Measuring the similarity of labeled graphs

Pierre-Antoine Champin, Christine Solnon

LIRIS, bât. Nautibus, University of Lyon I
43 Bd du 11 novembre, 69622 Villeurbanne cedex, France
{champin,csolnon}@bat710.univ-lyon1.fr

Abstract. This paper proposes a similarity measure to compare cases represented by labeled graphs. We first define an expressive model of directed labeled graph, allowing multiple labels on vertices and edges. Then we define the similarity problem as the search of a best mapping, where a mapping is a correspondence between vertices of the graphs. A key point of our approach is that this mapping does not have to be univalent, so that a vertex in a graph may be associated with several vertices of the other graph. Another key point is that the quality of the mapping is determined by generic functions, which can be tuned in order to implement domain-dependant knowledge. We discuss some computational issues related to this problem, and we describe a greedy algorithm for it. Finally, we show that our approach provides not only a quantitative measure of the similarity, but also qualitative information which can prove valuable in the adaptation phase of CBR.

1 Introduction

Case based reasoning (CBR) relies on the hypothesis that similar problems have similar solutions. Hence, a CBR system solves a new problem by retrieving a similar one, for which a solution is known, then reusing that solution in the current context [1]. The retrieving phase requires an accurate similarity measure, so that cases are actually reusable whenever they are considered similar to the problem at hand.

In many situations, cases can be represented as a set of attribute-value pairs (usually called "vector cases"). Similarity between such cases can be defined in a rather straightforward way, as a weighted combination of each attribute-value pair similarity, and it can be efficiently computed [5]. However, more structured representations can be necessary for solving more complex problems, *e.g.*, feature terms [14], hierarchical decompositions [15], directed or non-directed labeled graphs [7, 13].

In this paper, we focus on directed labeled graphs, *i.e.*, directed graphs whose vertices and edges are associated with one or more labels, and consider the problem of characterizing and computing their similarity. Such labeled graphs provide a rich mean for modeling structured objects. In particular in the Ardeco project [4], from which this work is originated, we use them to represent design objects in a computer aided design (CAD) application. Figure 1 displays two design objects, borrowed from this application, and allows us to introduce some key points that should be addressed when measuring their similarity. Roughly speaking, one would say these two design objects are similar since the four beams a, b, c, d respectively correspond to the beams 1, 2, 3, 4, and the walls e and f correspond to 5. However, they are not absolutely similar: first, the beams have different shapes (I on the left and U on the right); besides, the number of walls is different from one case to the other (the wall 5 plays alone the role of the two walls e and f).

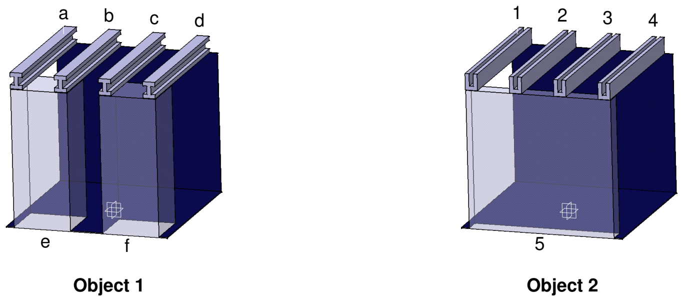


Fig. 1. Two similar design objects

This example first shows that, in order to measure the similarity between two objects, we must find a mapping which pairs their different components. Furthermore, this mapping should be the "best" one, i.e., the one that maps "similar" components as much as possible, where their similarity depends on the features they share, as well as the relations they have with other components. This example also shows that this mapping is not necessarily one-to-one: the role of wall 5 in object 2 is held by both walls e and f in object 1. To provide accurate comparison of complex structured objects, it is essential to allow such multiple mappings and to take them into account when defining a similarity measure. Finally, a similarity measure should not only be *quantitative*, indicating how much two objects are similar, but also *qualitative*, giving information about commonalities and differences between the two objects.

In the next section (2), we formally define labeled graphs, and illustrate their expressive power on the design example of figure 1. In section 3, we propose our general similarity measure based on the notion of mapping between graph vertices, and we show how application-dependant similarity knowledge can be implemented in our framework. Section 4 discusses some computational issues related to our similarity problem: we first study the tractability of a complete approach, and then describe an efficient greedy algorithm. In section 5, we discuss the benefits of our similarity measure in the reuse phase of CBR. Finally, we conclude by showing the genericity of this work with respect to other proposals for graph comparison, and we give further research directions.

2 Labeled graphs

2.1 Definitions and notations

A *labeled graph* is a directed graph such that vertices and edges are associated with labels. Without loss of generality, we shall assume that every vertex and edge is associated with at least one label: if some vertices (resp. edges) have no label, one can add an extra anonymous label that is associated with every vertex (resp. edge). More formally, given a finite set of vertex labels L_V, and a finite set of edge labels L_E, a labeled graph will be defined by a triple $G = \langle V, r_V, r_E \rangle$ such that:

- V is a finite set of vertices,
- $r_V \subseteq V \times L_V$ is the relation that associates vertices with labels, *i.e.*, r_V is the set of couples (v_i, l) such that vertex v_i has label l,
- $r_E \subseteq V \times V \times L_E$ is the relation that associates edges with labels, *i.e.*, r_E is the set of triples (v_i, v_j, l) such that edge (v_i, v_j) has label l. Note that from this edge relation r_E, one can define the set E of edges as $E = \{(v_i, v_j) | \exists l, (v_i, v_j, l) \in r_E\}$.

We respectively call the tuples from r_V and r_E, the *vertex features* and *edge features* of G. We then define the *descriptor* of a graph $G = \langle V, r_V, r_E \rangle$ as the set of all its features: $descr(G) = r_V \cup r_E$. This descriptor completely describes the graph and will be used to measure the similarity between two graphs. Finally, a *similarity problem* is defined by two graphs $G_1 = \langle V_1, r_{V_1}, r_{E_1} \rangle$ and $G_2 = \langle V_2, r_{V_2}, r_{E_2} \rangle$ that have disjoint sets of vertices, *i.e.*, $V_1 \cap V_2 = \emptyset$.

2.2 Example

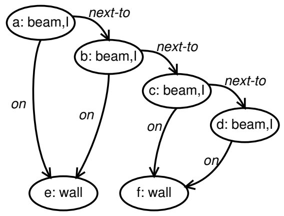
Let us consider again the two design objects displayed in figure 1. To represent these two objects with labeled graphs, we first define the sets of vertex and edge labels that respectively characterize their components and relationships between components:

$$L_V = \{\, beam, I, U, wall \,\}$$
$$L_E = \{\, on, next\_to \,\}$$

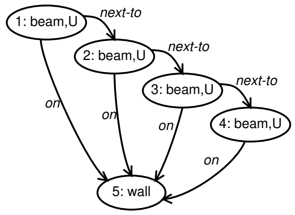
Given these sets of labels, the two design objects of figure 1 may be represented by the two labeled graphs of figure 2 (the left part of the figure displays their graphical representations; the right part gives their associated formal definitions by means of vertex and edge features).

Note that vertices a, b, c and d have the two labels *beam* and I, whereas vertices $1, 2, 3$ and 4 have the two labels *beam* and U. This allows us to express that the corresponding objects share the feature "being a beam", even though their shapes are different.

More generally, the fact that edges and vertices may have more than one label is used to express inheritance (specialization) relationships by inclusion of sets of labels. With such an inclusive expression of inheritance, the similarity of two vertices, or edges, can be defined by means of their common labels, corresponding to their common ancestors in the inheritance hierarchy.



$$G_1 = \langle \quad V_1 = \{ \, a, b, c, d, e, f \, \}$$
$$r_{V_1} = \{ \, (a, beam), (b, beam), (c, beam), (d, beam), (a, I), (b, I), (c, I), (d, I), (e, wall), (f, wall) \, \}$$
$$r_{E_1} = \{ \, (a, b, next\_to), (b, c, next\_to), (c, d, next\_to), (a, e, on), (b, e, on), (c, f, on), (d, f, on) \, \} \, \rangle$$



$$G_2 = \langle \quad V_2 = \{ \, 1, 2, 3, 4, 5 \, \}$$
$$r_{V_2} = \{ \, (1, beam), (2, beam), (3, beam), (4, beam), (1, U), (2, U), (3, U), (4, U), (5, wall) \, \}$$
$$r_{E_2} = \{ \, (1, 2, next\_to), (2, 3, next\_to), (3, 4, next\_to), (1, 5, on), (2, 5, on), (3, 5, on), (4, 5, on) \, \} \, \rangle$$

Fig. 2. Labeled graphs $G1$ and $G2$ describing objects 1 and 2 of figure 1

3 A measure of similarity for labeled graphs

3.1 Mapping of labeled graphs

To measure the similarity of two labeled graphs, one first has to find a mapping that matches their vertices in order to identify their common features. More precisely, a mapping between two labeled graphs $G_1 = \langle V_1, r_{V_1}, r_{E_1} \rangle$ and $G_2 = \langle V_2, r_{V_2}, r_{E_2} \rangle$, such that $V_1 \cap V_2 = \emptyset$, is a relation $m \subseteq V_1 \times V_2$. Note that such a mapping can associate to each vertex of one graph, zero, one, or more vertices of the other graph. By extension, we shall use the functional notation $m(v)$ to denote the set of vertices that are associated with a vertex v by the relation m, *i.e.*,

$$\forall v_1 \in V_1, \quad m(v_1) \doteq \{v_2 \in V_2 | \, (v_1, v_2) \in m\}$$
$$\forall v_2 \in V_2, \quad m(v_2) \doteq \{v_1 \in V_1 | \, (v_1, v_2) \in m\}$$

Even though we may use a functional notation, one should keep in mind that a mapping is a relation, *i.e.*, a set of couples of vertices, so that we can apply set operators on mappings.

Example: For the labeled graphs of figure 2, one can define, *e.g.*, the two following mappings

$$m_A = \{(a, 1), (b, 2), (c, 3), (d, 4), (e, 5), (f, 5)\}$$
$$m_B = \{(a, 1), (b, 2), (a, 3), (b, 4)), (e, 5)\}$$

The first mapping m_A respectively matches a, b, c and d to 1, 2, 3 and 4, and both e and f to 5; in this mapping, the set of vertices associated with, *e.g.*, 5 will be noted $m_A(5) = \{e, f\}$. The second mapping m_B matches both 1 and 3 to a, both 2 and 4 to b, and e to 5; in this mapping, the set of vertices associated with f will be noted $m_B(f) = \emptyset$.

3.2 Similarity with respect to a mapping

In order to measure the similarity between two objects, it is intuitive and usual to compare the amount of features which are common to both objects, to the total amount of their features [9]. In the field of psychology, Tversky [17] has demonstrated the cognitive plausibility of this intuition, giving the following mathematical model of a similarity measure between two objects a and b described respectively by two sets of features A and B:

$$sim_{\text{Tversky}}(a,b) = \frac{f(A \cap B)}{f(A \cup B) - \alpha f(A - B) - \beta f(B - A)}$$

where f is a non-decreasing positive function, monotonic with respect to inclusion, and α and β are positive values, allowing to define *asymmetrical* similarity measures. We shall set them to zero from now on, since in our case, their role can be held by function f; this will be discussed in section 3.5.

A labeled graph G, as we defined it in section 2, is described by the set $descr(G)$ of all its vertex and edge features. Hence, the similarity of two different graphs $G_1 = \langle V_1, r_{V_1}, r_{E_1} \rangle$ and $G_2 = \langle V_2, r_{V_2}, r_{E_2} \rangle$ depends on both the common features of $descr(G_1)$ and $descr(G_2)$, and the set of all their features. However, since $V_1 \cap V_2 = \emptyset$, the intersection of the two graphs descriptors will always be empty. We must instead compute this intersection *with respect* to a given mapping m, which pairs vertices from G_1 with vertices from G_2:

$$\begin{aligned} descr(G_1) \sqcap_m descr(G_2) \doteq\ & \{(v,l) \in r_{V_1} |\ \exists v' \in m(v),\ (v',l) \in r_{V_2}\} \\ \cup\ & \{(v,l) \in r_{V_2} |\ \exists v' \in m(v),\ (v',l) \in r_{V_1}\} \\ \cup\ & \{(v_i,v_j,l) \in r_{E_1} | \\ & \quad \exists v'_i \in m(v_i), \exists v'_j \in m(v_j)\, (v'_i,v'_j,l) \in r_{E_2}\} \\ \cup\ & \{(v_i,v_j,l) \in r_{E_2} | \\ & \quad \exists v'_i \in m(v_i), \exists v'_j \in m(v_j)\, (v'_i,v'_j,l) \in r_{E_1}\} \end{aligned}$$

This set contains all the features from both G_1 and G_2 whose vertices or edges are matched, according to m, to at least one vertex or edge with the same feature.

Given this definition of commonalities between G_1 and G_2, we can apply Tversky's formula to define the similarity between G_1 and G_2 with respect to a mapping m:

$$sim1_m(G_1,G_2) = \frac{f(descr(G_1) \sqcap_m descr(G_2))}{f(descr(G_1) \cup descr(G_2))} \tag{1}$$

Example: Let us consider the two labeled graphs, and their associated descriptors, of figure 2. The intersections of these descriptors, with respect to the mappings m_A and m_B proposed in section 3.1, are:

$$\begin{aligned}
descr(G_1) \sqcap_{m_A} descr(G_2) = \quad & descr(G_1) \cup descr(G_2) \\
& -\{\ (a, I), (b, I), (c, I), (d, I), \\
& \quad (1, U), (2, U), (3, U), (4, U)\ \} \\
descr(G_1) \sqcap_{m_B} descr(G_2) = \{\ & (a, beam), (b, beam), (e, wall), \\
& (1, beam), (2, beam), (3, beam), (4, beam), (5, wall), \\
& (a, b, next_to), \\
& (1, 2, next_to), (3, 4, next_to), \\
& (a, e, on), (b, e, on), \\
& (1, 5, on), (2, 5, on), (3, 5, on), (4, 5, on)\ \}
\end{aligned}$$

3.3 Evaluation of splits

The definition of $sim1_m$ in equation 1 is not entirely satisfying. For example, let us consider the two labeled graphs of figure 2 and suppose the beams of both objects have the same shape. In this case, the mapping $m_A = \{(a,1), (b,2), (c,3), (d,4), (e,5), (f,5)\}$ would match every feature of each graph to a perfectly similar one from the other graph, so that $descr(G_1) \sqcap_{m_A} descr(G_2) = descr(G_1) \cup descr(G_2)$ and $sim1_{m_A}(G_1, G_2) = 1$. This would mean that the two graphs are perfectly similar, whatever the chosen function f. Nevertheless, one might be annoyed that vertex 5 is paired with two vertices (e and f), and prefer another mapping such as, *e.g.*, $m_3 = \{(a,1), (b,2), (c,3), (d,4), (e,5)\}$, even though some features were not matched. We do not intend to mean that one mapping is strictly better than the other one, nor that only isomorphic graphs or subgraphs should be matched. Indeed, this obviously depends on the application domain. On the contrary, we want the model to be tunable in order to allow for either interpretation.

Therefore, to enhance the model, we introduce the function *splits* which returns the set of split vertices (*i.e.*, vertices paired with more than one vertex) together with the set s_v of vertices they are paired with. We also define an extended inclusion operator $\sqsubseteq$ on those sets, by considering inclusion on sets of paired vertices s_v.

$$splits(m) \doteq \{(v, s_v) |\ v \in V_1 \cup V_2,\ s_v = m(v),\ |m(v)| \geq 2\}$$

$$splits(m_1) \sqsubseteq splits(m_2) \Leftrightarrow \forall (v, s_v) \in splits(m_1),\ \exists (v, s'_v) \in splits(m_2),\ s_v \subseteq s'_v$$

For example, the mappings proposed in section 3.1 have the following splits: $splits(m_A) = \{\ (5, \{e, f\})\ \}$ and $splits(m_B) = \{\ (a, \{1, 3\}),\ (b, \{2, 4\})\ \}$.

We can now modify equation 1 so as to take the value of splits into account:

$$sim_m(G_1, G_2) = \frac{f(descr(G_1) \sqcap_m descr(G_2)) - g(splits(m))}{f(descr(G_1) \cup descr(G_2))} \tag{2}$$

where function g is defined with the same properties as f: it is positive, monotonic and non-decreasing with respect to extended inclusion $\sqsubseteq$. Hence the similarity will be decreasing as the number of splits is increasing.

3.4 Maximum similarity of labeled graphs

We have defined the similarity of two graphs with respect to a given mapping between the graphs vertices. Now, we can define the *maximum similarity* of two graphs G_1 and G_2 as the similarity induced by the best mapping:

$$sim(G_1, G_2) = \max_{m \subseteq V_1 \times V_2} \frac{f(descr(G_1) \sqcap_m descr(G_2)) - g(splits(m))}{f(descr(G_1) \cup descr(G_2))} \quad (3)$$

Note that, if the similarity value induced by a given mapping m (sim_m) is always between 0 and 1 when m does not contain any split (*i.e.*, when every vertex is associated with at most one vertex), it may become negative when some vertices are associated with more than one vertex: this occurs if the weight of splits, defined by g, is higher than the weight of common features, defined by f. However, in any cases, the maximum similarity sim will always be between 0 and 1 since the similarity induced by the empty mapping $m = \emptyset$ is null.

Note also that determining which of the mappings between G_1 and G_2 is the “best”, actually completely depends on functions f and g that respectively quantify vertex and edge features and splits. As a consequence, those functions must be carefully chosen depending on the application domain. This will be discussed in section 3.5.

3.5 Similarity knowledge

The notions presented so far about similarity measures are quite generic. However, in order to be accurate, an actual similarity measure has to take into account similarity knowledge which is of course application dependant. The functions f and g used in the above definitions are the place where such application dependant knowledge can be implemented. An easy way of defining them while ensuring their being monotonic, is to define them as a sum of positive (or null) weights assigned to each element of the measured sets (namely, features and splits), i.e., given a set of vertex and edge features F and a set of splits S,

$$f(F) = \sum_{(v,l) \in F} weight_{fV}(v, l) + \sum_{(v_1, v_2, l) \in F} weight_{fE}(v_1, v_2, l)$$
$$g(S) = \sum_{(v, s_v) \in S} weight_g(v, s_v)$$

Without much domain knowledge, one could assign the same weight to every feature (that is, f is the cardinality function). Every split (v, s_v) could receive the same weight or a weight proportional to the cardinality of s_v.

If *generic* knowledge is available, the weight of a feature could be defined on the basis of its label only: one could thus represent the facts that being a beam is more significant than being U-shaped, or that a composition relation is more significant than a “next-to” relation, for example.

Finally, if specific or *contextual* knowledge is available, one could assign a specific weight to each feature or split. For example, one could express the facts that it is more significant for a to be I-shaped than it is for b, that it is more significant for a and b to be in relation than it is for b and c, or that it is less annoying for a to be split into 1 and 2 than it is for c.

One may be concerned that we constrained parameters α and β in Tversky's formula to be zero. Those parameters make it possible to define asymmetrical similarity measures, which is often considered a desirable property in CBR. However, the definition of sim enables such asymmetrical similarity measures: features and splits can be weighted differently depending on which graph they belong to. For example, assigning a null weight to any feature of G_2 allows to measure how much G_1 "matches" (or "fits into") G_2.

4 Computing the maximum similarity of labeled graphs

Given two labeled graphs G_1 and G_2, we now consider the problem of computing their maximum similarity, *i.e.*, finding a mapping m that maximizes formula 2. One should note that the denominator $f(descr(G_1) \cup descr(G_2))$ of this formula does not depend on the mapping. Indeed, this denominator is introduced to normalize the similarity value between zero and one. Hence, to compute the maximum similarity between two graphs G_1 and G_2, one has to find the mapping m that maximizes the score function

$$score(m) = f(descr(G_1) \sqcap_m descr(G_2)) - g(splits(m))$$

This problem is highly combinatorial. Indeed, it is more general than, *e.g.*, the subgraph isomorphism problem[1] which is known to be NP-complete [12]. In this section, we first study the tractability of a complete search, and then propose a greedy incomplete algorithm for it.

4.1 Tractability of a complete search

The search space of the maximum similarity problem is composed of all different mappings —all subsets of $V_1 \times V_2$— and it contains $2^{|V_1|*|V_2|}$ states. This search space can be structured in a lattice by the set inclusion relation, and it can be explored in an exhaustive way with a "branch and bound" approach. Such a complete approach is actually tractable if there exists a "good" bounding function that can detect as soon as possible when a node can be pruned, *i.e.*, when the score of all the nodes that can be constructed from the current node is worse than the best score found so far. In our case, the potential successors of the node associated with a partial mapping $m \subset V_1 \times V_2$ are all the mappings that are supersets of m. However, the score function is not monotonic with respect to set inclusion, *i.e.,* the score of a mapping may either increase or decrease when one adds a new couple to it. Indeed, this score is defined as a difference between

[1] We do not present the proof here, since it involves some technical tricks which would require more space than available.

a function of the common features and a function of the splits, and both sides of this difference may increase when adding a couple to a mapping. More precisely, one can show that for all mappings m and m' such that $m \subseteq m'$,

$$\begin{aligned} & descr(G_1) \sqcap_m descr(G_2) \subseteq descr(G_1) \sqcap_{m'} descr(G_2) \\ \text{and} \quad & splits(m) \sqsubseteq splits(m') \end{aligned}$$

and therefore, since the f and g functions are monotonic,

$$\begin{aligned} & f(descr(G_1) \sqcap_m descr(G_2)) \leq f(descr(G_1) \sqcap_{m'} descr(G_2)) \\ \text{and} \quad & g(splits(m)) \leq g(splits(m')) \end{aligned}$$

However, we can use the fact that the intersection of graph features is bounded by the set of all graph features to define a bounding function. Indeed, for every mapping m, one can trivially show that

$$descr(G_1) \sqcap_m descr(G_2) \subseteq descr(G_1) \cup descr(G_2)$$

and, as f and g are monotonic, for every mapping m' such that $m' \supseteq m$,

$$\begin{aligned} score_{m'}(G_1, G_2) &= f(descr(G_1) \sqcap_{m'} descr(G_2)) - g(splits(m')) \\ &\leq f(descr(G_1) \cup descr(G_2)) - g(splits(m)) \end{aligned}$$

As a consequence, one can prune the node associated with a matching m if $f(descr(G_1) \cup descr(G_2)) - g(splits(m))$ is smaller or equal to the score of the best mapping found so far. In this case, all the nodes corresponding to mappings that are supersets of m will not be explored as their score cannot be higher than the best score found so far.

A first remark about this bounding function is that its effectiveness in reducing the search tree highly depends on the relative "weights" of functions f and g: the higher the weight of splits, the more nodes can be pruned. Actually, this bounding function is generic, and it can be applied to any kind of labeled graphs, with any f and g functions (provided that they are monotonic). More accurate bounding functions could be defined when considering more specific cases.

Also, one may introduce *ad-hoc* rules to reduce the search space. In particular, one can remove from the search space every mapping that contains a couple (v_i, v_j) such that v_i and v_j do not have any common features (vertex features, but also features of edges starting from or ending to v_i and v_j). Further more, during the exploration, one can remove from the partial search tree the root of which is a mapping m, every mapping $m' \supseteq m$ containing a couple (v_i, v_j) such that all the common features of v_i and v_j already belong to $descr(G_1) \sqcap_m descr(G_2)$.

Finally, the tractability of any complete branch and bound approach strongly depends on ordering heuristics, which determine an order for developing the nodes of the search tree: a good ordering heuristic allows the search to quickly find a good mapping, the score of which is high, so that more nodes can be cut. We describe in the next subsection a greedy algorithm that uses such ordering heuristics to quickly build a "good" mapping.

4.2 Greedy algorithm

Figure 3 describes a greedy algorithm that introduces ordering heuristics to build a mapping m: the algorithm starts from the empty mapping, and iteratively adds to this mapping a couple of vertices that most increases the score function. At each step, this set of candidate couples that most increase the score function — called *cand*— often contains more than one couple. To break ties between them, we look ahead the potentiality of each candidate $(u_1, u_2) \in cand$ by taking into account the features that are shared by edges starting from (resp. ending to) both u_1 and u_2 and that are not already in $descr(G_1) \sqcap_{m\cup\{(u_1,u_2)\}} descr(G_2)$. Hence, we select the next couple to enter the mapping within the set $cand'$ of couples whose looked-ahead common edge features maximize f.

function Greedy($G_1{=}\langle V_1, r_{V_1}, r_{E_1}\rangle$, $G_2{=}\langle V_2, r_{V_2}, r_{E_2}\rangle$) **returns** a mapping $m \subseteq V_1{\times}V_2$
  $m \leftarrow \emptyset$
  $best_m \leftarrow \emptyset$
  loop
    $cand \leftarrow \{(u_1, u_2) \in V_1{\times}V_2 - m \mid score(m \cup \{(u_1, u_2)\})$ is maximal$\}$
    $cand' \leftarrow \{(u_1, u_2) \in cand \mid f(look\_ahead(u_1, u_2))$ is maximal$\}$
      where
$$
\begin{aligned}
look\_ahead(u_1, u_2) = &\{(u_1, v_1, l) \in r_{E_1} \mid \exists v_2 \in V_2, (u_2, v_2, l) \in r_{E_2}\\
&\cup \{(u_2, v_2, l) \in r_{E_2} \mid \exists v_1 \in V_1, (u_1, v_1, l) \in r_{E_1}\\
&\cup \{(v_1, u_1, l) \in r_{E_1} \mid \exists v_2 \in V_2, (v_2, u_2, l) \in r_{E_2}\\
&\cup \{(v_2, u_2, l) \in r_{E_2} \mid \exists v_1 \in V_1, (v_1, u_1, l) \in r_{E_1}\\
&- descr(G_1) \sqcap_{m\cup\{(u_1,u_2)\}} descr(G_2)
\end{aligned}
$$
  exit loop when $\forall (u_1, u_2) \in cand'$, $score(m \cup \{(u_1, u_2)\}) \leq score(m)$ and $look\_ahead(u_1, u_2) = \emptyset$
    choose randomly one couple (u_1, u_2) in $cand'$
    $m \leftarrow m \cup \{(u_1, u_2)\}$
    if $score(m) > score(best_m)$ **then** $best_m \leftarrow m$
  end loop
  return $best_m$

Fig. 3. Greedy search of a mapping

Let us consider for example the two graphs of figure 2, and let us suppose that the f and g functions are both defined as the set cardinality function. At the first iteration, when the current mapping m is empty, the set of couples that most increase the score function contains every couple that matches two vertices sharing one vertex label, *i.e.*, $cand = \{a, b, c, d\} \times \{1, 2, 3, 4\} \cup \{e, f\} \times \{5\}$. Within this set of candidates, $(b, 2)$ (resp. $(c, 3)$) has $3+3$ potential common edge features, so that $f(look\_ahead(b, 2)) = f(look\_ahead(c, 3)) = 6$; also $(e, 5)$ and $(f, 5)$ have 6 potential common edge features (the 4 edges ending to 5 plus the 2 edges ending to e or f); other candidates all have a smaller number of potential common edge features. Hence, the greedy algorithm will randomly select one couple within the set $cand' = \{(b, 2), (c, 3), (e, 5), (f, 5)\}$. Let us suppose now that the selected couple is, *e.g.*, $(e, 5)$. Then, at the second iteration, *cand* will contain $(a, 1)$ and $(b, 2)$, which both increase the score function of $2+2$, whereas $cand'$ will only contain $(b, 2)$ as it has more potential common edge features than $(a, 1)$. Hence, the next couple to enter the mapping will be $(b, 2)$. At the third iteration, *cand* will only contain $(a, 1)$, which increases the score function of $3+3$, so that $(a, 1)$ will enter the mapping... and so on.

This greedy algorithm stops iterating when every couple neither directly increases the score function nor has looked-ahead common edge features. Note that the score of the mapping m under construction may decrease from one step to another: this occurs when the couple that enters the mapping introduces more new splits than new common features, but it has looked-ahead common edge features, so that the score function is expected to increase at the next iteration. Hence, the algorithm returns the best computed mapping —$best_m$— since the beginning of the run.

This greedy algorithm has a polynomial time complexity of $\mathcal{O}((|V_1| \times |V_2|)^2)$, provided that the computation of the f and g functions have linear time complexities with respect to the size of the mapping (note that the "look_ahead" sets can be computed in an incremental way). As a counterpart of this rather low complexity, this algorithm never backtracks and is not complete. Hence, it may not find the best mapping, the score of which is maximal; moreover, even if it actually finds the best mapping, it cannot be used to prove its optimality. Note however that, since this algorithm is not deterministic, we may run it several times for each case, and keep the best found mapping.

4.3 Discussion

Both algorithms have been implemented in C++. Generally speaking, first experiments have shown us that the complete branch and bound approach can be applied on small graphs only (up to 10 vertices in the general case), even though it performs better when increasing the relative weight of splits with respect to the weights of the common features. Experiments have also confirmed that the greedy algorithm is actually efficient, computing mappings for graphs with 50 nodes and 150 edges in a few seconds of CPU time.

Actually, tractability is all the more critical in CBR that the target case must be compared to numerous cases from the case base. Usually, a first pass filters out a majority of cases, then the most promising candidates are precisely compared. The filtering pass can use a less accurate but efficient algorithm [15], like the greedy algorithm presented here. It can also limit the comparison to representative sub-graphs [10], or even to a vector of characteristics [6]. In the ARDECO implementation, we plan to use a one-pass of our greedy algorithm to select from the case base the most promising candidates. Then, for these most promising candidates, we plan to run several more times the greedy algorithm, trying to find better mappings. As discussed in section 6, we shall further enhance our greedy algorithm by integrating some local search mechanisms.

5 Similarity and adaptation

5.1 Reusing qualitative similarity information

One of the goals pursued in this work was to provide not only a quantitative similarity measure, but also the *qualitative* information used to compute this measure. Indeed, from the best mapping m —the one that maximizes the two graphs similarity— one can define the set of *differences* between these two graphs as the set of both split vertices and features which are not shared by the graphs:

$$\mathit{diff}(G1, G2) = \mathit{splits}(m) \cup (\mathit{descr}(G1) \cup \mathit{descr}(G2)) - (\mathit{descr}(G1) \sqcap_m \mathit{descr}(G2))$$

We believe that this set of differences can be valuable when reusing a case. For example, the ARDECO assistant aims at helping designers (using a CAD application) to reuse their experience, represented as *design episodes* [4]. Episodes are composed of an initial state and a final state, both represented by labeled graphs. Reusability of an episode is estimated according to the similarity between its initial state and the current state of the designer's work. Adaptation is performed by transforming the current state into a new state which has the same differences with the final state than the current state has with the initial state (cf. figure 4).

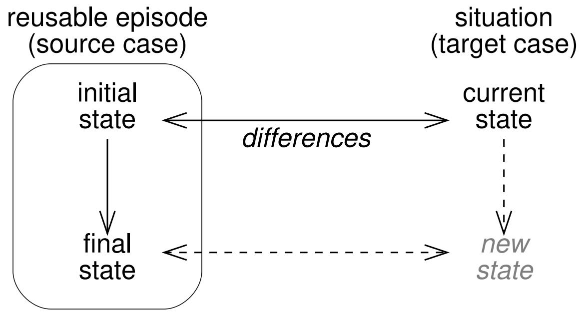


Fig. 4. Adapting a design episode in ARDECO

This approach can be compared to *similarity paths* proposed by [8]: the similarity between two cases is estimated by finding a path between them in the problem space, and this path is translated in the solution space in order to follow it "backward" during adaptation. This is another example of eliciting and reusing qualitative similarity information.

In our approach, the computed set of differences $\mathit{diff}(G1, G2)$ can be seen as a set of elementary operations (feature addition/deletion and vertex splitting/merging) which are reversible. Our model relies however on the assumption that such elementary operations are independent and unordered, while steps in similarity paths are strictly ordered and global (each of them can affect the whole graph).

5.2 Adaptation guided similarity

In the introduction, we underlined the fact that remembering and reusing are tightly related. Therefore from the point of view of CBR, a similarity measure is accurate if the cases considered similar are actually reusable. This has been pointed out by several authors [15, 8], the latter even prove that their retrieval mechanism (the similarity paths mentioned above) is correct and complete with respect to adaptation (meaning that all and only adaptable cases are retrieved).

It seems therefore important to emphasize again on the design of functions f and g. It has been discussed in section 3.5 how similarity knowledge can be implemented in these functions, but such knowledge must be consistent with adaptation knowledge: a feature must be all the more weighted that its addition (resp. deletion) is expensive or difficult in the adaptation process. Hence general purpose similarities as proposed by [9] are not necessarily adequate to CBR.

6 Conclusion

We have proposed a representation model for cases as labeled directed graphs, where vertices and edges may have more than one label. We have defined a similarity measure on such graphs, and discussed two approaches for computing that measure: the complete branch and bound algorithm appears to be intractable in the very general case, but an efficient greedy algorithm is proposed. This model is suited to CBR for it enables flexible modelling of similarity knowledge, and provides qualitative similarity information which can prove useful for adaptation.

6.1 A generic model

Graphs are versatile representation tools, that have been used and studied in a wide range of application domains. Comparing two graphs is an important problem which has been tackled under various modalities: are two graphs identical (problem of graph isomorphism), is one of them more general than the other (problem of subsumption), to what extent (quantitatively or qualitatively) are they different ?

Most of these graph comparison problems can be addressed by our similarity measure in a very straightforward way. For instance, isomorphism is trivially the search for a perfect mapping (with a similarity of 1) where f and g assign any non-null value to every feature and split. Subgraph isomorphism has already been discussed in section 4, and can easily be modelled, as well as partial graph or subgraph isomorphism. It has also been discussed in section 2 how multiple labels can be used to represent hierarchically structured types of vertex or edge. Determining if a graph G_1 subsumes a graph G_2, as does the projection algorithm [16], can be achieved by ignoring (*i.e.*, assigning a null weight to) every feature and every split from G_2, and looking for a similarity of 1. On the other hand, using a symmetrical similarity measure and considering similarity values lower than 1 corresponds to looking for the most specific subsumer of G_1 and G_2, as does the anti-unification algorithm [14].

In the field of knowledge representation, exact subsumption is sometimes too constraining —for ill-designed knowledge basis, or during incremental design. Approximative subsumption measures have been proposed [2, 18] to address this kind of problem. In the field of image recognition, [11] proposed an algorithm of fuzzy-projection, by extending the conceptual graphs formalism. Other theoretical work cope with error-tolerant isomorphism [3], computing similarity on the basis of how many elementary transformations are necessary to make the graphs isomorphic. However, all the above propositions miss some interesting features of our model, *e.g.*, approximate edge matching (most focus on vertices), handling splits, flexible tuning.

It is also worth mentioning the recent work of Petrovic *et al.* [13] about graph retrieval in the field of CBR. Their approach is quite similar to ours —it expresses similarity of a mapping between vertices, and looks for the maximizing mapping— but it is lacking some flexibility with regard to our requirements: splits are not allowed (mappings are univalent functions), graph elements have exactly one label each, and furthermore, function f is pre-defined and corresponds to the cardinality function. However, they went further in addressing tractability issues, by implementing an efficient Tabu search.

6.2 Further work

The genericity of our model explains to a large extent its complexity and the induced tractability issue. We plan to focus in three research directions to address that issue.

First, we shall further explore heuristic approaches. More work can be done on the *look_ahead* function in the greedy algorithm, in order to find a good trade-off between accurate predictions and efficiency. Other incomplete approaches, such as Tabu search, can also be used in order to refine results of the greedy algorithm by locally exploring the search space around the constructed mapping.

Second, in situations where graphs have remarkable properties, tractability issues are usually addressed by using *ad hoc* heuristics. We plan to study such heuristics from less generic approaches, in order to be able to integrate them, either by tuning f and g functions or by other means when necessary.

Finally, we are particularly interested in systems that aim at assisting the user (rather than standalone systems). In such situations, the retrieving and reusing phase of CBR should be performed interactively. This is possible thanks to the qualitative information elicited by our approach (mapping, commonalities and differences). Not only can interaction allow users to guide the system, thus reducing combinatorial complexity; but user guidance can also provide contextual knowledge about specific cases, which will make the system more knowledgeable and accurate for future uses.

Acknowledgment: We would like to thank Sébastien Sorlin for enriching discussions on this work, and the implementation of the two presented algorithms.

References

1. Agnar Aamodt and Enric Plaza. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*, 7(1):39–59, 1994.
2. Gilles Bisson. *Why and How to Define a Similarity Measure for Object Based Representation Systems*, pages 236–246. IOS Press, Amsterdam (NL), 1995.
3. Horst Bunke. Error Correcting Graph Matching: On the Influence of the Underlying Cost Function. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 21(9):917–922, 1999.
4. Pierre-Antoine Champin. *Modéliser l'expérience pour en assister la réutilisation : de la Conception Assistée par Ordinateur au Web Sémantique*. Thèse de doctorat en informatique, Université Claude Bernard, Lyon (FR), 2002.
5. Edwin Diday. *Éléments d'analyse de données*. Dunod, Paris (FR), 1982.
6. Christophe Irniger and Horst Bunke. Graph Matching: Filtering Large Databases of Graphs Using Decision Trees. In *IAPR-TC15 Workshop on Graph-based Representation in Pattern Recognition*, pages 239–249, 2001.
7. Jean Lieber and Amedeo Napoli. Using Classification in Case-Based Planning. In *proceedings of ECAI 96*, pages 132–136, 1996.
8. Jean Lieber and Amedeo Napoli. Correct and Complete Retrieval for Case-Based Problem-Solving. In *proceedings of ECAI 98*, pages 68–72, 1998.
9. Dekang Lin. An Information-Theoretic Definition of Similarity. In *proceedings of ICML 1998, Fifteenth International Conference on Machine Learning*, pages 296–304. Morgan Kaufmann, 1998.
10. Bruno T. Messmer and Horst Bunke. A New Algorithm for Error-Tolerant Subgraph Isomorphism Detection. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 20(5):493–504, 1998.
11. Philippe Mulhem, Wee Kheng Leow, and Yoong Keok Lee. Fuzzy Conceptual Graphs for Matching Images of Natural Scenes. In *proceedings of IJCAI 01*, pages 1397–1404, 2001.
12. Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, Boston, MA (US), 1994.
13. Sanja Petrovic, Graham Kendall, and Yong Yang. A Tabu Search Approach for Graph-Structured Case Retrieval. In *proceedings of STAIRS 02*, volume 78 of *Frontiers in Artificial Intelligence and Applications*, pages 55–64. IOS Press, 2002.
14. Enric Plaza. Cases as terms: A feature term approach to the structured representation of cases. In *proceedings of ICCBR 95*, number 1010 in LNCS, pages 265–276. Springer Verlag, 1995.
15. Barry Smyth and Matk T. Keane. Retrieval and Adaptation in Dj Vu, a Case-Based Reasoning System for Software Design. In *AAAI Fall Symposium on Adaptation of Knowledge for Reuse*. AAAI, 1995.
16. John Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. PWS Publishing Co., 1999.
17. Amos Tversky. Features of Similarity. *Psychological Review*, 84(4):327–352, 1977.
18. Petko Valtchev. *Construction automatique de taxonomies pour l'aide à la représentation de connaissances par objets*. Thèse de doctorat en informatique, Université Joseph Fourier, Grenoble (FR), 1999.