Approximation of Graph Edit Distance in Quadratic Time

Kaspar Riesen[1,3(✉)], Miquel Ferrer[1], Andreas Fischer[2], and Horst Bunke[3]

[1] Institute for Information Systems,
University of Applied Sciences and Arts Northwestern Switzerland,
Riggenbachstrasse 16, 4600, Olten, Switzerland
{kaspar.riesen,miquel.ferrer}@fhnw.ch
[2] DIUF Department, University of Fribourg, Switzerland and iCoSys Institute,
University of Applied Sciences and Arts Western Switzerland, Fribourg, Switzerland
andreas.fischer@unifr.ch
[3] Institute of Computer Science and Applied Mathematics, University of Bern,
Neubrückstrasse 10, 3012, Bern, Switzerland
{riesen,bunke}@iam.unibe.ch

Abstract. The basic idea of a recent graph matching framework is to reduce the problem of graph edit distance (GED) to an instance of a linear sum assignment problem (LSAP). The optimal solution for this simplified GED problem can be computed in cubic time and is eventually used to derive a suboptimal solution for the original GED problem. Yet, for large scale graphs and/or large scale graph sets the cubic time complexity remains a severe handicap of this procedure. Therefore, we propose to use suboptimal algorithms – with quadratic rather than cubic time complexity – for solving the underlying LSAP. In particular, we introduce several greedy assignment algorithms for approximating GED. In an experimental evaluation we show that there is great potential for further speeding up the GED computation. Moreover, we empirically confirm that the distances obtained by this procedure remain sufficiently accurate for graph based pattern classification.

1 Introduction

Graph edit distance (GED) is a widely accepted concept for general graph dissimilarity computation [1–5]. Yet, a well known drawback of GED is its computational complexity which is exponential in the number of nodes of the involved graphs. This means that for large graphs the exact computation of GED is intractable. In recent years, a number of methods addressing the high complexity of GED computation have been proposed [6–8]. In [9] the authors of the present paper introduced an algorithmic framework for the approximation of GED. The basic idea of this approach is to reduce the problem of GED computation to a *linear sum assignment problem* (LSAP). For LSAPs quite an arsenal of efficient (i.e. cubic time) algorithms exist [10].

C.-L. Liu et al. (Eds.): GbRPR 2015, LNCS 9069, pp. 3–12, 2015.
DOI: 10.1007/978-3-319-18224-7_1

The algorithmic procedure described in [9] consists of three major steps. In a first step the graphs to be matched are subdivided into individual nodes including local structural information. Next, in step 2, an LSAP solving algorithm is employed in order to find an optimal assignment of the nodes (plus local structures) of both graphs. Finally, in step 3, an approximate GED, which is globally consistent with the underlying edge structures of both graphs, is derived from the assignment of step 2.

In a recent paper [11] the optimal LSAP algorithm is replaced with a suboptimal greedy algorithm. From the theoretical point of view this approach is very appealing as it makes use of an approximation algorithm for an LSAP which in turn approximates the corresponding GED problem. The present paper introduces three advancements of this novel greedy graph edit distance approximation. All of these refinements are still greedy algorithms with quadratic time complexity. The proposed variants aim at improving the overall quality of the solution for the LSAP (and thus the GED approximation) by means of several heuristics.

Next, in Sect. 2 the original framework for GED approximation [9] is summarized. In Sect. 3 the greedy assignment algorithms are introduced. An experimental evaluation on diverse data sets is carried out in Sect. 4, and in Sect. 5 we draw conclusions and point out possible directions for future work.

2 Bipartite Graph Edit Distance Approximation

By reformulating GED to an instance of an LSAP (as introduced in [9] and denoted with *BP-GED*[1]), three major steps have to be carried out.

First Step: A *cost matrix* $\mathbf{C}$ 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, is established as follows.

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

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 $(\varepsilon \rightarrow v_j)$. That is, the left upper corner of $\mathbf{C} = (c_{ij})$ represents the costs of all possible node substitutions, while the diagonals of the right upper and left lower corner represent the costs of all possible node deletions and insertions, respectively (every node can be deleted or inserted at most once and thus any

[1] *Bipartite Graph Edit Distance* (LSAPs can be formulated by means of *bipartite graphs*).

non-diagonal element is set to ∞ in these parts). Substitutions of the form $(\varepsilon \to \varepsilon)$ should not cause any cost and therefore, any element in the right lower part is set to zero.

Second Step: Next, an LSAP is stated on cost matrix $\mathbf{C} = (c_{ij})$ and eventually solved. The LSAP optimization consists in finding a permutation $(\varphi_1, \ldots, \varphi_{n+m})$ of the integers $(1, 2, \ldots, (n + m))$ that minimizes the overall assignment cost $\sum_{i=1}^{(n+m)} c_{i\varphi_i}$. This permutation corresponds to the assignment

$$\psi = ((u_1 \to v_{\varphi_1}), (u_2 \to v_{\varphi_2}), \ldots, (u_{m+n} \to v_{\varphi_{m+n}}))$$

of the nodes of g_1 to the nodes of g_2. Note that assignment ψ includes node assignments of the form $(u_i \to v_j)$, $(u_i \to \varepsilon)$, $(\varepsilon \to v_j)$, and $(\varepsilon \to \varepsilon)$ (the latter can be dismissed, of course). Hence, the definition of $\mathbf{C} = (c_{ij})$ in Eq. 1 explicitly allows insertions and/or deletions to occur in the optimal assignment.

Assignment ψ does not take any structural constraints of the graphs into account as long as the individual entries in $\mathbf{C} = (c_{ij})$ consider the nodes of both graphs only. In order to integrate knowledge about the graph structure, to each entry c_{ij}, i.e. to each cost of a node edit operation $(u_i \to v_j)$, the minimum sum of edge edit operation costs, implied by the corresponding node operation, is added. This enables the LSAP to consider information about the local, yet not global, edge structure of a graph for optimizing the node assignment.

Third Step: The LSAP optimization finds an assignment ψ in which every node of g_1 and g_2 is either assigned to a unique node of the other graph, deleted or inserted. That is, ψ refers to a consistent node assignment and thus, the edge operations, which are implied by edit operations on their adjacent nodes, can be completely and consistently inferred from ψ. Hence, we get an edit path between the graphs under consideration. Yet, the edit path corresponding to ψ considers the edge structure of g_1 and g_2 in a global and consistent way while the optimal node assignment ψ is able to consider the structural information in an isolated way only (single nodes and their adjacent edges). Hence, the edit path found by this specific framework is not necessarily optimal and thus the distances are – in the best case – equal to, or – in general – larger than the exact graph edit distance.

3 Greedy Assignment Algorithms

3.1 Basic Greedy Assignment

In the second step of BP-GED an assignment of the nodes (plus local structures) of both graphs has to be found. For this task a large number of algorithms exist (see [10] for an exhaustive survey). For optimally solving the LSAP in the existing framework, Munkres' algorithm [12] also referred to as Kuhn-Munkres, or Hungarian algorithm, is deployed. The time complexity of this particular algorithm (as well as the best performing other algorithms for LSAPs) is cubic in the size of the problem, i.e. $O((n + m)^3)$ in our case.

Algorithm 1. Greedy-Assignment($\mathbf{C} = (c_{ij})$)

1. $\psi = \{\}$
2. **for** $i = 1, \ldots, (m + n)$ **do**
3. $\quad \varphi_i = \arg\min_{\forall j} c_{ij}$
4. $\quad$ Remove column φ_i from $\mathbf{C}$
5. $\quad \psi = \psi \cup \{(u_i \rightarrow v_{\varphi_i})\}$
6. **end for**
7. **return** ψ

In [11] a suboptimal (rather than an optimal) algorithm is used for solving the LSAP on cost matrix $\mathbf{C}$. In particular, a greedy algorithm is employed that iterates through $\mathbf{C}$ from top to bottom through all rows and assigns every element to the minimum unused element of the current row. This idea is formalized in Alg. 1. For each row i in the cost matrix $\mathbf{C} = (c_{ij})$ the minimum cost assignment is determined and the corresponding node edit operation $(u_i \rightarrow v_{\varphi_i})$ is added to ψ. By removing column φ_i in $\mathbf{C}$ it is ensured that every column of the cost matrix is considered at most once (i.e. $\forall j$ only refers to available columns in $\mathbf{C}$). Clearly, the complexity of this suboptimal assignment algorithm is only $O((n+m)^2)$. For the remainder of this paper we denote the graph edit distance approximation where the basic node assignment is computed by means of this greedy procedure with *Greedy-GED*.

3.2 Tie Break Strategy

A crucial question in Alg. 1 is how possible ties between two (or more) cost entries in the same row are resolved. In [11] a certain row is always assigned to the first minimum column that occurs from left to right. As a refinement of this coarse heuristic we propose the following procedure (denoted by *Greedy-Tie-GED* from now on).

Assume that in the i-th row the following $t > 1$ cost entries $\{c_{ij_1}, \ldots, c_{ij_t}\}$ offer minimal cost among all available columns. For each of the corresponding columns $\{j_1, \ldots, j_t\}$ we search the minimum cost row that has not yet been assigned. Eventually, row i is assigned to column $\varphi_i \in \{j_1, \ldots, j_t\}$ that holds the highest minimum entry. Formally, column

$$\varphi_i = \arg\max_{j \in \{j_1, \ldots, j_t\}} \left(\min_{\forall k} c_{kj} \right)$$

is assigned to row i.

The intuition behind this strategy is as follows. The minimum cost

$$\min_{\forall k} c_{kj}$$

refers to the best available alternative for column j besides row i ($\forall k$ refers to all unprocessed rows of matrix $\mathbf{C}$). With other words, if we do not select $(u_i \rightarrow v_j)$ as assignment, the alternative assignment costs at least $\min_{\forall k} c_{kj}$. Hence, we should select the assignment among $\{u_i \rightarrow v_{j_1}, \ldots, u_i \rightarrow v_{j_t}\}$ where the best possible alternative would lead to the highest cost.

3.3 Refined Greedy Assignment

A next refinement of the basic greedy algorithm is given in Alg. 2. Similar to Alg. 1 for every row i we search for the minimum cost column φ_i (in case of ties the strategy described in the previous section is applied). In addition to Alg. 1 we also seek for the minimum cost row k for column φ_i.

Algorithm 2. Greedy-Refined-Assignment($\mathbf{C} = (c_{ij})$)

1. $\psi = \{\}$
2. **while** row i in $\mathbf{C}$ is available **do**
3. $\quad \varphi_i = \arg\min_{\forall j} c_{ij}$ and $k = \arg\min_{\forall i} c_{i\varphi_i}$
4. $\quad$ **if** $c_{i\varphi_i} \leq c_{k\varphi_i}$ **then**
5. $\quad\quad \psi = \psi \cup \{(u_i \rightarrow v_{\varphi_i})\}$
6. $\quad\quad$ Remove row i and column φ_i from $\mathbf{C}$
7. $\quad$ **else**
8. $\quad\quad \psi = \psi \cup \{(u_k \rightarrow v_{\varphi_i})\}$
9. $\quad\quad$ Remove row k and column φ_i from $\mathbf{C}$
10. $\quad$ **end if**
11. **end while**
12. **return** ψ

That is, for row i column φ_i is the best assignment. Yet, considering the assignment process from column φ_i's point of view, row k would be the best choice. We select the assignment which leads to lower cost. That is, if $c_{i\varphi_i} \leq c_{k\varphi_i}$ the assignment $(u_i \rightarrow v_{\varphi_i})$ is added to ψ. This includes the special situation where $i = k$, i.e. column φ_i corresponds to the optimal assignment for row i and vice versa. Otherwise, if $c_{i\varphi_i} > c_{k\varphi_i}$, assignment $(u_k \rightarrow v_{\varphi_i})$ is selected. Regardless of the assignment actually added to ψ, the corresponding row and column are removed from $\mathbf{C}$.

Note that in contrast with Alg. 1 where the i-th assignment added to ψ always considers row i, this algorithm processes the rows of $\mathbf{C}$ not necessarily from top to bottom. However, the complexity of this assignment algorithm remains $O((n+m)^2)$. We denote the graph edit distance approximation where the LSAP is solved by means of this particular greedy assignment algorithm with *Greedy-Refined-GED*.

3.4 Greedy Assignment Regarding Loss

A further refinement of the greedy assignment is given in Alg. 3. Similar to Alg. 2 we first find both the minimum cost column φ_i for row i and the minimum cost row k for column φ_i (again the tie break strategy of Sect. 3.2 is applied). Now three cases have to be considered.

If i equals k, column φ_i is the best available assignment for row i and vice versa (and thus assignment $(u_i \rightarrow v_{\varphi_i})$ can safely be added to ψ). If $i \neq k$ two scenarios have to be distinguished. For both scenarios the second best assignment φ'_i for row i, as well as the best available assignment φ_k for row k (distinct from φ_i) are determined. In the first scenario row i is assigned to its best column φ_i

Algorithm 3. Greedy-Loss-Assignment($\mathbf{C} = (c_{ij})$)

```
1.  ψ = {}
2.  while row i in C is available do
3.      φ_i = arg min_{∀j} c_ij   and   k = arg min_{∀i} c_iφi
4.      if i == k then
5.          ψ = ψ ∪ {(u_i → v_φi)}
6.          Remove row i and column φ_i from C
7.      else
8.          φ'_i = arg min_{∀j≠φ_i} c_ij   and   φ_k = arg min_{∀j≠φ_i} c_kj
9.          if (c_iφi + c_kφk) < (c_iφ'i + c_kφi) then
10.             ψ = ψ ∪ {(u_i → v_φi), (u_k → v_φk)}
11.             Remove rows i, k and columns φ_i, φ_k from C
12.         else
13.             ψ = ψ ∪ {(u_i → v_φ'i), (u_k → v_φi)}
14.             Remove rows i, k and columns φ_i, φ'_i from C
15.         end if
16.     end if
17. end while
18. return ψ
```

and thus row k has eventually to be assigned to the best alternative φ_k. The sum of cost of this scenario amounts to $(c_{i\varphi_i} + c_{k\varphi_k})$. In the other scenario, row k is assigned to column φ_i and thus row i has to be assigned to its best possible alternative which is column φ'_i. The sum of cost of this scenario amounts to $(c_{i\varphi'_i} + c_{k\varphi_i})$. We choose the scenario with lower sum of cost and remove the corresponding rows and columns from $\mathbf{C}$.

We denote the graph edit distance approximation using this greedy approach with *Greedy-Loss-GED* (the complexity of this assignment algorithm remains $O((n+m)^2)$).

3.5 Relations to Exact Graph Edit Distance

Note that In contrast with the optimal permutation $(\varphi_1, \ldots, \varphi_{n+m})$ returned by the original framework, the permutations $(\varphi'_1, \ldots, \varphi'_{n+m})$ of the proposed greedy algorithms are suboptimal. That is, the sum of assignments costs of all greedy approaches are greater than, or equal to, the minimal sum of assignment cost provided by optimal LSAP solving algorithms. Formally, we have

$$\sum_{i=1}^{(n+m)} c_{i\varphi'_i} \geq \sum_{i=1}^{(n+m)} c_{i\varphi_i}$$

However, for the approximate graph distance values derived by BP-GED and any greedy approach no globally valid order relation exists. That is, the approximate graph edit distance derived from a greedy assignment can be greater than, equal to, or smaller than a distance value returned by BP-GED. Note that approximations derived from both optimal or suboptimal local assignments constitute upper bounds of the true graph edit distance. Hence, the smaller the approximated distance value is, the nearer it is to the exact graph edit distance.

4 Experimental Evaluation

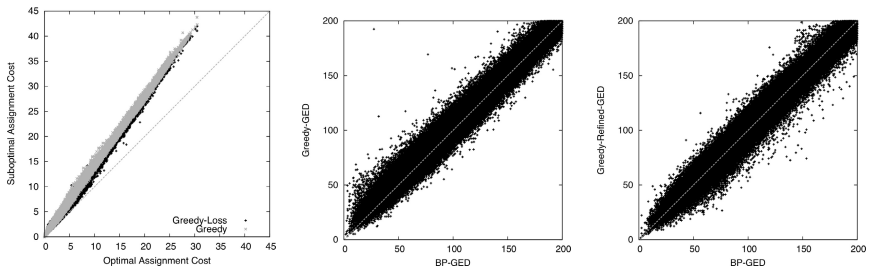
In Table 1 we show different characteristic numbers for BP-GED and all greedy variants for graph edit distance approximation on three different data sets from the IAM graph repository [13]. First we focus on the mean run time for one matching in ms ($\varnothing t$). On the relatively small graphs of the FP data set, the speed-up by Greedy-GED compared to BP-GED is rather small. Yet, on the other two data sets with larger graphs substantial speed-ups can be observed. That is, using Greedy-GED rather than BP-GED on the AIDS data set leads to a decrease of the mean matching time from 3.61ms to 1.21ms. On the MUTA data Greedy-GED is more than seven times faster than the original approximation. We also observe that the enhanced greedy algorithms, viz. Greedy-Tie, Greedy-Refined, and Greedy-Loss, do not need substantially more computation time compared to the plain approximation using Greedy-GED.

The characteristic number $\varnothing o$ measures the overestimation of one particular assignment compared to the optimal assignment. For comparison we take the difference between the optimal sum of costs and the sum of costs of the plain greedy assignment as 100%. Regarding the results in Table 1 we note that our novel tie resolution is not able to substantially reduce this overestimation. Yet, both Greedy-Refined and Greedy-Loss return overall assignment sums which are substantially nearer to the optimal sum than the plain greedy approach. For instance, compared with Greedy-GED the difference to the optimal sum of

Table 1. The mean run time for one matching in ms ($\varnothing t$), the relative overestimation of a greedy sum of assignment costs compared to the optimal sum of assignment costs ($\varnothing o$), the mean relative deviation of greedy GED algorithm variants compared with BP-GED in percentage ($\varnothing e$), and the accuracy of a 1NN classifier

Data		Algorithm				
		BP-GED	Greedy-GED	Greedy-Tie-GED	Greedy-Refined-GED	Greedy-Loss-GED
AIDS	$\varnothing t$ [ms]	3.61	1.21	1.23	1.25	1.24
	$\varnothing o$ [%]	–	+100.0	+99.9	+74.1	+75.0
	$\varnothing e$ [%]	–	$+7.1/-4.1$	$+6.5/-3.7$	$+5.8/-4.6$	$+7.0/-4.3$
	1NN [%]	99.07	98.93	99.20	99.13	98.87
FP	$\varnothing t$ [ms]	0.41	0.30	0.30	0.31	0.31
	$\varnothing o$ [%]	–	+100.0	+99.8	+88.0	+87.0
	$\varnothing e$ [%]	–	$+6.6/-15.7$	$+6.5/-15.9$	$+6.0/-17.5$	$+5.7/-18.1$
	1NN [%]	79.75	77.05	77.20	75.80	76.6
Muta	$\varnothing t$ [ms]	33.89	4.56	4.68	4.88	4.95
	$\varnothing o$ [%]	–	+100.0	+99.9	+2.8	+15.5
	$\varnothing e$ [%]	–	$+7.4/-3.8$	$+6.2/-4.6$	$+5.9/-5.1$	$+7.6/-4.9$
	1NN [%]	70.20	70.10	71.80	71.60	71.10

assignment costs is reduced by about 25% with Greedy-Refined and Greedy-Loss on the AIDS data set. Similar results are observable on the other data sets (note particularly the massive reduction on the Muta data set using Greedy-Refined). The reduction of overestimation of assignment costs can also be seen in the scatter plot in Fig. 1 (a) where the optimal assignment cost (x-axis) is plotted against the greedy assignment costs (y-axis) for Greedy (gray dots) and Greedy-Loss (black dots) on the AIDS data set.

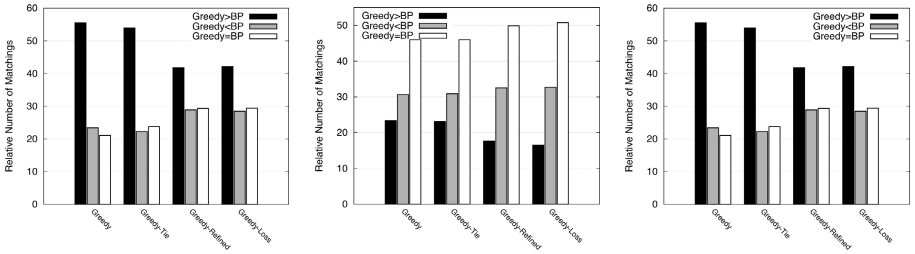


(a) (b) (c)

Fig. 1. (a) Optimal assignment cost (x-axis) vs. greedy assignment cost (y-axis) on the AIDS data sets using Greedy (gray dots) and Greedy-Loss (black dots), (b) distances of BP-GED (x-axis) vs. distances of Greedy-GED (y-axis), and (c) distances of BP-GED (x-axis) vs. distances of Greedy-Refined-GED (y-axis)

The characteristic number $\varnothing e$ in Table 1 measures the mean relative over- and underestimation of greedy graph edit distances compared with BP-GED. Note that both means are computed on the sets of distances where a greedy approach actually over- or underestimates the original approximation. We observe that our enhanced greedy assignment algorithms are able to reduce the mean overestimation compared with the plain greedy approach in eight out of nine cases. For instance, the mean overestimation of Greedy-GED amounts to +7.1% on the AIDS data, while the same parameter is reduced to +5.8% with Greedy-Refined-GED. Interestingly, the mean underestimation of Greedy-GED is further increased with our novel enhancements (also in eight out of nine cases). That is, compared with Greedy-GED the refined greedy variants are able to improve the overall distance quality in general. The reduction of overestimation and increase of underestimation by the refined algorithms can also be seen in Fig. 1 (b) and (c) where the distances of BP-GED (x-axis) are plotted against distances returned by Greedy-GED and Greedy-Refined-GED (y-axis), respectively (on the Muta data set).

Not only the mean over- and underestimation, but also the number of matchings, where greedy distances are equal to, smaller than, or greater than, the distances returned by BP-GED are crucially altered using our improved greedy assignments. Fig. 2 shows for every greedy algorithm the relative number of



(a) AIDS (b) FP (c) MUTA

Fig. 2. Relative number of matchings where the greedy algorithms return distances greater than BP-GED (black bars), smaller than than BP-GED (grey bars), or equal to BP-GED (white bars) on all data sets

matchings for these three cases. On the AIDS data set, for instance, the relative number of matchings where the greedy distances overestimate the distances returned by BP-GED are reduced from 55.5% (Greedy-GED) to 42.1% (Greedy-Loss-GED). Likewise, the number of matchings where the greedy approaches lead to equal, or even smaller distances than BP-GED is increased using the proposed refinement algorithms. Similar (or even better) results can be observed on the other data sets. That is, compared with Greedy-GED the novel algorithms decrease the number of matchings which overestimate BP-GED while the number of matchings with equal or even better approximations than BP-GED is increased.

Finally, Table 1 shows the recognition rate of a 1-nearest-neighbor classifier (1NN). The nearest neighbor paradigm is particularly interesting for the present evaluation because it directly uses the distances without any additional classifier training. In comparison with BP-GED we observe that Greedy-GED slightly deteriorates the recognition rates on all data sets. However, the proposed enhanced greedy assignment algorithms improve the recognition rate of a 1NN compared to Greedy-GED in 6 out of 9 cases. Moreover, at least one of the novel greedy algorithms outperforms the recognition rate of Greedy-GED on all data sets. For BP-GED the same observation holds for two of the three data sets.

5 Conclusions and Future Work

In the present paper we propose an extension of graph edit distance approximation algorithms developed previously. While in the original framework the nodes plus local edge structures are assigned to each other in an optimal way, the extension of the present paper uses various suboptimal greedy algorithms for this task. In particular we propose three enhanced versions of a simple greedy assignment algorithm. These novel enhancements allow the graph edit distance approximation in quadratic – rather than cubic – time. The speed up of the approximation is empirically verified on three different graph data sets. Moreover,

we show that the enhanced greedy algorithms are able to improve the distance quality of a plain greedy assignment. Finally, we observe that in most cases the novel greedy algorithms are able to keep up with the existing framework with respect to recognition accuracy using a 1NN classifier. In future work we plan to develop further suboptimal assignment algorithms and test their applicability in graph based pattern recognition scenarios. Moreover, we aim at testing our greedy approaches on additional graph data sets and larger graphs

Acknowledgement. This work has been supported by the *Hasler Foundation* Switzerland and the *Swiss National Science Foundation* projects 200021_153249 and PBBEP2_141453.

References

1. Bunke, H., Allermann, G.: Inexact graph matching for structural pattern recognition. Pattern Recognition Letters 1, 245–253 (1983)
2. 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)
3. Gao, X., Xiao, B., Tao, D., Li, X.: A survey of graph edit distance. Pattern Anal. Appl. 13(1), 113–129 (2010)
4. Robles-Kelly, A., Hancock, E.: Graph edit distance from spectral seriation. IEEE Transactions on Pattern Analysis and Machine Intelligence 27(3), 365–378 (2005)
5. Emms, D., Wilson, R., Hancock, E.: Graph edit distance without correspondence from continuous-time quantum walks. In: 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, pp. 5–14. Springer, Heidelberg (2008)
6. Boeres, M., Ribeiro, 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.: A binary linear programming formulation of the graph edit distance. IEEE Trans. on Pattern Analysis ans Machine Intelligence 28(8), 1200–1214 (2006)
9. Riesen, K., Bunke, H.: Approximate graph edit distance computation by means of bipartite graph matching. Image and Vision Computing 27(4), 950–959 (2009)
10. Burkard, R., Dell'Amico, M., Martello, S.: Assignment Problems. Society for Industrial and Applied Mathematics, Philadelphia (2009)
11. Riesen, K., Ferrer, M., Dornberger, R., Bunke, H.: Greedy graph edit distance (Submitted to MLDM)
12. Munkres, J.: Algorithms for the assignment and transportation problems. Journal of the Society for Industrial and Applied Mathematics 5(1), 32–38 (1957)
13. 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.) SSPR&SPR 2008. LNCS, vol. 5342, pp. 287–297. Springer, Heidelberg (2008)