A Comparative Study of Ant Colony Optimization and Reactive Search for Graph Matching Problems

Olfa Sammoud[1,2], Sébastien Sorlin[2], Christine Solnon[2] and Khaled Ghédira[1]

[1] SOIE, Institut Supérieur de Gestion de Tunis,
41 rue de la Liberté, Cité Bouchoucha, 2000 Le Bardo, Tunis
{olfa.sammoud,khaled.ghedira}@isg.rnu.tn
[2] LIRIS, CNRS UMR 5205, bât. Nautibus, University of Lyon I
43 Bd du 11 novembre, 69622 Villeurbanne cedex, France
{sammoud.olfa,sebastien.sorlin,christine.solnon}@liris.cnrs.fr

Abstract. Many applications involve matching two graphs in order to identify their common features and compute their similarity. In this paper, we address the problem of computing a graph similarity measure based on a multivalent graph matching and which is generic in the sense that other well known graph similarity measures can be viewed as special cases of it. We propose and compare two different kinds of algorithms: an Ant Colony Optimization based algorithm and a Reactive Search. We compare the efficiency of these two algorithms on two different kinds of difficult graph matching problems and we show that they obtain complementary results.

1 Introduction

Graphs are often used to model structured objects: vertices represent object components while edges represent binary relations between components. For example, graphs are used to model images [3, 5], design objects [10], molecules or proteins [1], course timetables [9]. In this context, object recognition, classification and identification involve comparing graphs, *i.e.,* matching graphs to identify their common features [11]. This may be done by looking for an exact graph or subgraph isomorphism in order to show graph equivalence or inclusion. However, in many applications, one looks for similar objects and not identical ones and exact isomorphisms cannot be found. As a consequence, error-tolerant graph matchings such as the maximum common subgraph and the graph edit distance have been proposed [6, 8, 7, 11]. Such matchings drop the condition that the matching must preserve all vertices and edges: the goal is to find a "best" matching, *i.e.,* one which preserves a maximum number of vertices and edges.

Most recently, three different papers ([10, 5, 3]) proposed to go one step further by introducing multivalent matchings, where a vertex in one graph may be matched with a set of vertices of the other graph in order to associate one single component of an object to a set of components of another object. This allows

one to compare objects described at different levels of granularity such as under- or over- segmented images [3], or a model of an image having a schematic aspect with real and over-segmented images [5].

We more particulary focus on the multi-labeled graph similarity measure of [10] as it has been shown in [20] that it is more general than the two other ones proposed in [5, 3]. Indeed, it is parameterized by similarity functions that allow one to easily express problem-dependent knowledge and constraints.

In section 2, we briefly present the generic graph similarity measure of [10]. In section 3 and 4, we propose two algorithms based on two different approaches for computing this graph similarity measure. The first one is based on Ant Colony Optimization (ACO) and the second one is based on Reactive Search (RS). In section 5, we experimentally compare the two proposed algorithms on two kinds of graph matching problems. Finally, we conclude on the complementarity of these two algorithms and we discuss some further work.

2 A generic similarity measure for multi-labeled graphs

Definition of multi-labeled graphs. A directed graph is defined by a couple $G = (V, E)$, where V is a finite set of vertices and $E \subseteq V \times V$ is a set of directed edges. Vertices and edges may be associated with labels that describe their properties. Without loss of generality, we assume that each vertex and each edge has at least one label. Given a set L_V of vertex labels and a set L_E of edge labels, a multi-labeled graph is 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 a relation associating labels to vertices, *i.e.,* r_V is the set of couples (v_i, l) such that vertex v_i is labeled by l,
- $r_E \subseteq V \times V \times L_E$ is a relation associating labels to edges, *i.e.,* r_E is the set of triples (v_i, v_j, l) such that edge (v_i, v_j) is labeled by l. Note that the set of edges of the graph can be defined by $E = \{(v_i, v_j) | \exists l, (v_i, v_j, l) \in r_E\}$.

Similarity measure. We now briefly describe the graph similarity measure introduced in [10]. We refer the reader to the original paper for more details.

The similarity measure is computed with respect to a matching of the vertices of the two graphs. We consider here a multivalent matching, *i.e.*, each vertex of one graph is matched with a –possibly empty– set of vertices of the other graph. More formally, a multivalent matching of two multi-labeled graphs $G = \langle V, r_V, r_E \rangle$ and $G' = \langle V', r_{V'}, r_{E'} \rangle$ is a relation $m \subseteq V \times V'$ which contains every couple $(v, v') \in V \times V'$ such that vertex v is matched with vertex v'.

Once a multivalent matching is defined, the next step is to identify the set of features that are common to the two graphs with respect to this matching. This set contains all the features from both G and G' whose vertices (resp. edges) are matched by m to at least one vertex (resp. edge) that has the same label. More formally, the set of common features $G \sqcap_m G'$ of two graphs $G = \langle V, r_V, r_E \rangle$ and

$G' = \langle V', r_{V'}, r_{E'} \rangle$, with respect to a matching $m \subseteq V \times V'$, is defined as follows:

$$\begin{aligned} G \sqcap_m G' \doteq & \{(v,l) \in r_V |\ \exists (v,v') \in m,\ (v',l) \in r_{V'}\} \\ & \cup \{(v',l) \in r_{V'} |\ \exists (v,v') \in m(v),\ (v,l) \in r_V\} \\ & \cup \{(v_i,v_j,l) \in r_E |\ \exists (v_i,v_i') \in m, \exists (v_j,v_j') \in m\, (v_i',v_j',l) \in r_{E'}\} \\ & \cup \{(v_i',v_j',l) \in r_{E'} |\ \exists (v_i,v_i') \in m, \exists (v_j,v_j') \in m\, (v_i,v_j,l) \in r_E\} \end{aligned}$$

Given a multivalent matching m, we also have to identify the set of split vertices, *i.e.*, the set of vertices that are matched to more than one vertex, each split vertex v being associated with the set s_v of its matched vertices:

$$\begin{aligned} splits(m) = & \{(v, s_v) \quad | \ v \in V, \quad s_v = \{v' \in V' | (v,v') \in m\},\ |s_v| \geq 2\} \\ & \cup \{(v', s_{v'}) \mid v' \in V',\ s_{v'} = \{v \in V | (v,v') \in m\},\ \ |s_{v'}| \geq 2\} \end{aligned}$$

The similarity of two graphs $G = \langle V, r_V, r_E \rangle$ and $G' = \langle V', r_{V'}, r_{E'} \rangle$ with respect to a matching m is then defined by:

$$sim_m(G,G') = \frac{f(G \sqcap_m G') - g(splits(m))}{f(r_V \cup r_{V'} \cup r_E \cup r_{E'})} \tag{1}$$

where f and g are two functions that are defined to weight features and splits, depending on the considered application.

Finally, the similarity $sim(G,G')$ of two graphs $G = \langle V, r_V, r_E \rangle$ and $G' = \langle V', r_{V'}, r_{E'} \rangle$ is the greatest similarity with respect to all possible matchings, *i.e.*,

$$sim(G,G') = \max_{m \subseteq V \times V'} sim_m(G,G')$$

Note that the denominator of formula (1) does not depend on the matching m —this denominator is introduced to normalize the similarity value between zero and one. Hence, it will be sufficient to find the matching m that maximizes the *score* function below:

$$score(m) = f(G \sqcap_m G') - g(splits(m))$$

Using this graph similarity measure to solve different graph matching problems. Thanks to the functions f and g of formula (1), the graph similarity measure of [10] is generic. [20] shows how this graph similarity measure can be used to solve many different graph matching problems such as the (sub)graph isomorphism problem, the graph edit distance, the maximum common subgraph problem and the multivalent matching problems of [5] and [10].

The graph matching problem has been shown to be NP-hard in [20]. A complete algorithm has been proposed for computing the matching which maximizes formula (1) in [10]. This kind of algorithm, based on an exhaustive exploration of the search space [16] combined with pruning techniques, guarantees solution optimality. However, this algorithm is limited to very small graphs (having less than 10 vertices in the worst case). Therefore, incomplete algorithms, that do not guarantee optimality but have a polynomial time complexity, appear to be good alternatives.

3 ACO for the graph matching problem

The ACO (Ant Colony Optimization) meta-heuristic is a bio-inspired approach [13, 12] that has been used to solve many hard combinatorial optimization problems. The main idea is to model the problem to solve as a search for an optimal path in a graph –called the construction graph– and to use artificial ants to search for 'good' paths. The behavior of artificial ants mimics the behavior of real ones: *(i)* ants lay pheromone trails on the components of the construction graph to keep track of the most promising components, *(ii)* ants construct solutions by moving through the construction graph and choose their path with respect to probabilities which depend on the pheromone trails previously laid, and *(iii)* pheromone trails decrease at each cycle simulating in this way the evaporation phenomena observed in the real world.

In order to solve graph matching problems, we have proposed in [18] a first ACO algorithm called `ANT-GM` (ANT-Graph Matching). However, if this algorithm appeared to be competitive with tabu search on sub-graph isomorphism problems, it was clearly outperformed on multivalent graph matching problems. The algorithm presented bellow called `ANT-GM'06` improves `ANT-GM` with respect to the following points: *(i)* we consider a new heuristic function in the definition of the transition probability, *(ii)* we consider a new pheromonal strategy, and *(iii)* we introduce a local search procedure to improve solutions constructed by ants.

Algorithmic scheme. At each cycle, each ant constructs a complete matching in a randomized greedy way. Once every ant has generated a matching, a local search procedure takes place to improve the quality of the best matching of the cycle. Pheromone trails are updated according to this improved matching. This process stops iterating either when an ant has found an optimal matching, or when a maximum number of cycles has been performed.

Contrary to the algorithm introduced in [18], `ANT-GM'06` follows the *Max-Min Ant System*[21]: we explicit impose lower and upper bounds τ_{min} and τ_{max} on pheromone trails (with $0 < \tau_{min} < \tau_{max}$). The goal is to favor a larger exploration of the search space by preventing the relative differences between pheromone trails from becoming too extreme during processing. Also and in order to achieve a higher exploration of the search space at the first cycles, pheromone trails are set to τ_{max} at the beginning.

Construction graph. The construction graph is the graph on which artificial ants lay pheromone trails. Vertices of this graph are solution components that are selected by ants to generate solutions. In our graph matching application, ants build matchings by iteratively selecting couples of vertices to be matched. Hence, given two attributed graphs $G = \langle V, r_V, r_E \rangle$ and $G' = \langle V', r_{V'}, r_{E'} \rangle$, the construction graph is the complete non-directed graph that associates a vertex to each couple $(u, u') \in V \times V'$.

Pheromone trails. A key point when developing any ACO algorithm is to decide where pheromone trails should be laid and how they should be exploited and updated. In our case, we may consider two different possibilities:

- the first one turns into laying pheromone on the vertices of the construction graph. So, the amount of pheromone on a vertex (u, u') represents the learnt desirability to match u with u' when constructing matchings.
- the second consists in laying pheromone on the edges of the construction graph. The amount of pheromone on an edge $< (u, u'), (v, v') >$ represents the learnt desirability to match together u with u' and v with v' when constructing matchings.

Experimental results presented in [18] have been obtained with the second strategy, that appears to be the best-performing one on the maximum clique problem [19] and multiple knapsack problem [2]. Since then, we have compared this second strategy with the first one, and experiments showed us that when we choose to lay pheromone on vertices (instead of edges), better results are obtained, also the algorithm is much less time consuming (pheromone laying and evaporation has a linear complexity with respect to the number of vertices of the construction graph and not a quadratic one).

So, in `ANT-GM'06`, pheromone is laid on vertices and not on edges like in the initial version of `ANT-GM`. The amount of pheromone on a vertex (u, u') will be noted $\tau(u, u')$.

Matching construction by ants. At each cycle, each ant constructs a complete matching: starting from an empty matching $m = \emptyset$, by iteratively adding couples of vertices that are chosen within the set $cand = \{(u, u') \in V \times V' - m\}$. As usually in ACO algorithm, the choice of the next couple to be added to m is done with respect to a probability that depends on pheromone and heuristic factors. More formally, given a matching m and a set of candidates $cand$, the probability $p_m(u, u')$ of selecting $(u, u') \in cand$ is:

$$p_m(u, u') = \frac{[\tau(u, u')]^\alpha \cdot [\eta_m(u, u')]^\beta}{\sum_{(v,v') \in cand} [\tau(v, v')]^\alpha \cdot [\eta_m(v, v')]^\beta} \tag{2}$$

where:

- $\tau(u, u')$ is the pheromone factor (when choosing the first couple, $\tau_m(u, u') = 1$, so that the probability only depends on the heuristic factor), and
- $\eta_m(u, u')$ is a heuristic factor that aims at favoring couples that most increase the score function, *i.e.,* $\eta_m(u, u') = score(m \cup \{(u, u')\}) - score(m)$.
- α and β are two parameters that determine the relative importance of the two factors.

Ants stop adding new couples to the matching when the addition of every candidate couple decreases the score function or when the score function has not been increased since the last three iterations.

Local search procedure. The best performing ACO algorithms for many combinatorial problems are hybrid algorithms that combine probabilistic solution construction by a colony of ants with local search. In a same perspective, we have tried to improve the performance of `ANT-GM'06` by coupling it with a local search procedure. In our case, we have chosen a local search which achieves a 'good' compromise between quality and time consuming. Once every ant has constructed a matching, we try to improve the quality of the best matching constructed during the cycle as follows: the three worse couples of the matching are removed from it, and the resulting matching is completed in a greedy way, *i.e.*, by iteratively adding couples of vertices that most increase the *score* function. This local search process is iterated until no more improvement is obtained.

Note that the *'goodness'* of a couple is judged according to its contribution in our *score* function, and couples to be removed at each step of a local search improvement are couples which have not already been removed.

Pheromone updating step. Once each ant has constructed a matching, and the best of these matchings has been improved by local search, pheromone trails are updated according to the *Max-Min Ant System*. First, evaporation is simulated by multiplying every pheromone trail by a pheromone persistance rate ρ such that $0 \leq \rho \leq 1$. Then, the best ant of the cycle deposits pheromone. More precisely, let m_k be the best matching (with respect to the score function) built during the cycle and improved by local search, and m_{best} be the best matching built since the beginning of the run (including the current cycle), the quantity of pheromone laid is inversely proportional to the gap of score between m_k and m_{best}, *i.e.* it is equal to $1/(1 + score(m_{best}) - score(m_k))$.

As, we have chosen to put pheromone on the vertices of the construction graph, the quantity of pheromone to be added is deposited on each couple of vertices (u, u') in m_k.

4 Reactive search for the graph matching problem

Greedy algorithm. [10] proposed a greedy algorithm to solve the graph matching problem. We briefly describe it because it is used as a starting point of our Reactive Search algorithm. More information can be found in [10]. The algorithm starts from an empty matching $m = \emptyset$, and iteratively adds to m couples of vertices chosen within the set of candidate couples $cand = V \times V' - m$. This greedy addition of couples to m is iterated until m is locally optimal, *i.e.*, until no more couple addition can increase the similarity. At each step, the couple to be added is randomly chosen within the set of couples that most increase the *score* function. This greedy algorithm has a polynomial time complexity of $\mathcal{O}((|V| \times |V'|)^2)$, provided that the computation of the f and g functions have linear time complexities with respect to the size of the matching. As a counterpart of this rather low complexity, this algorithm never backtracks and is not complete.

Local search. The greedy algorithm of [10] returns a "locally optimal" matching in the sense that adding or removing one couple of vertices to this matching cannot improve it. However, it may be possible to improve it by adding and/or removing more than one couple to this matching. A local search [14, 15] tries to improve a solution by locally exploring its neighborhood: the neighbours of a matching m are the matchings which can be obtained by adding or removing one couple of vertices to m:

$$\forall m \in \wp(V \times V'), neighbourhood(m) = \{m \cup \{(v,v')\} | (v,v') \in (V \times V') - m\} \\ \cup \{m - \{(v,v')\} | (v,v') \in m\}$$

From an initial matching, computed by the greedy algorithm, the search space is explored from neighbour to neighbour until the optimal solution is found (when the optimal value is known) or until a maximum number of moves have been performed. The next neighbour to move on at each step is selected according to the Tabu meta-heuristic.

Tabu meta-heuristic. *Tabu* search [14, 17] is one of the best known heuristic to choose the next neighbour to move on. At each step, one chooses the best neighbour with respect to the *score* function. To avoid staying around locally optimal matchings by always performing the same moves, a Tabu list is used. This list has a length k and memorizes the last k moves (*i.e.,* the last k added/removed couples) in order to forbid backward moves (*i.e.,* to remove/add a couple recently added/removed). An exception named "aspiration" is added: if a forbidden move reaches a better matching than the best known matching, the move is nevertheless done.

Reactive Search. The length k of the tabu list is a critical parameter that is hard to set: if the list is too long, search diversification is too strong so that the algorithm converges too slowly; if the list is too short, intensification is too strong so that the algorithm may be stuck around local maxima and fail in improving the current solution. To solve this parameter tuning problem, [4] introduced *Reactive Search* where the length of the Tabu list is dynamically adapted during the search. To make the Tabu algorithm reactive, one must evaluate the need for diversifying the search. When the same matching is explored twice, the search must be diversified. In order to detect such redundancies, a hashing key is memorized for each explored matching. When a collision occurs in the hash table, the list length is increased. On the contrary, when there is no collision during a fixed number of moves, thus indicating that search is diversified enough, one can reduce the list length. Hashing keys are incrementally computed so that this method has a negligible added cost.

Iterated Reactive Search. This reactive search process is iterated from different starting points: the total number of allowed moves $maxMoves$ is divided

by k and k executions of reactive search having each one $maxMoves/k$ allowed moves are launched. Finally, we keep the best matching found during the k executions.

5 Experimental comparison results of RS and ACO

Problem instances. We compare our two algorithms on two different sets of multivalent matching problems: a randomly generated one and a set of seven instances introduced in [5].

Test suite 1. We have used a random graph generator to generate "similar" pairs of graphs: it randomly generates a first graph and applies some vertex splitting/merging and some edge and vertex insertion/suppression to build a second graph which is similar to the first one. When graph components have many different labels, the best matching is trivially found as nearly all vertices/edges have different labels. Therefore, to obtain harder instances, we have generated 100 graphs such that all vertices and edges have the same label. These graphs have between 80 and 100 vertices and between 200 and 360 edges. The second graph is obtained by doing 5 vertex merging/splitting and 10 edge or vertex insertion/suppression. We define function f of formula (1) as the cardinality function and function g as a weighted sum:

$g(S) = w * \sum_{(v,s_v)\in S} (|s_v| - 1)$ where w is the weight of a split.

The chosen weight w can drastically change the hardness of instances: with a null weight, problem is trivially solved (one can make as many splitted vertices as needed to recover labels), whereas, with a high weight, optimal solutions do not split vertices and problem turns into an univalent graph matching problem. With intermediate weights, the problem is harder: optimal solutions must do a balancing between the number of splitted vertices and the number of recovered labels. We display experimental results obtained for two different "intermediate" split weights in order to compare the capacity of our algorithms to deal with splitted vertices. We first consider instances where $w = 1$, so that optimal solutions may contain several splits. We also consider instances where $w = 3$, so that optimal solutions contain less splitted vertices. We keep only the 13 hardest instances (*i.e.*, the ones that cannot be solved by the iterated greedy algorithm of [10]).

Test suite 2. A non-bijective graph matching problem was introduced in [5] to find the best matching between models and over-segmented images of brains. Given a model graph G=(V, E) and an image graph G'=(V', E'), a matching is defined as a function $\phi : V \rightarrow \wp(V')$ which associates to each vertex of the model graph G a non empty set of vertices of G', and such that *(i)* each vertex of the image graph G' is associated to exactly one vertex of the model graph G, *(ii)* for some forbidden couples $(v, v') \in V \times V'$, v' must not belong to $\phi(v)$, and *(iii)* the subgraph induced by every set $\phi(v)$ must be connected. A weight $s^v(v_i, v'_i)$

(resp. $s^e(e_i, e'_i)$) is associated with each couple of vertices $(v_i, v'_i) \in V \times V'$ (resp. of edges $(e_i, e'_i) \in E \times E'$). The goal is to find the matching which maximizes a function depending on these weights of matched vertices and edges.

One can define functions f and g so that the matching which maximizes formula (1) corresponds to the best matching as defined in [5]. We refer the reader to [20] for more details on the definition of these two functions. We have taken the 7 instances of the non-bijective graph matching problem of [5]. Scheme graphs have between 10 and 50 vertices while image graphs have between 30 and 250 vertices. For these instances, we compare our two algorithms with $LS+$, a randomized construction algorithm proposed by [5] that quickly computes a set of possible non-bijective graph matchings and improves the best of these matchings with a local search algorithm until a locally optimal point is reached. For more details on these instances and on the $LS+$ algorithm, please refer to [5].

Experimental settings for ACO. For `ANT-GM'06`, we have set the pheromone factor weight α to 1 and 2 respectively on test suite 1 and test suite 2, the pheromone persistance rate ρ to 0.98, the heuristic factor weight β to 10, the maximum number of cycles $MaxCycle$ respectively to 1000 and 2000, the number of ants $nbAnts$ to 20, the pheromone lower and upper bound τ_{min} and τ_{max} to 0.01 and 6.

To evaluate the benefit of integrating local search within `ANT-GM'06`, we display results obtained without and with local search. We respectively call these algorithms `ANT-GM'06` and `ANT-GM'06+LS`.

Experimental settings for RS. Reactive Search needs 5 parameters: the minimum (min) and the maximum (max) length of the list, the length of extension (and shortening) *diff* of the list when a reaction process occurs, the frequency $freq$ of reduction of the list and the maximum number of allowed moves $nbMoves$. The initial length of the list is always set to min. For the two test suites, max is set to 50, *diff* is set to 15 and the number of moves is set to 50000. In order to obtain better results, the two others parameters must be chosen depending on the considered problem. On instances of test suite 1, min is set to 15 and $freq$ is set to 5000 whereas on instances of test suite 2, min is set to 10 and $freq$ is set to 1000. As 50000 moves of RS are performed much quicker than 1000 cycles of `ANT-GM'06`, we iterated RS from different calls to the greedy algorithm. The number of iterations of RS is setted in such a way that both algorithms spend the same time. Note however that execution of our reactive local search is deterministic on instances of test suite 2: the weights used are real numbers and as a consequence couples of vertices are never chosen randomly. As a consequence, it is useless to use iterated version of RS and we perform only one execution of RS.

We made at least 20 executions of each algorithm on each instance.

Problem				RS			ANT-GM'06			ANT-GM'06+LS		
Nbr	(| V_1 |,| E_1 |)	(| V_2 |,| E_2 |)	L	Best	Avg	T.	Best	Avg	T.	Best	Avg	T.
1	(80, 200)	(74, 186)	1512	511	511.00	57	511	511.00	131	**512**	**511.10**	140
2	(80, 240)	(82, 261)	1415	644	644.00	60	644	644.00	266	644	644.00	239
3	(80, 320)	(83, 362)	1445	821	820.97	279	821	820.50	498	**822**	**821.20**	660
4	(80, 340)	(72, 302)	1174	753	753.00	55	753	753.00	111	753	753.00	130
5	(80, 360)	(77, 367)	1139	**856**	**855.97**	187	855	855.00	321	855	855.00	249
6	(80, 360)	(78, 367)	1196	863	863.00	21	863	863.00	187	**864**	**863.94**	565
7	(90, 300)	(91, 307)	1670	762	762.00	98	762	762.00	326	762	762.00	213
8	(90, 320)	(87, 310)	1611	780	780.00	51	780	780.00	572	780	780.00	409
9	(90, 320)	(90, 339)	1716	816	**816.00**	69	816	815.45	546	816	815.45	602
10	(100, 260)	(96, 263)	2093	697	696.63	628	697	696.90	976	697	**697.00**	812
11	(100, 300)	(100, 304)	2078	780	780.00	148	780	780.00	278	780	780.00	279
12	(100, 320)	(98, 331)	2080	828	828.00	46	828	828.00	286	828	828.00	218
13	(100, 360)	(99, 371)	2455	915	915.00	90	915	915.00	267	915	915.00	152

Table 1. Results on multivalent graph matching problems with splits weighted to 1. For each instance, the table reports the number of vertices and edges of the two graphs, the CPU time limit L for one run of each algorithm (on a Pentium IV 1.7 GHz) and, for each algorithm, the best *score* and the average *score* found over at least 20 runs and the average time needed (in seconds) to get the best *score*.

Problem				RS			ANT-GM'06			ANT-GM'06+LS		
Nbr	(| V_1 |,| E_1 |)	(| V_2 |,| E_2 |)	L	Best	Avg	T.	Best	Avg	T.	Best	Avg	T.
1	(80, 200)	(74, 186)	659	496	496.00	28	496	496.00	132	496	496.00	121
2	(80, 240)	(82, 261)	798	624	624.00	26	624	624.00	108	624	624.00	88
3	(80, 320)	(83, 362)	896	801	801.00	17	801	801.00	213	801	801.00	218
4	(80, 340)	(72, 302)	737	732	732.00	27	732	732.00	185	732	732.00	194
5	(80, 360)	(77, 367)	852	846	846.00	198	846	846.00	116	846	846.00	77
6	(80, 360)	(78, 367)	855	840	840.00	36	840	840.00	94	840	840.00	67
7	(90, 300)	(91, 307)	1140	748	748.00	82	748	748.00	186	748	748.00	150
8	(90, 320)	(87, 310)	1079	766	766.00	44	766	766.00	187	766	766.00	187
9	(90, 320)	(90, 339)	1127	802	802.00	70	802	802.00	167	802	802.00	163
10	(100, 260)	(96, 263)	1346	683	683.00	114	683	682.75	556	683	683.00	354
11	(100, 300)	(100, 304)	1466	769	769.00	358	769	769.00	274	769	769.00	285
12	(100, 320)	(98, 331)	1463	814	814.00	51	814	814.00	241	814	814.00	201
13	(100, 360)	(99, 371)	1528	900	900.00	54	900	900.00	245	900	900.00	243

Table 2. Results on multivalent graph matching problems with splits weighted to 3. For each instance, the table reports the number of vertices and edges of the two graphs, the CPU time limit L for one run of each algorithm (on a Pentium IV 1.7 GHz) and, for each algorithm, the best *score* and the average *score* found over at least 20 runs and the average time needed (in seconds) to get the best *score*.

Results. Table 1 displays results on the 13 instances of test suite 1 with splits weighted to 1. First, we can see that our 3 algorithms seems to be robust in the sense that their average results are close to their best results on 20 executions (for 9 of the 13 instances, the average result is equal to the best result). Also, we can note that RS performs better than ANT-GM'06: it obtains better

Problem			LS+	RS		ANT-GM'06			ANT-GM'06+LS		
GM-i (| V_1 |, | V_2 |)		L	Sim	Sim	T.	Sim	Avg	T.	Sim	Avg	T.
5	(10, 30)	18	.5474	.5481	0.9	.5601	.5598	16	**.5608**	.5604	15
5a	(10, 30)	19	.5435	.5529	4.6	.5638	.5638	10	**.5645**	.5641	7
6	(12, 95)	269	.4248	.4213	0.0	**.4252**	.4251	211	**.4252**	.4251	215
7	(14, 28)	13	.6319	.6333	2.1	.6369	.6369	7	**.6376**	.6369	5
8	(30, 100)	595	.5186	.5210	1.3	.5229	.5226	462	**.5232**	.5228	229
8a	(30, 100)	595	.5222	.5245	1.3	.5263	.5261	456	**.5269**	.5264	241
9	(50, 250)	6018	.5187	.5199	81.7	.5201	.5201	4133	**.5203**	.5202	2034

Table 3. Results on non-bijective graph matching of [5]. For each instance, the table reports its name, the number of vertices of the two graphs, the CPU time limit L for one run of each algorithm (on a Pentium IV 1.7 GHz), the similarities of the best solutions obtained by LS+[5], RS, ANT-GM'06 without LS and with LS (the best solution, the average solution found over 20 runs and the average time in seconds).

result on 1 instance and is always as fast as ANT-GM'06. Integrating local search within ANT-GM'06 actually improves the solution process so that ANT-GM'06 obtains better (resp. worse) results than RS on 3 (resp. 1) instances, whereas they obtain same results on 9 instances. On these instances, RS and ANT-GM'06+LS obtain complementary results: ANT-GM'06+LS outperforms RS more frequently than RS outperforms ANT-GM'06+LS but RS is much quicker than ANT-GM'06+LS, even when the two algorithms obtain the same results. Finally, note that if one cycle of ANT-GM'06+LS is more time consuming than one cycle of ANT-GM'06, ANT-GM'06+LS does not generally need more CPU time than ANT-GM'06 to find a solution: the local search procedure speed up the convergence of ANT-GM'06 and less cycles are needed to find the best solution.

Table 1 does not show results for the first ANT-GM algorithm described in [18] but one should note that the new ACO algorithm ANT-GM'06 clearly outperforms this first one. Actually, on all the considered instances, ANT-GM'06 computes much better solutions in less CPU time than ANT-GM. For example, on instance 1, the best *score* found by ANT-GM is 505 and is found in 8648 seconds, whereas ANT-GM'06 finds a *score* of 511 in 131 seconds.

Table 2 displays results on the 13 instances of test suite 1 with splits weighted to 3. On each instance, our three algorithms always find the same best score and the same average score (except for ANT-GM'06 on one instance). However, RS finds the solution in shorter times than ANT-GM'06 and ANT-GM'06+LS except for only two instances. These results show that on these instances, one clearly has to use our RS algorithm.

On the 7 instances of test suite 2, our three algorithms obtain better results than $LS+$, the reference algorithm of [5] (6 instances on 7 are better solved by RS and 7 instances on 7 are better solved by ACO algorithms). Note that, because the considered weights are real numbers, an execution of RS is deterministic. As a consequence, RS is less randomized than for multivalent matching problems, it quickly converges to "good" matchings but can easily be trapped into local optimum. So, results show that, as for non-bijective graph matching problems, ACO gives better results than RS but needs much more time.

In conclusion, the local search procedure helps `ANT-GM'06` to improve the quality of the results. As a consequence, `ANT-GM'06+LS` usually obtains better results but is slower than `RS`. The time limits allowed to our three algorithms have been set depending on the ACO algorithms which generaly need a long time to converge. So, these time limits penalize RS which, within a shorter time, can generaly find a better solution then ACO algorithms. `ANT-GM'06+LS` and `RS` are complementary: if we need to compute quickly a "good" solution of hard instances or if instances are easy, we can use RS but if we have more time to spend on computation or if we want to solve very hard instances, we can use `ANT-GM'06+LS`.

6 Conclusion and further work

In this paper, we address the problem of computing the generic graph similarity measure of [10]. We propose and compare two different kinds of algorithms: an Ant Colony Optimization (ACO) based algorithm boosted with local search and a Reactive Search based on a tabu local search heuristic. We compare the efficiency of these two algorithms on two different kinds of difficult graph matching problems. We show that ACO usually obtains better results but is slower than Reactive Search. These two algorithms are complementary: if we need to compute quickly a "good" solution of hard instances or if instances are easy, we can use RS but if we have more time to spend on computation or if we want to solve very hard instances, we can use ACO.

In further work, we would like to compare these algorithms on some other graph matching problems such as maximum common subgraph problems. For ACO, we would like to speed up the convergence of the algorithm. This could be done by using a better local search strategy to repair built matchings. For RS, we would like to diversify the search. This could be done by using an other strategy than the elitist greedy algorithm of [10] to choose the starting points and then, start shorter tabu searches from many starting points.

References

1. Tatsuya Akutsu. Protein structure alignment using a graph matching technique. Technical report, Bioinformatics Center, Institute for Chemical Research, Kyoto University. Uji, Kyoto 611-0011, Japan, 1995.
2. I. Alaya, C. Solnon, and K. Ghédira. Ant algorithm for the multi-dimensional knapsack problem. *International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2004)*, pages 63–72, 2004.
3. R. Ambauen, S. Fischer, and H. Bunke. Graph edit distance with node splitting and merging and its application to diatom identification. *4th IAPR-TC15 Wk on Graph-based Representations in Pattern Recognition*, LNCS 2726-Springer:95–106, 2003.
4. R. Battiti and M. Protasi. Reactive local search for the maximum clique problem. In Springer-Verlag, editor, *Algorithmica*, volume 29, pages 610–637, 2001.

5. M. Boeres, C. Ribeiro, and I. Bloch. A randomized heuristic for scene recognition by graph matching. In *Wkshp on Experimental and Efficient Algorithms (WEA 2004)*, pages 100–113, 2004.
6. H. Bunke. On a relation between graph edit distance and maximum common subgraph. *PRL: Pattern Recognition Letters*, 18:689–694, 1997.
7. H. Bunke and X. Jiang. *Graph Matching and Similarity*, volume Teodorescu, H.-N., Mlynek, D., Kandel, A., Zimmermann, H.-J. (eds.): Intelligent Systems and Interfaces, chapter 1. Kluwer Academic Publishers, 2000.
8. H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. *Pattern recognition letters*, 19(3–4):255–259, 1998.
9. E.K Burke, B. MacCarthy, S. Petrovic, and R. Qu. Case based reasoning in course timetabling: An attribute graph approach. *Proc. 4th Int. Conf. on Case-Based Reasoning (ICCBR-2001)*, LNAI 2080-Springer:90–104, 2001.
10. P.-A. Champin and C. Solnon. Measuring the similarity of labeled graphs. In *5th Int. Conf. on Case-Based Reasoning (ICCBR 2003)*, volume LNAI 2689-Springer, pages 80–95, 2003.
11. D. Conte, P. Foggia, C. Sansone, and M. Vento. 30 years of graph matching in pattern recognition. *Int. Jour. of Pattern Recogn. and AI*, 18(3):265–298, 2004.
12. M. Dorigo and G. Di Caro. The ant colony optimization meta-heuristic. *In D. Corne, M. Dorigo, and F. Glover, editors, New Ideas in Optimization. McGraw Hill, London, UK*, pages 11–32, 1999.
13. M. Dorigo and T. Stützle. Ant colony optimization. *MIT Press*, 2004.
14. F. Glover. Tabu search - part I. *Journal on Computing*, pages 190–260, 1989.
15. S. Kirkpatrick, S. Gelatt, and M. Vecchi. Optimisation by simulated annealing. In *Science*, volume 220, pages 671–680, 1983.
16. B.T. Messmer and H.Bunke. Efficient subgraph isomorphism detection: a decomposition approach. *IEEE Trans. on Knowledge and Data Engineering*, 12-2:307–323, 2000.
17. Sanja Petrovic, Graham Kendall, and Yong Yang. A tabu search approach for graph-structured case retrieval. In IOS Press, editor, *Proc. of the STarting Artificial Intelligence Researchers Symposium (STAIRS 2002)*, pages 55–64, 2002.
18. O. Sammoud, C. Solnon, and K. Ghédira. An ant algorithm for the graph matching problem. *5th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2005)*, LNCS 3448 - Springer:213–223, 2005.
19. C. Solnon and S. Fenet. A study of aco capabilities for solving the maximum clique problem. *To appear in Journal of Heuristics- Springer*, 2006.
20. S. Sorlin and C. Solnon. Reactive tabu search for measuring graph similarity. *5th IAPR Workshop on Graph-based Representations in Pattern Recognition (GbR 2005)*, LNCS 3434 - Springer:172–182, 2005.
21. T. Stützle and H.H. Hoos. Max-min Ant System. *Journal of Future Generation Computer Systems*, 16:889–914, 2000.