Ant Algorithm for the Graph Matching Problem

Olfa Sammoud[1], 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
christine.solnon@liris.cnrs.fr

Abstract. This paper describes a new Ant Colony Optimization (ACO) algorithm for solving Graph Matching Problems, the goal of which is to find the best matching between vertices of multi-labeled graphs. This new ACO algorithm is experimentally compared with greedy and reactive tabu approaches on subgraph isomorphism problems and on multivalent graph matching problems.

1 Introduction

Numerous applications require to measure the similarity of objects. For instance, Case-Based Reasoning (CBR) relies on the hypothesis that similar problems have similar solutions, so that CBR systems solve new problems by retrieving similar ones, for which solutions are known and can be adapted [1]. Also, information retrieval systems must be able to measure the similarity of documents and images in order to retrieve relevant documents from a database.

In many of these applications, objects are described by graphs, so that measuring objects similarity turns into determining graphs similarity, *i.e.,* matching graph vertices to identify their common features [6, 8, 10]. This may be done by looking for an exact graph or subgraph isomorphism in order to show graph equivalence or inclusion. However, the objects to be compared are usually not identical and the assumption of the existence of an isomorphism between the corresponding graphs is usually too strong. As a consequence, error-tolerant graph matchings such as maximum common subgraph and graph edit distance have been proposed [7, 10]. Such matchings drop the condition that all vertices and edges must be preserved: 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 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 [4], graph matching is used for model-based pattern recognition of brain images. In this case, the assumption of a bijection between regions of models and images is too strong: models have schematic aspects easy to segment

while images are noised and usually over-segmented. Therefore, scene recognition is better expressed as a multivalent matching problem where a set of vertices of the scene may be linked to a same vertex of the model.
- Guided by very similar motivations, [2] proposes a new graph edit distance that introduces two new edit operations —vertex splitting and merging— in order to handle the fact that images may be over- or under- segmented.
- In [9], graphs are used to model design objects in a computer-aided design application. In this context, vertices are used to represent object components and one single component of an object may play the same role than a set of components of another object, depending of the granularity of object description. Therefore, the authors introduce a similarity measure based on multivalent matchings so that one vertex in a graph may be associated with a set of vertices of the other graph.

The graph similarity measure of [9] is generic and is parameterized by functions that allow one to express domain dependent knowledge. Hence, [16] shows that the matchings introduced in [4] and [2] are special cases of the graph similarity measure of [9].

In this paper, we address the problem of computing this graph similarity measure. Indeed, [9] has proposed a first greedy algorithm that incrementally builds multivalent matchings. These matchings are quickly computed but are usually far from optimality. Hence, we propose to improve the quality of the constructed matchings by using the Ant Colony Optimization (ACO) metaheuristic [11]: the idea is to use pheromone trails to keep track of the best components of matchings built by the greedy algorithm.

Section 2 briefly describes the generic graph similarity measure and the associated greedy algorithm introduced in [9]. Section 3 describes a new ACO algorithm —Ant Graph Matching (ANT-GM)— for computing this measure. Section 4 presents experimental results on different graph matching problems, and compares ANT-GM with the greedy algorithm of [9] and a reactive tabu search algorithm introduced in [16].

2 A generic similarity measure for multi-labeled graphs

2.1 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. 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 E of edges of the graph can be defined by $E = \{(v_i, v_j) | \exists l, (v_i, v_j, l) \in r_E\}$.

We shall call the tuples of r_V and r_E the vertex and edge features of G. The set $descr(G) = r_V \cup r_E$ of all vertex and edge features of a graph G completely describes the graph G.

2.2 Similarity measure

We now briefly describe the graph similarity measure introduced in [9], we refer the reader to [9] for more details. This similarity measure is defined for two multi-labeled graphs $G = \langle V, r_V, r_E \rangle$ and $G' = \langle V', r_{V'}, r_{E'} \rangle$, defined over the same sets of vertex and edge labels L_V and L_E, and such that $V \cap V' = \emptyset$.

The first step for measuring graph similarity is to match vertices. The matching function considered here is multivalent, *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 graphs G and G' is a set $m \subseteq V \times V'$ 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 feature. More formally, the set of common features $descr(G) \sqcap_m descr(G')$, with respect to a matching m, is defined as follows:

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

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:

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

The similarity of G and G' with respect to a matching m is then defined by:

$$sim_m(G, G') = \frac{f(descr(G) \sqcap_m descr(G')) - g(splits(m))}{f(descr(G) \cup descr(G'))} \tag{1}$$

where f and g are two functions that are defined to weight features and splits, depending on the considered application. For example, if f is the cardinality function and g is the null function, then the similarity is proportional to the number of common features with respect to the total number of features. If g

is the cardinality function, instead of the null function, then the similarity is decreased proportionally to the number of split vertices.

Finally, the maximal similarity $sim(G, G')$ of two graphs G and G' 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 in the definition 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 that maximizes the *score* function below:

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

2.3 Greedy algorithm

A greedy algorithm for approximating $sim(G, G')$ is introduced in [9]. We briefly describe it as it is used as a starting point of our ACO algorithm.

The algorithm starts from the empty matching $m = \emptyset$, and iteratively adds to this matching couples of vertices that are chosen within the set $cand = V \times V' - m$ in a greedy way: at each step, the algorithm first selects the set of couples $(u, u') \in cand$ that most increase the *score* function. This set of best scored couples often contains more than one couple. To break ties between them, the potentiality of each candidate (u, u') is looked ahead by taking into account the features that are shared by edges starting from (resp. ending to) both u and u' and that are not already in $descr(G) \sqcap_{m \cup \{(u,u')\}} descr(G')$. More formally, one defines the set $look\_ahead_m(u, u')$ of potential common edge features by:

$$\begin{aligned} &\{(u, v, l) \in r_E \mid \exists v' \in V', (u', v', l) \in r_{E'}\} \cup \{(u', v', l) \in r_{E'} \mid \exists v \in V, (u, v, l) \in r_E\} \\ \cup\ &\{(v, u, l) \in r_E \mid \exists v' \in V', (v', u', l) \in r_{E'}\} \cup \{(v', u', l) \in r_{E'} \mid \exists v \in V, (v, u, l) \in r_E\} \\ -\ &descr(G) \sqcap_{m \cup \{(u,u')\}} descr(G') \end{aligned}$$

The next couple to enter the matching is randomly selected within the set of couples (u, u') that most increase the *score* function and that maximize $f(look\_ahead(u, u'))$.

This greedy algorithm stops iterating when every couple neither directly increases the score function nor has looked-ahead common edge features.

3 Description of ANT-GM

The greedy algorithm may be run several times, in order to compute different matchings. We propose to combine such iterated greedy constructions with the Ant Colony Optimization (ACO) meta-heuristic, in order to take benefit of the previously computed matchings when building new ones.

The ACO meta-heuristic is a bio-inspired approach that has been used to solve different hard combinatorial optimization problems [14, 12]. The main idea is to model the problem to solve as the search for a minimum cost path

in a graph —called construction graph— and to use artificial ants to search for good paths. The behavior of artificial ants is inspired from real ants: they lay pheromone trails on graph components and they choose their path with respect to probabilities that depend on pheromone trails that have been previously laid, these pheromone trails progressively decrease by evaporation. Intuitively, this indirect stigmergic communication means aims at giving information about the quality of path components in order to attract ants, in the following iterations, towards the corresponding areas of the search space.

The proposed ACO algorithm for computing graph similarity follows the classical ACO algorithmic scheme for static combinatorial optimization problems [11]. At each cycle, each ant constructs a complete matching in a randomized greedy way, and then pheromone trails are updated. The algorithm stops iterating either when an ant has found an optimal matching, or when a maximum number of cycles has been performed.

3.1 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 $G = (V, r_V, r_E)$ and $G' = (V', r_{V'}, r_{E'})$, the construction graph is the complete non-directed graph that associates a vertex to each couple $(u, u') \in V \times V'$.

3.2 Pheromone trails

Ants communicate by laying pheromone trails on edges of the construction graph[3]. The amount of pheromone on an edge $< (u, u'), (v, v') >$ is noted $\tau_{<(u,u'),(v,v')>}$ and represents the learnt desirability of matching together u with u' and v with v'. Hence, to reward a matching m_i, ants lay pheromone trails between every pair of matched vertices $((u, u'), (v, v')) \in m_i^2$. Then, when constructing a new matching m_k, vertices that are matched in m_i will be more likely to be matched in m_k *if* m_k *already contains some matched vertices of* m_i. More precisely, the more m_k will contain matched vertices of m_i, the more the other matched vertices of m_i will be attractive.

3.3 Construction of a matching by an ant

At each cycle, each ant constructs a matching, starting from the 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

[3] We have defined another ACO algorithm where pheromone trails are laid on vertices of the construction graph (instead of edges). However, experiments showed us that this algorithm obtains much worse results than when pheromone is laid on edges.

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:

$$\frac{[\tau_m(u,u')]^\alpha \cdot [h1_m(u,u')]^{\beta_1} \cdot [h2_m(u,u')]^{\beta_2}}{\sum_{(v,v')\in cand}[\tau_m(v,v')]^\alpha \cdot [h1_m(v,v')]^{\beta_1} \cdot [h2_m(v,v')]^{\beta_2}} \quad (2)$$

where

- $\tau_m(u, u')$ is the pheromone factor and is defined by the sum of all pheromone trails laying between the candidate (u, u') and every couple (v, v') already selected in m, *i.e.,*

$$\tau_m(u,u') = \sum_{(v,v')\in m} \tau_{<(u,u'),(v,v')>}$$

  When $m = \emptyset$, *i.e.,* when choosing the first couple, $\tau_m(u, u') = 1$ so that the probability only depends on heuristic factors).
- $h1_m(u, u')$ is a first heuristic factor that aims at favoring couples that most increase the score function, *i.e.,*

$$h1_m(u,u') = score(m \cup \{(u,u')\}) - score(m)$$

- $h2_m(u, u')$ is a second heuristic factor that aims at favoring couples that have many looked-ahead features (as defined for the greedy algorithm described in Section 2), *i.e.,*

$$h2_m(u,u') = f(look\_ahead_m(u,u'))$$

- α, β_1, and β_2 are three parameters that determine the relative importance of the three factors.

Ants stop adding new couples to the matching m when every couple neither directly increases the score function nor has looked-ahead common edge features, or when the score function has not been increased since the last three iterations.

3.4 Pheromone updating step

Once every ant has constructed a matching, pheromone trails are updated according to the ACO meta-heuristic. First, evaporation is simulated by multiplying every pheromone trail $\tau_{<(u,u'),(v,v')>}$ by $(1 - \rho)$, where ρ is the pheromone evaporation rate 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 (if there are several best matchings, ties are randomly broken), 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 $1/(1 + score(m_{best}) - score(m_k))$. This quantity of pheromone is deposited on every edge $((u, u'), (v, v'))$ connecting two different couples (u, u') and (v, v') of m_k.

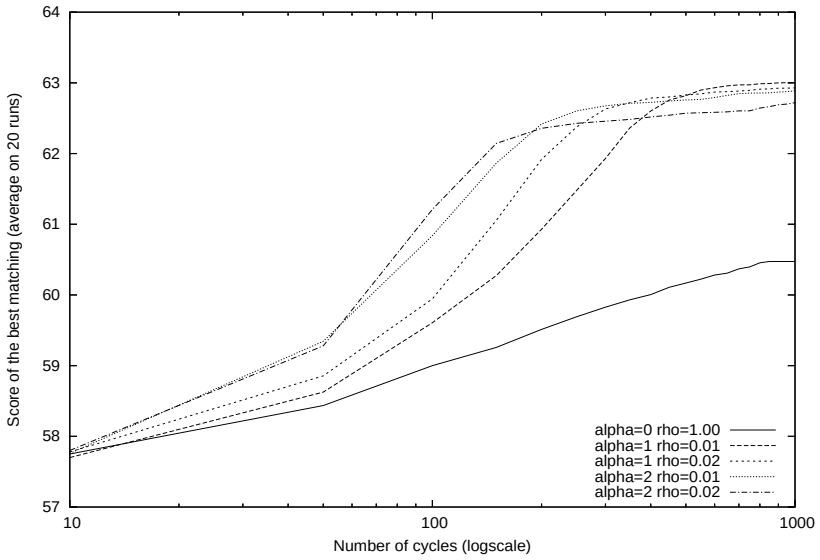

Fig. 1. Evolution of the score of the best found matching w.r.t. the number of cycles, for different settings of α and ρ (with 10 ants, $\beta_1 = 8$ and $\beta_2 = 3$)

4 Experimental study of ANT-GM

4.1 Influence of pheromone on the solution quality

As usually in ACO algorithms, the behavior of ANT-GM depends on its parameters, and more particularly on α, the pheromone factor weight, and ρ, the evaporation rate. Diversification can be emphasized both by decreasing α, so that ants become less sensitive to pheromone trails, and ρ, so that pheromone evaporates more slowly. When increasing the exploratory ability of ants in this way, better solutions are found, but as a counterpart it takes more longer time.

This is illustrated in Figure 1 on the $si2r001s80$ UNINA instance [13]. On this figure, one can remark that when α or ρ increase, ants converge quicker towards a matching: convergence occurs around cycle 500 when α=1 and ρ=0.01, around cycle 350 when α=1 and ρ=0.02, and around cycle 200 when α=2 and ρ=0.02. As a counterpart, ants find better matchings, at the end of the solution process, when α and ρ are set to lower values such as α=1 and ρ=0.01.

Note also that when α=0 and ρ=1, *i.e.,* when pheromone is totaly ignored, so that the solution process is a pure randomized greedy one, the constructed matchings have a much lower score and hardly reach 60.5 at the end of the

solution process, instead of more than 62.5 when pheromone is used. This shows that pheromone improves the solution quality.

4.2 Comparison of ANT-GM with Greedy and Reactive approaches

Considered algorithms. We compare our ACO algorithm (`ANT-GM`) with the Greedy Search algorithm (`GS`) of [9] described in Section 2 and a Reactive Tabu Search algorithm (`RTS`) described in [16].

`RTS` improves a matching built by the greedy search algorithm of [9] by performing local search: the idea is to iteratively move from a matching to one of its neighbours (obtained by either adding or removing one couple of vertices) until the optimal solution is found or until a maximum number of moves have been performed. At each step, the search moves towards the best neighbour of the current matching (with respect to the same criteria than for the greedy algorithm). To avoid being trapped in locally optimal matchings, a Tabu list is used that memorizes the last moves in order to forbid backward moves. As proposed in [3], the length of this Tabu list is dynamically adapted during the search, depending on the need for diversification/intensification.

Experimental Setup. `ANT-GM`, `GS`, and `RTS` have been implemented in C++, and run on a 1.8Ghz pentium M with 512Mo RAM.

For `ANT-GM`, we have set α to 1, ρ to 0.01, β_1 to 8, β_2 to 3, the maximum number of cycles $MaxCycle$ to 1000 and the number of ants $nbAnts$ to 10, so that each run builds $10,000$ matchings. Parameters of `RTS` have been set as recommended in [16].

To compare algorithms independently from implementation issues, all runs on a given instance are limited to a same number of moves, where a move is defined by the addition or removal of one couple of vertices to a matching. This limit on the number of moves depends on the considered instance. Indeed, one run of `ANT-GM` builds $10,000$ matchings, but the size of these matchings, and therefore the number of moves performed by `ANT-GM`, depends on the considered instance. Hence, let x be the average size of the matchings built by `ANT-GM` for a given instance, the number of moves performed by `ANT-GM` is $x * 10,000$ so that the maximum number of moves for this instance is set to $x * 10,000$.

Each algorithm has been run 20 times on each instance of each benchmark.

Results on subgraph isomorphism problems. We first consider 11 benchmarks of subgraph isomorphism problems, for non labeled graphs, issued from a UNINA benchmark [13] and available at `http://amalfi.dis.unina.it.graph`. For each of these 11 benchmarks, we have considered the 30 first instances.

Each instance is composed of two graphs $G=(V,E)$ and $G'=(V',E')$ such that $|V| \leq |V'|$, and the goal is to find an injective function $\phi : V \rightarrow V'$ such that $(v_1, v_2) \in E \Rightarrow (\phi(v_1), \phi(v_2)) \in E'$.

To solve subgraph isomorphism problems with the generic similarity measure of formula (1), we define function g as the cardinality function and function f as a weighted sum where the weight of the features of G (resp. G') is 1 (resp.

Benchmark	ANT-GM				GS				RTS			
Name (nb vertices)	GSR	ISR	Mv	T	GSR	ISR	Mv	T	GSR	ISR	Mv	T
si2r001s100 (20/100)	76.8	86.7	40492	33.2	33.3	33.3	89	0.2	67.5	100.0	9758	6.6
si2r001s80 (16/80)	93.3	100.0	42240	10.1	33.3	33.3	37	0.0	90.0	100.0	5585	2.4
si2r001s60 (12/60)	99.7	100.0	22164	2.8	46.7	46.7	15	0.0	99.2	100.0	1590	0.4
si4r001s80 (32/80)	81.3	90.0	110818	44.1	23.3	23.3	507	0.5	85.7	100.0	8292	7.5
si4r001s60 (24/60)	99.2	100.0	44539	9.0	40.0	40.0	39	0.1	93.2	100.0	5066	2.5
si4r001s40 (16/40)	100.0	100.0	8634	0.7	53.3	53.3	41	0.0	99.7	100.0	1759	0.4
si4r001s20 (8/20)	100.0	100.0	166	0.0	83.3	83.3	9	0.0	100.0	100.0	219	0.0
si4r005s40 (16/40)	89.7	96.7	34976	4.4	6.7	6.7	67	0.0	88.0	96.7	4647	1.0
si6r001s60 (36/60)	99.7	100.0	79738	21.0	63.3	63.3	110	0.1	94.5	100.0	6964	5.2
si6r001s40 (24/40)	100.0	100.0	16547	1.9	86.7	86.7	44	0.0	98.3	100.0	3101	1.0
si6r001s20 (12/20)	100.0	100.0	352	0.0	93.3	93.3	24	0.0	100.0	100.0	266	0.0
Average	94.5	97.6	36424	11.6	51.2	51.2	89	0.1	92.4	99.7	4295	2.45

Table 1. Results on 11 benchmark sets of subgraph isomorphism problems. For each benchmark set, the table first reports its name and the number of vertices of the two graphs to be matched. Then, for each algorithm, it reports the global success rate (GSR), *i.e.,* the percentage of successful runs over all runs for all instances of the benchmark, the instance success rate (ISR), *i.e.,* the percentage of instances that have been solved at least once over the twenty runs, and the number of moves (Mv) and the CPU time (T) spent to find the solution (average on successful runs only).

0). In this case, $sim(G, G')$=1 if and only if there exists a mapping m such that $descr(G) \subseteq descr(G) \sqcap_m descr(G')$ (as $f(descr(G) \cup descr(G')) = |descr(G)|$) and $splits(m) = \emptyset$, *i.e.*, $sim(G, G') = 1$ if and only if there exists a subgraph isomorphism.G and G'.

Table 1 reports results obtained on these subgraph isomorphism problems. These results first show that GS is much less successfull than both ANT-GM and RTS, being able to solve nearly twice as less instances. Moreover, global and instance success rates of GS are always equal and, when a solution is found, the number of moves performed to find it is always very low. Indeed, the search is not much diversified in GS: random choices are performed only to break ties between candidates that have equally highest scores. As a consequence, GS always computes very similar matchings and, given an instance, either it very quickly finds a solution, or it never finds it.

When comparing ANT-GM with RTS, one can note that the global success rate of ANT-GM is nearly always greater or equal to the one of RTS: 94.5% of the $20 * 30 * 11$ runs of ANT-GM have succeeded instead of 92.4% for RTS. However, the instance rate of ANT-GM is always smaller or equal to the instance success rate of RTS: 97.6% of the $30 * 11$ considered instances have been solved at least once over the 20 runs of ANT-GM instead of 99.7% for RTS. Actually, given an instance, the result of an execution of ANT-GM is nearly always the same (*i.e.,* either it nearly always fail or it nearly always succeed), whereas the result of an execution of RTS is more variable and highly depends on the starting point of the local search.

Problem	ANT-GM			RTS		
name	Sim	Mv	T	Sim	Mv	T
hom-v20-e60	0.795	303167	30.9	0.798	17747	2.2
hom-v30-e90	0.863	512746	155.0	0.865	14187	4.4
hom-v40-e120	0.885	685155	477.9	0.895	24801	13.7
hom-v45-e135	0.895	717767	709.5	0.904	60085	40.5
hom-v50-e150	0.804	847699	1075.6	0.913	53922	47.9
Average	0.848	613307	489.8	0.875	34149	21.7

Table 2. Results on 5 multivalent matching problems. For each problem and for each algorithm, the table displays the average similarity (Sim), the average number of moves (Mv) and the average CPU time in seconds (T) needed to find the best solution.

Table 1 also shows that ANT-GM performs 8.5 times as more moves as RTS to find a solution. However, as one move of ANT-GM is performed twice as fast, RTS is 4.7 times as fast as ANT-GM.

Experimental comparison on multivalent matching problems. We have also compared ANT-GM and RTS on 5 multivalent graph matching problems that have been randomly generated. Each problem named hom-vN-eM is composed of a couple of non labeled graphs such that the first graph has N vertices and M edges (randomly generated) and the second graph is obtained by randomly removing 6 vertices and their incident egdes of the first graph, and then randomly splitting 5 vertices and their incident edges.

Table 2 shows the results obtained by ANT-GM and RTS on these multivalent matching problems. On this table, one can note that similarities computed by ANT-GM are slightly worse than those computed by RTS. Moreover, when graph sizes increase, this difference in quality becomes more important. Also, ANT-GM needs more moves to converge towards its best solution, and therefore it is more time consuming.

5 Conclusion

We have introduced in this paper ANT-GM, a new ACO algorithm for solving multivalent graph matching problems. First experiments on benchmarks of subgraph isomorphism problems showed us that ANT-GM is able to solve to optimality a wide majority of these problems. A key point of the multivalent graph matching problem is that each vertex may be mapped to a set of vertices, so that it can be used to evaluate the similarity of two graphs, and not only their equivalence, inclusion or intersection. As there does not yet exist benchmarks dedicated to this problem, we have generated random instances. Experiments showed us that, on these problems, ANT-GM is outperformed by a Reactive Tabu Search approach.

Further work will mainly concern the integration within ANT-GM of some local search technics such as the one used by RTS. Indeed, experiments showed

us that results obtained by ANT-GM and RTS are rather complementary, each algorithm being able to solve instances that the other one cannot solve. Actually, 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 [12, 15].

References

1. A. Aamodt and E. Plaza. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. AI Communications, IOS Press, Amsterdam (NL), 7(1):39-59, 1994.
2. R. Ambauen, S. Fischer, and H. Bunke. Graph Edit Distance with Node Splitting and Merging, and Its Application to Diatom Identification. IAPR-TC15 Wksp on Graph-based Representation in Pattern Recognition, LNCS, Springer Verlag, 95-106, 2003.
3. R. Battiti and M. Protasi. Reactive Local Search for the Maximum Clique Problem. Algorithmica, Springer-Verlag, (29), 610-637, 2001.
4. M. Boeres, C. Ribeiro, and I. Bloch. A Randomized Heuristic for Scene Recognition by Graph Matching. WEA 2004, 100-113, 2004.
5. H. Bunke. Error-tolerant Graph Matching: A Formal Framework and Algorithms. Lecture Notes in Computer Science. Springer, Berlin, 1998.
6. H. Bunke and B.T. Messmer. Recent advances in graph matching. International Journal of Pattern Recognition and Artificial Intelligence, (11):169-203, 1997.
7. H. Bunke and K. Shearer. A graph distance metric based on maximal common subgraph. Pattern recognition letters, (19):255-259, 1998.
8. H. Bunke and X. Jiang. Graph matching and similarity. Volume Teodorescu, H-N, Mlynek, D. Kandel, A. Zimmermann, H-J. (ds.): Intelligent Systems and Interfaces, chapter 1, 2000.
9. P. Champin and C. Solnon. Measuring the similarity of labeled graphs. 5th International Conference on Case-Based Reasoning (ICCBR). Lecture Notes in Computer Science - Springer Verlag, 2003.
10. D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. International Journal of Pattern Recognition and Artificial Intelligence, 18(3):265-298, 2004.
11. 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.
12. M. Dorigo and L. Gambardella. Ant Colony System: A cooperative learning approach to traveling salesman problem. IEEE transactions on evolutionary computation, 1(1):53-66, 1997.
13. P. Foggia, C. Sansone, and M. Vento. A database of graphs for isomorphism and sub-graph isomorphism benchmarking. In 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern recognition, pages 176-187, 2001.
14. V. Maniezzo and A.Colorni. The Ant System Applied to the Quadratic Assignement Problem. IEEE Transactions on Data and Knowledge Engineering, 11(5):769-778, 1999.
15. T. Stützle and H.H. Hoos. $\mathcal{MAX} - \mathcal{MIN}$ Ant System. Journal of Future Generation Computer Systems, 16:889-914,2000.

16. S. Sorlin and C. Solnon. Reactive Tabu Search for Measuring Graph Similarity. to appear in 5th IAPR Workshop on Graph-based Representations in Pattern Recognition (GbR 2005), LNCS, Springer Verlag, 2005.