Structured Cases, Trees and Efficient Retrieval

Francesco Ricci and **Luca Senter**

Istituto per la Ricerca Scientifica e Tecnologica
via Sommarive
38050 Povo (TN)
Italy
email: {ricci,senter}@itc.it

Abstract. A set of efficient algorithms for case retrieval from a case base of trees labeled on both nodes and edges is proposed. They extend the best known algorithm for solving the subtree-isomorphism problem. A branch and bound technique and a general definition of similarity between trees is considered. Both the case structure, i.e. the graph structure, and his semantic part, the labels, is taken into account when evaluating similarity. The comparison with a conventional state-space search algorithm on randomly generated case bases, shows that significant speed up can be obtained.

1 Introduction

Given a problem in the form of a partially defined query case, a case-based reasoning system (CBR) starts the solution process retrieving from the memory a case that is both *similar* to the query and *reusable* to solve the given problem. Similarity is usually provided by a distance metric.

Therefore, designing efficient retrieval algorithms is a key issue in CBR. In rather simple application domains a case can be represented by a vector of attributes. This representation originates from pattern recognition and is widely used in CBR systems mainly because retrieval becomes simple and fast.

In more complex domains, e.g. planning [9], vision [14], software engineering [8] and design [6], the vector representation is not enough expressive for describing real cases. In these domains, structured representations are more appropriate [3, 9, 12, 2]. Cases are therefore modeled by semantic networks, a particular type of labeled graphs in which every node is associated to a concept and the edges represent relations between concepts. In this framework retrieval algorithms essentially search the case base for a graph that contains a subgraph isomorph, or partially isomorph (maximal common subgraph), to the query case [4, 2, 6].

Unfortunately the graph isomorphism problem for generic graphs is NP-Hard (exponential in the number of graph's nodes). Many attempts were made to cope with that, but there are some positive results only with case bases of small size. It still seems very hard to scale up on case bases with hundreds of cases. Börner et al. [2] reduce the maximal common subgraph problem to the problem of searching for the maximal clique in a combination graph. Infact both the subgraph isomorphism problem and the maximal common subgraphs problem can be addressed in two way: by searching for maximal clique [13] or by constraints satisfaction [18, 11, 13]. But this does not modify the complexity of the problem.

A group of approaches extract from a case library similarities between cases and use these similarities to build a hierarchy of graphs. Bunke and Messmer [4] call this hierarchy "network of model graphs" (NMG). In a NMG the common subgraphs of a graph library are organized in a lattice, where the relation is the usual subgraph relation. That structure (exponential in space) can be exploited when a query is made. They show encouraging results when the library of graphs is large and when the stored cases are very similar to each other.

Another approach in this direction has been proposed by Börner et al. [2]. They exploit similarities among cases by clustering cases according to a structural similarity metric, so that matching is performed in a two stages process. First a good cluster is found and then a good set of graphs is retrieved. They also use a hierarchical structure similar to that presented in [4] but restricted to trees.

In this paper we follow a different approach, i.e., trading expressive power with efficiency. The subgraph isomorphism problem becomes polynomial when both the query case and the cases in the case base are trees [19], and this result generalizes also to a class of planar graphs. For that reason, we propose to focus on tree structured cases. We believe that notwithstanding this limitation a large class of real situations can still be managed. For instance, Jones et al. [7] represent cases with trees in a CBR system for intelligent retrieval of historical meteorological data, and Surma describes aggregation taxonomies with trees [16]. Moreover labeled trees are at the base of object oriented representation languages and document representation in information retrieval.

We have designed and implemented a set of efficient algorithms for case retrieval from a case base of trees labeled on both nodes and edges. These algorithms are not based on state-space search and backtracking and they use a general definition of similarity between trees that consider both the case structure, i.e. the graph structure, and his semantic part, the labels. The complexity of the retrieval resides on the fact that trees here considered are general trees, i.e. they are not ordered[1] as other approaches assume [17]. This feature is important in many applications, e.g. planning, and enables one to apply the proposed algorithms to trees that partially represent a graph, for instance a spanning tree. The proposed algorithms take the move from the best known algorithm for solving the subtree isomorphism problem on unlabeled trees, proposed by D.W. Matula [10] and M. Chung [5]. We have added a user definable similarity metric and exploited a branch and bound technique. The proposed retrieval algorithms solve three different problems:

1. the first searches, in the case base, among all the subtrees isomorph to the query case the most semantically similar to the query tree (*best complete*

[1] The children of a given node are not linearly ordered and this order must not respected in the matching nodes.

similarity morphism);
2. the second searches the subtree simultaneously most semantically and structurally similar (*best incomplete similarity morphism*) with the constraint that the root of the query tree is matched;
3. the third searches the most semantically and structurally common subtree between the query tree and the trees in the case base.

All the above algorithms are polynomial. We conducted an empirical evaluation on randomly generated case bases. We compared our algorithms with a classical state-space search algorithm. The results show significant speed-up with respect to the classical approach.

2 Graphs and Structured Cases

This Section contains the definitions needed for the retrieval algorithms (Section 3). A *directed graph* is a structure $G =< V_G, E_G >$ in which V_G is a finite set of nodes and $E_G \subseteq V_G \times V_G$ is a set of edges. If $G =< V_G, E_G >$ and $H =< V_H, E_H >$ are two graphs, H is a *subgraph* of G if $V_H \subseteq V_G$ and $E_H \subseteq E_G$. If $G =< V_G, E_G >$ is a graph and $V' \subseteq V$ then $H =< V', E' >$ is the *subgraph induced* by V' if $E' = \{(u, v) \in E_G \text{ s.t. } u, v \in V'\}$.

We are interested in, given a query graph H and a family of graphs $\{G_i\}$, identifying the subgraphs of some element G_i that are isomorphic to the given query graph. We now precisely define the terms we are using [13, 19].

Let $G =< V_G, E_G >$ and $H =< V_H, E_H >$ be two graphs.

- $f : V_H \to V_G$ is a *morphism* iff $(u, v) \in E_H \Rightarrow (f(u), f(v)) \in E_G$.
- G and H are *isomorphic* iff there exists a morphism $f : V_H \to V_G$ s.t. f bijection and the inverse of f is also a morphism.
- H is *isomorphic* to a subgraph of G iff there exists an injective morphism $f : V_H \to V_G$, s.t. H is isomorphic to the subgraph induced by $f(V)$ in G.

A morphism $f : V_H \to V_G$ is said *incomplete* if it is not defined on all the nodes of H, otherwise is called *complete*.

In the rest of the paper we shall be concerned only with trees, i.e., connected acyclic graphs. Let $G =< V_G, E_G >$ be a (directed) tree and $x \in V_G$, the *father* of x, $F(x)$, is the unique node s.t. $(F(x), x) \in E_G$; the *children* of x, $C(x)$, are the nodes y s.t. $(x, y) \in E_G$; the *root* node r is the unique node that has no father.

In Figure 1 are shown three trees. From H to G_1 there are six injective complete morphism. $\{(a, A), (b, B), (c, D), (d, F), (e, E), (f, I), (g, L)\}$ and $\{(a, D), (b, F), (c, E), (d, I), (e, L), (f, N), (g, M)\}$ are two injective complete morphisms. These trees are not ordered, this means that if in the first morphism we switch I with L we still have a (different) morphism. From H to G_2 there are no complete morphisms but many incomplete ones. $\{(a, A), (b, B), (c, C), (e, F), (g, I)\}$ is a morphism that does not match both d and f. $\{(c, A), (d, C), (e, B), (f, D), (g, E)\}$ is a morphism that does not match the root node a.

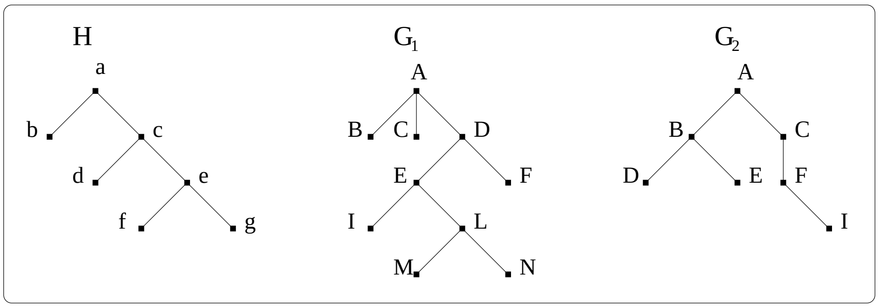

Fig. 1. Examples of trees.

A tree is labeled on nodes and edges if two functions $\lambda_n : V \to L$ and $\lambda_e : V \times V \to L$ are defined, where L is the label set. To extend the definition of morphism to labeled trees we need a measure of *similarity* between labels $s : L \times L \to [0, 1]$. If x, y, w, z are nodes we shall more compactly write $s(x, y)$ instead of $s(\lambda_n(x), \lambda_n(y))$, and $s((x, y), (w.z))$ instead of $s(\lambda_e(x, y), \lambda_e(w, z))$.

Let H and G be two labeled trees, $f : V_H \to V_G$ is an *similarity morphism* (*tsm*) iff it is an incomplete injective morphism and $\forall(x, y) \in E_H$ s.t. $f(x)$ and $f(y)$ are defined, the following conditions hold:

- $s(x, f(x)) > 0$, $s(y, f(y)) > 0$ and $s((x, y), (f(x), f(y))) > 0$,
- the set $\{x \in V_H$ s.t. $f(x)$ is defined$\}$ is a subtree of H.

We say that H is *similar isomorphic* to a subtree of G iff exists an *tsm* f from H to G. If $f(x)$ is not defined we shall also write $f(x) = \$$, in this case we shall also assume that $s(x, \$) = 0$ and $s((x, y), (f(x), f(y))) = 0$.

A distance between trees (*tsd*) is a function $d : T \times T \to [0, \infty)$, where T is the space of all trees, s.t.

$$d_f(H, G) = \sum_{x \in V_H} w_x(1 - s(x, f(x))) + w_{F(x),x}(1 - s((F(x), x), (f(F(x)), f(x)))) \quad (1)$$

$$d(H, G) = min_f\{d_f(H, G)\} \quad (2)$$

where $f : V_H \to V_G$ is a *tsm* between H and G, w_x and $w_{F(x),x}$ are positive real numbers. We note that this distance function between labeled trees has the property that if a node $x \in V_H$ is not mapped by f to a node of G then the contribution of this mismatch to the sum in Equation 1 is given by w_x , $w_{F(x),x}$ and all w_y , $w_{F(y),y}$ s.t. y is a descendant of x.

3 Retrieval Algorithms

3.1 Tree Isomorphism Algorithm

In this Section we present the basic algorithm for solving the subtree isomorphism problem. Let us assume that $H_r = (V_H, E_H)$ and $G_{r'} = (V_G, E_G)$ are two

```
SubTreeIso(H_r, G_r')
input: two rooted trees H_r and G_r'
output: Boolean
1  notvoid ← ⊤
2  for each node p ∈ V_H
3      if p is a leaf
4         S^r(p) ← V_G
5         mark(p) = ⊤
6      else S^r(p) ← ∅
7          mark(p) ← ⊥
8  while notvoid
            and exists p s.t. mark(p) = ⊥
            and (∀c_p ∈ C(p) mark(c_p) = ⊤)
9        S^r(p) ← ComputeS(H_r, p, G_r')
10       mark(p) ← ⊤
11       if S^r(p) = ∅
12          notvoid ← ⊥
13 return notvoid
```

```
ComputeS(H_r, G_r', p)
input: two rooted trees H_r, G_r'
and a node p of H_r
output: a subset S^r(p) of V_G
1  S^r(p) ← ∅
2  for each g ∈ V_G
3      Build(B_p,g)
4      m ← ComputeMatching(B_p,g)
5      if |m| = |C(p)|
6         S^r(p) ← S^r(p) ∪ {g}
7  return S^r(p)
```

Fig. 2. The algorithm for solving the subtree isomorphism problem.

trees with root r and r' respectively. For every node $p \in V_H$ we define the set $S^r(p)$ as follow:

$$S^r(p) = \{g \in V_G : \exists \text{ an injective morphism } m : H_r(p) \to G_{r'}(g) \text{ with } m(p) = g\}$$

where $H_r(p)$ is the subgraph induced by the set made of p and all the descendants of p. If $S^r(r)$ is not void then H_r is isomorphic to a subtree of $G_{r'}$. If $r' \in S^r(r)$ then H_r is isomorphic to a subtree rooted in the same root of $G_{r'}$. If $p \in V_H$ and $g \in V_G$ then we can build the bipartite graph $B_{p,g} = (C(p) \cup C(g), E)$, where $C(p)$ and $C(g)$ are the children of p and g and $(c_p, c_g) \in E$ iff $c_g \in S^r(c_p)$. The following proposition holds [5]:

Proposition 1 *$g \in S^r(p)$ iff there exists a matching in $B_{p,g}$ that cover $C(p)$.*

A matching of $G =< V, E >$ is a subset $M \subseteq E$ such that if $(x, y), (z, w) \in M$ then $x \neq z$, $x \neq w$, $y \neq z$ and $y \neq w$ [19]. A matching $M \subseteq E$ *cover* a set $V' \subseteq V$ if for all $x \in V'$ there exists $y \in V$ s.t. $(x, y) \in M$ or $(y, x) \in M$. Proposition 1 yields immediately a recursive algorithm for testing if H_r is isomorphic to a subtree of $G_{r'}$. The algorithm in Figure 2 shows an iterative version that is equivalent to that presented in [13].

In *SubTreeIso* there is a first initialization block (lines $2 \div 7$) of the sets S^r; the second block (lines $8 \div 12$) computes the $S^r(p)$ $\forall p \in V_H$ starting from the leaves and ending to the root of the tree. Note that *SubTreeIso* stops and exits with false when one tree $H_r(p)$ is not isomorph to any subtree of $G_{r'}$.

The function *ComputeMatching* applies a maximal bipartite matching algorithm [19] to find (lines $5 \div 6$ of function *ComputeS*) if a node $g \in V_G$ is in $S^r(p)$, according to proposition 1. This algorithm can be implemented quite easily with an $O(ne)$ worst case complexity, where n and e are the number of vertices and edges in $B_{p,g}$.

The algorithm in Figure 2 does not explicitly list all the injective morphisms of H_r in $G_{r'}$. To attain that goal an additional procedure must be used [15]. It is simple to show that when all the $S^r(p)$ have been computed a backtrack-free search can output all the isomorphisms. Therefore the time complexity of this phase is linear in the number of the subtree isomorphisms.

3.2 Complete Similarity Tree Isomorphism Algorithm

The algorithm depicted in Figure 2 can be modified to find the complete isomorphism that minimizes the distance function (Equation 2) among all the isomorphisms.

First, $SubTreeIso$ has to be modified in order to compute the new sets $S^r_c(h)$, $h \in V_H$, which are defined as $S^r(h)$, except that now the morphism is substituted with a complete similarity morphism. We need only two changes:

1. at the line 2 of the function $ComputeS$ we repeat the search for all $g \in V_G$ s.t. $s(h,g) > 0$;
2. $B_{hg} =< C(h) \cup C(g), E >$, and $(c_h, c_g) \in E$ if and only if $c_g \in S^r_c(c_h)$ and $s((h, c_h), (g, c_g)) > 0$.

Having done this, to find the best complete isomorphism, we use a branch and bound search in the space of all isomorphisms. This algorithm is very fast because it is restricted to the space of all the complete similarity isomorphisms which is normally smaller than the space of all the possible maps between two trees.

3.3 Incomplete Similarity Tree Isomorphism Algorithm

The algorithm presented in this Section further generalizes the idea presented in the previous one allowing an incomplete match of the tree structure not only a partial match of the labels. Moreover we want that the root of the query be matched. To obtain this goal we must again slightly change the definition of S^r.

Let H_r and $G_{r'}$ be two rooted trees and $f : V_H \to V_G$, be a (incomplete) tree similarity morphism tsm, then for all $v \in V_H$, $S^r_i(v)$ is the set of triples $(g, d_{vg}, f|_{H_r(v)})$ where:

1. $g \in V_G$ and $f(v) = g$;
2. d_{vg} is the *matching distance* between $H_r(v)$ and $G_{r'}(g)$ computed with respect to f (Equation 1);
3. $f|_{H_r(v)}$ is the tsm f restricted to the subtree of H_r rooted at v that gives minimal matching distance between $H_r(v)$ and $G_{r'}(g)$;
4. the similarity between the labels of the nodes v and g and of his incident edges is greater than 0.

SubTreeSimMatching($H_r, G_{r'}$)

input: two rooted trees H_r and $G_{r'}$

output: The best incomplete similarity isomorphism between H_r and a subtree of $G_{r'}$.

1 **for each** node $h \in V_H$

2   **if** h is a leaf

3     $S_i^r(h) \leftarrow \emptyset$

4   **for each** node $g \in V_G$

5   **if** $ss(h,g) > 0$

6         $S_i^r(h) \leftarrow S_i^r(h) \cup (g, [1 - s(h,g)] + [1 - s((F(h),h),(F(g),g))], f|_{H_r(h)})$

7     $mark(h) = \top$

8   **else** $mark(h) \leftarrow \bot$

9 **while**($\exists h \in V_H \mid mark(h) = \bot$) **and** $(\forall c_h \in C(h)\ mark(c_h) = \top)$

10     $S_i^r(h) \leftarrow$ComputeS($H_r, h, G_{r'}$)

11     $mark(h) \leftarrow \top$

12 **return** Best($S_i^r(r)$)

ComputeS($H_r, h, G_{r'}$)

input: two rooted trees H_r, $G_{r'}$, a node $h \in H_r$

output: $S_i^r(h)$

$S_i^r(h) \leftarrow \emptyset$

for each $g \in V_G$

  if $ss(h,g) > 0$

    Build($B_{h,g}$)

    $m \leftarrow$ ComputeMatching($B_{h,g}$)

    $d_{hg} \leftarrow$ ComputeDist(h, g, m)

    $S_i^r(h) \leftarrow S_i^r(h) \cup (g, d_{hg}, f|_{H_r(h)})$

return $S_i^r(h)$

Fig. 3. The algorithm for solving the incomplete similarity isomorphism problem of H_r with a subtree of $G_{r'}$. $ss(h,g) > 0$ means $(s(h,g) > 0$ and $s((F(h),h),(F(g),g)) > 0)$.

$S_i^r(v)$ is said the *match set* of v. $S_i^r(r)$ is the set of all incomplete similarity isomorphisms of H_r with subtrees of $G_{r'}$ together with the distance evaluated on the corresponding *tsm*. The best incomplete similarity isomorphism is the element, in $S_i^r(r)$, which have minimal distance (as shown in Equation 2).

In [15] is proved that $S_i^r(v)$ can be computed from the sets $S_i^r(x)$, $\forall x \in C(v)$. This property allows a fast search of the best incomplete similarity isomorphism. Figure 3 shows the algorithm for computing the best incomplete similarity isomorphism. There are two blocks:

- in the first block (lines $1 \div 8$) is computed the set $S_i^r(h)$ for each leaf node h of H_r;
- the second block (lines $9 \div 11$) computes $S_i^r(h)$ for each internal node h.

In the procedure *ComputeS* given two nodes $h \in V_H$ and $g \in V_G$, if the similarity between the labels of the two nodes and of their incident edges is greater than 0 then the matching distance d_{hg} is computed solving a *maximum weight bipartite matching problem* [19]. The matching distance d_{hg}, computed in *ComputeDist*, has two parts: the first is determined by the labels of h and g and the labels of the incident edges, the second depends on the matching of the children nodes of h with the children nodes of g. The maximum weight bipartite matching algorithm computes this second component. The nodes of B_{hg} are

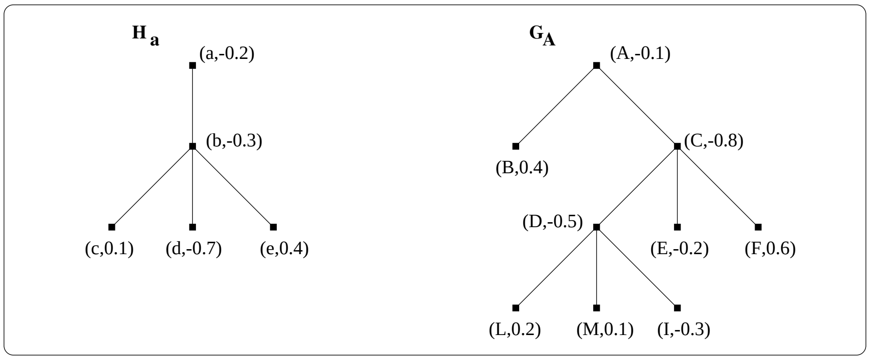

Fig. 4. Two labeled trees (H is the query). Only the nodes are labeled with numbers. Each node is associated with a pair (x, n) where x is the name of the node and n his label.

the children of h and g and there is an edge between two nodes $x \in C(h)$ and $y \in C(g)$ if and only if $(y, d_{xy}, f|_{H_r(x)}) \in S_i^r(x)$; the weight of the edge (x, y) in B_{hg} is $(\sum_{v \in V_{H_r(x)}} w_v + w_{F(v),v}) - d_{xy}$. The last term is called the (maximal) *matching similarity* between x and y.

Let us consider an example to show the behavior of $SubTreeSimMatching$ (see Figure 4). The two trees shown are labeled only on the nodes. The similarity function is 0 if two labels have opposite sign and is one minus the module of the difference otherwise. For example $s(b, B) = 0$ and $s(b, A) = 0.2$.

The initialization phase (lines 1-8 in $SubTreeSimMatching$) generates the sets $S_i^a(x)$ $\forall$ x leaf node of H:

$$S_i^a(c) = \{(B, 0.3, \{(c, B)\}), (F, 0.5, \{(c, F)\}), (L, 0.1, \{(c, L)\}), (M, 0, \{(c, M)\})\}$$
$$S_i^a(d) = \{(A, 0.6, \{(d, A)\}), (C, 0.1, \{(d, C)\}), (D, 0.2, \{(d, D)\}), (I, 0.4, \{(d, I)\}), (E, 0.5, \{(d, E)\})\}$$
$$S_i^a(e) = \{(B, 0, \{(e, B)\}), (F, 0.2, \{(e, F)\}), (L, 0.2, \{(e, L)\}), (M, 0.3, \{(e, M)\})\}$$

After that the father of c is considered (lines 9-11), and the set $S_i^a(b)$ is computed. Figure 5 shows the bipartite graphs used to match b with D, C and A respectively. Note that each edge (x, y) is weighted with the matching similarity of the two nodes x and y.

$$S_i^a(b) = \{(D, 0.8, \{(b, D), (c, M), (d, I), (e, L)\}), (I, 3, \{(b, I)\}), (E, 3.1, \{(b, E)\}),$$
$$(C, 1.9, \{(b, C), (d, D), (e, F)\}), (A, 2.1, \{(b, A), (d, C)\})\}$$

The matching distance between b and D is calculated (see Equation 1) as the sum of one minus the similarity between b and D $(1 - s(b, D) = 0.2)$ and the matching distance of their children $(= d_{cM} + d_{dI} + d_{eL} = 0 + 0.2 + 0.4 = 0.6)$. This match is found by the maximum weight bipartite matching algorithm.

$SubTreeSimMatching$ terminates after generating the set $S_i^a(a)$ (Figure 6 shows the bipartite graphs generated).

$$S_i^a(a) = \{(I, 4.1, \{(a, I)\}), (D, 3.3, \{(a, D), (b, I)\}), (C, 1.4, \{(a, C), (b, D), (c, M), (d, I), (e, L)\}),$$
$$(E, 4, \{(a, E)\}), (A, 2, \{(a, A), (b, C), (d, D), (e, F)\})\}$$

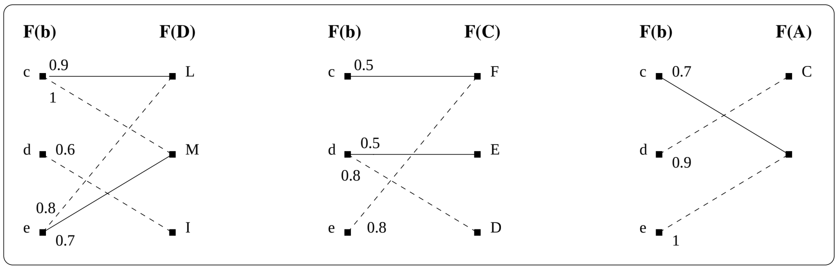

Fig. 5. The bipartite graphs generated during the first phase of the algorithm. The edges showed in dashed lines represent the maximal match.

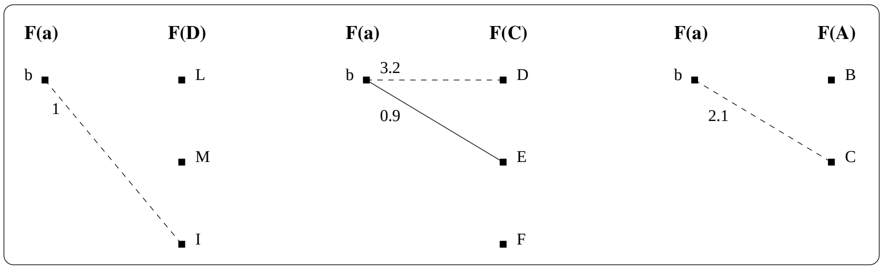

Fig. 6. The bipartite graphs generated during the second phase of the algorithm. The edges showed in dashed lines represent the maximal match.

The *best complete similarity morphism* is $\{(a, C), (b, D), (c, M), (d, I), (e, L)\})$ with matching distance 1.4.

To apply $SubTreeSimMatching$ to a real retrieval algorithm we have to consider not only a tree $G_{r'}$, but a collection of trees $\{G^i_{r'}\}$. We then search the subtree in $\{G^i_{r'}\}$ that has minimal distance with the query tree H_r. We call NN-TSDr the Nearest Neighbor algorithm that uses the tsd distance as defined in Equation 2 and computed by the procedure $SubTreeSimMatching$. Our implementation of NN-TSDr exploits a branch and bound technique that discards a partial match $(g, d_{hg}, f|_{H_r(h)})$ in the $SubTreeSimMatching$ computation when d_{hg} is greater than a previously found tsd distance. NN-TSD differs from NN-TSDr in relaxing the condition that the root r of the query tree is matched.

If $CB = \{G_i =< V_{G_i}, E_{G_i} >, i = 1 \ldots m\}$ is a collection of labeled trees (case-base) and $H =< V_H, E_H >$ is a query tree, then the worst case complexity of NN-TSDr and NN-TSD is ([15]):

$$O(m * |V_H| * |V_{G_i}| * (|V_H| + |V_{G_i}|)).$$

We shall show experimentally in the next Section that in the average case the proposed algorithms are linear with respect to $|V_H|$ and $|V_{G_i}|$.

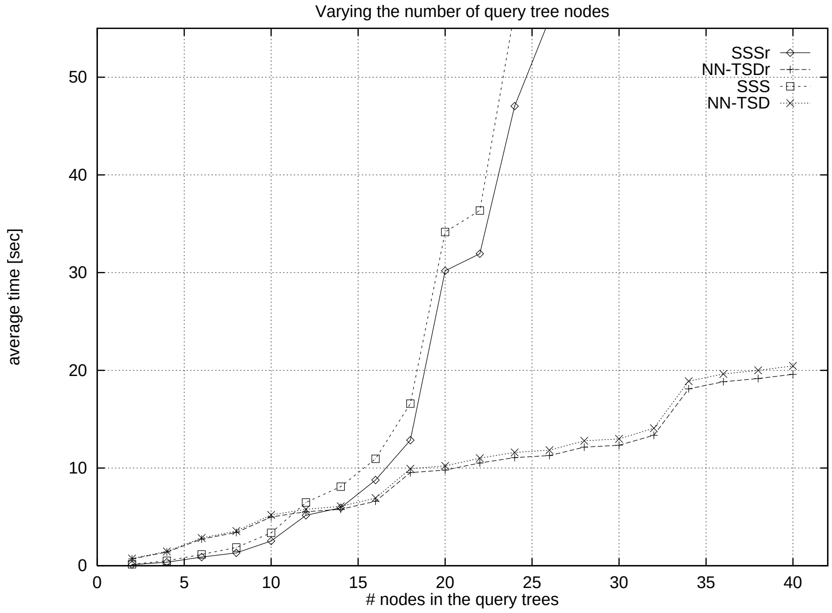

Fig. 7. Average retrieval times on 20 retrievals varying the number of nodes in the query tree. The number of nodes in the case base trees is 50, the number of trees is 100, the maximum number of child nodes is 5 and the maximum number of child nodes in the query trees is 3.

4 Empirical Evaluation

In this Section we experimentally compare NN-TSD and NN-TSDr with a classical state-space search algorithm (SSS solves the same problem of NN-TSD, SSSr the same of NN-TSDr) that also uses the branch and bound technique mentioned above. NN-TSD and SSS (equivalently NN-TSDr and SSSr) solve exactly the same problem and the results are equal. The experiments here illustrated are executed on randomly generated case bases. The trees contain both numerical and nominal labels on the nodes and only nominal labels on the edges. There are only two labels on edges, "numerical" and "nominal". The edge (x, y) is labeled "nominal" ("numeric") if y has a nominal ("numeric") label. On the nominal nodes there are 5 possible labels and the similarity is 1 between two equal labels and 0 otherwise. On the numeric nodes the similarity is taken as one minus the Euclidean distance. This is only an example, a different similarity metric can be used on both nodes and edges.

Figure 7 shows the average time of one retrieval varying the number of nodes in the query tree. Note that NN-TSD and NN-TSDr have linear behavior, while SSS has, as expected, an exponential one. Figure 8 shows the retrieval time varying the number of nodes in the case base trees. All the compared algorithms seem linear but NN-TSD is more efficient than SSS. The SSS curve stops at 30

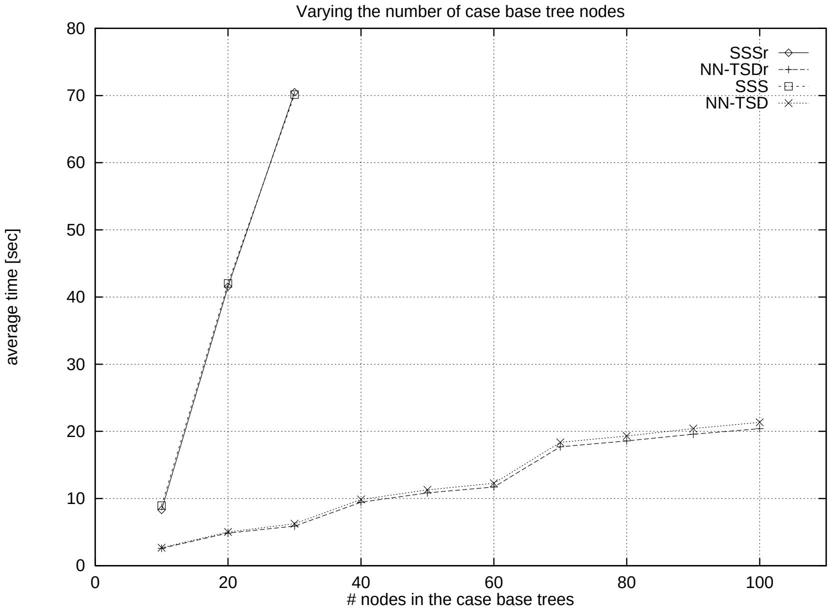

Fig. 8. Average retrieval times on 20 retrievals varying the number of nodes in the case base trees. The maximum number of child nodes in the case base trees is 15, the number of trees is 100, the number of nodes in the query trees is 20 and the maximum number of child nodes is 10.

nodes, in fact on trees with 40 nodes SSS requires more than 100 Mega byte of memory making impractical any further computation. It is worth noting that these curves show an ideal situation without any memory limit (swap time is not included). In a real situation SSS is much slower because of disk swapping. On the contrary our algorithms show a poor use of memory, in all the experiments done NN-TSD never passed 15 Mega bytes of memory.

5 Conclusions

In this paper we propose a set of efficient algorithms for case retrieval from a case base of trees labeled on both nodes and edges. These algorithms are not based on state-space search and backtracking, they use a branch and bound technique and a general definition of similarity between trees that consider both the case structure, i.e. the graph structure, and his semantic part, the labels. The comparison with a conventional state-space search algorithm on randomly generated case bases, shows that significant speed up can be obtained. This algorithms have been integrated in a C++ library for case-based reasoning and data mining called CBET [1].

References

1. P. Avesani, A. Perini, and F. Ricci. Cbet: a case base exploration tool. In Springer-Verlag, editor, *Fifth Congress of the Italian Association for Artificial Intelligence (AI*IA 97)*, Roma (Italy), 1997, September 16-19 1997.
2. K. Börner, E. Pippig, E.-C. Tammer, and K.-H. Coulon. Structural similarity and adaptation. In *European Workshop on CBR*, Lausanne, 1996.
3. L. K. Braiting. Building explanations from rules and structured cases. *International Journal of Man-Machine Studies*, 34:797–837, 1991.
4. H. Bunke and B. Messmer. Similarity measures for structured representations. In S. Wess, K.-D. Althoff, and M. M. Richter, editors, *Topics in Case-Based Reasoning*, Kaiserslautern, Germany, 1994. Springer-Verlag.
5. M. Chung. $o(n^{2.5})$ time algorithms for the subgraph homeomorphism problem in trees. *Journal of Algorithms*, 8:106–122, 1987.
6. F. Gebhardt, A. Voß, W. Gräther, and B. Schmidt-Belz. *Reasoning with complex cases.* Kluwer, 1997.
7. E. K. Jones and A. Roydhouse. Intelligent retrieval of historical meteorological data. *AI Applications*, 8(3):43–54, 1994.
8. P. Katalagarianos and Y. Vassiliou. On the reuse of software: a case-based approach employing a repository. *Automated Software Engineering*, 2:55–86, 1995.
9. B. Kettler, J. Hendler, W. A. Anderson, and M. P. Evett. Massively parallel support for case-bases planning. *IEEE Expert*, pages 8–14, 1994.
10. D. W. Matula. Subtree isomorphism in $O(n^{5/2})$. *Ann. Discrete Math.*, 2:91–106, 1978.
11. J. J. McGregor. Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. *Information Science*, 19:229–250, 1979.
12. E. Plaza. Cases as terms: a feature term approach to the structured representation of cases. In *International Conference on Case-Based Reasoning (ICCBR-95), Sesimbra, Portugal, Oct. 23-26.* Springer Verlag, 1995.
13. J.-C. Régin. *Développment d'outils algorithmiques pour l'Intelligence Artificielle. applicatin à la chimique organique.* Thèse de doctorat, Université Montpellier II, 1995.
14. R. J. Schalkoff. *Pattern recognition: statistical, structural and neural approaches.* John Wiley, 1992.
15. L. Senter. Accoppiamento inesatto di alberi e ragionamento basato su casi. Master's thesis, Univeristà di Padova, Facoltà di Ingegneria, 1998.
16. J. Surma. A similarity measure for aggregation taxonomies. In *ECML Workshop Notes on Case-Based Learning: Beyond Classification of Feature Vectors*, Prague, 1997.
17. E. Tanaka and K. Tanaka. The tree-to-tree editing problem. *International Journal of pattern recognition and artificial intelligence*, 2(2):224–240, 1988.
18. J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM*, 1(23):31–42, 1976.
19. J. van Leeuwen. Graphs algorithms. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 525–631. Elsevier, 1990.

This article was processed using the LaTeX macro package with LLNCS style