Improving Approximate Graph Edit Distance by Means of a Greedy Swap Strategy

Kaspar Riesen[1] and Horst Bunke[2]

[1] Institute for Information Systems, University of Applied Sciences and Arts Northwestern Switzerland, Riggenbachstrasse 16, 4600 Olten, Switzerland
kaspar.riesen@fhnw.ch
[2] Institute of Computer Science and Applied Mathematics, University of Bern, Neubrückstrasse 10, 3012 Bern, Switzerland
bunke@iam.ch

Abstract. The authors of the present paper previously introduced a fast approximation framework for the graph edit distance problem. The basic idea of this approximation is to build a square cost matrix $\mathbf{C} = (c_{ij})$, where each entry c_{ij} reflects the cost of a node substitution, deletion or insertion plus the matching cost arising from the local edge structure. Based on $\mathbf{C}$ an optimal assignment of the nodes and their local structure is established in polynomial time. Yet, this procedure considers the graph structure only in a local way, and thus, an overestimation of the true graph edit distance has to be accepted. The present paper aims at reducing this overestimation by means of an additional greedy search strategy that builds upon the initial assignment. In an experimental evaluation on three real world data sets we empirically verify a substantial gain of distance accuracy while run time is nearly not affected.

1 Introduction

A large number of methods for graph matching have been proposed in recent years (see [1,2] for exhaustive surveys). Graph edit distance [3,4] is one of the most flexible and versatile error-tolerant graph matching models. That is, graph edit distance is able to cope with directed and undirected, as well as with labeled and unlabeled graphs. If there are labels on nodes, edges, or both, no constraints on the respective label alphabets have to be considered. Moreover, through the use of an appropriate cost function graph edit distance can be adopted and tailored to various problem specifications (e.g. diatom identification [5][1] or clustering of color images [6], to name just two applications).

Given two graphs, the source graph g_1 and the target graph g_2, the basic idea of graph edit distance is to transform g_1 into g_2 using some distortion operations. A standard set of distortion operations is given by *insertions*, *deletions*, and *substitutions* of both nodes and edges. We denote the substitution of two nodes

[1] Diatoms are unicellular algae found in humid places where light provides the basis for photosynthesis.

A. Elmoataz et al. (Eds.): ICISP 2014, LNCS 8509, pp. 314–321, 2014.

u and v by $(u \to v)$, the deletion of node u by $(u \to \varepsilon)$, and the insertion of node v by $(\varepsilon \to v)$[2]. 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, one introduces a cost for each edit operation, measuring the strength of the corresponding operation (commonly defined with respect to the node/edge labels). The *edit distance* of two graphs is then defined by the minimum cost edit path between two graphs. The computation of exact graph edit distance is usually carried out by means of a tree search algorithm 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 (e.g. using an A* algorithm [7]). The problem of exact graph edit distance computation belongs to the family of *quadratic assignment problems* (QAPs), which in turn belong to the class of $\mathcal{NP}$*-complete* problems. Consequently, exact edit distance can be computed for graphs of rather small size only.

In recent years, a number of methods addressing the high computational complexity of graph edit distance computation have been proposed (e.g. [8,9,10,11]). The authors of the present paper also introduced an algorithmic framework which allows the approximate computation of graph edit distance in a substantially faster way than traditional methods [12]. The basic idea of this approach is to reduce the QAP of graph edit distance to a linear sum assignment problem (LSAP). To this end, the graphs to be matched are subdivided into singular nodes plus local structures in a first step. Next, these independent sets of nodes including local structures are optimally assigned to each other.

LSAPs can be formulated on sets of independent entities only. Hence, reformulating the graph edit distance problem to an instance of an LSAP implies that only local, rather than global, edge structure of the graphs can be considered. This limitation allows the computation to be accomplished in polynomial time but leads to an overestimation of the actual graph edit distance. The main objective of the present paper is to reduce this overestimation in our approximation framework. To this end, the distance approximation found by the procedure of [12] is systematically improved using a greedy search strategy.

The present work is closely related to two recent approaches [13,14]. In these two papers the node assignment from the original approximation is also regarded as starting point for an additional search procedure. The basic idea of these two approaches is to systematically manipulate the underlying matching costs and eventually recompute optimal assignments based on these cost variations. Rather than altering the approximation via cost manipulation, in the present approach we vary the original assignment by means of pairwise swaps of node assignments. Hence, in contrast with [13,14] the time consuming recomputation of optimal node assignments based on slightly varied cost models can be omitted.

The remainder of this paper is organized as follows. Next, in Sect. 2 the original framework for graph edit distance approximation [12] is summarized. In Sect. 3 the extension of this specific framework using a greedy search procedure

[2] For edges we use a similar notation.

is introduced. An experimental evaluation on diverse data sets is carried out in Sect. 4, and in Sect. 5 we draw conclusions and outline directions for future work.

2 Bipartite Graph Edit Distance Approximation

In the framework presented in [12], for matching two graphs g_1 and g_2 with nodes $V_1 = \{u_1, \ldots, u_n\}$ and $V_2 = \{v_1, \ldots, v_m\}$, respectively, a cost matrix $\mathbf{C}$ is first established as follows:

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

Entry c_{ij} thereby denotes the cost of a node substitution $u_i \rightarrow v_j$, $c_{i\varepsilon}$ denotes the cost of a node deletion $u_i \rightarrow \varepsilon$, and $c_{\varepsilon j}$ denotes the cost of a node insertion $\varepsilon \rightarrow v_j$.

Obviously, 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. Note that each node can be deleted or inserted at most once. Therefore any non-diagonal element of the right-upper and left-lower part is set to ∞. The bottom right corner of the cost matrix is set to zero since substitutions of the form $(\varepsilon \rightarrow \varepsilon)$ should not cause any costs.

Note that the described extension of cost matrix $\mathbf{C}$ to to dimension $(n+m) \times (n+m)$ is necessary since assignment algorithms for LSAPs expect every entry of the first set to be assigned with exactly one entry of the second set (and vice versa), and we want the optimal matching to be able to possibly include several node deletions and/or insertions.

In order to integrate knowledge about the graph structure, to each entry c_{ij}, i.e. to each cost of a node edit operation, the minimum sum of edge edit operation costs, implied by the corresponding node operation, is added (i.e. the matching cost arising from the local edge structure is encoded in c_{ij}).

On the basis of the square cost matrix $\mathbf{C}$ a bipartite assignment algorithm is executed (referred to as *first step* from now on). In [12], for instance, Munkres' algorithm [15] is employed for this task. The result returned by this optimization procedures corresponds to the minimum cost mapping m of the nodes and their local edge structure of g_1 to the nodes and their local edge structure of g_2. Formally, assignment algorithms find a permutation $p = p_1, \ldots, p_{n+m}$ of the integers $1, 2, \ldots, (n+m)$ that minimizes the overall mapping cost $\sum_{i=1}^{(n+m)} c_{ip_i}$.

This permutation corresponds to a mapping

$$m = \{u_1 \to v_{p_1}, u_2 \to v_{p_2}, \ldots, u_{m+n} \to v_{p_{m+n}}\}$$

of nodes[3].

Mapping m can be seen as a partial edit path where each edit operation reflects an operation on nodes from V_1 and/or V_2 (deletions, insertions or substitutions). In a second step the edit path between g_1 and g_2 is completed according to mapping m. Note that edit operations on edges are implied by edit operations on their adjacent nodes, i.e. whether an edge is substituted, deleted, or inserted, depends on the edit operations performed on all of its adjacent nodes. Hence, given the set of node operations in m the global edge structures from g_1 and g_2 can be edited accordingly. The cost of the complete edit path is finally returned as an approximate graph edit distance. We denote the approximated edit distance between graphs g_1 and g_2 according to mapping m with $d_{\langle m \rangle}(g_1, g_2)$. For the remainder of this paper we denote this graph edit distance approximation algorithm with *BP* (*Bipartite*).

Note that during the optimization process of step 1 no information about neighboring node mappings is available. Hence, in comparison with optimal search methods for graph edit distance, this algorithmic framework might cause additional edge operations in the second step, which would not be necessary in a globally optimal graph matching. Hence, the distances found by this specific framework are – in the optimal case – equal to, or – in a suboptimal case – larger than the exact graph edit distance.

3 Improving Graph Edit Distance Approximations Using a Greedy Swap Algorithm

In several experimental evaluations we observed that the suboptimality of BP is very often due to a few incorrectly assigned nodes in m. That is, only few node assignments from the first step are responsible for the additional edge operations in the second step (and the resulting overestimation of the true edit distance). Our novel procedure ties in at this observation. Rather than returning the approximate edit distance directly, a greedy search procedure based on mapping m is started. The complete algorithmic procedure is given in Alg. 1. Note that the first three lines of Alg. 1 correspond to the original framework *BP*, resulting in a (first) approximation value d_{best}, while line 4 to 26 describe the proposed extension, denoted by *Greedy-Swap* from now on.

For each pair of node assignments $(u_i \to v_{p_i}, u_j \to v_{p_j})$ in m we compute its total assignment cost $cost_{orig} = c_{ip_i} + c_{jp_j}$. Furthermore, we compute $c_{ip_j} + c_{jp_i}$ (cost of swapped mappings $u_i \to v_{p_j}$ and $u_j \to v_{p_i}$) and buffer this sum in $cost_{swap}$ (line 7–10). In order to decide whether or not the swap is further investigated, we verify if the absolute value of difference between $cost_{orig}$ and

[3] Note that this mapping includes node assignments of the form $u_i \to v_j$, $u_i \to \varepsilon$, $\varepsilon \to v_j$, and $\varepsilon \to \varepsilon$.

Algorithm 1. Greedy-Swap(g_1, g_2) (Meta Parameter: θ)

```
1.  Build cost matrix C according to the input graphs g1 and g2
2.  Compute optimal node assignment m = {u1 → v_p1, u2 → v_p2, ..., u_{m+n} → v_{p_{m+n}}} on C
3.  d_best = d_⟨m⟩(g1, g2)
4.  swapped = true
5.  while swapped do
6.     swapped = false
7.     for i = 1, ..., (m + n − 1) do
8.        for j = i + 1, ..., (m + n) do
9.           cost_orig = c_{ip_i} + c_{jp_j}
10.          cost_swap = c_{ip_j} + c_{jp_i}
11.          if |cost_orig − cost_swap| ≤ θ · cost_orig then
12.             m' = m − {u_i → v_{p_i}, u_j → v_{p_j}} ∪ {u_i → v_{p_j}, u_j → v_{p_i}}
13.             Derive approximate edit distance d_⟨m'⟩(g1, g2)
14.             if d_⟨m'⟩(g1, g2) < d_best then
15.                d_best = d_⟨m'⟩(g1, g2)
16.                best-swap = {i, p_j, j, p_i}
17.                swapped = true
18.             end if
19.          end if
20.       end for
21.    end for
22.    if swapped then
23.       update m according to best-swap
24.    end if
25. end while
26. return d_best
```

$cost_{swap}$ lies below a certain threshold. In our procedure we use a threshold that depends on the cost of the original assignment, viz. $\theta \cdot cost_{orig}$ where θ is a user defined parameter (line 11)[4]. The intuition behind this procedure is that two node assignments with similar cost values might have been mixed up in the first step of our procedure. Thus, it is possibly beneficial to change the respective node assignment for further investigations.

Next, we integrate the assignment swap in our original matching m. That is, we derive a novel matching $m' = m - \{u_i \to v_{p_i}, u_j \to v_{p_j}\} \cup \{u_i \to v_{p_j}, u_j \to v_{p_i}\}$ and compute the corresponding edit distance $d_{\langle m' \rangle}(g_1, g_2)$ (line 12–13). If this distance value constitutes a better approximation than the currently best approximation value d_{best}, d_{best} is replaced by $d_{\langle m' \rangle}(g_1, g_2)$ and the swap actually carried out is buffered in a list named *best-swap* (line 14–17).

After testing all pairs of node assignments (i.e. the two **for**-loops from line 7 to 21 have been completely executed), the best individual swap *best-swap* (if any) is actually carried out on matching m (line 23).

This procedure is repeated as long as in each complete iteration through all possible swaps a beneficial swap constellation can be found. This is handled by means of **while**-loop (line 5 to 25) and the boolean variable *swapped* which turns *true* if the swap under consideration leads to an overall better approximation value than the currently best distance approximation (line 17).

[4] For instance, defining $\theta = 0.1$ implies that the cost of a swap can differ at most 10% from the original cost to be further considered.

Note that our procedure carries out at most one single swap in every iteration through the **while**-loop. This restriction prevents the computation of inconsistent node matchings (which assign, for instance, two different nodes u_i and u_j node from g_1 to the same node v_k from g_2). With other words, our procedure guarantees that the modified matching m remains globally consistent at any given time. That is, every node of g_1 is assigned to a single node of g_2 (or deleted) and every node of g_2 is assigned to a single node of g_1 (or inserted).

4 Experimental Evaluation

For experimental evaluations, three data sets from the IAM graph database repository for graph based pattern recognition and machine learning are used. The first graph data set involves graphs that represent molecular compounds (AIDS), the second graph data set consists of graphs representing fingerprint images (FP), and the third data set consists of graphs representing symbols from architectural and electronic drawings (GREC). For details about the underlying data and/or the graph extraction processes on all data sets we refer to [16].

In Table 1 the achieved results are shown. On each data set and for each graph edit distance algorithm two characteristic numbers are computed, viz. the mean relative overestimation of the exact graph edit distance ($\varnothing o$) and the mean run time to carry out one graph matching ($\varnothing t$). The algorithms employed are A* and BP (reference systems), five differently parametrized versions of our novel procedure Greedy-Swap ($\theta \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$) and an unparametrized version of our procedure (not yet discussed – details follow below).

Table 1. The mean relative overestimation of the exact graph edit distance ($\varnothing o$) and the mean run time for one matching in ms ($\varnothing t$) using a specific graph edit distance algorithm

Algorithm	**Data Set**					
	AIDS		**FP**		**GREC**	
	$\varnothing\ o$	$\varnothing\ t$	$\varnothing\ o$	$\varnothing\ t$	$\varnothing\ o$	$\varnothing\ t$
A* (Exact)	-	5629.53	-	5000.85	-	3103.76
BP	12.68	0.44	6.38	0.56	2.98	0.43
Greedy-Swap(0.1)	4.72	0.88	1.91	0.84	1.22	0.99
Greedy-Swap(0.3)	3.97		1.24		1.17	
Greedy-Swap(0.5)	3.97		0.94		1.17	
Greedy-Swap(0.7)	3.95		0.82		1.17	
Greedy-Swap(0.9)	3.95		0.80		1.17	
Greedy-Swap (unparametrized)	2.04	1.84	0.35	2.06	0.41	2.45

First we focus on the degree of overestimation (and exclude the unparametrized version for the present). The original framework (BP) overestimates the graph distance by 12.68% on average on the AIDS data. On the Fingerprint and GREC data the overestimations of the true distances amount to 6.38% and 2.98%, respectively. These values can be reduced with our extended framework on all data sets. For instance on the AIDS data, the mean relative overestimation can be reduced to 3.95% in the best case. That is, the mean relative overestimation of our novel framework amounts to less than a quarter of the original overestimation. On the Fingerprint data the overestimation can be heavily reduced from 6.38% to 0.80% and on the GREC data set the mean relative overestimation is reduced from 2.98% to 1.17% in the best case.

The run time shown for differently parametrized Greedy-Swap procedures corresponds to the mean run time in ms measured over all five parameter values of θ (parameter θ has only marginal influence on the run time behavior). Comparing the mean run time of our novel procedure with the original framework, we observe that our extension approximately doubles the average runtime for one matching on all data sets. Yet, the average run time still lies below 1ms per matching on every data set, and moreover, compared to the huge run time for exact computation, the slight increase of the run time becomes negligible.

In our experimental evaluation. we observe that the larger parameter θ is set, the better the resulting approximation. This leads to the conclusion that one could use our novel procedure either with a large fixed parameter θ or even without parametrization (i.e. omitting line 9–11 in Alg. 1). In this unparametrized version all pairs of assignments are swapped and eventually tested. The results of this version are shown in the last line of Table 1. We observe that this version of our procedure achieves the best results on all data sets at the expense of another run time increase (which is clearly due to the increased amount of swap tests that have to be carried out without parameter θ).

5 Conclusion and Future Work

In the present paper we propose an extension of our previous graph edit distance approximation algorithm (BP). The major idea of our work is to use the suboptimal graph edit distance and the underlying node assignment in a greedy search procedure to improve the approximation accuracy. That is, the initial node assignment is systematically altered by means of pairwise node assignment swaps. A sequence of swaps is created using a greedy search strategy. With several experimental results we show that this extension leads to a substantial reduction of the overestimations typical for BP while the run time remains below 1ms on average per matching. In future work we plan to employ other search strategies based on the same idea of swapping pairwise node assignments.

Acknowledgements. This work has been supported by the *Hasler Foundation* Switzerland.

References

1. Conte, D., Foggia, P., Sansone, C., Vento, M.: Thirty years of graph matching in pattern recognition. Int. Journal of Pattern Recognition and Artificial Intelligence 18(3), 265–298 (2004)
2. Foggia, P., Percannella, G., Vento, M.: Graph matching and learning in pattern recognition in the last 10 years. Int. Journal of Pattern Recognition and Art. Intelligence Online Ready (2014)
3. Sanfeliu, A., Fu, K.: A distance measure between attributed relational graphs for pattern recognition. IEEE Transactions on Systems, Man, and Cybernetics (Part B) 13(3), 353–363 (1983)
4. Bunke, H., Allermann, G.: Inexact graph matching for structural pattern recognition. Pattern Recognition Letters 1, 245–253 (1983)
5. Ambauen, R., Fischer, S., Bunke, H.: Graph edit distance with node splitting and merging and its application to diatom identification. In: Hancock, E., Vento, M. (eds.) GbRPR 2003. 95–106, vol. 2726, pp. 95–106. Springer, Heidelberg (2003)
6. Robles-Kelly, A., Hancock, E.: Graph edit distance from spectral seriation. IEEE Transactions on Pattern Analysis and Machine Intelligence 27(3), 365–378 (2005)
7. Hart, P., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions of Systems, Science, and Cybernetics 4(2), 100–107 (1968)
8. 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)
9. 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)
10. Justice, D., Hero, A.: A binary linear programming formulation of the graph edit distance. IEEE Trans. on Pattern Analysis ans Machine Intelligence 28(8), 1200–1214 (2006)
11. 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 2006 and SPR 2006. LNCS, vol. 4109, pp. 163–172. Springer, Heidelberg (2006)
12. Riesen, K., Bunke, H.: Approximate graph edit distance computation by means of bipartite graph matching. Image and Vision Computing 27(4), 950–959 (2009)
13. Riesen, K., Dornberger, R., Bunke, H.: Iterative bipartite graph edit distance approximation. Accepted for Publication in Proc. 11th IAPR Int. Workshop on Document Analysis Systems
14. Riesen, K., Fischer, A., Bunke, H.: Improving approximate graph edit distance using genetic algorithms. Accepted for Publication in Proc. IAPR Joint Int. Workshop on S+SSPR
15. Munkres, J.: Algorithms for the assignment and transportation problems. Journal of the Society for Industrial and Applied Mathematics 5(1), 32–38 (1957)
16. Riesen, K., Bunke, H.: IAM graph database repository for graph based pattern recognition and machine learning. In: da Vitoria Lobo, N., Kasparis, T., Roli, F., Kwok, J.T., Georgiopoulos, M., Anagnostopoulos, G.C., Loog, M. (eds.) S+SSPR 2008. LNCS, vol. 5342, pp. 287–297. Springer, Heidelberg (2008)