allow the construction of a refinement graph and several languages of feature terms of increasing expressiveness. Sections 4 and 5 define the anti-

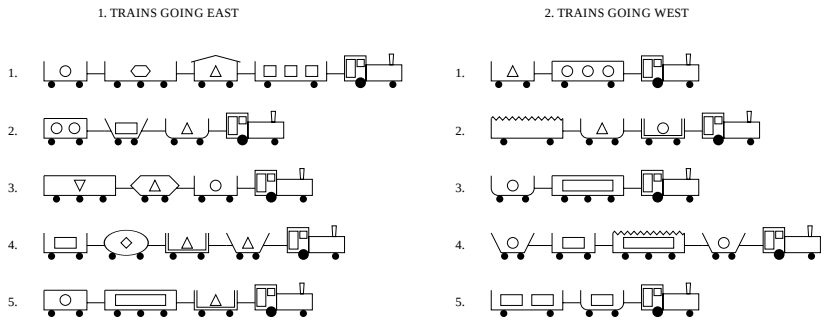


Fig. 1 Trains data set as introduced by Michalski [27].

unification-based and the property-based similarity respectively, while Section 6 introduces a weighted version of the property-based similarity. Then, Section 7 presents an empirical evaluation of these measures. Section 8 discusses related work on relational similarity measures, and Section 9 discusses the contributions of this paper and outlines future lines of research.

2 An Overview of Feature Terms

Feature terms [3,13] (also called Order-Sorted Feature logics, 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-oriented capabilities of declarative languages. Feature terms correspond to a different subset of first-order logics than description logics. However, as argued by Aït-Kaci [2], they have the same expressive power, only differing in their basic reasoning mechanisms. In this paper we use a specific formalization that may slightly differ from those of Aït-Kaci [3] or Carpenter [13].

As an example, consider the apparently simple *Trains* data set shown in Figure 1, introduced by Michalski [27]. The original task is to learn the rule that discriminates east-bound from west-bound trains. If we were to represent such data set using a feature vector, we would need to define features for each one of the cars of a train (size, shape, load, and number of wheels), and determine beforehand a maximum number of cars per train (since feature vector representations have a fixed number of features). Notice, however, that not all the trains have the same number of cars, and that, in principle, a train may have an unbounded number of cars. Thus, it is difficult 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 load each train is carrying.

For instance, Figure 2 represents the first west-bound train from Figure 1 using a graphical representation of a feature term. Such train (X_1) is composed of three cars (X_2, X_3 and X_4), the first of which (X_2) is a long engine car with 2 wheels, the second (X_3) is a long closed rectangle with 2 wheels holding three circles, and the last one (X_4) is a short open rectangle with 2 wheels holding one triangle.

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 sort symbols, which includes $\bot$ (also called "any") representing the most general sort and $\top$ (also called "none") representing the most specific sort. $\leq$ is an order relation

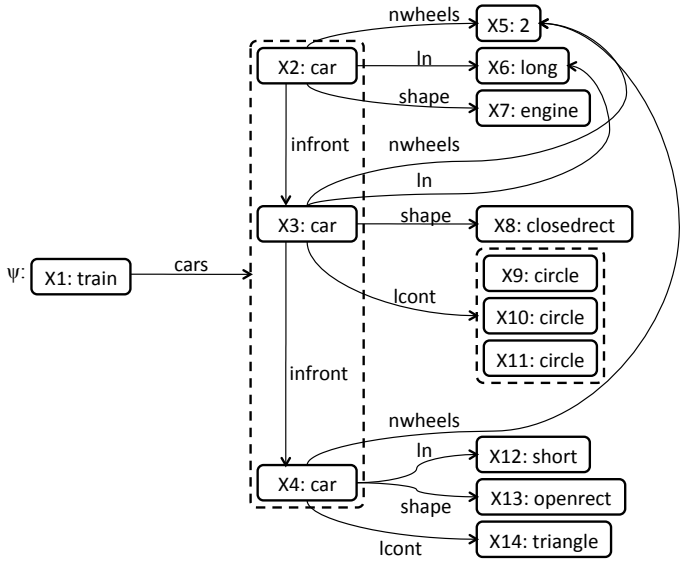


Fig. 2 A graphical depiction of a train represented using the feature terms representation formalism.

inducing a single inheritance hierarchy among the sorts in $\mathcal{S} \setminus \top$, and where $\bot \leq s \leq \top$ for any $s \in \mathcal{S}$. $\mathcal{F}$ is a set of feature symbols, and $\mathcal{V}$ is a set of variable names.

We 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 $\{x_1, ..., x_m\}$ (where elements are feature terms or variables). When the value Ψ_i of a feature f_i is an already defined variable, we say that there is a *variable equality*. Moreover, in our formalization of feature terms we impose the restriction that each element in a set must be different.

Using this term notation, Figure 3 represents the same train pictured previously in Figure 2, where we can see that the term is composed of 14 variables (corresponding to the 14 nodes in the graphical representation in Figure 2). The term contains two set-valued features (indicated by a curly bracket): in the feature *cars* of variable X_1, and in the feature *lcont* of variable X_3. Finally, we can also see that there are several variable equalities in this term: since the value of the feature *infront* of variable X_2 is the, already defined, variable X_3 (we note $X_2.infront \doteq X_3$), and also $X_3.infront \doteq X_4$. Additionally, the number of wheels in all the cars is the same, and the length of the first two cars is also the same.

In the remainder of this paper we will use $vars(\psi)$ to represent the set of variables in a given feature term ψ, $features(X)$ to denote the set of features defined for a variable X, and $sort(X)$ to denote the sort defined for a variable X. Finally, we will define the set $reachable(X)$ as the set of variables reachable from the features of X. For instance $reachable(X_3) = \{X_2, X_6, X_8, X_9, X_{10}, X_{11}, X_4, X_{12}, X_{13}, X_{14}\}$ and $reachable(X_{14}) = \emptyset$. Moreover, if for a particular term ψ there is a variable

$$\psi ::= X_1 : train \left[cars \doteq \left\{ \begin{array}{l} X_2 : car \left[\begin{array}{l} nwheels \doteq X_5 : 2 \\ ln \doteq X_6 : long \\ shape \doteq X_7 : engine \\ infront \doteq X_3 \end{array} \right] \\ X_3 : car \left[\begin{array}{l} nwheels \doteq X_5 \\ ln \doteq X_6 \\ shape \doteq X_8 : closedrect \\ lcont \doteq \left\{ \begin{array}{l} X_9 : circle \\ X_{10} : circle \\ X_{11} : circle \end{array} \right. \\ infront \doteq X_4 \end{array} \right] \\ X_4 : car \left[\begin{array}{l} nwheels \doteq X_5 \\ ln \doteq X_{12} : short \\ shape \doteq X_{13} : openrect \\ lcont \doteq X_{14} : triangle \end{array} \right] \end{array} \right. \right]$$

Fig. 3 The same train in Figure 2, represented in term notation.

$X \in vars(\psi)$ such that $X \in reachable(X)$, we will say that ψ has a *circular variable equality*, or simply a *cycle*.

In our formalization of feature terms, we add the notion of an *ontology*, which restricts the set of feature terms that can be formed. Thus, in addition to the signature Σ, we will define an ontology O as a set of restrictions of the form: $s_1.f \stackrel{\circ}{=} s_2$, meaning that any value of a defined feature $f \in \mathcal{F}$ of any variable of sort s_1 (or more specific) must be of sort s_2 (or more specific). Moreover, if no restriction appears for a particular feature f for a particular sort s, then f is not allowed in s. For instance, the ontology for the Trains example would be $O = \{train.cars \stackrel{\circ}{=} car, train.ncar \stackrel{\circ}{=} integer, car.infront \stackrel{\circ}{=} car, car.nwheels \stackrel{\circ}{=} integer, car.ln \stackrel{\circ}{=} car\text{-}size, car.shape \stackrel{\circ}{=} car\text{-}shape, car.lcont \stackrel{\circ}{=} load\}$. This notion of ontology is a particular case of the full OSF theories presented in [4], but sufficient for the purposes of this paper.

Feature terms can be represented in three different formalisms: they can be represented structurally (as graphs), as depicted in Figure 2, in term notation, as shown in Figure 3, or in clause notation, as shown in Figure 4. The three notations are equivalent, and we will use them indistinguishably in the remainder of this paper. The equivalence between the term form and the clause form is described by Aït-Kaci [4], so that given a term:

$$\psi ::= X : s[f_1 \doteq \psi_1, ..., f_n \doteq \psi_n]$$

the clause form can be obtained by a process called *dissolving* as:

$$\phi ::= X : s \ \& \ X.f_1 \doteq X_1 \ \& \ ... \ \& \ X.f_n \doteq X_n$$

where X_i represents the root variable of the term ψ_i. In the case where there is any set-valued feature $X.f \doteq \{X_1 ... X_n\}$, it can be dissolved as $X.f \doteq X_1 \ \& \ ... \ \& \ X.f \doteq X_n$. For instance, in Figure 3, the value of the feature *cars* of variable X_1 is a set of three elements (X_2, X_3, and X_4), and thus it is dissolved as $X_1.cars \doteq X_2 \ \& \ X_1.cars \doteq X_3 \ \& \ X_1.cars \doteq X_4$ (as shown in Figure 4).

The basic operation between feature terms is *subsumption*: we will use $\psi_1 \sqsubseteq \psi_2$ to express that a term ψ_1 subsumes another term ψ_2 – that is to say ψ_1 is more general (or

$$\begin{aligned}\phi ::= & X_1 : train \ \& \ X_1.cars \doteq X_2 \ \& \ X_1.cars \doteq X_3 \ \& \ X_1.cars \doteq X_4 \ \& \\ & X_2 : car \ \& \ X_2.nwheels \doteq X_5 \ \& \ X_2.ln \doteq X_6 \ \& \ X_2.shape \doteq X_7 \ \& \\ & X_2.infront \doteq X_3 \ \& \ X_5 : 2 \ \& \ X_6 : long \ \& \ X_7 : engine \ \& \ X_3 : car \ \& \\ & X_3.nwheels \doteq X_5 \ \& \ X_3.ln \doteq X_6 \ \& \ X_2.shape \doteq X_8 \ \& \ X_3.lcont \doteq X_9 \ \& \\ & X_3.lcont \doteq X_{10} \ \& \ X_3.lcont \doteq X_{11} \ \& \ X_3.infront \doteq X_4 \ \& \\ & X_8 : closedrect \ \& \ X_9 : circle \ \& \ X_{10} : circle \ \& \ X_{11} : circle \ \& \\ & X_4 : car \ \& \ X_4.nwheels \doteq X_5 \ \& \ X_4.ln \doteq X_{12} \ \& \ X_4.shape \doteq X_{13} \ \& \\ & X_4.lcont \doteq X_{14} \ \& \ X_{12} : short \ \& \ X_{13} : openrect \ \& \ X_{14} : triangle\end{aligned}$$

Fig. 4 Representation in clause notation of the train shown in Figure 2.

equal) than ψ_2[1]. Another interpretation of subsumption is that of an "informational content" order: $\psi_1 \sqsubseteq \psi_2$ means that all the information in ψ_1 (all that is true for ψ_1) is also contained in ψ_2 (is also true for ψ_2). In our work, we use the definition of subsumption introduced in [6] which has a slightly different definition than the traditional *θ-subsumption*. Specifically, the difference is that we introduce the constraint that all the elements in a set have to be different. See Appendix A for a formal definition.

The subsumption relation induces a partial order in the set of feature terms, i.e. the pair $\langle \mathcal{L}, \sqsubseteq \rangle$ is a *poset* for a given set of terms $\mathcal{L}$ (called a language); additionally, $\mathcal{L}$ contains the infimum element $\bot$ (or "any"), and the supremum element $\top$ (or "none") with respect to the subsumption order. We will work with several languages $\mathcal{L}$ of increasing expressiveness, some of which satisfying that $\langle \mathcal{L}, \sqsubseteq \rangle$ is a lattice[2]. In the remainder of this paper we will use $G(\psi) = \{\psi' \in \mathcal{L} | \psi' \sqsubseteq \psi\}$ to denote the set of *subsumers* of ψ in a given language $\mathcal{L}$, i.e., all the terms which subsume ψ (including $\bot$ and ψ itself).

Given the subsumption relation, for any two terms ψ_1 and ψ_2 we can define their *anti-unification* ($\psi_1 \sqcap \psi_2$) as their *least general generalization* [36]:

Definition 1 (Anti-unification) The *anti-unification* of two terms ψ_1 and ψ_2, noted as $\psi_1 \sqcap \psi_2$, is the most specific term that subsumes both:

$$\psi_1 \sqcap \psi_2 = \psi : (\psi \sqsubseteq \psi_1 \ \wedge \ \psi \sqsubseteq \psi_2) \wedge (\nexists \psi' \sqsupset \psi : \ \psi' \sqsubseteq \psi_1 \ \wedge \ \psi' \sqsubseteq \psi_2)$$

Anti-unification is relevant for defining similarity measures, since it contains *all* the information that is common to both ψ_1 and ψ_2. If two terms have nothing in common, then $\psi_1 \sqcap \psi_2 = \bot$. Thus, anti-unification encapsulates in a single description *all* that is shared by two given terms. Additionally, we can generalize the idea of anti-unification to a set of terms, noted by $\bigsqcap(\{\psi_1, ..., \psi_n\})$, representing the most specific term that subsumes all the terms in $\{\psi_1, ..., \psi_n\}$. Appendix B presents an algorithm to compute one anti-unification of a given set of terms (the algorithm is not specific to feature terms, but can be applied to any representation formalism for which an adequate refinement operator can be defined). A complementary operation to the anti-unification is that of *unification*, or *most general specialization*:

[1] Notice that in the description logics notation, subsumption is written in the reverse order since it is seen as "set inclusion" of their interpretations. In machine learning terms, $A \sqsubseteq B$ means that A is more general than B, while in description logics it has the opposite meaning.

[2] Albeit every partial order with supremum and infimum is trivially a lattice, we will reserve the name of *lattice* for those partial orders $\langle \mathcal{L}, \sqsubseteq \rangle$ where we will define a unification and anti-unification that correspond to the meet and join operations of such lattice.

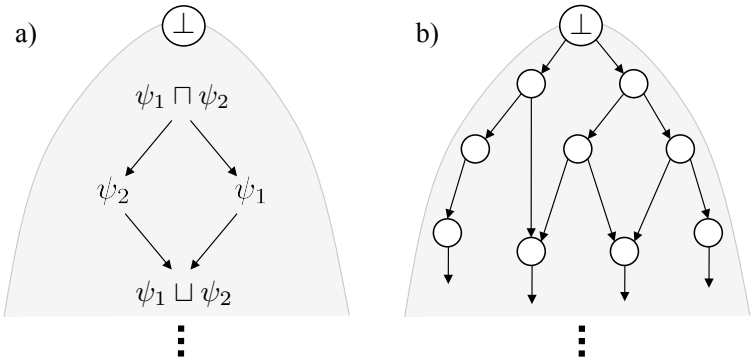


Fig. 5 a) Two terms ψ_1 and ψ_2 and with their unification $\psi_1 \sqcup \psi_2$ and anti-unification, $\psi_1 \sqcap \psi_2$; b) a *refinement graph*, where each node represents a term and the most general node is $\bot$.

Definition 2 (Unification) The *unification* of two terms ψ_1 and ψ_2, noted as $\psi_1 \sqcup \psi_2$, is the most general term that is subsumed by both:

$$\psi_1 \sqcup \psi_2 = \psi : (\psi_1 \sqsubseteq \psi \ \wedge \ \psi_2 \sqsubseteq \psi) \wedge (\nexists \psi' \sqsubset \psi : \ \psi_1 \sqsubseteq \psi' \ \wedge \ \psi_2 \sqsubseteq \psi')$$

When two terms have contradictory information then they have no unifier – which is equivalent to say that their unifier is "none": $\psi_1 \sqcup \psi_2 = \top$. As before, we can generalize unification to a set of terms, noted by $\bigsqcup(\{\psi_1, ..., \psi_n\})$, representing the most general term subsumed by all terms in $\{\psi_1, ..., \psi_n\}$.

Figure 5.a graphically illustrates both concepts. Notice that both unification and anti-unification are operations over the subsumption graph: anti-unification corresponds to finding the most specific common "parent" (generalization), where as unification corresponds to finding the most general common "descendant" (specialization). Moreover, unification and anti-unification might be unique or not depending on the structure of the subsumption graph, as we will see later. To simplify the notation, in the remainder of this paper we will assume anti-unification and unification are unique when it does not make a difference.

3 Refinement Graphs for Feature Terms

From the poset $\langle \mathcal{L}, \sqsubseteq \rangle$ we can derive a *refinement graph* as the poset $\mathcal{R} = \langle \mathcal{L}, \prec \rangle$, where $\psi_1 \prec \psi_2$ represents that ψ_2 is a *specialization refinement* of ψ_1. Conversely, ψ_1 is a generalization refinement of ψ_2. Formally, the refinement relation used in this paper is defined as:

Definition 3 (Refinement) Two feature terms hold a refinement relation $\psi_1 \prec \psi_2$ iff

$$\psi_1 \sqsubseteq \psi_2 \ \wedge \ \nexists \psi' : \psi_1 \sqsubset \psi' \sqsubset \psi_2$$

That is to say, the relation $\psi_1 \prec \psi_2$ holds when specializing from ψ_1 to ψ_2 is a minimal specialization step: there is no intermediate term ψ' in $\langle \mathcal{L}, \sqsubseteq \rangle$ between ψ_1 and ψ_2. Figure 5.b shows an illustration of the refinement graph, where each node represents a term, and arrows represent refinements. Typically, such refinement graphs have been used in inductive learning and in pattern mining to define the hypothesis

(ρ_s) Sort Specialization:

$$\begin{bmatrix} s_1 < s_2 \wedge \nexists s : s_1 < s < s_2 \wedge \\ \forall X.f \doteq Y \in \phi, \exists s_2.f \stackrel{\circ}{=} s_3 \in O \wedge \\ sort(Y) \geq s_3 \end{bmatrix} \qquad \frac{\phi \;\&\; X : s_1}{\phi \;\&\; X : s_2}$$

(ρ_i) Variable Introduction Without Sets:

$$\begin{bmatrix} Y \in \mathcal{V} \;\wedge\; Y \notin vars(\phi) \wedge \\ s.f \stackrel{\circ}{=} s' \in O \wedge \\ f \notin features(X) \end{bmatrix} \qquad \frac{\phi \;\&\; X : s}{\phi \;\&\; X : s \;\&\; X.f \doteq Y \;\&\; Y : s'}$$

(ρ_v) Variable Introduction:

$$\begin{bmatrix} Y \in \mathcal{V} \;\wedge\; Y \notin vars(\phi) \wedge \\ s.f \stackrel{\circ}{=} s' \in O \end{bmatrix} \qquad \frac{\phi \;\&\; X : s}{\phi \;\&\; X : s \;\&\; X.f \doteq Y \;\&\; Y : s'}$$

(ρ_n) Non-Circular Variable Equality:

$$\begin{bmatrix} X, Y \in vars(\psi) \;\wedge\; X \neq Y \;\wedge \\ sort(X) \geq sort(Y) \;\vee\; sort(Y) \geq sort(X) \wedge \\ X \notin reachable(Y) \wedge \\ Y \notin reachable(X) \end{bmatrix} \frac{\phi}{\phi \;\&\; X = Y}$$

(ρ_e) Variable Equality:

$$\begin{bmatrix} X, Y \in vars(\psi) \;\wedge\; X \neq Y \;\wedge \\ sort(X) \geq sort(Y) \;\vee\; sort(Y) \geq sort(X) \end{bmatrix} \qquad \frac{\phi}{\phi \;\&\; X = Y}$$

Fig. 6 Specialization refinement operators, represented as rewriting rules.

space of inductive learning methods [32]. However, in this paper we will show how to use refinement graphs to define similarity measures.

A refinement graph is defined by a *refinement operator*, that can be either a *specialization* (or *downward*) refinement operator or a *generalization* (or *upward*) refinement operator. Specifically, a specialization refinement operator is defined as follows:

$$\rho(\psi) = \{\psi' \in \mathcal{L} | \psi \prec \psi'\}$$

Whereas a generalization refinement operator is defined as follows:

$$\gamma(\psi) = \{\psi' \in \mathcal{L} | \psi' \prec \psi\}$$

Terms are related by refinements paths, as follows:.

Definition 4 (Refinement Path) A finite sequence of terms $(\psi_1, ..., \psi_n)$ is a *refinement path* $\psi_1 \xrightarrow{\rho} \psi_n$ between two terms ψ_1 and ψ_n when for each $1 \leq i < n$, $\psi_{i+1} \in \rho(\psi_i)$. The same definition applies for the generalization refinement operator: $\psi_n \xrightarrow{\gamma} \psi_1$.

This allows us to define a language as follows.

Definition 5 (Language) A *language* $\mathcal{L}$ (for a refinement operator ρ) is the set of terms $\mathcal{L} = \{\psi | \bot \xrightarrow{\rho} \psi\} \cup \{\bot, \top\}$.

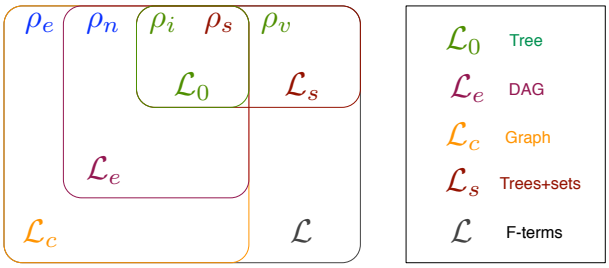


Fig. 7 Different sublanguages of feature terms generated by different subsets of refinement operators.

That is to say, a language is the set of all terms ψ that are reachable from $\bot$ with a refinement path using a particular specialization refinement operator ρ, plus $\bot$ and $\top$.

The refinement graph $\mathcal{R}$ might be more or less complex depending on the representation language being used. For instance, unification and anti-unification might be unique or not. Since the complexity of the refinement graph determines the computational cost of operations defined over it, it is useful to define several languages (different sets of terms $\mathcal{L}$) of feature terms with different expressive power, clearly delineating the areas in which, for instance, the unification is unique (and $\mathcal{R} = \langle \mathcal{L}, \prec \rangle$ is a lattice).

In order to define languages of different expressiveness we will specify several different refinement operators. The collection of specialization refinement operators for feature terms are outlined in Figure 6 as rewriting rules. A rewriting rule is composed of three parts: a top part, with the clause representation of a term, a lower part which represents the refinement of that term, and the applicability conditions of the rewrite rule (shown between square brackets in the left hand side of the definition).

Let us briefly review each operator:

- ρ_s generates refinements by substituting the sort s_1 of a variable in a term by a more specific sort s_2. Notice that the applicability condition ensures that s_1 is substituted only by a direct descendant sort s_2 (i.e. $\nexists s \in \mathcal{S} : s_1 < s < s_2$) while satisfying the restrictions in the ontology.
- ρ_i generates refinements by adding a feature and its value to a term that previously didn't have that feature.
- ρ_v generates refinements by adding a value to a feature: if the feature was undefined the operator will add the feature and its value or (if the feature was already defined) add a value to the a set of values of that feature.
- ρ_n generates refinements by adding a non-circular variable equality.
- ρ_e generates refinements by adding a variable equality (either circular or non circular).

Moreover, notice that all the refinement operators always generate terms that satisfy the given ontology O. Using these refinement operators we will define five languages of increasing expressiveness. Figure 7 shows the relationship between these languages, where set inclusion among languages means that $\mathcal{L} \subset \mathcal{L}'$. Let us briefly review them:

- $\mathcal{L}_0$ is defined by the refinement operators $\{\rho_s, \rho_i\}$[3] shown in Figure 6. $\mathcal{L}_0$ contains all the feature terms that do not have any set-valued feature or any variable equality.

[3] Notice that in Definition 5 we defined a language from a single refinement operator, and here were are defining languages form sets of refinement operators. In fact, a set of refinement

Unification and anti-unification are unique in $\mathcal{L}_0$. Using the graphical representation of feature terms, the terms in $\mathcal{L}_0$ are always trees.

- $\mathcal{L}_e$ is defined by the refinement operators $\{\rho_s, \rho_i, \rho_n\}$. $\mathcal{L}_e$ is a super set of $\mathcal{L}_0$ corresponding to all the terms that do not have any set-valued feature or any circular variable equality (non-circular variable equalities are allowed). Unification and anti-unification are also unique. Terms in $\mathcal{L}_e$ are always DAGs (directed acyclic graphs).
- $\mathcal{L}_c$ is defined by the refinement operators $\{\rho_s, \rho_i, \rho_e\}$. $\mathcal{L}_c$ is a super set of $\mathcal{L}_e$ which allows terms with circular variable equalities. Both unification and anti-unification are still unique. One difference of this language with respect to $\mathcal{L}_0$ and $\mathcal{L}_e$ is that some terms might have an infinite number of subsumers, i.e. $G(\psi)$ might be infinite for some $\psi \in \mathcal{L}_c$ (see Appendix C).
- $\mathcal{L}_s$ is defined by the refinement operators $\{\rho_s, \rho_v\}$. $\mathcal{L}_s$ is a super set of the base language $\mathcal{L}_0$ which allows set-valued features. This language has non-unique unification or anti-unification, but all terms have a finite number of subsumers.
- $\mathcal{L}$ is defined by the refinement operators $\{\rho_s, \rho_v, \rho_e\}$. $\mathcal{L}$ is a superset of all the previous languages and contains all feature terms as defined in Section 2. $\mathcal{L}$ has non-unique unification and anti-unification, and also some terms might have an infinite number of subsumers.

Notice that the three languages on left part of Figure 7 ($\mathcal{L}_0$,$\mathcal{L}_e$ and $\mathcal{L}_c$) form a lattice with respect the refinement order relation ($\prec$) and have unique unification and anti-unification operations. The other two languages ($\mathcal{L}_s$ and $\mathcal{L}$) are just posets with respect to refinement order relation ($\prec$) —since unification and anti-unification may be non-unique when set-valued features are allowed [37].

Refinement operators may be characterized according to three properties: *completeness*, *properness* and *local finiteness* [44]. Completeness states that there are no refinements of a term which are not generated by the operator, properness means that a term is not equivalent to any of its refinements, and local finiteness means that the number of refinements generated for any given term by the refinement operator is finite. The set of refinement operators $\{\rho_s, \rho_v, \rho_e\}$ for feature terms presented here is complete, finite and proper [4]. Notice that for other refinement graphs (such as the ones defined for description logics [29] or the ones defined by θ-subsumption [44]) complete, finite and proper refinement operators do not exist[5].

Figure 8 presents the generalization operators for feature terms. Notice that the generalization operators are the same regardless of the language of feature terms used. Nevertheless, depending of the language used, the operators will be complete or not. Concerning the generalization operator, it is not possible to define a complete and still locally finite operator when there are circular variable equalities (see Appendix C). However, for the purposes of this paper (similarity assessment among terms), it suffices with generalization operators that ensure that $\bot$ is reachable by generalizing any term. The operators shown in Figure 8 do ensure that $\bot$ is reached, but they are not complete. Specifically:

operators R is equivalent to a single refinement operator ρ which generates the union of all the refinements generated by the operators in R.

[4] The operators ρ_n and ρ_e shown in Figure 6 are a simplification that might generate some terms that are not refinements, but it's easy to filter those out using subsumption tests.

[5] The conditions identified by van der Laag and Nienhuys-Cheng [44] for proving the inexistence are not satisfied in our case, so the proof does not apply to our formalization.

(γ_s) Sort Generalization:

$$\left[\begin{array}{l} s_1 < s \ \wedge \ \nexists s_2 : \ s_1 < s_2 < s \ \wedge \\ \forall X.f \doteq Y \in \phi \exists s_1.f \stackrel{\circ}{=} s_3 \in O \ \wedge \\ sort(Y) \geq s_3 \end{array}\right] \quad \frac{\phi \ \& \ X : s}{\phi \ \& \ X : s_1}$$

(γ_v) Variable Elimination:

$$\left[\begin{array}{l} s.f \stackrel{\circ}{=} s' \in O \ \wedge \\ features(Y) = \emptyset \end{array}\right] \quad \frac{\phi \ \& \ X : s \ \& \ X.f \doteq Y \ \& \ Y : s'}{\phi \ \& \ X : s}$$

(γ_e) Variable Equality Elimination:

$$\left[\ Z_1 \notin vars(\phi)\ \right] \quad \frac{\phi \ \& \ X.f \doteq Z \ \& \ Y.f' \doteq Z}{\phi \ \& \ X.f \doteq Z \ \& \ Y.f' \doteq Z_1 \ \& \ Z_1 : sort(Z)}$$

(γ_r) Root Variable Equality Elimination:

$$\left[\begin{array}{l} Z_1 \notin vars(\phi) \ \wedge \\ root(\psi) = Z \end{array}\right] \quad \frac{\phi \ \& \ X.f \doteq Z}{\phi \ \& \ X.f \doteq Z_1 \ \& \ Z_1 : sort(Z)}$$

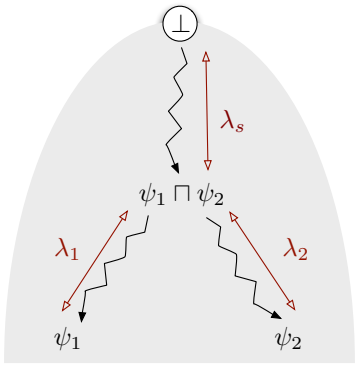
Fig. 8 Generalization operators for feature terms. Notice that this operators are not complete, but that they ensure reaching $\bot$ from any feature term in the language.

- γ_s generalizes terms by substituting the sort of one of the variables in the term by a more general sort.
- γ_v generalizes a term by removing the value of one of the features in one variable of the term. Notice that the operator only removes a variable Y if $features(Y) = \emptyset$, i.e. if Y has no defined features.
- γ_e and γ_r generalize a term by removing a variable equality. A circular variable equality can be removed in an infinite number of ways (see Appendix C), and this is the cause of the generalization operators not being complete, although they still ensure that $\bot$ can be reached from any term.

This section has defined refinement operators for feature terms. Nevertheless, the techniques presented in the remainder of this paper are applicable to any other representation formalism for which (1) a complete and locally finite specialization refinement operator can be defined, and (2) there is a generalization operator that ensures that the most general term ($\bot$) can be reached from any term through a finite number of steps.

4 Anti-Unification-based Similarity

The anti-unification of two feature terms $\psi_1 \sqcap \psi_2$ is commonly described as their least general generalization. But it is also a symbolic representation of that which is shared by ψ_1 and ψ_2 and *all that is shared*. For this reason, anti-unification has been used in case-based reasoning (CBR) as a form of *symbolic similitude* [34]. Although this symbolic similitude can be used for explanatory purposes in CBR, another important issue, that we want to address here, is how to quantitatively measure the similarity.



$$S_\lambda(\psi_1, \psi_2) = \frac{\lambda_s}{\lambda_s + \lambda_1 + \lambda_2}$$

Fig. 9 Illustration of the anti-unification based similarity for two feature terms ψ_1 and ψ_2

Algorithm $\Lambda(\psi_a, \psi_b, \rho)$
ForEach ($\psi \in \rho(c_t)$) Do
If ($\psi \sqsubseteq \psi_b \wedge \psi \not\sqsubseteq \psi_a$) Then Return $\Lambda(\psi, \psi_b, \rho) + 1$
EndForEach
Return 0
EndAlgorithm

Fig. 10 Given two terms ψ_a and ψ_b, such that $\psi_a \sqsubseteq \psi_b$, and a refinement operator ρ, the algorithm Λ returns the length of the refinement path from ψ_a to ψ_b in the refinement graph defined by the operator ρ.

That is to say, answering the question "How large is $\psi_1 \sqcap \psi_2$" might give us a measure that estimates how similar ψ_1 and ψ_2 are.

Our proposal is to estimate the *informational content* of $\psi_1 \sqcap \psi_2$, since the larger $\psi_1 \sqcap \psi_2$ is the more similar ψ_1 and ψ_2 are. The refinement graph gives us a direct way to estimate the informational content of any term ψ in a language $\mathcal{L}$: it is the length of the minimal path of refinements that leads from $\bot$ (the most general feature term) to ψ itself. In other words, the number of times that a refinement operator has to be applied, starting from $\bot$, to reach ψ. The intuition here is that every time a term is specialized, an extra piece of information is added to it. Using Definition 4 (of *refinement path*) we can characterize any term ψ by the path $\bot \xrightarrow{\rho} \psi$, i.e. the refinement path from $\bot$ to ψ; moreover, an estimation of the informational content of any term ψ is given by the length (λ) of that path: $\lambda_\psi = \lambda(\bot \xrightarrow{\rho} \psi)$.

Therfore, the length $\lambda(\bot \xrightarrow{\rho} \psi_1 \sqcap \psi_2)$ of the refinement path from $\bot$ to $\psi_1 \sqcap \psi_2$ estimates the informational content of that which is common to ψ_1 and ψ_2 (this length is called λ_s in Figure 9). In order to define a similarity measure we need to compare what is common to ψ_1 and ψ_2 with that which is not. The information in ψ_1 that is not common to both terms is exactly what is added by the path $\psi_1 \sqcap \psi_2 \xrightarrow{\rho} \psi_1$ (whose length is called λ_1 in Figure 9); and equivalently for ψ_2 with path $\psi_1 \sqcap \psi_2 \xrightarrow{\rho} \psi_2$ (whose length is called λ_2 in Figure 9).

Definition 6 (Anti-unification-based similarity) The anti-unification-based similarity S_λ between two terms ψ_1 and ψ_2 is:

$$S_\lambda(\psi_1, \psi_2) = \frac{\lambda(\bot \xrightarrow{\rho} \psi_1 \sqcap \psi_2)}{\lambda(\bot \xrightarrow{\rho} \psi_1 \sqcap \psi_2) + \lambda(\psi_1 \sqcap \psi_2 \xrightarrow{\rho} \psi_1) + \lambda(\psi_1 \sqcap \psi_2 \xrightarrow{\rho} \psi_2)}$$

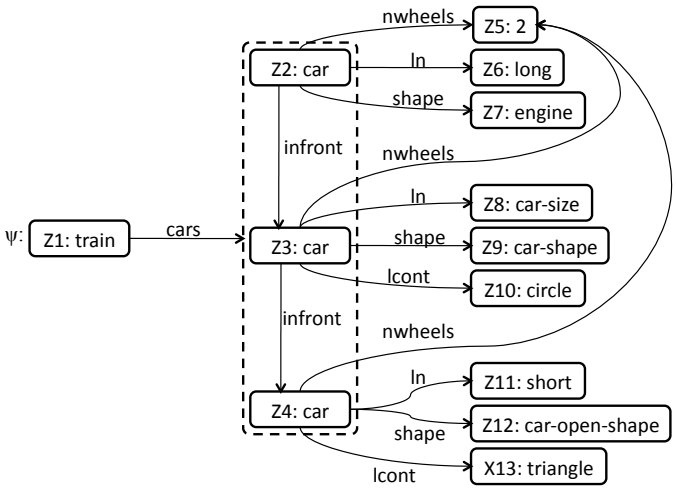


Fig. 11 Anti-unification of the two first west-bound trains.

The measure S_λ estimates the ratio between the amount of shared information and the total informational content (i.e. the shared plus the non-shared information). The numerator estimates the shared informational content as $\lambda_s = \lambda(\bot \xrightarrow{\rho} \psi_1 \sqcap \psi_2)$, while the denominator estimates the total amount of information of ψ_1 and ψ_2 together by adding the shared informational content λ_s with the informational content of ψ_1 that is not shared $\lambda_1 = \lambda(\psi_1 \sqcap \psi_2 \xrightarrow{\rho} \psi_1)$ and the informational content of ψ_2 that is not shared $\lambda_2 = \lambda(\psi_1 \sqcap \psi_2 \xrightarrow{\rho} \psi_2)$.

$S_\lambda(\psi_1, \psi_2)$ requires computing two things: the anti-unification $\psi_1 \sqcap \psi_2$, and the three lengths λ_s, λ_1, λ_2. Appendix B describes an anti-unification algorithm that starts with $\bot$ and proceeds by iteratively applying refinement until the anti-unification is reached, returning the anti-unifier $\psi_1 \sqcap \psi_2$ and the length of the refinement path λ_s.

The other two lengths (λ_1 and λ_2) are computed by the algorithm Λ shown in Figure 10: Λ takes two terms ψ_a and ψ_b (such that $\psi_a \sqsubseteq \psi_b$) and a refinement operator ρ, and returns the number of specialization refinements required to go from ψ_a to ψ_b. Therefore, the two lengths can be computed as $\lambda_1 = \Lambda(\psi_1 \sqcap \psi_2, \psi_1, \rho)$ and $\lambda_2 = \Lambda(\psi_1 \sqcap \psi_2, \psi_2, \rho)$ where ρ is the refinement operator for the language being used.

4.1 Exemplification

To illustrate the anti-unification-based similarity, let us walk through the process of computing the similarity between the first two west bound trains in Michalski's data set (Figure 1), which we will call ψ_6 and ψ_7. In order to represent this (apparently simple) data set, the full language $\mathcal{L}$ is required. The anti-unification of those two trains ($\psi_6 \sqcap \psi_7$) is shown in Figure 11, clearly capturing the key similarities among the two trains: both of them have 3 cars in common, the first car is the engine, which is long and has 2 wheels; both of them have a second car with 2 wheels and which contains at least one circle; and both of them have a third car, which is short, has two wheels, has at least one triangle, and has an open shape (one has an open rectangle shape and the other one has a U shape).

This anti-unification can be computed using the algorithm presented in Appendix B, that also yields the length of the refinement path from $\bot$ to $\psi_6 \sqcap \psi_7$, which in this case is $\lambda_s = 40$. Then, the algorithm Λ in Figure 10 counts the refinement path length from $\psi_6 \sqcap \psi_7$ to ψ_6, which is $\lambda_1 = 8$, and from $\psi_6 \sqcap \psi_7$ to ψ_7, which is $\lambda_2 = 19$. Notice that the length from $\psi_6 \sqcap \psi_7$ to the first westbound train is very small (only 8 refinements), while the length to the second westbound train is larger (19 refinements), since the second train has more non-shared content (4 cars, instead of 3).

Thus, the similarity between these two trains is: $S_\lambda(\psi_6, \psi_7) = \frac{40}{40+8+19} = 0.597$. The interpretation of S_λ is that the ratio 40/67 estimates that these two train descriptions share almost 60% of the total informational content. Moreover, the *symbolic similitude* (shown in Figure 11) is capable of conveying an explanation of that which is behind the numerical similarity value 0.597.

4.2 Discussion

The requirement for S_λ to be applied to a refinement graph is that the specialization refinement operators are locally finite and complete. Thus, the anti-unification-based similarity can be applied to any other representation formalisms such that a refinement graph can be defined with a subsumption relation that is compatible with the definition of specialization refinement operators satisfying this requirement.

There are two main issues concerning S_λ worth discussing. The first one is the computational cost of S_λ. Although computing the anti-unification of two terms requires (using the algorithm described in Appendix B) a linear number of subsumption tests in function of the size of the terms, testing for subsumption might have a different computational cost depending on the representation language used. In expressive languages, like $\mathcal{L}_s$ and $\mathcal{L}$, subsumption has a higher computational cost than in less expressive languages, such as $\mathcal{L}_0$, $\mathcal{L}_e$, or $\mathcal{L}_c$.

Moreover, in languages where anti-unification might not be unique ($\mathcal{L}_s$ and $\mathcal{L}$), ensuring that the maximum similarity value for similarity S_λ is found would involve computing all possible anti-unifications $\psi_1 \sqcap \psi_2$ and taking the one that maximizes Definition 6. Computing only one anti-unification (which is the approach taken in the experiments reported in Section 7) results only in an estimation of S_λ that is not ensured to be maximal. Thus, in domains where the instances are large structures and where the language used is very expressive, this trade-off (non-maximal but efficient) may be useful.

A second issue is that S_λ considers each refinement in the refinement graph as equally important; in general, assuming that all pieces of information generated from refinement operators have the same usefulness might not hold. Thus, being able to determine a different weight or importance degree for each of the refinements in the similarity computation is a significant issue that we address on Section 6.

Nevertheless, S_λ has two main advantages: first of all, S_λ is a conceptually intuitive similarity measure, and second, during the computation of S_λ a symbolic similitude term is also computed: the anti-unification term $\psi_1 \sqcap \psi_2$. As argued by Plaza [35], such term describes in what aspects two instances are similar, and can be used for explanation purposes in Case-based Reasoning systems and other systems estimating similarity among complex descriptions. Additionally the anti-unification term can also be used for case adaptation purposes in CBR (since it makes the similarities between a problem and the retrieved case explicit [12]).

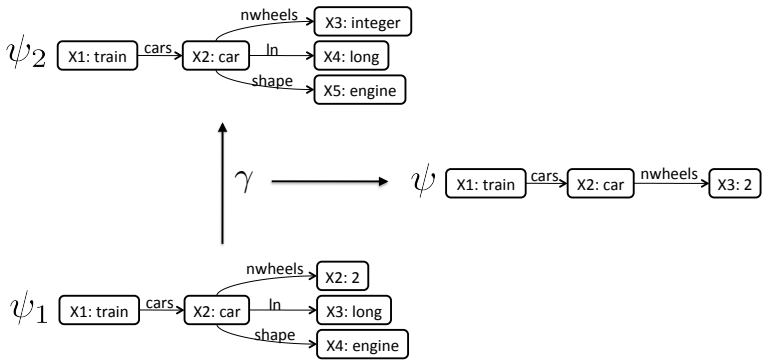


Fig. 12 A refinement operator γ that generalizes ψ_1 into ψ_2 by subtracting a piece of information ψ called the *remainder* of the refinement.

5 Property-based Similarity

This section introduces the *property-based similarity*, a new approach to assess similarity that is also based on the idea that every specialization refinement adds a piece of informational content to a description (and, conversely, every generalization refinement subtracts a piece of informational content from a description), but addresses the two issues of S_λ mentioned above. These pieces of information added or removed by a refinement operator are called *properties*, i.e. a property is some condition that some terms satisfy and some do not. For example, in the Trains domain introduced before, a property might be that "a train has 3 cars", and some trains might satisfy it while some others might not.

The core idea of the property-based similarity is to *disintegrate* a term into a collection of smaller terms, which we call properties, and then count how many of those properties they share. Furthermore, we will show it is feasible to *integrate* those properties again in order to reconstruct the original term, so no information is lost.

There are several issues that we have to address to define the property-based similarity. First we will define what constitutes a property; second, we will specify how to disintegrate a term into a collection of properties; and finally, we will define a measure of similarity based on the properties of two disintegrated terms.

5.1 Disintegrating a Term into Properties

Intuitively we would like a property to capture an individual piece of information contained in a term. For instance, in the Trains example used previously, the fact that a train has 4 cars, or that one of the cars is carrying a triangle, are examples of two properties. Generalization refinement operators generate refinements by subtracting pieces of information from terms, and thus making them more general. These pieces of information that are subtracted from terms when generalizing are properties, and will be represented also as feature terms. Dually, specialization refinement operators add pieces of information (properties) to terms. In order to formally define the properties of a term, we will first define the *remainder* of a generalization refinement operator.

Algorithm Disintegrate(ψ, γ)
  $D = \emptyset, t = 0, \psi_0 = \psi$
  While ($\psi_t \neq \bot$) Do
    $\psi_{t+1} \in \gamma(\psi_t)$
    $D = D \cup \{r(\psi_t, \psi_{t+1})\}$
    $t = t + 1$
  EndWhile
  Return D
EndAlgorithm

Fig. 13 Algorithm to disintegrate a term ψ into a property set $D(\psi)$.

Definition 7 (Remainder) Given a term $\psi_2 \in \gamma(\psi_1)$, where γ is a generalization refinement, the *remainder* $r(\psi_1, \psi_2)$ of such generalization is a term ψ such that $\psi \sqcup \psi_2 = \psi_1$ and $\nexists \psi' \in G(\psi_1)$ such that $\psi' \sqsubset \psi$ and $\psi' \sqcup \psi_2 = \psi_1$.

That is to say, the remainder of a generalizing refinement γ from ψ_1 to ψ_2 is the most general term ψ such that when unified with the generalization ψ_2 obtains back the original term ψ_1. We will call this remainder ψ a *property* of ψ_1. Notice that the remainder is the most general term that captures which is the "property" that ψ_1 has and that is not present in ψ_2, i.e. the informational content that the generalization operator removed. Figure 12 illustrates this idea, where a train is generalized with a refinement operator, and the property subtracted is the fact that the car of that train has 2 wheels. The remainder of a specialization refinement ρ can be defined similarly.

Now, if we iterate this generalization refinement over the resulting term and keep generalizing it, we will obtain a collection of properties as remainders of each step. In the end, the iterative generalization process will reach $\bot$, the empty term, and we will have a collection of properties satisfied by the initial term. This is the intuitive idea of *term disintegration*: generalize a term repeatedly until reaching $\bot$ while collecting a property at each step by getting the remainder of the generalization operation.

Definition 8 (Disintegration) Given a finite refinement path $p = \psi_1 \xrightarrow{\gamma} \bot$ consisting of a sequence of terms $(\psi_1, ..., \psi_n = \bot)$, a *disintegration* of a term ψ_1 is the set $D_p(\psi_1) = \{r(\psi_i, \psi_{i+i}) | 1 \leq i < n\}$,

That is to say, $D_p(\psi_1)$ is the set of remainders resulting from each generalization step in p from ψ_1 to $\bot$.

Given a refinement path $\psi \xrightarrow{\gamma} \bot$, and having in mind that refinement operators represent the most fine-grained steps in which terms can be specialized or generalized, the remainders obtained from such paths correspond to the most primitive pieces of information contained in a term ψ. Therefore, the disintegration of a term is a process that breaks up a term into its most constituent and primitive pieces of information (with respect to a particular language); each one of these pieces of information is also represented as a term, and is what we call a *property*.

Figure 13 presents an algorithm to compute the disintegration of a term ψ. Given a term ψ, and a generalization refinement operator γ, the algorithm proceeds iteratively, generalizing ψ using γ, until $\bot$ is reached. At each iteration t of the algorithm, a new generalization ψ_{t+1} is generated by taking one of the generalizations (it does not matter which one) generated by γ from the current term ψ_t. Then, the property set D is expanded by adding the remainder $r(\psi_t, \psi_{t+1})$ of generalizing ψ_t into ψ_{t+1}. When $\bot$ is reached, the algorithm returns the set D containing all the properties generated so far, corresponding to a disintegration of the term ψ.

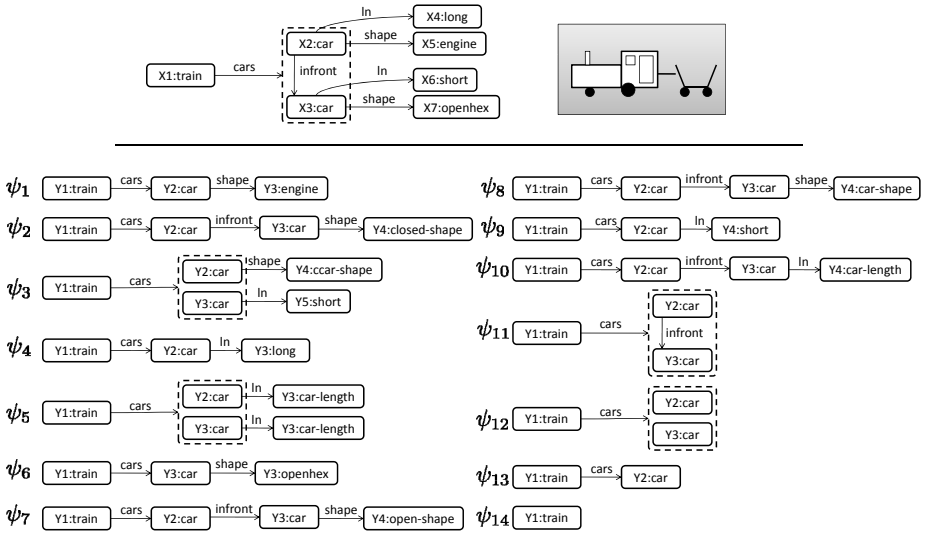


Fig. 14 An example feature term disintegrated into properties using the algorithm in Figure 13.

The *integration* of a property set is the process opposite to disintegration, and is defined as $integrate(D(\psi)) = \bigsqcup(D(\psi))$, that is to say the unification of all properties of a disintegrated term ψ. Integrating a disintegrated term allows as to recover the original term, as shown in the following Lemma.

Lemma 1 *One of the unifications of all the properties of a term ψ is exactly the term ψ, i.e. $\psi \in \bigsqcup(D(\psi))$. When unification is unique, then $\psi = \bigsqcup(D(\psi))$.*

Proof Let ψ_1 be a term that when disintegrated using the refinement path $(\psi_1, ..., \psi_n = \bot)$ yields the properties $(r_1, \ldots, r_{n-1})$. Definition 7 (Remainder) ensures that $\psi_{i+1} \sqcup r_i = \psi_i$ (or that ψ_i is one of the unifications if $\psi_{i+1} \sqcup r_i$ is not unique). Let us consider first the case when unification is unique. Iteratively unifying the properties in the reverse order in which they were generated, we can reconstruct the refinement path: $\psi_{n-1} = r_{n-1}$, $\psi_{n-2} = r_{n-1} \sqcup r_{n-2}$, $\psi_{n-3} = (r_{n-1} \sqcup r_{n-2}) \sqcup r_{n-3}$, etc. Thus, $\psi_1 = \bigsqcup_{i=n-1\ldots1} r_i$, which is precisely $\bigsqcup(D(\psi))$. When unification is not unique, we know by Definition 7 that: $\psi_{n-1} = r_{n-1}$, $\psi_{n-2} \in r_{n-1} \sqcup r_{n-2}$, $\psi_{n-3} \in (r_{n-1} \sqcup r_{n-2}) \sqcup r_{n-3}$, etc. Thus, $\psi_1 \in \bigsqcup_{i=n-1\ldots1} r_i$, which is precisely $\bigsqcup(D(\psi))$. □

Figure 14 shows an example of the disintegration process, where a simple train represented as a feature term (top half) has been disintegrated into properties (bottom half). Disintegration extracted 14 properties from this train, since the refinement path used had 14 generalization steps. Also, notice that different refinement paths might generate different disintegrations; Section 5.3 discusses this and other properties of the disintegration operation within the different languages $\mathcal{L}$ we have defined. But let us first present the formal definition of the property-based similarity.

5.2 Property-based Similarity Measure

The *property-based similarity* S_π of two terms ψ_1 and ψ_2 uses disintegration to obtain their their property sets $D(\psi_1)$ and $D(\psi_2)$ and compares how many properties are shared and how many are not.

Definition 9 (Property-based Similarity Measure)

$$S_\pi(\psi_1, \psi_2) = \frac{|D(\psi_1) \cap D(\psi_2)|}{|D(\psi_1) \cup D(\psi_2)|}$$

That is to say, S_π is the ratio between number of shared properties, and the total number of properties that at least one of the two terms satisfies. An equality among feature terms is required in order to perform set intersection and union of property sets, which we define in the usual way: $\psi \equiv \psi' \iff (\psi \sqsubseteq \psi' \wedge \psi' \sqsubseteq \psi)$.

Notice that in the disintegration process each property corresponds to one refinement. Therefore, S_π should provide results similar to those of S_λ, since each refinement in the path $\bot \xrightarrow{\rho} \psi_1 \sqcap \psi_2$ corresponds to a shared property in S_π (see Section 5.3 below).

The advantage of the property-based similarity S_π is that its computational cost is lower than that of S_λ. For data sets with complex cases, computing the anti-unification of two terms (as required by S_λ) might be very costly, while properties can be extracted at a reasonable cost. Although the formal analysis of computational complexity of both similarity measures is outside the scope of this paper, Section 7 shows an empirical evaluation of their average computational cost. Intuitively, the idea behind the property-based similarity is to go from unification and anti-unification of terms to union and intersection of properties. This is a powerful idea, since unification and anti-unification are expensive operations, while union and intersection are cheaper; the only cost is associated with subsumption for computing term equality, since $\psi_1 \equiv \psi_2 \iff (\psi_1 \sqsubseteq \psi_2 \wedge \psi_2 \sqsubseteq \psi_1)$. Subsumption among properties is a fast and efficient operation in practice, as long as we are in a domain where properties are represented by small-sized terms.

Finally, the requirements for S_π (and the disintegration operation) to be applied to a refinement graph are: (1) there is a unification operation, and (2) the generalization refinement operators ensure that for every ψ there is a path $\psi \xrightarrow{\gamma} \bot$.

5.3 Disintegration and Feature Term Languages

Depending on the refinement path used for disintegration and on the language used, we might obtain different property sets. Specifically, depending on the language being used, term disintegration following Definition 8 can be either *weakly* or *strongly complete*.

Definition 10 (Weak Completeness) Disintegration is *weakly complete*, when, given any term ψ, we have that $\psi \in \bigsqcup(D(\psi))$ —i.e. integrating all the properties using unification recovers the original term ψ.

For a given term ψ, the property set $D(\psi)$ obtained by the disintegration algorithm in Figure 13 is always at least weakly complete since (as shown in Lemma 1) $\psi \in \bigsqcup(D(\psi))$.

Definition 11 (Strong Completeness) Disintegration is *strongly complete* when, given any two terms ψ_1 and ψ_2, we have that $\psi_1 \sqcap \psi_2 \subseteq \bigsqcup(D(\psi_1) \cap D(\psi_2))$ – integrating the properties common to both disintegrations recovers the anti-unification of the two terms, $\psi_1 \sqcap \psi_2$.

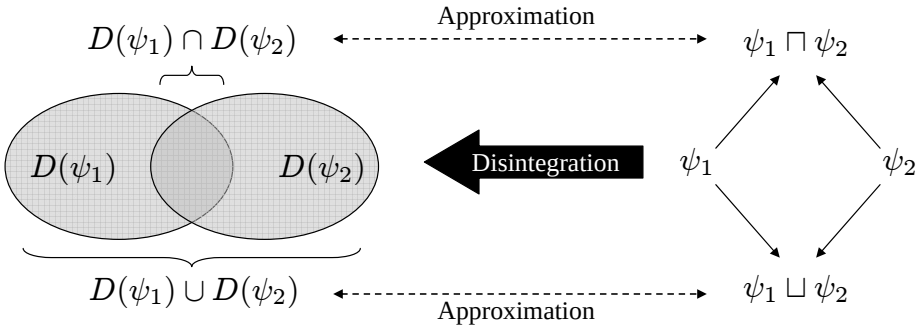


Fig. 15 Disintegration maps a term ψ into a property set $D(\psi)$ allows to go from unification and anti-unification of terms (on the right side) to union and intersection of properties (to the left side), which are computationally cheaper. Thus, the term resulting from integrating the intersection of the properties of two terms is an approximation of its anti-unification.

Notice that strong completeness implies that given any pair of terms ψ and ψ' such that $\psi' \sqsubseteq \psi$, there is a subset of properties $D' \subseteq D(\psi)$ such that $\psi' \in \bigsqcup(D')$, i.e. any term which is a subsumer of ψ can be generated as the unification of a subset of properties of ψ. This is because if $\psi' \sqsubseteq \psi$ then $\psi \sqcap \psi' = \psi'$, and thus $\psi' \in \bigsqcup(D(\psi) \cap D(\psi'))$; moreover, $D' = D(\psi) \cap D(\psi')$.

Disintegration is strongly complete in the language $\mathcal{L}_0$. The reason is that, regardless of which refinement path is used, the property set obtained is the same —the properties are just obtained in a different order (this is a direct implication of terms in $\mathcal{L}_0$ being always trees). Given two terms, ψ_1 and ψ_2, we can always find refinement paths from ψ_1 to $\bot$ and from ψ_2 to $\bot$ that traverse their anti-unification term $\psi_a = \psi_1 \sqcap \psi_2$. Therefore, the set of properties generated for ψ_1 or for ψ_2 by disintegration using those paths will contain the set of properties of ψ_a. Since, regardless of the path, we obtain the same property sets, the disintegrations of ψ_1 and ψ_2 each contain all the properties of ψ_a, and thus the unification of the intersection of their property sets is exactly ψ_a. Additionally, there is a more efficient algorithm to obtain the remainders in $\mathcal{L}_0$ [33].

If we move to more expressive languages like $\mathcal{L}_s$ and $\mathcal{L}_e$, the property set generated will be different depending on the refinement path used to disintegrate a term, and the disintegration operation cannot be guaranteed to be strongly complete. New ways of disintegrating terms are part of our future work. Additionally, studying whether different refinement paths for disintegration affects S_π, in the sense that some might be more efficient or accurate, is also part of our future work.

Finally, disintegration can never be strongly complete in languages $\mathcal{L}_c$ and $\mathcal{L}$, even if other disintegration algorithms are devised. The reason is that, since they allow cyclic variable equalities, the set $G(\psi)$ of some ψ might be infinite (see Appendix C). For having strong completeness, we need to have a subset of properties in $D(\psi)$ which can recover every term in $G(\psi)$, i.e. there is a mapping between the parts of $D(\psi)$ and elements of $G(\psi)$. Since $D(\psi)$ is finite, it is not possible to find such mapping when $G(\psi)$ is infinite.

5.4 Approximation of S_λ by S_π

We will now show that S_π approximates S_λ. The difference between the two is working with unification and anti-unification between terms versus working with union and intersection of properties, as shown in Figure 15. In fact, they are equal when disintegration is strongly complete, i.e. for language $\mathcal{L}_0$, where $S_\lambda = S_\pi$, because (as we saw) property sets are unique and thus $\psi_1 \sqcap \psi_2 = \bigsqcup(D(\psi_1) \cap D(\psi_2))$. The approximation is partial for more complex languages for which disintegration is not strongly complete. That is to say, we cannot ensure that $\psi_1 \sqcap \psi_2$ is equal to $\bigsqcup(D(\psi_1) \cap D(\psi_2))$.

When disintegration is weakly complete, the term resulting from unification of the shared properties might be more general than the anti-unification of the terms.

Lemma 2 *The unification of all the shared properties of two terms ψ_1 and ψ_2 is always more general or equal to their anti-unification, i.e.*

$$\forall \psi \in \bigsqcup(D(\psi_1) \cap D(\psi_2)), \exists \psi_a \in \psi_1 \sqcap \psi_2 : \psi \sqsubseteq \psi_a$$

Proof The proof is a straightforward implication of the definition of unification and anti-unification. All the properties in $D(\psi_1) \cap D(\psi_2)$ subsume both ψ_1 and ψ_2, and thus they must also subsume at leas one of their anti-unification(s), i.e. $\forall \psi \in D(\psi_1) \cap D(\psi_2) : \psi \sqsubseteq \psi_a$ (for some $\psi_a \in \psi_1 \sqcap \psi_2$) – see Definition 1. Therefore, by Definition 2, $\bigsqcup(D(\psi_1) \cap D(\psi_2)) \sqsubseteq \psi_a$. □

Notice that the term resulting from unification of the shared properties might be more general than the anti-unification of the terms when disintegration is weakly complete, since there might be some properties shared by ψ_1 and ψ_2 that are not in their disintegrations.

As a consequence, S_π is an underestimator of S_λ ($S_\pi \leq S_\lambda$), assuming the refinement paths used in both similarities are the same. The reason is that, in Definitions 6 and 9, the numerator in S_λ is greater or equal to the numerator in S_π, and the denominator in S_λ is lower or equal to the denominator in S_π. First, notice that $|D(\psi_1)| = \lambda_s + \lambda_1$ and $|D(\psi_2)| = \lambda_s + \lambda_2$, since the number of properties in the disintegration of the terms is identical to the lengths of the refinement paths used to disintegrate them. However, $|D(\psi_1) \cap D(\psi_2)| \leq \lambda_s$ since, even when the paths are the same, the set of properties may be different, and there are some properties shared by ψ_1 and ψ_2 that are not in both of their disintegrations. Consequently, the numerator S_λ is greater or equal to the numerator in S_π and the denominator satisfies $|D(\psi_1) \cup D(\psi_2)| = |D(\psi_1)| + |D(\psi_2)| - |D(\psi_1) \cap D(\psi_2)| \geq |D(\psi_1)| + |D(\psi_2)| - \lambda_s = \lambda_s + \lambda_1 + \lambda_2$.

6 Weighted Property-based Similarity

Since each property represents a piece of information, S_π measures how many pieces of information are shared among two given terms. However, when using this similarity measure for a particular learning task, not all the pieces of information shared among two terms might be equally important.

An interesting advantage of the property-based similarity is that we can assess the importance of each individual property for the task at hand and thus determine a weight for each property. For classification tasks, assume our training set is $T = (c_1, \ldots, c_n)$

Data set	Examples	Classes	Variables	Sets	Properties	L
Soybean	307	18	8 - 38	0	12 - 72	$\mathcal{L}_0$
Demospongiae-280	280	3	20 - 48	0 - 18	32 - 106	$\mathcal{L}_s$
Demospongiae-503	503	8	20 - 51	0 - 18	32 - 106	$\mathcal{L}_s$
Trains	10	2	14 - 23	3 - 6	46 - 78	$\mathcal{L}$
Kinship	24	2	14	0 - 8	67 - 110	$\mathcal{L}$
PTC-MR	297	2	6 - 138	0 - 64	7 - 523	$\mathcal{L}$
PTC-FR	296	2	6 - 138	0 - 76	7 - 491	$\mathcal{L}$
PTC-MM	296	2	5 - 138	0 - 76	7 - 523	$\mathcal{L}$
PTC-FM	319	2	5 - 138	0 - 76	7 - 523	$\mathcal{L}$

Table 1 Data set size and complexity comparison, showing the number of examples and classes of each data set, the ranges in number of variables and set-values features in the examples, the ranges in number of properties produced by disintegration of examples, and the language L used to represent the data set.

and an example is a pair $c_i = (p_i, s_i)$, i.e. a problem description p_i and a solution class s_i. In order to asses property weights, we take T's problem descriptions (p_i) and disintegrate them obtaining $D(p_i)$ for each example in T. Then, the *property dictionary* $\mathbf{D}(T)$ of the training set T is obtained by the set of all (non-equivalent) properties of the examples, i.e. $\mathbf{D}(T) = \bigcup_{i=1,\ldots,n} D(p_i)$.

The second step is to estimate a weight $w(\psi)$ for each property ψ in the property dictionary $\mathbf{D}(T)$. Each property ψ divides the set of training instances in two sets: T_ψ and $\overline{T}_\psi$, where $T_\psi = \{c_i \in T | \psi \in D(p_i)\}$ is the subset of examples that satisfy a given property ψ and $\overline{T}_\psi = \{c_i \in T | \psi \notin D(p_i)\}$ is the subset of those that don't. A measure such as Quinlan's Information Gain [38] can be used to compute property weights. The following equation determines the weight of each property $\psi \in \mathbf{D}(T)$ using Information Gain:

$$w(\psi) = H(T) - \frac{H(T_\psi) \times |T_\psi| + H(\overline{T}_\psi) \times |\overline{T}_\psi|}{|T|}$$

where $H(X)$ represents the entropy of the set of instances X with respect to the partition induced by the solution classes, and $|X|$ is the cardinality of set X. Thus, given that $D(\psi)$ is the subset of properties $D(\psi) \subset \mathbf{D}(T)$ that a particular term ψ satisfies, the weighted property-based similarity $S_{w\pi}$ between two terms is defined as follows:

Definition 12 (Weighted Property Similarity)

$$S_{w\pi}(\psi_1, \psi_2) = \frac{\sum_{\psi_i \in D(\psi_1) \cap D(\psi_2)} w(\psi_i)}{\sum_{\psi_j \in D(\psi_1) \cup D(\psi_2)} w(\psi_j)}$$

That is to say, $S_{w\pi}$ is the sum of the weights of those properties shared by two terms ψ_1 and ψ_2, divided by the sum of the weights of all the properties that at least one of the terms ψ_1 and ψ_2 satisfies. Clearly, $S_{w\pi}$ is independent of the Information Gain measure, and any other way to estimate importance of properties could be used. Nevertheless, the experiments reported in the next section use Information Gain for estimating the weight of properties.

7 Experimental Evaluation

In order to evaluate our similarity measures, we measured the performance of a nearest neighbor method using them. We used five different data sets: *Soybean*, *Demospongiae*,

Trains, *Kinship* and *Toxicology* (PTC). The different characteristics of these data sets are shown in Table 1. Trains is the data set shown in Figure 1, as presented by Michalski [27]; the full language $\mathcal{L}$ is required to represent this data set. Soybean is a propositional data set (representable using $\mathcal{L}_0$) from the UCI machine learning repository consisting of 307 cases and 18 solution classes.

Kinship is a small but complex relational data set consisting of two families where the goal is to learn family relations like *uncle* [22]. Each family has 12 members (thus 24 persons in total); the full language $\mathcal{L}$ is required to represent this data set. The representation is purely relational, and each family is a graph (with most features introducing circular variable equalities); there are 4 positive examples and 20 negative examples. We selected the Trains and Kinship data sets to highlight domains that are highly relational and where the value of features is not as important as the structure of the terms. The Demospongiae data set is a relational data set from the UCI machine learning repository (with sets, but no variable equalities, and representable in $\mathcal{L}_s$) composed of 503 Demospongiae belonging to 8 different solution classes. For the Demospongiae data set, we report results both using the complete data set as well as using a subset consisting of 280 Demospongiae and 3 solution classes.

Finally, the Toxicology data set (PTC) is a highly relational data set introduced as a challenge in the ECML/PKDD 2001 conference [21]. We have selected PTC to evaluate the scalability of our similarity measures rather than their accuracy; thus, we present results for Toxicology in a separate section. The original dataset consists of a collection of Prolog facts, so, for our evaluation we used the version created by Armengol and Plaza [8], who converted the dataset to feature terms using a chemical ontology and which contains 371 examples. The Toxicology dataset consists of a collection of molecules and the task is to predict their carcinogenicity (positive or negative) for four different types (MR, FR, MM, FM) given by sex (M and F) or species (M and R). For each different type, a different subset of the 371 examples is used, and they are shown in Table 1 as separate data sets.

Additionally, in order to better measure the complexity of each dataset, Table 1 shows some statistics of each data set. Specifically, the first two columns in Table 1 shows the number of examples of each dataset and the number of solution classes. The third column shows the number of variables present in the feature terms representing the examples of each data set; we show the range from minimum to maximum number of variables. The larger the number of variables, the larger the examples – for instance the term in Figure 2 has 14 variables. As Table 1 shows, the larger examples are found in the Toxicology dataset, where some examples have up to 138 variables. The fourth column of Table 1 shows the total number of variables occurring in set-valued features (minimum and maximum); this factor plays an important role in the computational cost of subsumption. The only data set without set-valued features is Soybean. The fifth column shows the number of properties obtained by disintegrating the examples of a dataset: the more properties the more complex the examples are. This number is equivalent to measuring the length of a refinement path from $\bot$ to a given example (i.e. it is a measure of the amount of information contained in each example); we show the minimum and maximum number of properties. Again, this shows that the Toxicology data set has some very large examples, some of them generating up to 523 properties. The last column in Table 1 shows the language required to represent the examples in each data set.

	S_λ	S_π	$S_{w\pi}$	SHAUD	RIBL	Kashima
	1-NN					
Soybean	91.53	91.53	91.21	91.53	91.53	**92.18**
Demospongiae-280	95.00	94.64	**96.79**	95.71	91.67	90.71
Demospongiae-503	89.66	90.26	**92.44**	88.27	88.93	83.10
Trains	50.00	**60.00**	**60.00**	-	50.00	20.00
Kinship	**100.00**	87.50	**100.00**	-	83.33	83.33
	3-NN					
Soybean	88.93	**89.58**	88.27	88.93	88.93	87.95
Demospongiae-280	94.29	95.71	**96.79**	95.00	91.67	90.71
Demospongiae-503	88.27	**90.66**	90.46	87.08	86.43	83.10
Trains	60.00	**70.00**	**70.00**	-	**70.00**	30.00
Kinship	**91.67**	83.33	83.33	-	83.33	83.33

Table 2 Classification accuracy (in percentage) measured using a leave-one-out method for different similarity measures.

7.1 Classification Accuracy Comparison

Table 2 shows the classification accuracy for several similarity measures in the data sets used for our evaluation —except the PTC data set that is addressed later in Section 7.2. We report results for S_λ and for both S_π and $S_{w\pi}$, as well as two other relational similarity measures (SHAUD [7] and RIBL [17]) and a graph kernel [25] for comparison purposes. For each similarity measure we measured classification accuracy using both 1-nearest neighbor and a 3-nearest neighbor by means of a leave-one-out method. SHAUD is a relational similarity measure 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 actually be converted to FOL predicates without losing information. We used such conversion to evaluate RIBL.

Moreover, RIBL and SHAUD require knowing the ranges of each numeric feature in order to compute the similarity values. We used the minimum and maximum values observed in the data set to define such ranges. RIBL also requires a maximum depth parameter, that was set to 10 in our experiments. Since SHAUD works only for trees, it could not be applied to the Kinship or Trains data set.

We also used the random-walk graph kernel [25], that we will refer to as Kashima's kernel. Given two graphs, Kashima's kernel computes the expected similarity between a random walk from one graph and a random walk from the other graph. This kernel has a low computational cost for acyclic graphs, but requires inverting a matrix with $n \times m$ rows and $n \times m$ columns (where n and m are the number of nodes of the two graphs) when there are cycles in the graph, and thus it can only be calculated using a numerical approximation. Since Kashima's kernel works on labelled graphs, we used the graph representation of feature terms. Moreover, Kashima's kernel has two parameters: γ, that corresponds to the probability of a random walk to end, and a kernel to assess similarity among the labels in the graph. We experimented with different values for γ, and used $\gamma = 0.1$ in our experiments, since it gave the best results overall. We used the sort ontology to define a kernel for the labels of the graph (to ensure that the kernel can exploit all the information available to the other similarity measures).

Table 2 shows the classification accuracy of these methods, with the highest accuracy for a given data set shown in bold; most of the times the difference between

the accuracy shown in bold and the rest is statistically significant using a t-test with $p < 0.05$, with the following exceptopns: in the Trains data set (which only has 10 instances), in the Demospongiae-503 data set with 3-NN (where the difference between S_π and $S_{w\pi}$ is not statistically significant), and in the Demospongiae-280 data set, where it is only significant with $p < 0.25$. Table 2 shows that the weighted property-based similarity ($S_{w\pi}$) achieves the highest classification accuracy in all data sets except for Soybean (but where the difference is not statistically significant when using 3-NN). The Soybean data set is a propositional one, where the most important issue is the value of the features, and not the relational structure in the instances; however, S_π achieves the same classification accuracy as both SHAUD and RIBL, and only slightly under that achieved by Kashima's kernel.

Structure is the only important factor in the Kinship data set. SHAUD cannot be applied since instances are not trees, and RIBL has problems also, since there are no numerical or symbolic values in any of the terms in Kinship, only a graph relating each member of the family to each other. In fact RIBL achieves an accuracy of 83.33% only because it always predicts "negative", and there are only 4 positive examples out of 24. S_λ and $S_{w\pi}$ are able to capture the structure of the instances, and achieve an accuracy of 100.00% when used in a 1-NN. S_π also achieves a high accuracy, although not a 100.00%.

Trains is an apparently simple but complex data set, since out of the numerous features in each train only two are key to determine the class, and it is hard to learn this with only 10 instances. $S_{w\pi}$ and RIBL perform the best in this data set.

In the Demospongiae data set, $S_{w\pi}$ achieves the best results. SHAUD, S_π, and S_λ achieve also good results but not as good, and finally Kashima's kernel gets the lowest accuracy. RIBL does not perform well in this data set either, because it does not exploit completely the information in the sort taxonomy, which is important in this data set. Moreover, notice 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 $S_{w\pi}$), and thus we used uniform weights. Thus, the results reported here for RIBL might be suboptimal, although weights would not allow RIBL to improve in this data set.

Comparing with Kashima's kernel, Table 2 shows that the kernel achieves good results only in the Soybean dataset. The issue seems to be that Kashima's kernel simply estimates the similarity among two graphs; for instance, in the Kinship dataset, the graphs representing some of the examples are identical (since all the examples belong to the same family). However, both RIBL and our measures S_π and S_λ consider the similarity among instances that, albeit are graphs, start at a specific node (called the root node in feature terms). To determine whether this was the case, we performed a second set of experiments (not shown on Table 2) where we used Kashima's kernel, but only considering random walks starting from a root node. Using this second version, we achieved better results in the Trains and Kinship datasets. In the Trains dataset, accuracy went up to 50% for 1-NN and 60% for 3-NN, while in the Kinship dataset, accuracy went up to 87.5% both using 1-NN and 3-NN.

The computational cost of different similarity measures, evaluated as the average execution time to compute the similarity between two instances in different data sets, is shown in Table 3. The most computationally expensive similarity measures are S_λ and SHAUD, since they perform the anti-unification of the two instances being compared. For instance, in the Demospongiae data set, S_λ takes 66.23 milliseconds[6] on average

[6] Times were measured in a Macbook Pro laptop with a 2.53GHz CPU and 4GB of RAM.

	S_λ	$S_\pi / S_{w\pi}$	SHAUD	RIBL	Kashima
Soybean	66.30ms	0.36ms	23.86ms	1.90ms	1.37ms
Demospongiae-280	66.23ms	2.65ms	28.24ms	5.26ms	3.09ms
Demospongiae-503	62.72ms	3.17ms	23.00ms	5.25ms	4.36ms
Trains	158.35ms	4.29ms	-	7.81ms	1.35ms
Kinship	85.84ms	5.39ms	-	9.69ms	15.87ms

Table 3 Average time (in milliseconds) required to compute the similarity of two instances for different similarity measures.

	S_π	$S_{w\pi}$	RIBL
	1-NN		
PTC-MR	57.24	**58.59**	57.91
PTC-FR	63.04	61.39	**66.33**
PTC-MM	58.06	**60.87**	50.83
PTC-FM	**62.07**	58.62	56.43
	3-NN		
PTC-MR	51.85	57.58	**59.59**
PTC-FR	56.44	63.37	**64.03**
PTC-MM	57.19	**65.22**	56.52
PTC-FM	58.93	**60.19**	54.23

Table 4 Classification accuracy for PTC in percentage measured using a leave-one-out method for different similarity measures.

to assess one similarity. SHAUD takes 23.86ms, and the property based similarities S_π and $S_{w\pi}$ take 2.65ms. RIBL is also a fast similarity measure, taking only 5.26 milliseconds per similarity in the Demospongiae data set. Kashima's kernel is very fast in the Soybean, Demospongiae and Trains data sets, since they do not contain cycles. However, Kashima's kernel has a much higher computational cost in the Kinship data set (15.87ms), since it contains cycles[7].

Computing the weights of properties in $S_{w\pi}$, as well as disintegrating an example into properties, are processes typically performed offline, since they only need to be performed once, and their cost is not included in the time for computing the similarity among 2 instances. Computing weights for $S_{w\pi}$ in our experiments takes 79.82ms in the Soybean dataset, 389.20ms and 983.92ms in the Demospongiae-280 and Demospongiae-503 data sets, 1.50ms in the Trains data set, and 49.68ms in the Kinship data set. Disintegrating the complete Soybean data set takes 8.12 seconds, the Demospongiae data set requires 328.77 seconds, while the Trains and Kinship data sets take 7.45 and 18.85 seconds respectively.

The next two subsections evaluate the scalability of our similarity measures by using first a data set with very large examples and then a data set with a large number of examples.

7.2 Evaluation with Large Examples

In order to evaluate the scalability of S_λ and S_π, we selected the Toxicology data set, which contains some very large molecules (with up to 138 variables in the terms representing the examples). SHAUD could not be applied to Toxicology, since molecules

[7] We used the *Colt* Java matrix library for Kashima's kernel for high performance matrix computations: `http://acs.lbl.gov/software/colt/`

	$S_\pi / S_{w\pi}$	RIBL
PTC-MR	4.05ms	5.85ms
PTC-FR	9.76ms	5.84ms
PTC-MM	10.26ms	5.84ms
PTC-FM	10.11ms	5.79ms

Table 5 Average time (in milliseconds) required to compute the similarity of two instances for different similarity measures in the Toxicology data set.

are cyclic graphs (SHAUD was applied to a subset of the Toxicology data set [7], consisting of molecules without cycles when represented as feature terms). Additionally, due to the size of the molecules, S_λ could not be applied to the Toxicology data set, since there is a small set of molecules that are too large for computing anti-unification in a reasonable amount of time.

Table 4 shows the classification accuracy results for the Toxicology data set in its 4 animal groups. RIBL performs well for the MR and FR animals, but rather poorly in MM and FM. $S_{w\pi}$ outperforms S_π in all tests except in FR and FM using a 1-NN. Using a 3-NN, $S_{w\pi}$ always outperforms S_π. The accuracy levels achieved for the Toxicology data set are comparable to the results obtained with other techniques. For instance, using graph-kernels, Kashima et al. [25] report between 54.1% to 58.4% accuracy for MR depending on the value set for the parameter γ of their algorithm (the termination probability of random walks). They also report between 62.1% to 66.1% for FR, 62.2% to 64.3% for MM and between 59.3% to 63.4% for FM. Moreover, the maximum and minimum accuracy for each type were achieved with a different value of that parameter. The results obtained with $S_{w\pi}$ using a 3-NN are inside or above those ranges, without the need to tune any parameter, so the results achieved with $S_{w\pi}$ are comparable to those obtained with Kashima's kernel.

Table 5 shows the average time required to compute the S_π similarity between examples in the Toxicology data set, showing that the cost of assessing similarity even in this complex data set is not too high. As before, $S_{w\pi}$ needs to compute weights for all the properties, which takes 1.17s, 1.02s, 1.19s and 1.28s for the PTC-MR, PTC-FR, PTC-MM and PTC-FM data sets respectively; notice that, using a leave-one-out method, the weights were recomputed for each problem using the remaining examples as training set (consisting of n-1 examples). The Toxicology data set takes a considerable amount of time to disintegrate compared with the other data sets, requiring 22,614.14 seconds (over 6 hours) to disintegrate. However, notice that data sets only need to be disintegrated once.

7.3 Evaluation with a Large Number of Examples

In order to better evaluate the scalability of S_λ and S_π, we used Muggleton's Train data set generator [31], which can generate Michalsky-style Train data sets, but of arbitrary size. Specifically, we generated data sets with 100, 1,000 and 10,000 trains. Tables 6 and 7 show the results of our experiments over those data sets using a leave-one-out method. For each different size of the Trains data set and similarity metric, we evaluated the performance of both a 1-NN and a 3-NN algorithm in classification accuracy and in time used to classify a given problem. Notice that, using a k-NN, the time used to solve a problem includes the application of the similarity metric $n - 1$ times (where n is the size of the data set). We evaluated problem solving time, instead

	S_λ	S_π	$S_{w\pi}$	RIBL	Kashima
	1-NN Accuracy				
Trains-100	**71.00%**	**71.00%**	**70.00%**	56.00%	62.00%
Trains-1000	84.80%	80.20%	**93.90%**	67.10%	63.90%
Trains-10000	-	95.58%	**99.01%**	97.86%	67.20%
	3-NN Accuracy				
Trains-100	64.00%	67.00%	**73.00%**	62.00%	**73.00%**
Trains-1000	88.50%	82.40%	**93.30%**	71.20%	67.60%
Trains-10000	-	97.73%	**98.51%**	96.71%	68.39%

Table 6 Classification accuracy for different sizes of the Trains data set measured using a leave-one-out method for different similarity measures.

	S_λ	$S_\pi / S_{w\pi}$	RIBL	Kashima
Trains-100	2.13s	9.88ms	114.87ms	255.60ms
Trains-1000	27.26s	18.30ms	813.98ms	2.14s
Trains-10000	-	95.39ms	11.13s	28.96s

Table 7 Time to solve a problem using different similarity measures for different sizes of the training set.

of the time required to assess the similarity between two instances (as in previous sections), since the time required to assess similarity does not change as we increase the number of instances in a data set. As before, times in Table 7 do not include the time required to disintegrate a data set for S_π and $S_{w\pi}$, or the computation of weights for $S_{w\pi}$. Disintegration time was 30.26, 235.78 and 2,287 seconds for the 100, 1,000 and 10,000 trains data sets respectively, while computing weights takes 2.24ms, 141.74ms, and 39.9s for the 100, 1,000, and 10,000 trains data sets respectively.

As expected, Table 6 shows that all the similarity measures achieve much higher performance, in terms of classification accuracy, in data sets with a larger number of examples. In particular, $S_{w\pi}$ is the one that increases faster, achieving an accuracy of 99.01% in the data set with 10,000 trains. Kashima's graph kernel does not perform well in this data set in terms of accuracy, since it seems unable to capture the target concept (namely, trains are labelled as "east" if they have one car that is, at the same time "short" and "closed").

For the 10,000 trains data set, we show no results for S_λ, since it takes a prohibitive amount of time to complete a leave-one-out run with such large amount of examples. A simple estimation allows us to predict that the amount of time S_λ would take to solve a problem in the 10,000 trains data set is between 200 to 300 seconds. S_λ is slow since it is based on antiunification, if a very large number of similarity assessments is required, its cost can be prohibitive. RIBL and Kashima's kernel are in the middle, they are much faster than S_λ, but not as fast as S_π and $S_{w\pi}$. Specifically, Kashima's kernel is quite efficient in this data set, since the graphs representing trains do not have cycles.

S_π and $S_{w\pi}$ are much faster than the other similarity measures. For example, the time to solve a problem in the 10,000 trains data set is only 95.39 milliseconds. In summary, we can see that S_π and $S_{w\pi}$ are scalable to large data sets, since the expensive operation is disintegration, and it only needs to be performed once. $S_{w\pi}$ also requires to compute weights for the different properties, but this only needs to be done once for each training set.

	Soybean	Demospongiae-503	*Trains*	*Kinship*
S_λ vs S_π	1.000	0.984	0.950	0.894
S_λ vs $S_{w\pi}$	0.987	0.938	0.700	0.829

Table 8 Correlation between the similarity values of S_λ and S_π and $S_{w\pi}$.

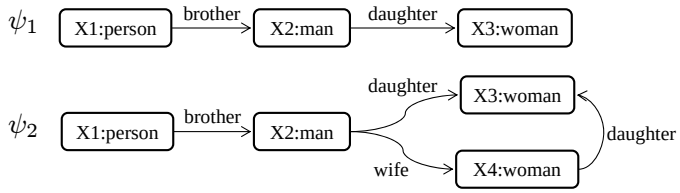


Fig. 16 Two of the highest weighted properties in the Kinship data set. Notice that both properties imply that the person corresponding to this pattern is an uncle or an aunt, and that the target concept to learn from the Kinship data set is "uncle".

7.4 Correlation Between S_λ and S_π

Since S_π approximates S_λ by moving from unification and anti-unification of terms to union and intersection of properties, we would like to evaluate how good is this approximation. Theoretically, the two similarity measures are identical when disintegration is *strongly complete*, as defined in Section 5.3 (e.g. when working in the language $\mathcal{L}_0$), while they may deviate when disintegration is only *weakly complete*. In order to evaluate how does this effect manifests in different data sets, we have evaluated the Pearson correlation coefficient among the similarity values produced by our three similarity measures (S_λ, S_π, and $S_{w\pi}$) on the 4 data sets shown in Table 8.

In the Soybean data set, as expected, S_λ and S_π compute identical similarity values, and thus their correlation is 1. This is because the Soybean data set can be represented in the language $\mathcal{L}_0$, in which disintegration is strongly complete, and thus S_π approximates S_λ completely. In the Demospongiae data set the correlation diminishes to 0.984. This means that there are already some differences, but they are quite small. The reason is that this data set, which is represented using language $\mathcal{L}_s$, has only very few attributes that cannot be represented with $\mathcal{L}_0$ (it contains a few features that are set valued). In data sets where more complex languages are required, such as the Trains and the Kinship data sets, S_λ and S_π start deviating, as expected, since disintegration is only weakly complete. Finally, we can observe that the difference between S_λ and S_π is smaller than between S_λ and $S_{w\pi}$, since the weights in $S_{w\pi}$ modify the similarity estimation.

7.5 Discussion

As mentioned earlier, both RIBL and SHAUD implement the idea of "hierarchical aggregation," and therefore have a bias that makes them deem features closer to the root of an instance as more important than those that are deeper into the structure. The similarity measures presented in this paper do not have this bias. The weighted property-based similarity $S_{w\pi}$ assigns weights to properties according to how discriminant they are with respect to the task at hand, regardless of whether they are deep into the structure or if they are closer to the root. For instance, Figure 16 shows two of

the highest weighted properties in the Kinship data set by the $S_{w\pi}$ similarity (many properties were tied with the highest weight, and these are just two of them). Recall the the target concept for this data set is that of "uncle". The first one represents "a person with a brother who has a daughter", and the second one, "a person with a brother, who has a daughter and a wife". Notice that if a particular instance satisfied any of those properties, it means that the term represents either an "uncle" or an "aunt". Therefore, those properties are highly discriminant for the task at hand, which is discriminating uncles from non-uncles. On the other hand, Kashima's graph kernel does not have the hierarchical aggregation bias, but weights each of the different random walks that can be generated for a given graph according to its probability, which might not be correlated with its importance to the task at hand. For that reason, it would be interesting to incorporate a different way to weight the random walks, like $S_{w\pi}$ does with properties.

Moreover, notice that properties do not correspond to individual features in propositional machine learning, but to complete patterns that capture some piece of information of an instance. For instance, the second property shown in Figure 16 captures the fact that X_2, the brother of a particular person, and the wife of X_2 have a daughter in common; this shows that the disintegration operation will construct the property as the most general term which still captures that piece of information, regardless of the depth or the number of features or variables needed to represent it. This illustrates that the combination of disintegration and weight assessment can discover which are the relevant properties or patterns for a given classification task.

In summary, $S_{w\pi}$ is the most balanced similarity overall, achieving the highest classification accuracy in most data sets while being computationally efficient. Moreover, S_λ, S_π and $S_{w\pi}$ are conceptually very simple (they measure the amount of shared information) whereas in more complex measures, such as SHAUD and RIBL, it is hard to conceptually understand what is exactly being measured. Comparing S_λ to S_π and $S_{w\pi}$, S_λ has the advantage of computing an explicit symbolic similarity and of being conceptually very simple, however it is computationally expensive. S_π and $S_{w\pi}$ on the other hand are computationally less expensive and more accurate. Additionally, $S_{w\pi}$ has the interesting side effect of discovering which are the most relevant properties for a given classification task, as illustrated above.

8 Related Work

Hutchinson [24] presented a distance based on the anti-unification of two terms. The Hutchinson distance is the addition of the sizes of the variable substitutions required to move from the anti-unification of two terms to each of these terms. This measure is related to S_λ, since it is similar to the addition $\lambda_1+\lambda_2$. However, counting only variable substitutions fails to take into account some information. 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, while in S_λ moving from *number* to *integer* counts as one refinement, whereas moving from *number* to 45 requires two refinements —thus S_λ is more fine-grained. Moreover, Hutchinson's distance does not take into account the amount of information shared (that S_λ estimates as the length λ_s of the path from $\bot$ to the anti-unification).

RIBL (Relational Instance-Based Learning) is an approach to apply lazy learning techniques while using Horn clauses as the representation formalism [17]. RIBL's

similarity measure follows a "hierarchical decomposition" approach: the similarity of two instances (terms) is the average of the similarity of the value of their attributes (calling this function recursively if the values are themselves terms). Thus, RIBL is better suited for acyclic graphs. Moreover, they define special similarity measures if the values are numeric or symbolic. Our S_λ and S_π 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 a refinement graph. For S_λ and S_π, the fact that terms are trees, graphs or lists only affects the computational cost of the process, but not their algorithms. Moreover, RIBL's recursive computation of similarity adds a bias in that values deep in the instance tree are bound to have less importance, while S_λ and S_π do not present this bias. An earlier similarity measure related to RIBL was that of Bisson [11].

Horváth et al [23] presented an extension of RIBL that is able to deal with 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 RIBL's basic similarity measure. The downside of the RIBL approach (including this improvement) is that specialized measures have to be defined for different types of data, whereas S_λ and S_π can uniformly handle any kind of tree or graph.

Bergmann and Stahl [10] 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). This similarity is defined in a recursive way, thus following the same "hierarchical decomposition" idea as RIBL, and limiting the approach to tree representations.

SHAUD, presented by Armengol and Plaza [7], is another similarity measure following the "hierarchical decomposition" approach but designed for feature terms. SHAUD also assumes that the terms do not have cycles, and in the same way as RIBL and Bergmann and Stalh's it can handle numerical values by using specialized similarity measures for different data types. An advantage of S_λ and S_π with respect to RIBL and SHAUD is that S_λ and S_π can handle comparisons among generalizations (i.e. terms that have unbound variables). Hutchinson distance can handle generalizations by using the "single representation trick" to map variables into constants, and Bergmann and Stahl define some special instances to handle this situation. Notice that this is because S_λ and S_π do not make any assumptions about the data other than assuming a subsumption relation and a refinement graph.

Similarity measures for complex molecular structures in domains of biology or chemistry have been widely studied [45], and they are typically grouped into two classes [39]: *fingerprint-based* and *graph-based*. Fingerprint similarities transform each molecule in a sequence of binary features where each feature determines whether a particular molecule exhibits some particular property or contains a certain substructure. Of special interest for the work presented in this paper are graph-based similarities, which represent molecules as graphs (where each vertex corresponds to an atom). Graph-based similarity measures for molecules are typically based on computing the *maximum common subgraph* (MCS) of two graphs. This is a computationally expensive process, and thus there are a number of strategies to simplify the computations. For instance Raymond et al. [40] propose to first compute an upper bound of the similarity measure, and only compute the actual MCS for those molecules for which the upper bound is over a given threshold. Computing the MCS is a very related problem to that of finding the anti-unification in refinement graphs, and thus S_λ is related to graph-based similarities

for molecules. Moreover, S_π can be seen as either a fingerprint similarity (because it breaks every term into a collection of properties, each of which could be represented as a binary feature) or a graph-based similarity (since S_π is an approximation of S_λ).

Another related area is that of kernels for graphs. Kernels for graphs are interesting since they allow the application of machine learning techniques to complex data represented as graphs (such as chemical molecules). Typically, kernels for graphs are based on the idea of finding substructures (the idea is that if substructures are similar, then graphs are similar). For example, Kashima et al. [25] present a kernel for graphs based on random-walks. Although these kernels are defined specifically for particular kinds of graphs (e.g. directed labeled graphs), an interesting idea is that they are based on approximations rather than on exact calculations. Our similarity measures, however, are based on exact subsumption calculations. An interesting idea for future work is to explore similarity measures for refinement graphs based on approximations of either the subsumption, anti-unification or remainder operations. Moreover, notice that feature terms can also be represented as labelled graphs, thus, kernels for graphs could be applied to our data. However, when we represent a term as a labelled graph, some information (such as the sort information) might be lost.

Concerning the applicability of S_λ and S_π to other formalisms, feature terms can represent naturally object oriented data, making our approach applicable to those representations. Moreover, other authors have proposed refinement operators for different subsets of first-order logics such as description logics, which is the first step towards applying our similarity metrics to those representation formalisms. For instance, Laag and Nienhuys-Cheng [26], Shapiro [42] or Lehmann and Hitzler [30].

A related area is that of similarity measures for description logics. Fanizzi et al. [18] present a similarity metric based on the idea of a "committee of concepts". In their work, they consider each single concept in the T-box to be a feature that can be 0 or 1 for each individual (belonging or not to that concept). The ratio of concepts that two individuals share corresponds to their similarity. González et al. [20] present a similarity measure for description logics designed for case-based reasoning systems. This similarity measure shares ideas with SHAUD, but in the context of description logics. In the same way as SHAUD, it has problems with circular variable equalities, and thus they preprocess the instances to remove the roles that introduce such circularities before processing. For that reason, this similarity would not be suitable for data sets such as Kinship.

Kaci and Sasaki [5] introduced algorithms for both unification and anti-unification of feature terms based on clause rewriting rules (which operate over the clause form of a feature term). The anti-unification algorithm presented in Appendix B, however, is different, since it is based on refinement operators.

In an earlier paper [33] we introduced two informal definitions of S_λ and S_π. In this paper, we have formally defined several languages in order to analyze the behavior of S_λ and S_π within representations of different complexity and presented general algorithms for refinement graphs. Moreover, the disintegration process is defined here for the first time, while the algorithm for finding properties in [33] is adequate only for $\mathcal{L}_0$, since it would not generate weakly complete property sets in other languages.

9 Conclusions

In this paper we have presented similarity measures for structured representations based on the notion of refinement graphs, which can be used for case-based reasoning and instance-based learning systems. Specifically, we presented the anti-unification-based similarity, S_λ, which generalizes the classical notions of similarity between concepts in a taxonomy to compute similarity between instances in a refinement graph. We also presented the property-based similarity, S_π, a measure that is an approximation of S_λ and, thanks to the idea of term disintegration, is computationally more efficient and allows the addition of weights. We have evaluated both similarity measures in a collection of propositional and relational data sets and compared them with other similarity measures for structured representation of instances.

In order to define S_λ and S_π, we presented a refinement graph for feature terms, but both similarity measures could, in principle, be implemented for other representation formalisms. Our similarity measures only require a refinement graph with (1) unification, (2) specialization refinement operators that are finite and complete, and (3) generalization refinement operators which are finite and which ensure that $\bot$ is reachable from any term by generalization. Specifically, S_λ requires (2) while S_π requires (1) and (3).

The main contributions of our work are twofold. The first contribution is the idea of defining similarity over a refinement graph, which allows our similarity measures to be independent of the representation formalism. Whereas previous work in similarity measures presented specialized algorithms to assess similarity for specific data types, in our work, the language in which the instances are represented (whether it is relational or propositional, or whether we allow cycles or not, etc.) only has an effect on the computational cost, but not on the algorithm used to assess similarity. Ontologies, as defined in this paper, were introduced to restrict the search space and increase the efficiency, but they are not strictly required for the definition of the similarity measures.

The second contribution is term disintegration. Disintegration allows us to move from performing unification and anti-unification of terms to performing union and intersection of properties, which are more efficient operations, hence improving the efficiency of similarity assessment over refinement graphs. Moreover, working directly with property sets offers interesting possibilities. For instance, in this paper we have exploited such property sets with feature weighting to obtain a similarity measure ($S_{w\pi}$) which can discover the most relevant properties for a given classification task.

Additionally, we have defined refinement operators for feature terms for several languages whose refinement graphs satisfy the requirements above. The two similarity measures that we have introduced are general with respect to the refinement graphs satisfying those requirements. Moreover, they are capable of both estimating a similarity degree as well as generating a symbolic description of such similarity.

As future work, we will study how our approach on similarity could be used in other representation formalisms such as Horn clauses or description logics (a preliminary study on this can be found at [41]). This study basically needs to define refinement operators that satisfy the requirements (1-3) above, so that our similarity measures apply. This entails solving some theoretical issues, since current results characterizing refinement graphs on Horn clauses [44] or description logics [19] lack some of the requirements outlined in our approach. We will also explore the use of term disintegration in other fields of structured machine learning such as induction, i.e. using the property sets as an alternative representation of relational instances. Finally, the rela-

tionship (established by disintegration and integration) between the "space of terms" (with unification and anti-unification) and the "space of properties" in languages of different expressiveness seems to merit also a deeper study.

Acknowledgements We thank Eva Armengol for her support in using and integrating the PTC dataset and the anonymous reviewers for their insightful suggestions. Support for this work came from the project Next-CBR TIN2009-13692-C03-01 (co-sponsored by EU FEDER funds).

Appendix A: Formal Definition of Subsumption for Feature Terms

Intuitively, a feature term ψ_1 subsumes another one ψ_2 if all the information contained in ψ_1 is also in ψ_2; or equivalently, when all that is true for ψ_1 is also true for ψ_2. Formally, a feature term ψ_1 subsumes another one ψ_2 ($\psi_1 \sqsubseteq \psi_2$) if all the following conditions are met:

- There is a total mapping $m : vars(\psi_1) \rightarrow vars(\psi_2)$,
- $root(\psi_2) = m(root(\psi_1))$,
- For each variable $X \in vars(\psi_1)$:
  - $sort(X) \leq sort(m(X))$,
  - $features(X) \subseteq features(m(X))$,
  - For each feature $f \in features(X)$, where $X.f = \Psi_1$ and $m(X).f = \Psi_2$ (i.e. Ψ_1 and Ψ_2 are the set of feature terms that are the value of feature f in ψ_1 and ψ_2 respectively), we have that:
    - $\forall Y \in \Psi_1, \exists Z \in \Psi_2 | m(Y) = Z$,
    - $\forall Y, Z \in \Psi_1, Y \neq Z \Rightarrow m(Y) \neq m(Z)$,

    i.e. that for each variable in the set Ψ_1, its mapping is in the set Ψ_2, and that each different variable in Ψ_2 has a different mapping.

The main difference from this definition of subsumption to the traditional θ-subsumption is that in our definition, all variables in a set are considered different, i.e. a set of n variables will never subsume a set of less than n variables. This is interesting, since it eliminates the problem described in [44], and allows finite, complete and proper specialization refinement operators (although it does not resolve the problem for the case of the generalization operator as explained in Appendix C).

It is important to remark the difference between subsumption ($\sqsubseteq$) as defined here, as a relation that defines an informational content ordering, and classical θ-subsumption. A basic idea concerning feature terms is that they can be understood as partial descriptions, which allows us to have both instances and generalizations in the same representation language. The relation ordering partial descriptions ($\psi \sqsubseteq \psi'$) makes sense only if all that is true for ψ (all information contained in ψ) is also true for ψ' (the information contained in ψ' includes at least all information contained in ψ).

An example may clarify how subsumption ($\sqsubseteq$) but not θ-subsumption induce such an informational order upon partial descriptions. Consider three partial descriptions (ψ_1, ψ_2 and ψ_3) of a person such that they are equal in all but the *children* feature where they have (respectively) one, two, or three children. Clearly, subsumption ($\sqsubseteq$) captures the desired ordering among partial descriptions, namely $\psi_1 \sqsubset \psi_2 \sqsubset \psi_3$. However, θ-subsumption does not respect this order, since we can substitute the two variables of ψ_2 (representing two children) for the same variable in ψ_1 (representing one children) and thus ψ_2 would also θ-subsume ψ_1. Thus, θ-subsumption allows losing information, in this case about having two children as stated in ψ_2. Moreover, unification makes sense for our subsumption since, as expected when $\psi_1 \sqsubset \psi_2$, unifying a term with a subsumer yields the same term: $\psi_1 \sqcup \psi_2 = \psi_2$ – while this property does not hold for θ-subsumption.

Appendix B: Efficient Search for an Anti-Unification

Finding the set of all the anti-unifications of a set of given terms might be computationally expensive. However, if we are interested in finding just one anti-unification of the set of possible anti-unifications, an efficient algorithms exists.

Algorithm AntiUnification($T, \rho, \psi_0 = \bot$)
  ForEach ($\psi \in \rho(\psi_0)$) Do
    If ($\psi \not\sqsupseteq \psi_0 \wedge \forall \psi' \in T, \psi \sqsubseteq \psi'$) Then
      $\langle \psi_t, t \rangle$ = AntiUnification(T, ρ, ψ)
      Return $\langle \psi_t, t+1 \rangle$
  EndForEach
  Return $\langle \psi_0, 0 \rangle$
EndAlgorithm

Fig. 17 Algorithm to compute an anti-unification of a set of feature terms T using depth first search. It returns a pair containing both the anti-unification, and the length of the refinement path from $\bot$ to the anti-unification.

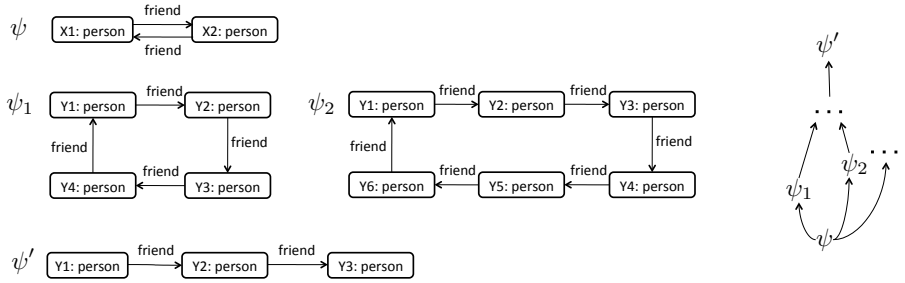


Fig. 18 A term ψ with a circular variable equality. This term has an infinite number of generalizations: ψ_1 and ψ_2 are two examples. The generalization operator γ_e generates ψ' to avoid infinite paths.

Given a refinement operators ρ that is *finite* and *complete* (see Section 3), Figure 17 shows an algorithm to compute one anti-unification of a set of feature terms T. The algorithm works as follows: the search starts by having $c_0 = \bot$ as initial candidate to be the anti-unification. At each step of the algorithm, a set $\rho(c_0)$ will be generated with the specialization refinements of the current candidate c_0 using the refinement operator ρ. If any of those refinements ψ subsumes all the terms in T, and it is more specific than the current candidate c_0 (this test is required since we do not require the refinement operator to be *proper*) then a recursive call is made using ψ as the candidate anti-unification. When none of the refinements of c_0 subsume all of the terms in T, we will know that c_0 is an anti-unification of all the terms in T.

Given that all the feature terms in T have a finite number of variables, it is possible to proof that only a finite number of refinements are required to reach an anti-unification from any starting feature term ψ, and in particular from $c_0 = \bot$. Moreover, the number of refinements required is linear as a function of the number of variables, features and the depth of the sort taxonomy.

The number of recursive calls represents the number of refinements required to compute the anti-unification, i.e. the length of the refinement path in the refinement graph from $\bot$ to the anti-unification. The algorithm returns a pair $\langle c_0, t \rangle$ with both the anti-unification and refinement path length.

Appendix C: Finite and Infinite Paths in a Refinement Graph

Languages $\mathcal{L}_c$ and $\mathcal{L}$, allowing circular variable equalities, define refinement graphs where there may be *infinite monotonic refinement paths*[8]. This means that starting from some terms, infinite sequences of terms, where each term is a generalization of the previous one, can be created. Let us again show this with an example.

[8] A refinement path is monotonic when all the steps in the path are a generalization, or all the steps in the path are a specialization.

Figure 18 shows a term ψ which contains a circular variable equality: it describes a person X_1 which has a friend X_2, and X_2 also has a friend, which is X_1, so it has a cycle of size 2. Let us show that circular variable equalities cause infinite generalization paths. The term ψ_1 in Figure 18 is a generalization of ψ ($\psi_1 \sqsubseteq \psi$), and has a cycle of size 4. ψ_1 subsumes ψ because with the mapping $m(Y_1) = X_1$, $m(Y_2) = X_2$, $m(Y_3) = X_1$ and $m(Y_4) = X_2$ all the conditions specified in Appendix A are satisfied. It is easy to see that a simple term with a cycle of size a where $a = k \times b$ (with a, b and k being natural numbers) subsumes a term with a cycle of size b. Thus, we can create an infinite generalization path by starting with ψ which has a cycle of size 2, continuing with a term with a cycle of size 4, then one of size 8, 16, 32, etc.

Moreover, two terms which have cycles of size a and b respectively do not subsume each other if a and b are not a multiple of each other. For example, the term ψ_2 in Figure 18 has a cycle of size 6 and it subsumes ψ, but not ψ_1. Thus, there are also infinite terms which are generalizations of ψ, but that do not subsume each other. This is why a circular variable equality can be removed in an infinite number of ways, as we mentioned in Section 3. This is also the reason for which the set of subsumers $G(\psi)$ is infinite when there are circular variable equalities.

This is a problem for implementing generalization refinement operators. For that reason, the generalization operator γ_e in Section 3 generates the term ψ' as a generalization of ψ but does not generate any of the terms ψ_1, ψ_2, etc. In this way, we avoid infinite paths, but at the cost of not being complete. In practice this is not a problem, since the disallowed generalizations are terms with new variables (w.r.t. the set of examples) and they are not required in the disintegration process where γ_e is used.

References

1. Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7(1):39–59, 1994.
2. Hassan Aït-Kaci. Description logic vs. order-sorted feature logic. In *Description Logics*, 2007.
3. Hassan Aït-Kaci and A. Podelski. Towards a meaning of LIFE. Technical Report 11, Digital Research Laboratory, 1992.
4. Hassan Aït-Kaci, Andreas Podelski, and Seth Copen Goldstein. Order-sorted feature theory unification. In *ILPS '93: Proc. 1993 International Symposium on Logic programming*, pages 506–524, Cambridge, MA, USA, 1993. MIT Press.
5. Hassan Aït-Kaci and Yutaka Sasaki. An axiomatic approach to feature term generalization. In *EMCL '01: Proceedings of the 12th European Conference on Machine Learning*, pages 1–12, London, UK, 2001. Springer.
6. Josep Lluís Arcos. *The NOOS representation language*. PhD thesis, Universitat Politècnica de Catalunya, 1997.
7. Eva Armengol and Enric Plaza. Relational case-based reasoning for carcinogenic activity prediction. *Artificial Intelligence Review*, 20(1-2):121–141, 2003.
8. Eva Armengol and Enric Plaza. Lazy learning for predictive toxicology based on a chemical ontology. In W. Dubitzky and F. Azuaje, editors, *Artificial Intelligence Methods and Tools for Systems Biology*, volume 5, pages 1–18. Springer, 2005.
9. Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
10. Ralph Bergmann and Armin Stahl. Similarity measures for object-oriented case representations. In *Advances in Case-Based Reasoning, EWCBR-98*, volume 1488 of *Lecture Notes in Artificial Intelligence*, pages 8–13. Springer Verlag, 1998.
11. Gilles Bisson. Learing in FOL with a similarity measure. In *Proceedings of AAAI 1992*, pages 82–87, 1992.
12. Katy Börner. Structural similarity as a guidance in case-based design. In *Topics in Case-Based Reasoning: EWCBR'93*, volume 837 of *Lecture Notes in Computer Science*, pages 197–208, 1993.
13. Bob Carpenter. *The Logic of Typed Feature Structures*, volume 32 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1992.

14. T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.
15. Claudia d'Amato, Steffen Staab, and Nicola Fanizzi. On the influence of description logics ontologies on conceptual similarity. In *EKAW '08, Proc. 16th Int. Conf. on Knowledge Engineering: Practice and Patterns*, volume 5268 of *Lecture Notes in Computer Science*, pages 48–63, 2008.
16. Thomas Dietterich, Pedro Domingos, Lise Getoor, Stephen Muggleton, and Prasad Tadepalli. Structured machine learning: the next ten years. *Machine Learning*, pages 3–23, 2008.
17. Werner Emde and Dietrich Wettschereck. Relational instance-based learning. In Lorenza Saitta, editor, *Proceedings 13th International Conference on Machine Learning*, pages 122–130. Morgan Kaufmann, 1996.
18. Nicola Fanizzi, Claudia d'Amato, and Floriana Esposito. Induction of optimal semi-distances for individuals based on feature sets. In *Proc. 2007 International Workshop on Description Logics*. CEUR-WS, 2007.
19. Nicola Fanizzi, Claudia D'Amato, and Floriana Esposito. DL-FOIL concept learning in description logics. In *ILP '08: Proceedings of the 18th Int. Conf. Inductive Logic Programming*, volume 5194 of *Lecture Notes in Computer Science*, pages 107–121, Berlin, Heidelberg, 2008. Springer.
20. Pedro A. González-Calero, Belén Díaz-Agudo, and Mercedes Gómez-Albarrán. Applying DLs for retrieval in case-based reasoning. In *In Proceedings of the 1999 Description Logics Workshop (DL'99)*, 1999.
21. C. Helma, R. King, S. Kramer, and A. Srinivasan. The predictive toxicology challenge 2000- 2001. *Bioinformatics*, 17:107–108, 2001.
22. Geoffrey E. Hinton. Learning distributed representations of concepts. In *Proceedings of 8th Annual Conf. Cognitive Science Society*, pages 1–12, 1986.
23. Tamás Horváth, Stefan Wrobel, and Uta Bohnebeck. Relational instance-based learning with lists and terms. *Machine Learning*, 43(1/2):53–80, 2001.
24. Alan Hutchinson. Metrics on terms and clauses. In *ECML '97: Proceedings of the 9th European Conference on Machine Learning*, volume 1224 of *Lecture Notes in Computer Science*, pages 138–145. Springer, 1997.
25. Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the Twentieth International Conference (ICML 2003)*, pages 321–328. AAAI Press, 2003.
26. P.R.J. van der Laag and S.-H. Nienhuys-Cheng. Subsumption and refinement in model inference. Discussion Paper 7, Erasmus University Rotterdam, Faculty of Economics, 1992.
27. James Larson and Ryszard S. Michalski. Inductive inference of VL decision rules. *SIGART Bulletin*, 63:38–44, 1977.
28. Nada Lavrač and Sašo Džeroski. *Inductive Logic Programming. Techniques and Applications*. Ellis Horwood, 1994.
29. Jens Lehmann and Pascal Hitzler. Foundations of refinement operators for description logics. In *Proceedings of the 17th International Conference on Inductive Logic Programming (ILP)*, volume 4894 of *Lecture Notes in Computer Science*, pages 161–174. Springer, 2008.
30. Jens Lehmann and Pascal Hitzler. A refinement operator based learning algorithm for the ALC description logic. In *Proceedings of the 17th International Conference on Inductive Logic Programming (ILP)*, volume 4894 of *Lecture Notes in Computer Science*, pages 147–160. Springer, 2008.
31. D. Michie, S. Muggleton, D. Page, and A. Srinivasan. To the international computing community: A new East-West challenge. Technical report, Oxford University Computing Laboratory, Oxford,UK, 1994.
32. Tom Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, 1982.
33. Santiago Ontañón and Enric Plaza. On similarity measures based on a refinement lattice. In D. Wilson and L. McGinty, editors, *Proc. 8th Int. Conf. on Case-Based Reasoning*, volume 5650 of *Lecture Notes in Computer Science*, pages 240–255. Springer, 2009.
34. Enric Plaza. Cases as terms: A feature term approach to the structured representation of cases. In M. Veloso and A. Aamodt, editors, *Case-Based Reasoning, ICCBR-95*, volume 1010 of *Lecture Notes in Artificial Intelligence*, pages 265–276. Springer, 1995.
35. Enric Plaza, Eva Armengol, and Santiago Ontañón. The explanatory power of symbolic similarity in case-based reasoning. *Artificial Intelligence Review*, 24(2):145–161, 2005.

36. Gordon D Plotkin. A note on inductive generalization. In *Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press, 1970.
37. Carl J. Pollard and M. Drew Moshier. Unifying partial descriptions of sets. In Philip P. Hanson, editor, *Information, Language and Cognition*, volume 1, pages 167–184. University of British Columbia Press, 1990.
38. J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.
39. John W. Raymond, C. John Blankley, and Peter Willett. Comparison of chemical clustering methods using graph- and fingerprint-based similarity measures. *Journal of Molecular Graphics and Modelling*, 21(5):421–433, 2003.
40. John W. Raymond, Eleanor J. Gardiner, and Peter Willett. Rascal: Calculation of graph similarity using maximum common edge subgraphs. *Computer J.*, 45(6):631–644, 2002.
41. Antonio A. Sánchez-Ruiz, Santiago Ontañón, Pedro A. González-Calero, and Enric Plaza. Measuring similarity in description logics using refinement operators. In *Proc. 19th Int. Conf. on Case-Based Reasoning*, volume 6880 of *Lecture Notes in Computers Science*, pages 289–303. Springer, 2011.
42. Ehud Y. Shapiro. Inductive inference of theories from facts. Technical Report 192, Dept. of Computer Science, Yale University, 1981.
43. Amos Tversky. Features of similarity. In *Psychological Review*, volume 84, pages 327–352, 1977.
44. Patrick R. J. van der Laag and Shan-Hwei Nienhuys-Cheng. Existence and nonexistence of complete refinement operators. In *ECML-94: Proceedings of the European Conference on Machine Learning*, pages 307–322. Springer New York, Inc., 1994.
45. Peter Willett, John M. Barnard, and Geoffrey M. Downs. Chemical similarity searching. *Journal of Chemical Information and Computer Sciences*, 38(6):983–996, 1998.