A First Step Towards Exact Graph Edit Distance Using Bipartite Graph Matching

Miquel Ferrer[1(✉)], Francesc Serratosa[2], and Kaspar Riesen[1]

[1] Institute for Information Systems, University of Applied Sciences and Arts, Riggenbachstrasse 16, Olten, 4600, Switzerland
miquel.ferrer@fhnw.ch

[2] Departament d'Enginyeria Informàtica i Matemàtiques, Universitat Rovira i Virgili, Avda. Països Catalans 26, 43007 Tarragona, Spain

Abstract. In recent years, a powerful approximation framework for graph edit distance computation has been introduced. This particular approximation is based on an optimal assignment of local graph structures which can be established in polynomial time. However, as this approach considers the local structural properties of the graphs only, it yields suboptimal solutions that overestimate the true edit distance in general. Recently, several attempts for reducing this overestimation have been made. The present paper is a starting point towards the study of sophisticated heuristics that can be integrated in these reduction strategies. These heuristics aim at further improving the overall distance quality while keeping the low computation time of the approximation framework. We propose an iterative version of one of the existing improvement strategies. An experimental evaluation clearly shows that there is large space for further substantial reductions of the overestimation in the existing approximation framework.

1 Introduction

Graph edit distance [1, 2] is one of the most flexible and versatile approaches to error-tolerant graph matching. In particular, graph edit distance is able to cope with directed and undirected, as well as with labeled and unlabeled graphs. In addition, no constraints have to be considered on the alphabets for node and/or edge labels. Moreover, through the concept of cost functions graph edit distance can be adapted and tailored to diverse applications [3, 4]. An extensive survey about graph edit distance can be found in [5].

The major drawback of graph edit distance is its high computational complexity that restrict its applicability to graphs of rather small size. In fact, graph edit distance belongs to the family of *quadratic assignment problems* (QAPs), which belong to the class of $\mathcal{NP}$*-complete* problems. Therefore, exact computation of graph edit distance can be solved in exponential time complexity only.

In recent years, a number of methods addressing the high computational complexity of graph edit distance have been proposed (e.g. [6–9]). Beyond these works, an algorithmic framework based on bipartite graph matching has been

C.-L. Liu et al. (Eds.): GbRPR 2015, LNCS 9069, pp. 77–86, 2015.
DOI: 10.1007/978-3-319-18224-7_8

introduced recently [10, 11]. The main idea behind this approach is to convert the difficult problem of graph edit distance to a *linear sum assignment problem* (LSAP). LSAPs basically constitute the problem of finding an optimal assignment between two independent sets of entities, for which a collection of polynomial algorithms exists [12]. In [10, 11] the LSAP is formulated on the sets of nodes including local edge information. The main advantage of this approach is that it allows the approximate computation of graph edit distance in a substantially faster way than traditional methods. However, during the node assignment only local instead of global structural information is taken into account. Hence, this might lead to incorrect node assignments compared with an exact matching and thus, the derived edit distance is equal to, or larger than, the exact graph edit distance.

In order to overcome this problem and reduce the overestimation of the true graph edit distance, a variation of the original framework [10] has been proposed in [13]. Given the initial assignment found by the bipartite framework, the main idea is to introduce a post-processing step such that the number of incorrect assignments is decreased (which in turn reduces the overestimation). The proposed post-processing varies the original node assignment by systematically swapping the target nodes of two node assignments. In order to search the space of assignment variations a *beam search* (i.e. a tree search with pruning) is used. One of the most important observations derived from [13] is that given an initial node assignment, one can substantially reduce the overestimation using this local search method. Yet, beam search is sub-optimal in the sense of possibly pruning the optimal solution in an early stage of the search process.

Now the crucial question arises, how the space of assignment variations could be explored such that promising parts of the search tree are not (or at least not too early) pruned. In [13] the initial assignment is systematically varied without using any kind of heuristic or additional information to keep the best potential assignment unpruned in the tree. In particular it is not taken into account that certain nodes and/or local assignments have greater impact than other on the graph edit distance approximation and should thus be considered first in the beam search process. Clearly, considering more important node assignments and/or nodes in an early stage of the beam search process might reduce the risk of pruning the optimal assignment. In this sense we argue that the introduction of procedures and heuristics that guide the order in which the assignments are varied during beam search is a rewarding line of research.

The main objective of the present paper is to start the investigation towards new heuristics that improve the overall quality of the node assignment. In particular, we propose an iterative version of [13] to derive randomized permutations of the original mapping which serve as starting point for beam search improvements. Hence, the present paper introduces a heuristic in order to answer the general question to what extent the ordering of the nodes affects the mapping quality.

Next, in Section 2, the original bipartite framework for graph edit distance approximation [10] as well as its recent extension [13], named *BP-Beam*, are summarized. In Section 3 our novel iterative version of *BP-Beam* is described. An experimental evaluation on diverse data sets is carried out in Section 4. Finally, in Section 5 we draw conclusions and outline some possible tasks and extensions for future work.

2 Approximate Graph Edit Distance Computation

Given two graphs, g_1 and g_2, the basic idea of graph edit distance is to transform g_1 into g_2 using edit operations, namely, *insertions*, *deletions*, and *substitutions* of both nodes and edges. The substitution of two nodes u and v is denoted by $(u \rightarrow v)$, the deletion of node u by $(u \rightarrow \epsilon)$, and the insertion of node v by $(\epsilon \rightarrow v)$[1]. A sequence of edit operations $e_1, \ldots, e_k$ that transform g_1 completely into g_2 is called an edit path between g_1 and g_2.

To find the most suitable edit path out of all possible edit paths between two graphs, a cost measuring the strength of the corresponding operation is introduced. The edit distance between two graphs g_1 and g_2 is then defined by the minimum cost edit path between them. Exact computation of graph edit distance is usually carried out by means of a tree search algorithm (e.g. A*) which explores the space of all possible mappings of the nodes and edges of the first graph to the nodes and edges of the second graph.

2.1 Bipartite Graph Edit Distance Approximation

The computational complexity of exact graph edit distance is exponential in the number of nodes of the involved graphs. That is considering n nodes in g_1 and m nodes in g_2, the set of all possible edit paths contains $O(n^m)$ solutions to be explored. This means that for large graphs the computation of edit distance is intractable. In order to reduce its computational complexity, in [10], the graph edit distance problem is transformed into a linear sum assignment problem (LSAP). To this end, based on the node sets $V_1 = \{u_1, \ldots, u_n\}$ and $V_2 = \{v_1, \ldots, v_m\}$ of g_1 and g_2 respectively, a cost matrix C is first established as follows:

$$C = \left[\begin{array}{cccc|cccc} c_{11} & c_{12} & \cdots & c_{1m} & c_{1\epsilon} & \infty & \cdots & \infty \\ c_{21} & c_{22} & \cdots & c_{2m} & \infty & c_{2\epsilon} & \cdots & \infty \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nm} & \infty & \infty & \cdots & c_{n\epsilon} \\ \hline c_{\epsilon 1} & \infty & \cdots & \infty & 0 & 0 & \cdots & 0 \\ \infty & c_{\epsilon 2} & \cdots & \infty & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \infty & \vdots & \vdots & \ddots & \vdots \\ \infty & \infty & \cdots & c_{\epsilon m} & 0 & 0 & \cdots & 0 \end{array}\right]$$

[1] Similar notation is used for edges.

Entry c_{ij} denotes the cost of a node substitution $(u_i \rightarrow v_j)$, $c_{i\epsilon}$ denotes the cost of a node deletion $(u_i \rightarrow \epsilon)$, and $c_{\epsilon j}$ denotes the cost of a node insertion $(\epsilon \rightarrow v_j)$. The left upper corner of the cost matrix represents the costs of all possible node substitutions, the diagonal of the right upper corner the costs of all possible node deletions, and the diagonal of the bottom left corner the costs of all possible node insertions. In each entry c_{ij}, not only the cost of node operation is taken into account but also the minimum sum of edge edit operation costs, implied by the corresponding node operation. That is, the matching cost of the local edge structure is encoded in the individual entries of C.

In a second step of [10], an assignment algorithm is applied to the square cost matrix $C = (c_{ij})$ in order to find the minimum cost assignment of the nodes (and their local edge structure) of g_1 to the nodes (and their local edge structure) of g_2. Note that this task exactly corresponds to an instance of an LSAP and can thus be optimally solved in polynomial time by several algorithms [12].

Any of the LSAP algorithms will return a permutation $(\varphi_1, \ldots, \varphi_{n+m})$ of the integers $(1, 2, \ldots, (n+m))$, which minimizes the overall mapping cost $\sum_{i=1}^{(n+m)} c_{i\varphi_i}$. This permutation corresponds to the mapping

$$\psi = \{u_1 \rightarrow v_{\varphi 1}, u_2 \rightarrow v_{\varphi 2}, \ldots, u_{m+n} \rightarrow v_{\varphi_{m+n}}\}$$

of the nodes of g_1 to the nodes of g_2. Note that ψ does not only include node substitutions $(u_i \rightarrow v_j)$, but also deletions and insertions $(u_i \rightarrow \epsilon)$, $(\epsilon \rightarrow v_j)$ and thus perfectly reflects the definition of graph edit distance (mappings of the form $(\epsilon \rightarrow \epsilon)$ can be dismissed, of course). Hence, mapping ψ can be interpreted as partial edit path between g_1 and g_2, which considers operations on nodes only.

In a third step, the partial edit path ψ between g_1 and g_2 is completed with respect to the edges. This can be accomplished since edge edit operations are implied by edit operations on their adjacent nodes. That is, whether an edge is substituted, deleted, or inserted, depends on the edit operations performed on its adjacent nodes. The total cost $d_{\langle\psi\rangle}(g_1, g_2)$ of the completed edit path between graphs g_1 and g_1 is finally returned as approximate graph edit distance. We refer to this graph edit distance approximation algorithm as *BP*(g_1, g_2).

2.2 Improving the Approximation Using Beam Search

Several experimental evaluations indicate that the suboptimality of *BP*, i.e. the overestimation of the true edit distance, is very often due to a few incorrectly assigned nodes in ψ with respect to the optimal edit path. The extension presented in [13] ties in at this observation. In particular, the node assignment ψ is used as a starting point for a subsequent search in order to improve the quality of the distance approximation.

In [13], the original node assignment ψ is systematically varied by swapping the target nodes v_{φ_i} and v_{φ_j} of two node assignments $(u_i \rightarrow v_{\varphi_i}) \in \psi$ and $(u_j \rightarrow v_{\varphi_j}) \in \psi$. For each swap it is verified whether (and to what extent) the derived distance approximation stagnates, increases or decreases. For a systematic variation of mapping ψ a tree search is used.

Algorithm 1. *BP-Beam*(g_1, g_2, ψ, b)

1. $d_{best} = d_{\langle\psi\rangle}(g_1, g_2)$
2. Initialize $open = \{(\psi, 0, d_{\langle\psi\rangle}(g_1, g_2))\}$
3. **while** *open* is not empty **do**
4. Remove first tree node in *open*: $(\psi, q, d_{\langle\psi\rangle}(g_1, g_2))$
5. **for** $j = (q+1), \ldots, (m+n)$ **do**
6. $\psi' = \psi \setminus \{u_{q+1} \to v_{\varphi_{q+1}}, u_j \to v_{\varphi_j}\} \cup \{u_{q+1} \to v_{\varphi_j}, u_j \to v_{\varphi_{q+1}}\}$
7. Derive approximate edit distance $d_{\langle\psi'\rangle}(g_1, g_2)$
8. $open = open \cup \{(\psi', q+1, d_{\langle\psi'\rangle}(g_1, g_2))\}$
9. **if** $d_{\langle\psi'\rangle}(g_1, g_2) < d_{best}$ **then**
10. $d_{best} = d_{\langle\psi'\rangle}(g_1, g_2)$
11. **end if**
12. **end for**
13. **while** size of $open > b$ **do**
14. Remove tree node with highest approximation value $d_{\langle\psi\rangle}$ from *open*
15. **end while**
16. **end while**
17. **return** d_{best}

The tree nodes in the search procedure correspond to triples $(\psi, q, d_{\langle\psi\rangle})$, where ψ is a certain node assignment, q denotes the depth of the tree node in the search tree and $d_{\langle\psi\rangle}$ is the approximate distance value corresponding to ψ. The root node of the search tree refers to the optimal node assignment ψ found by *BP*. Hence, the root node (with depth $= 0$) is given by the triple $(\psi, 0, d_{\langle\psi\rangle})$. Subsequent tree nodes $(\psi', q, d_{\langle\psi'\rangle})$ with depth $q = 1, \ldots, (m+n)$ contain node assignments ψ' with swapped element $(u_q \to v_{\varphi_q})$.

As usual in tree search based methods, a set *open* is employed that holds all of the unprocessed tree nodes. The tree nodes in *open* are kept sorted in ascending order according to their depth in the search tree (known as *breadth-first search*). As a second order criterion the approximate edit distance $d_{\langle\psi\rangle}$ is used.

The extended framework with the tree search based improvement is given in Alg. 1. As long as *open* is not empty, we retrieve (and remove) the triple $(\psi, q, d_{\langle\psi\rangle})$ at the first position in *open*, generate the successors of this specific tree node and add them to *open*. To this end all pairs of node assignments $(u_{q+1} \to v_{\varphi_{q+1}})$ and $(u_j \to v_{\varphi_j})$ with $j = (q+1), \ldots, (n+m)$ are individually swapped resulting in two new assignments $(u_{q+1} \to v_{\varphi_j})$ and $(u_j \to v_{\varphi_{q+1}})$. In order to derive node mapping ψ' from ψ, the original node assignment pair is removed from ψ and the swapped node assignment is added to ψ'. Since index j starts at $(q+1)$ we also allow that a certain assignment $u_{q+1} \to v_{\varphi_{q+1}}$ remains unaltered at depth $(q+1)$ in the search tree.

Since every tree node in our search procedure corresponds to a complete solution and the cost of these solutions neither monotonically decrease nor increase with growing depth in the search tree, we need to buffer the best possible distance approximation found during the tree search in d_{best} (which is returned as soon as *open* is empty)

As stated before, given a mapping ψ from *BP*, the derived edit distance overestimates the true edit distance in general. This overestimation is due to some incorrect node mappings in ψ. Hence, the objective of any post-processing should

be to find a variation ψ' of the original mapping ψ such that $d_{\langle\psi'\rangle} < d_{\langle\psi\rangle}$. However, the search space of all possible permutations of ψ contains $(n+m)!$ possibilities, making an exhaustive search (starting with ψ) both unreasonable and intractable. Therefore, only the b assignments with the lowest approximate distance values are kept in *open* at all time (known as *beam search*). Note that parameter b can be used as trade-off parameter between run time and approximation quality. That is, it can be expected that larger values of b lead to both better approximations and increased run time (and vice versa). From now on we refer to this variant of the approximation framework as *BP-Beam*(g_1, g_2, ψ, b).

3 Iterative BP-Beam

Note that the the successors of tree node $(\psi, q, d_{\langle\psi\rangle})$ are generated in fixed order in *BP-Beam*. In particular, the assignments of the original node matching ψ are processed according to the depth q of the current search tree node. That is, at depth q the assignment $(u_q \to v_{\varphi_q})$ is processed and swapped with other assignments. Note that beam search prunes quite large parts of the tree during the search process. Hence, processing correct node assignments at the top of the search tree is somewhat useless and moreover runs the risk of potentially pruning crucial parts of the tree at an early stage of the search. That is, the fixed order processing, which does not take any information about the individual node assignments into account, is a clear drawback of the procedure described in [13].

Clearly, it would be highly favorable to process important node assignments as early as possible in the tree search. Our hypothesis is that there should exist some heuristics that indicate which node assignments of ψ are the most important or critical ones and should thus be processed first. Finding such heuristics that indicate the impact of a single node assignment on the approximation quality turns out to be a highly non-trivial task. Moreover, it is not yet proven whether the order of the assignment processing actually has a great impact on the resulting distance quality.

As a starting point towards the question of whether or not the ordering of the assignment processing in *BP-Beam* has great influence on the quality of the distance approximation, we propose a procedure with random re-orderings of the individual assignments. Thus, given a mapping ψ obtained from *BP*, we propose to perform several iterations over *BP-Beam*. In every iteration the original mapping ψ is randomly reordered and fed into *BP-Beam*. This leads to an iterative version of *BP-Beam* called *IBP-Beam* from now on. *IBP-Beam* takes two parameters, namely the number of iterations k and the beam size b.

The algorithm *IBP-Beam* is given in Alg. 2. The first three lines correspond to the three major steps of the original approximation. Then, the main loop is carried out k times. In each iteration a newly ordered assignment ψ' is generated by randomly permuting the original assignment ψ derived from *BP*. Then, the assignment and the corresponding distance are possibly improved using *BP-Beam* taking ψ' as starting point for the tree search. Whenever the *BP-Beam*

Algorithm 2. IBP-Beam(g_1,g_2, k, b)

1. Build cost matrix $C = (c_{ij})$ according to the input graphs g_1 and g_2
2. Compute optimal node assignment $\psi = \{u_1 \to v_{\varphi 1}, \ldots, u_{m+n} \to v_{\varphi_{m+n}}\}$ on C
3. $d_{best} = d_{\langle\psi\rangle}(g_1, g_2)$
4. $i = 1$
5. **while** $i \leq k$ **do**
6. $\quad \psi' = RandomPermutation(\psi)$
7. $\quad d_{beam} = BP\text{-}Beam(g_1, g_2, \psi', b)$
8. $\quad d_{best} = \min\{d_{best}, d_{beam}\}$
9. $\quad i++$
10. **end while**
11. **return** d_{best}

is able to further decrease the distance approximation, the best solution d_{best} is replaced by the novel approximation.

Clearly, further reductions of the overestimation with this random iterative procedure, would highly encourage the hypothesis that there might be heuristics that would solve the same task of reordering in a deterministic manner (in particular when the number of iterations k needed remains small).

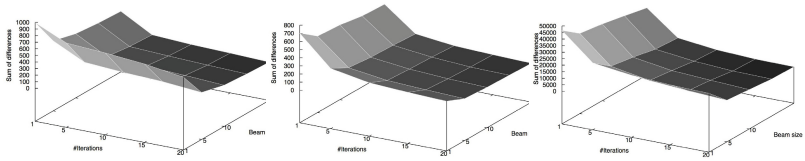
4 Experimental Evaluation

The goal of the experimental evaluation is to verify whether the proposed extension is able to reduce the overestimation of graph edit distance approximation returned by *BP* and in particular *BP-Beam*, and how the iterative process affects the computation time. Three data sets from the IAM graph database repository involving molecular compounds (AIDS), fingerprint images (Fingerprint), and symbols from architectural and electronic drawings (GREC) are used to carry out this experimental part. For further details about these data sets we refer to [14]. For all data sets, subsets of 100 graphs are randomly selected on which 10,000 pairwise graph edit distance computations are performed.

4.1 Impact of Meta Parameter *b* and *k*

In this first experiment we aim at researching the impact of the two parameters b and k on *IBP-Beam*. To this end we perform several executions of the *IBP-Beam* with $b, k \in \{1, 5, 10, 15, 20\}$, leading to 25 different distance approximations. We compute the sum of the difference between the exact distance (A^*) and the distance obtained by *IBP-Beam*. Figure 1 shows such sum of differences as a function of the number of iterations k (x-axis) and the beam size b (y-axis).

First, we observe that the sum of differences is monotonically reduced as long as k and b are increased. We can also observe that we are able to obtain distance values very close to the exact distance on all data sets. Finally (and probably most importantly), we note that the major part of the reduction in the sum of differences is already obtained with values of k and b of 5. Further increases of both parameters lead to relatively small further reductions of the overestimation.



(a) AIDS (b) Fingerprint (c) GREC

Fig. 1. Sum of difference between A^* and *IBP-Beam(k,b)* as a function of the number of iterations k (x-axis) and the beam size b (y-axis)

4.2 Relative Overestimation and Computation Time

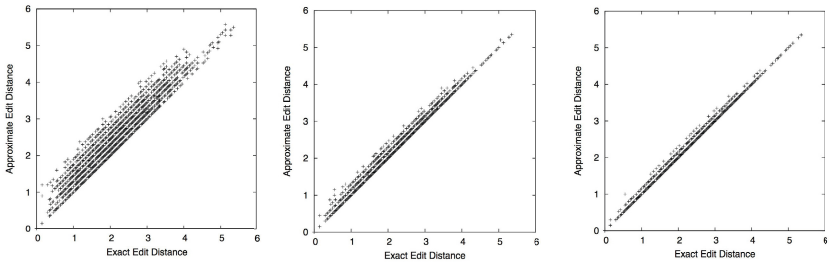
Next we measure the mean relative overestimation ϕo [%] and the mean computation time ϕt [ms] for all algorithms (see Table 1). The mean relative overestimation ϕo of a certain approximation is computed as the relative difference to the sum of distances returned by A^*. The relative overestimation of A^* is thus zero and the value of ϕo for *BP* is taken as reference value and corresponds to 100%. The computation times ϕt measures the average matching time for a pair of graphs. Note that for both *BP-Beam* and *IBP-Beam* the beam size b is fixed to 5 (thus this parameter is not shown in the Table 1, but only k).

Table 1. The mean relative overestimation of the exact distance (ϕo) in %, and the mean run time for one matcing (ϕt) in ms. for each data set and for a given algorithm. The beam size for *BP-Beam* and *IBP-Beam* algorithm is set to 5, and parameter k is varied from 5 to 20 for *IBP-Beam*.

	AIDS		**Fingerprint**		**GREC**	
Algorithm	ϕo	ϕt	ϕo	ϕt	ϕo	ϕt
A^*	0.00	25750.22	0.00	31645.08	0.00	7770.81
BP	100.00	0.28	100.00	0.35	100.00	0.27
BP-Beam	15.09	1.82	24.57	1.45	16.98	2.62
IBP-Beam(5)	9.27	7.81	8.63	5.70	8.53	12.01
IBP-Beam(10)	6.11	15.27	6.11	10.98	5.72	23.57
IBP-Beam(15)	4.85	22.79	5.21	16.14	4.57	35.81
IBP-Beam(20)	4.26	30.41	4.51	20.94	3.73	47.19

Regarding the overestimation ϕo we observe a substantial improvement of the distance quality using *BP-Beam* rather than *BP*. For instance, on the AIDS data the overestimation is reduced by 85% (similar results are obtained on the other data sets). By using *IBP-Beam* further substantial reductions of the overestimation are possible on all data sets. For instance, on the AIDS data set using *IBP-Beam* with $k = 5$ rather than *BP-Beam* enables a reduction from 15.09% to 9.27% (similar or even better results are obtained on the other data sets). Increasing the number of iterations further decreases the overestimation such that a distance accuracy very near to the exact edit distance is possible with our

novel approach. These substantial reductions of the overestimation from *BP* to *BP-Beam* and from *BP-Beam* to *IBP-Beam* can also be seen in Figure 2 where for each pair of graphs the exact distance (x-axis) is plotted vs. the distance obtained by an approximation algorithm (y-axis). In fact, for *IBP-Beam* the line-like scatter plot along the diagonal suggests that the approximation is very near to the optimal distance.



(a) *BP* (b) *BP-Beam(5)* (c) *IBP-Beam(20,5)*

Fig. 2. Exact (x-axis) vs. approximate (y-axis) edit distance on the AIDS dataset computed with (a) *BP*, (b) *BP-Beam(5)*, (c) *IBP-Beam(20,5)*

Regarding the computation time ϕt we can report that *BP* provides the lowest computation time on all data sets (approximately 0.3ms per matching on all data sets). Yet, remember that this fast matching time is at the expense of the highest overestimation. *BP-Beam* increases the computation time to approximately 2ms per matching and *IBP-Beam* further increases the average run time to several milliseconds per matching. As expected, the run time of *IBP-Beam* linearly grows with parameter k. However, it is important to remark that in all cases the computation time is much lower than those of A^*. Overall *IBP-Beam(5)* seems to be a good trade-off between computation time and reduction of the overestimation.

5 Conclusions and Future Work

In recent years a framework based on bipartite graph matching to derive approximate solutions of the graph edit distance has been presented. In its original version it suffers from a high overestimation of the computed distance with respect to the true edit distance. In this paper, we propose an iterative extension of one of the existing bipartite-based graph edit distance approximation algorithm. The aim of the paper is to empirically investigate the influence of the order in which the assignments are explored in a post processing search process on the distance quality. The experimental evaluation on three different databases verifies that this order is actually one of the critical factors to improve the overall distance quality. Though the run times are increased when compared to our former framework (as expected), they are still far below the run times of the exact algorithm. The presented approach can be seen as a first step towards

finding determinant heuristics to guide the search through the space of possible assignment variants (starting with ψ).

Acknowledgements. This work has been supported by the Swiss National Science Foundation (SNSF) project Nr. 200021_153249, the Hasler Foundation Switzerland, and by the Spanish CICYT project DPI2013–42458–P and TIN2013–47245–C2–2–R.

References

1. Sanfeliu, A., Fu, K.-S.: A distance measure between attributed relational graphs for pattern recognition. IEEE Transactions on Systems, Man and Cybernetics SMC-13(3), 353–362 (1983)
2. Bunke, H., Allermann, G.: Inexact graph matching for structural pattern recognition. Pattern Recognition Letters 1(4), 245–253 (1983)
3. Neuhaus, M., Bunke, H.: A graph matching based approach to fingerprint classification using directional variance. In: Kanade, T., Jain, A., Ratha, N.K. (eds.) AVBPA 2005. LNCS, vol. 3546, pp. 191–200. Springer, Heidelberg (2005)
4. Robles-Kelly, A., Hancock, E.R.: Graph edit distance from spectral seriation. IEEE Trans. Pattern Anal. Mach. Intell. 27(3), 365–378 (2005)
5. Gao, X., Xiao, B., Tao, D., Li, X.: A survey of graph edit distance. Pattern Anal. Appl. 13(1), 113–129 (2010)
6. Boeres, M.C., Ribeiro, C.C., Bloch, I.: A randomized heuristic for scene recognition by graph matching. In: Ribeiro, C.C., Martins, S.L. (eds.) WEA 2004. LNCS, vol. 3059, pp. 100–113. Springer, Heidelberg (2004)
7. Sorlin, S., Solnon, C.: Reactive tabu search for measuring graph similarity. In: Brun, L., Vento, M. (eds.) GbRPR 2005. LNCS, vol. 3434, pp. 172–182. Springer, Heidelberg (2005)
8. Justice, D., Hero, A.O.: A binary linear programming formulation of the graph edit distance. IEEE Trans. PAMI 28(8), 1200–1214 (2006)
9. Neuhaus, M., Riesen, K., Bunke, H.: Fast suboptimal algorithms for the computation of graph edit distance. In: Yeung, D.-Y., Kwok, J.T., Fred, A., Roli, F., de Ridder, D. (eds.) SSPR&SPR 2006. LNCS, vol. 4109, pp. 163–172. Springer, Heidelberg (2006)
10. Riesen, K., Bunke, H.: Approximate graph edit distance computation by means of bipartite graph matching. Image Vision Comput. 27(7), 950–959 (2009)
11. Serratosa, F.: Fast computation of bipartite graph matching. Pattern Recognition Letters 45, 244–250 (2014)
12. Burkard, R.E., Dell'Amico, M., Martello, S.: Assignment Problems. SIAM (2009)
13. Riesen, K., Fischer, A., Bunke, H.: Combining bipartite graph matching and beam search for graph edit distance approximation. In: El Gayar, N., Schwenker, F., Suen, C. (eds.) ANNPR 2014. LNCS, vol. 8774, pp. 117–128. Springer, Heidelberg (2014)
14. Riesen, K., Bunke, H.: IAM graph database repository for graph based pattern recognition and machine learning. In: da Vitoria Lobo, et al. (eds.) [15], pp. 287–297
15. da Vitoria Lobo, N., Kasparis, T., Roli, F., Kwok, J.T., Georgiopoulos, M., Anagnostopoulos, G.C., Loog, M. (eds.): SSPR&SPR 2008. LNCS, vol. 5342. Springer, Heidelberg (2008)