Approximation of Graph Edit Distance by Means of a Utility Matrix

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

[1] Institute for Information Systems, University of Applied Sciences FHNW,
Riggenbachstrasse 16, 4600 Olten, Switzerland
kaspar.riesen@fhnw.ch

[2] Department of Informatics, University of Fribourg and HES-SO,
1700 Fribourg, Switzerland
andreas.fischer@unifr.ch

[3] Institute of Computer Science and Applied Mathematics, University of Bern,
Neubrückstrasse 10, 3012 Bern, Switzerland
bunke@iam.ch

Abstract. Graph edit distance is one of the most popular graph matching paradigms available. By means of a reformulation of graph edit distance to an instance of a linear sum assignment problem, the major drawback of this dissimilarity model, viz. the exponential time complexity, has been invalidated recently. Yet, the substantial decrease of the computation time is at the expense of an approximation error. The present paper introduces a novel transformation that processes the underlying cost model into a utility model. The benefit of this transformation is that it enables the integration of additional information in the assignment process. We empirically confirm the positive effects of this transformation on three standard graph data sets. That is, we show that the accuracy of a distance based classifier can be improved with the proposed transformation while the run time remains nearly unaffected.

1 Introduction

Graphs are recognized as versatile alternative to feature vectors and thus, they found widespread application in pattern recognition and related fields [1,2]. However, one drawback of graphs, when compared to feature vectors, is the significant increase of the complexity of many algorithms. Regard, for instance, the algorithmic comparison of two patterns (which is actually a basic requirement for pattern recognition). Due to the homogeneous nature of feature vectors, pairwise comparisons is straightforward and can be accomplished in linear time with respect to the length of the two vectors. Yet, the same task for graphs, commonly referred to as *graph matching*, is much more complex, as one has to identify common parts of the graphs by considering all of their subsets of nodes. Regarding that there are $O(2^n)$ subsets of nodes in a graph with n nodes, the inherent difficulty of graph matching becomes obvious.

In the last four decades a huge number of procedures for graph matching have been proposed in the literature [1,2]. They range from *spectral methods* [3,4], over

F. Schwenker et al. (Eds.): ANNPR 2016, LNAI 9896, pp. 185–194, 2016.
DOI: 10.1007/978-3-319-46182-3_16

graph kernels [5,6], to reformulations of the discrete graph matching problem to an instance of a *continuous optimization problem* (basically by relaxing some constraints) [7]. *Graph edit distance* [8,9], introduced about 30 years ago, is still one of the most flexible graph distance models available and topic of various recent research projects.

In order to compute the graph edit distance often A* based search techniques using some heuristics are employed (e.g. [10]). Yet, exact graph edit distance computation based on a tree search algorithm is exponential in the number of nodes of the involved graphs. Formally, for two graphs with m and n nodes we observe a time complexity of $O(m^n)$. This means that for large graphs the computation of the exact edit distance is intractable.

In [11] authors of the present paper introduced an algorithmic framework for the approximation of graph edit distance. The basic idea of this approach is to reduce the difficult problem of graph edit distance to a *linear sum assignment problem* (LSAP), for which an arsenal of efficient (i.e. cubic time) algorithms exist [12]. In two recent papers [13,14] the optimal algorithm for the LSAP has been replaced with a suboptimal greedy algorithm which runs in quadratic time. Due to the lower complexity of this suboptimal assignment process, a substantial speed up of the complete approximation procedure has been observed. However, it was also reported that the distance accuracy of this extension is slightly worse than with the original algorithm. Major contribution of the present paper is to improve the overall distance accuracy of this recent procedure by means of an elaborated transformation of the underlying cost model.

The remainder of this paper is organized as follows. Next, in Sect. 2, the computation of graph edit distance is thoroughly reviewed. In particular, it is shown how the graph edit distance problem can be reduced to a linear sum assignment problem. In Sect. 3, the transformation of the cost model into a utility model is outlined. Eventually, in Sect. 4, we empirically confirm the benefit of this transformation in a classification experiment on three graph data sets. Finally, in Sect. 5, we conclude the paper.

2 Graph Edit Distance (GED)

2.1 Exact Computation of GED

A graph g is a four-tuple $g = (V, E, \mu, \nu)$, where V is the finite set of nodes, $E \subseteq V \times V$ is the set of edges, $\mu : V \rightarrow L_V$ is the node labeling function, and $\nu : E \rightarrow L_E$ is the edge labeling function. The labels for both nodes and edges can be given by the set of integers $L = \{1, 2, 3, \ldots\}$, the vector space $L = \mathbb{R}^n$, a set of symbolic labels $L = \{\alpha, \beta, \gamma, \ldots\}$, or a combination of various label alphabets from different domains. Unlabeled graphs are obtained by assigning the same (empty) label $\varnothing$ to all nodes and edges, i.e. $L_V = L_E = \{\varnothing\}$.

Given two graphs, $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$, the basic idea of *graph edit distance* (*GED*) [8,9] is to transform g_1 into g_2 using edit operations, viz. *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 $(u \rightarrow \varepsilon)$, and the insertion of node v by $(\varepsilon \rightarrow v)$[1]. A set of edit operations $\lambda(g_1, g_2) = \{e_1, \ldots, e_k\}$ that transform g_1 completely into g_2 is called an edit path between g_1 and g_2.

Note that edit operations on edges are uniquely defined by the edit operations on their adjacent nodes. That is, whether an edge (u, v) is substituted with an existing edge from the other graph, deleted, or inserted actually depends on the operations performed on both adjacent nodes u and v. Thus, we define that an edit path $\lambda(g_1, g_2)$ explicitly contains the edit operations between the graphs' nodes V_1 and V_2, while the edge edit operations are implicitly given by these node edit operations.

A cost function that measures the strength of an edit operation is commonly introduced for graph edit distance. The edit distance between two graphs g_1 and g_2 is then defined by the sum of cost of the minimum cost edit path $\lambda_{\min}$ between g_1 and g_2. In fact, the problem of finding the minimum cost edit path $\lambda_{\min}$ between g_1 and g_2 can be reformulated as a *quadratic assignment problem* (QAP). Roughly speaking, QAPs deal with the problem of assigning n entities of a first set $S = \{s_1, \ldots, s_n\}$ to n entities of a second set $Q = \{q_1, \ldots, q_n\}$ under some (computationally demanding) side constraints. A common way to formally represent assignments between the entities of S and Q is given by means of permutations $(\varphi_1, \ldots, \varphi_n)$ of the integers $(1, 2, \ldots, n)$. A permutation $(\varphi_1, \ldots, \varphi_n)$ refers to the assignment where the first entity $s_1 \in S$ is mapped to entity $q_{\varphi_1} \in Q$, the second entity $s_2 \in S$ is assigned to entity $q_{\varphi_2} \in Q$, and so on.

By reformulating the graph edit distance problem to an instance of a QAP, two major issues have to be resolved. First, QAPs are generally stated on sets with equal cardinality. Yet, in case of graph edit distance the elements to be assigned to each other are given by the sets of nodes (and edges) with unequal cardinality in general. Second, solutions to QAPs refer to assignments of elements in which every element of the first set is assigned to exactly one element of the second set and vice versa (i.e. a solution to a QAP corresponds to a bijective assignment of the underlying entities). Yet, GED is a more general assignment problem as it explicitly allows both deletions and insertions to occur on the basic entities (rather than only substitutions).

These two issues can be simultaneously resolved by adding an appropriate number of empty "nodes" ε to both graphs g_1 and g_2. Formally, assume that $|V_1| = n$ and $|V_2| = m$, we extend V_1 and V_2 according to

$$V_1^+ = V_1 \cup \overbrace{\{\varepsilon_1, \ldots, \varepsilon_m\}}^{m \text{ empty nodes}} \quad \text{and} \quad V_2^+ = V_2 \cup \underbrace{\{\varepsilon_1, \ldots, \varepsilon_n\}}_{n \text{ empty nodes}}.$$

Since both graphs g_1 and g_2 have now an equal number of nodes, viz. $(n+m)$, their corresponding adjacency matrices $\mathbf{A}$ and $\mathbf{B}$ offer also equal dimension.

[1] A similar notation is used for edges.

These adjacency matrices of g_1 and g_2 are defined by

$$\mathbf{A} = \begin{array}{c|ccc|ccc} & 1 & \dots & n & 1 & \dots & m \\ \hline 1 & a_{11} & \dots & a_{1n} & \varepsilon & \dots & \varepsilon \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ n & a_{n1} & \dots & a_{nn} & \varepsilon & \dots & \varepsilon \\ \hline 1 & \varepsilon & \dots & \varepsilon & \varepsilon & \dots & \varepsilon \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ m & \varepsilon & \dots & \varepsilon & \varepsilon & \dots & \varepsilon \end{array} \quad \text{and} \quad \mathbf{B} = \begin{array}{c|ccc|ccc} & 1 & \dots & m & 1 & \dots & n \\ \hline 1 & b_{11} & \dots & b_{1m} & \varepsilon & \dots & \varepsilon \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ m & b_{m1} & \dots & b_{mm} & \varepsilon & \dots & \varepsilon \\ \hline 1 & \varepsilon & \dots & \varepsilon & \varepsilon & \dots & \varepsilon \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ n & \varepsilon & \dots & \varepsilon & \varepsilon & \dots & \varepsilon \end{array} \tag{1}$$

If there actually is an edge between node $u_i \in V_1$ and $v_j \in V_1$, entry a_{ij} refers to this edge $(u_i, v_j) \in E_1$, and otherwise to the empty "edge" ε. Note that there cannot be any edge from an existing node in V_1 to an empty node ε and thus the corresponding entries $a_{ij} \in \mathbf{A}$ with $i > n$ and/or $j > n$ are also empty. The same observations account for entries b_{ij} in $\mathbf{B}$.

Next, based on the extended node sets V_1^+ and V_2^+ of g_1 and g_2, respectively, a *cost matrix* $\mathbf{C}$ can be established as follows.

$$\mathbf{C} = \begin{array}{c|cccc|cccc} & v_1 & v_2 & \dots & v_m & \varepsilon_1 & \varepsilon_2 & \dots & \varepsilon_n \\ \hline u_1 & c_{11} & c_{12} & \cdots & c_{1m} & c_{1\varepsilon} & c_{1\varepsilon} & \cdots & c_{1\varepsilon} \\ u_2 & c_{21} & c_{22} & \cdots & c_{2m} & c_{2\varepsilon} & c_{2\varepsilon} & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \ddots & \vdots \\ u_n & c_{n1} & c_{n2} & \cdots & c_{nm} & c_{n\varepsilon} & \cdots & c_{n\varepsilon} & c_{n\varepsilon} \\ \hline \varepsilon_1 & c_{\varepsilon 1} & c_{\varepsilon 1} & \cdots & c_{\varepsilon 1} & 0 & 0 & \cdots & 0 \\ \varepsilon_2 & c_{\varepsilon 2} & c_{\varepsilon 2} & \ddots & \vdots & 0 & 0 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & 0 \\ \varepsilon_m & c_{\varepsilon m} & \cdots & c_{\varepsilon m} & c_{\varepsilon m} & 0 & \cdots & 0 & 0 \end{array} \tag{2}$$

Entry c_{ij} thereby denotes the cost $c(u_i \to v_j)$ of the node substitution $(u_i \to v_j)$, $c_{i\varepsilon}$ denotes the cost $c(u_i \to \varepsilon)$ of the node deletion $(u_i \to \varepsilon)$, and $c_{\varepsilon j}$ denotes the cost $c(\varepsilon \to v_j)$ of the node insertion $(\varepsilon \to v_j)$. Obviously, the left upper part of the cost matrix represents the costs of all possible node substitutions, the right upper part the costs of all possible node deletions, and the bottom left part the costs of all possible node insertions. The bottom right part of the cost matrix is set to zero since substitutions of the form $(\varepsilon \to \varepsilon)$ should not cause any cost.

Given the adjacency matrices $\mathbf{A}$ and $\mathbf{B}$ as well as the cost matrix $\mathbf{C}$ (Eqs. 1 and 2), the following optimization problem can now be stated.

$$(\varphi_1, \dots, \varphi_{(n+m)}) = \underset{(\varphi_1, \dots, \varphi_{(n+m)}) \in \mathcal{S}_{(n+m)}}{\arg\min} \left[\sum_{i=1}^{n+m} c_{i\varphi_i} + \sum_{i=1}^{n+m} \sum_{j=1}^{n+m} c(a_{ij} \to b_{\varphi_i \varphi_j}) \right],$$

where $\mathcal{S}_{(n+m)}$ refers to the set of all $(n+m)!$ possible permutations of the integers $1, 2, \dots, (n+m)$. Note that this optimal permutation $(\varphi_1, \dots, \varphi_{(n+m)})$ (as well as any other valid permutation) corresponds to a bijective assignment

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

of the extended node set V_1^+ of g_1 to the extended node set V_2^+ of g_2. That is, assignment λ includes node edit operations of the form $(u_i \rightarrow v_j)$, $(u_i \rightarrow \varepsilon)$, $(\varepsilon \rightarrow v_j)$, and $(\varepsilon \rightarrow \varepsilon)$ (the latter can be dismissed, of course). In other words, an arbitrary permutation $(\varphi_1, \ldots, \varphi_{(n+m)})$ perfectly corresponds to a valid edit path λ between two graphs.

The optimization problem stated in Eq. 2.1 exactly corresponds to a standard QAP. Note that the linear term $\sum_{i=1}^{n+m} c_{i\varphi_i}$ refers to the sum of cost of all node edit operations, which are defined by the permutation $(\varphi_1, \ldots, \varphi_{n+m})$. The quadratic term $\sum_{i=1}^{n+m} \sum_{j=1}^{n+m} c(a_{ij} \rightarrow b_{\varphi_i\varphi_j})$ refers to the implied edge edit cost defined by the node edit operations. That is, since node $u_i \in V_1^+$ is assigned to a node $v_{\varphi_i} \in V_2^+$ and node $u_j \in V_1^+$ is assigned to a node $v_{\varphi_j} \in V_2^+$, the edge $(u_i, u_j) \in E_1 \cup \{\varepsilon\}$ (stored in $a_{ij} \in \mathbf{A}$) has to be assigned to the edge $(v_{\varphi_i}, v_{\varphi_j}) \in E_2 \cup \{\varepsilon\}$ (stored in $b_{\varphi_i\varphi_j} \in \mathbf{B}$).

2.2 Approximate Computation of GED

In fact, QAPs are very hard to solve as they belong to the class of NP-hard problems. Authors of the present paper introduced an algorithmic framework which allows the approximation of graph edit distance in a substantially faster way than traditional methods [11]. The basic idea of this approach is to reduce the QAP of graph edit distance computation to an instance of a *Linear Sum Assignment Problem* (*LSAP*). LSAPs are similar to QAPs in the sense of also formulating an assignment problem of entities. Yet, in contrast with QAPs, LSAPs are able to optimize the permutation $(\varphi_1, \ldots, \varphi_{(n+m)})$ with respect to the linear term $\sum_{i=1}^{n+m} c_{i\varphi_i}$ only. That is, LSAPs consider a single cost matrix $\mathbf{C}$ without any side constraints. For solving LSAPs a large number of efficient (i.e. polynomial) algorithms exist (see [12] for an exhaustive survey on LSAP solvers).

Yet, by omitting the quadratic term $\sum_{i=1}^{n+m} \sum_{j=1}^{n+m} c(a_{ij} \rightarrow b_{\varphi_i\varphi_j})$ during the optimization process, we neglect the structural relationships between the nodes (i.e. the edges between the nodes). In order to integrate knowledge about the graph structure, to each entry $c_{ij} \in \mathbf{C}$, i.e. to each cost of a node edit operation $(u_i \rightarrow v_j)$, the minimum sum of edge edit operation costs, implied by the corresponding node operation, can be added. Formally, for every entry c_{ij} in the cost matrix $\mathbf{C}$ one might solve an LSAP on the ingoing and outgoing edges of node u_i and v_j and add the resulting cost to c_{ij}. That is, we define

$$c^*_{ij} = c_{ij} + \min_{(\varphi_1, \ldots, \varphi_{(n+m)}) \in \mathcal{S}_{(n+m)}} \sum_{k=1}^{n+m} c(a_{ik} \rightarrow b_{j\varphi_k}) + c(a_{ki} \rightarrow b_{\varphi_k j}),$$

where $\mathcal{S}_{(n+m)}$ refers to the set of all $(n+m)!$ possible permutations of the integers $1, \ldots, (n+m)$. To entry $c_{i\varepsilon}$, which denotes the cost of a node deletion, the cost of the deletion of all incident edges of u_i can be added, and to the entry $c_{\varepsilon j}$, which denotes the cost of a node insertion, the cost of all insertions of the incident edges of v_j can be added. We denote the cost matrix which is enriched with structural information with $\mathbf{C}^* = (c^*_{ij})$ from now on.

In [11] the cost matrix $\mathbf{C}^* = (c^*_{ij})$ as defined above is employed in order to optimally solve the LSAP by means of *Munkres Algorithm* [15][2]. 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 $\sum_{i=1}^{(n+m)} c^*_{i\varphi^*_i}$. Similar to the permutation $(\varphi_1, \ldots, \varphi_{n+m})$ obtained on the QAP, the permutation $(\varphi^*_1, \ldots, \varphi^*_{n+m})$ corresponds to a bijective assignment of the entities in V_1^+ to the entities in V_2^+. In other words, the permutation $(\varphi^*_1, \ldots, \varphi^*_{(n+m)})$ refers to an admissible and complete (yet not necessarily minimal cost) edit path between the graphs under consideration. We denote this approximation framework with *BP-GED* from now on.

Recently, it has been proposed to solve the LSAP stated on $\mathbf{C}^*$ with an approximation rather than with an exact algorithm [13,14]. This algorithm iterates through $\mathbf{C}^*$ from top to bottom through all rows and assigns every element to the minimum unused element in a greedy manner. Clearly, the complexity of this suboptimal assignment algorithm is $O((n+m)^2)$. For the remainder of this paper we denote the graph edit distance approximation where the LSAP on $\mathbf{C}^*$ is solved by means of this greedy procedure with *GR-GED*.

3 Building the Utility Matrix

Similar to [13,14] we aim at solving the basic LSAP in $O(n^2)$ time in order to approximate the graph edit distance. Yet, in contrast with this previous approach, which considers the cost matrix $\mathbf{C}^* = (c^*_{ij})$ directly as its basis, we transform the given cost matrix into a *utility matrix* with equal dimension as $\mathbf{C}^*$ and work with this matrix instead.

The rationale behind this transformation is based on the following observation. When picking the minimum element c_{ij} from cost matrix $\mathbf{C}^*$, i.e. when assigning node u_i to v_j, we exclude both nodes u_i and v_j from any future assignment. However, it may happen that node v_j is not only the best choice for u_i but also for another node u_k. Because v_j is no longer available, we may be forced to map u_k to another, very expensive node v_l, such that the total assignment cost becomes higher than mapping node u_i to some node that is (slightly) more expensive than v_j. In order to take such situations into account, we incorporate additional information in the utility matrix about the the minimum and maximum value in each row, and each column.

Let us consider the i-th row of the cost matrix $\mathbf{C}^*$ and let *row-min*$_i$ and *row-max*$_i$ denote the minimum and maximum value occurring in this row, respectively. Formally, we have

$$\textit{row-min}_i = \min_{j=1,\ldots,(n+m)} c^*_{ij} \quad \text{and} \quad \textit{row-max}_i = \max_{j=1,\ldots,(n+m)} c^*_{ij}.$$

[2] The time complexity of this particular algorithm is cubic in the size of the problem, i.e. $O((n+m)^3)$.

If the node edit operation $(u_i \rightarrow v_j)$ is selected, one might interpret the quantity

$$\textit{row-win}_{ij} = \frac{\textit{row-max}_i - c^*_{ij}}{\textit{row-max}_i - \textit{row-min}_i}$$

as a *win* for $(u_i \rightarrow v_j)$, when compared to the locally worst case situation where v_k with $k = \arg\max_{j=1,\ldots,(n+m)} c^*_{ij}$ is chosen as target node for u_i. Likewise, we might interpret

$$\textit{row-loss}_{ij} = \frac{c^*_{ij} - \textit{row-min}_i}{\textit{row-max}_i - \textit{row-min}_i}$$

as a *loss* for $(u_i \rightarrow v_j)$, when compared to selecting the minimum cost assignment which would be possible in this row. Note that both $\textit{row-win}_{ij}$ and $\textit{row-loss}_{ij}$ are normalized to the interval $[0, 1]$. That is, when $c^*_{ij} = \textit{row-min}_i$ we have a maximum win of 1 and a minimum loss of 0. Likewise, when $c^*_{ij} = \textit{row-max}_i$ we observe a minimum win of 0 and a maximum loss of 1.

Overall we define the *utility* of the node edit operation $(u_i \rightarrow v_j)$ with respect to row i as

$$\textit{row-utility}_{ij} = \textit{row-win}_{ij} - \textit{row-loss}_{ij} = \frac{\textit{row-max}_i + \textit{row-min}_i - 2c^*_{ij}}{\textit{row-max}_i - \textit{row-min}_i}.$$

Clearly, when $c_{ij} = \textit{row-min}_i$ we observe a row utility of $+1$, and vice versa, when $c_{ij} = \textit{row-max}_i$ we have a row utility of -1.

So far the utility of a node edit operation $(u_i \rightarrow v_j)$ is quantified with respect to the i-th row only. In order to take into account information about the j-th column, we seek for the minimum and maximum values that occur in column j by

$$\textit{col-min}_j = \min_{i=1,\ldots,(n+m)} c^*_{ij} \qquad \text{and} \qquad \textit{col-max}_j = \max_{i=1,\ldots,(n+m)} c^*_{ij}.$$

Eventually, we define

$$\textit{col-win}_{ij} = \frac{\textit{col-max}_j - c^*_{ij}}{\textit{col-max}_j - \textit{col-min}_j} \qquad \text{and} \qquad \textit{col-loss}_{ij} = \frac{c^*_{ij} - \textit{col-min}_j}{\textit{col-max}_j - \textit{col-min}_j}.$$

Similarly to the utility of the node edit operation $(u_i \rightarrow v_j)$ with respect to row i we may define the utility of the same edit operation with respect to column j as

$$\textit{col-utility}_{ij} = \textit{col-win}_{ij} - \textit{col-loss}_{ij} = \frac{\textit{col-max}_j + \textit{col-min}_j - 2c^*_{ij}}{\textit{col-max}_j - \textit{col-min}_j}.$$

To finally estimate the utility u_{ij} of a node edit operation $(u_i \rightarrow v_j)$ with respect to both row i and column j we compute the sum

$$u_{ij} = \textit{row-utility}_{ij} + \textit{col-utility}_{ij}.$$

Since both $\textit{row-utility}_{ij}$ and $\textit{col-utility}_{ij}$ lie in the interval $[-1, 1]$, we have $u_{ij} \in [-2, 2]$ for $i, j = 1, \ldots, (n+m)$. We denote the final utility matrix by $\mathbf{U} = (u_{ij})$.

4 Experimental Evaluation

In the experimental evaluation we aim at investigating the benefit of using the utility matrix $\mathbf{U}$ instead of the cost matrix $\mathbf{C}^*$ in the framework GR-GED. In particular, we aim at assessing the quality of the different distance approximations by means of comparisons of the sum of distances and by means of a distance based classifier. Actually, a nearest-neighbor classifier (NN) is employed. Note that there are various other approaches to graph classification that make use of graph edit distance in some form. Yet, the nearest neighbor paradigm is particularly interesting for the present evaluation because it directly uses the distances without any additional classifier training.

We use three real world data sets from the IAM graph database repository [16][3]. Two graph data sets involve graphs that represent molecular compounds (AIDS and MUTA). These data set consists of two classes, which represent molecules with activity against HIV or not (AIDS), and molecules with and without the *mutagen* property (MUTA), respectively. The third data set consists of graphs representing proteins stemming from six different classes (PROT).

Table 1. The mean run time for one matching ($\varnothing t$), the relative increase of the sum of distances compared with BP-GED, and the recognition rate (rr) of a nearest-neighbor classifier using a specific graph edit distance algorithm.

Data Set	Algorithm								
	BP-GED(C^*)			GR-GED(C^*)			GR-GED(U)		
	$\varnothing t$	sod	rr	$\varnothing t$	sod	rr	$\varnothing t$	sod	rr
AIDS	3.61	-	99.07	1.21	1.92	98.93	1.34	2.40	99.00
MUTA	33.89	-	70.20	4.56	1.50	70.10	5.06	0.68	71.60
PROT	25.54	-	67.50	13.31	10.86	64.50	14.11	2.71	66.00

In Table 1 the results obtained with three different graph edit distance approximations are shown. The first algorithm is BP-GED($\mathbf{C}^*$), which solves the LSAP on $\mathbf{C}^*$ in an optimal manner in cubic time [11]. The second algorithm is GR-GED($\mathbf{C}^*$), which solves the LSAP on $\mathbf{C}^*$ in a greedy manner in quadratic time [13,14]. Finally, the third algorithm is GR-GED($\mathbf{U}$), which operates on the utility matrix $\mathbf{U}$ instead of $\mathbf{C}^*$ (also using the greedy assignment algorithm).

We first focus on the mean run time for one matching in ms ($\varnothing t$) and compare BP-GED with GR-GED that operates on the original cost matrix $\mathbf{C}^*$. On all data sets substantial speed-ups of GR-GED($\mathbf{C}^*$) can be observed. On the AIDS data set, for instance, the greedy approach GR-GED($\mathbf{C}^*$) is approximately three times faster than BP-GED. On the MUTA data set the mean matching time is decreased from 33.89ms to 4.56ms (seven times faster) and on the PROT data the greedy approach approximately halves the matching time (25.43 ms

[3] www.iam.unibe.ch/fki/databases/iam-graph-database.

vs. 13.31 ms). Comparing GR-GED($\mathbf{C}^*$) with GR-GED($\mathbf{U}$) we observe only a small increase of the matching time when the latter approach is used. The slight increase of the run time, which is actually observable on all data sets, is due to the computational overhead that is necessary for transforming the cost matrix $\mathbf{C}^*$ to the utility matrix $\mathbf{U}$.

Next, we focus on the distance quality of the greedy approximation algorithms. Note that all of the employed algorithms return an upper bound on the true edit distance, and thus, the lower the sum of distances of a specific algorithm is, the better is its approximation quality. For our evaluation we take the sum of distances returned by BP-GED as reference point and measure the relative increase of the sum of distances when compared with BP-GED (*sod*). We observe that GR-GED($\mathbf{C}^*$) increases the sum of distances by 1.92 % on the AIDS data when compared with BP-GED. On the other two data sets the sum of distances is also increased (by 1.50 % and 10.86 %, respectively). By using the utility matrix $\mathbf{U}$ rather than the cost matrix $\mathbf{C}$ in the greedy assignment algorithm, we observe smaller sums of distances on the MUTA and PROT data sets. Hence, we conclude that GR-GED($\mathbf{U}$) is able to produce more accurate approximations than GR-GED($\mathbf{C}$) in general.

Finally, we focus on the recognition rate (*rr*) of a NN-classifier that uses the different distance approximations. We observe that the NN-classifier that is based on the distances returned by GR-GED($\mathbf{C}^*$) achieves lower recognition rates than the same classifier that uses distances from BP-GED (on all data sets). This loss in recognition accuracy may be attributed to the fact that the approximations in GR-GED are coarser than those in BP-GED. Yet, our novel procedure, i.e. GR-GED($\mathbf{U}$), improves the recognition accuracy on all data sets when compared to GR-GED($\mathbf{C}^*$). Moreover, we observe that GR-GED($\mathbf{U}$) is inferior to BP-GED in two out of three cases only.

5 Conclusions and Future Work

In the present paper we propose to use a utility matrix instead of a cost matrix for the assignment of local substructures in a graph. The motivation for this transformation is based on the greedy behavior of the basic assignment algorithm. More formally, with the transformation of the cost matrix into a utility matrix we aim at increasing the probability of selecting a correct node edit operation during the optimization process. With an experimental evaluation on three real world data sets, we empirically confirm that our novel approach is able to increase the accuracy of a distance based classifier, while the run time is nearly not affected.

In future work we aim at testing other (greedy) assignment algorithms on the utility matrix $\mathbf{U}$. Moreover, there seems to be room for developing and researching variants of the utility matrix with the aim of integrating additional information about the trade-off between wins and losses of individual 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. J. Pattern Recognit. Art Intell. **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. J. Pattern Recognit. Art Intell. **28**(1), 1450001 (2014)
3. Luo, B., Wilson, R.C., Hancock, E.R.: Spectral feature vectors for graph clustering. In: Caelli, T.M., Amin, A., Duin, R.P.W., Kamel, M.S., de Ridder, D. (eds.) SPR 2002 and SSPR 2002. LNCS, vol. 2396, p. 83. Springer, Heidelberg (2002)
4. Wilson, R.C., Hancock, E.R., Luo, B.: Pattern vectors from algebraic graph theory. IEEE Trans. Pattern Anal. Mach. Intell. **27**(7), 1112–1124 (2005)
5. Gaüzère, B., Brun, L., Villemin, D.: Two new graphs kernels in chemoinformatics. Pattern Recognit. Lett. **33**(15), 2038–2047 (2012)
6. Borgwardt, K., Kriegel, H.-P.: Graph kernels for disease outcome prediction from protein-protein interaction networks. Pac. Symp. Biocomput. **2007**, 4–15 (2007)
7. Torsello, A., Hancock, E.: Computing approximate tree edit distance using relaxation labeling. Pattern Recognit. Lett. **24**(8), 1089–1097 (2003)
8. Bunke, H., Allermann, G.: Inexact graph matching for structural pattern recognition. Pattern Recognit. Lett. **1**, 245–253 (1983)
9. Sanfeliu, A., Fu, K.S.: A distance measure between attributed relational graphs for pattern recognition. IEEE Trans. Syst. Man Cybern. (Part B) **13**(3), 353–363 (1983)
10. Fischer, A., Plamondon, R., Savaria, Y., Riesen, K., Bunke, H.: A hausdorff heuristic for efficient computation of graph edit distance. In: Fränti, P., Brown, G., Loog, M., Escolano, F., Pelillo, M. (eds.) S+SSPR 2014. LNCS, vol. 8621, pp. 83–92. Springer, Heidelberg (2014)
11. Riesen, K., Bunke, H.: Approximate graph edit distance computation by means of bipartite graph matching. Image Vis. Comput. **27**(4), 950–959 (2009)
12. Burkard, R., Dell'Amico, M., Martello, S.: Assignment Problems. Society for Industrial and Applied Mathematics, Philadelphia (2009)
13. Riesen, K., Ferrer, M., Dornberger, R., Bunke, H.: Greedy graph edit distance. In: Perner, P. (ed.) MLDM 2015. LNCS, vol. 9166, pp. 3–16. Springer, Heidelberg (2015)
14. Riesen, K., Ferrer, M., Fischer, A., Bunke, H.: Approximation of graph edit distance in quadratic time. In: Liu, C.-L., Luo, B., Kropatsch, W.G., Cheng, J. (eds.) GbRPR 2015. LNCS, vol. 9069, pp. 3–12. Springer, Heidelberg (2015)
15. Munkres, J.: Algorithms for the assignment and transportation problems. J. Soci. Ind. Appl. Math. **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.) Structural, Syntactic, and Statistical Pattern Recognition. LNCS, vol. 5342, pp. 287–297. Springer, Heidelberg (2008)