Neural Process Lett
DOI 10.1007/s11063-017-9739-7

On the Impact of Using Utilities Rather than Costs for Graph Matching

Kaspar Riesen[1] · **Andreas Fischer**[2] · **Horst Bunke**[3]

© Springer Science+Business Media, LLC 2017

Abstract The concept of graph edit distance constitutes one of the most flexible graph matching paradigms available. The major drawback of graph edit distance, viz. the exponential time complexity, has been recently overcome by means of a reformulation of the edit distance problem to a linear sum assignment problem. However, the substantial speed up of the matching is also accompanied by an approximation error on the distances. Major contribution of this paper is the introduction of a transformation process in order to convert 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 five benchmark graph sets with respect to the accuracy and run time of a distance based classifier.

Keywords Structural pattern recognition · Graph matching · Greedy graph edit distance · Utility matrix

1 Introduction

The use of feature vectors for pattern representation implicates two limitations. First, vectors are based on a predefined set of features, and thus all vectors in a given application have to

✉ Kaspar Riesen
kaspar.riesen@fhnw.ch

Andreas Fischer
andreas.fischer@unifr.ch

Horst Bunke
bunke@inf.unibe.ch

[1] Institute for Information Systems, University of Applied Sciences FHNW, Riggenbachstrasse 16, 4600 Olten, Switzerland

[2] Department of Informatics, University of Fribourg and HES-SO, 1700 Fribourg, Switzerland

[3] Institute of Computer Science, University of Bern, Neubrückstrasse 10, 3012 Bern, Switzerland

Published online: 09 November 2017

preserve the same length regardless of the size or complexity of the corresponding pattern. Second, there is no direct possibility to describe binary (or higher-order) relationships that might exist among different parts of a pattern. These two drawbacks are severe, particularly when the patterns under consideration are characterized by complex structural relationships rather than the statistical distribution of a fixed set of pattern features.

Graphs are able to overcome both limitations and are thus regarded as versatile alternative to feature vectors. Due to their power and flexibility graphs found widespread application in pattern recognition and related fields [10,17]. 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 many pattern recognition algorithms). 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. However, the same task for graphs is much more complex, as one has to identify common parts of the graphs by considering all of their subsets of nodes.

With the rise of graph kernels [18] as well as graph embedding methods [39], the gap between vector based and graph based pattern recognition has been bridged. However, both approaches crucially depend on similarity or dissimilarity computation on graphs, commonly referred to as *graph matching*. The overall aim of graph matching is to find a correspondence between the nodes and edges of two graphs that satisfies some, more or less, stringent constraints. In the last 4 decades a huge number of procedures for graph matching have been proposed in the literature (e.g. [11,20,47]).

Graph edit distance [8,48], introduced about 30 years ago, is still one of the most flexible graph distance models available and topic of various recent research projects. In fact, the concept of graph edit distance is particularly interesting because it 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.

In order to compute the graph edit distance often tree search techniques endowed with some heuristics are employed (e.g. [16]). However, this type of search algorithm is exponential in the number of nodes of the involved graphs. In [38] authors of this paper introduced an algorithmic framework for a suboptimal computation 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 [9]. In two recent papers [41,42] 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 this paper is to improve the overall distance accuracy of this recent procedure by means of an elaborated transformation of the underlying cost model.

This paper is based on a preliminary contribution presented in [45]. The current paper has been extended with respect to both the theoretic foundation and the underlying method. Moreover, the description of the novel approach as well as the underlying concepts are more detailed as in the preliminary paper. Last but not least, compared to the preliminary contribution the experimental evaluation as well as the discussion has been substantially extended. In particular, two additional real world data sets are employed for empirical investigations.

The remainder of this paper is organized as follows. Next, in Sect. 2, the computation of graph edit distance is 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 thoroughly described. Eventually, in Sect. 4, we empirically confirm the benefit of this transformation in a classification experiment on five graph data sets. Finally, in Sect. 5, we conclude the paper and outline possible future research activities.

2 Graph Edit Distance

Originally, the paradigm of edit distance has been proposed for sequential data structures such as strings [28,58]. Eventually, the idea of edit distance has been extended to more general data structures such as trees [49] and graphs [8,13,48,56,57].

Definition 1 (*Graph*) A graph g is defined as $g = (V, E)$, where V refers to the finite set of nodes and $E \subseteq V \times V$ is the set of edges.

In this work we consider graphs to be labeled. Formally, we use $\mu : V \rightarrow L_V$ and $\nu : E \rightarrow L_E$ as node and edge labeling functions, respectively. The label alphabets L_V and L_E for both nodes and edges are generally not restricted to any domain. That is, the alphabets 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. Note that unlabeled graphs can be seen as special cases of labeled graphs by assigning the same (empty) label $\varnothing$ to all nodes and edges, i.e. $L_V = L_E = \{\varnothing\}$.

The idea of *graph edit distance* is to define a dissimilarity measure based on the number as well as the strength of so called *edit operations* that have to be applied to transform a graph $g_1 = (V_1, E_1, \mu_1, \nu_1)$ into another graph $g_2 = (V_2, E_2, \mu_2, \nu_2)$. These edit operations are commonly given by *insertions*, *deletions*, and *substitutions* of both nodes and edges. However, other edit operations such as *merging* or *splitting* might be useful in some applications but not considered in the current work (we refer to [3] for an application of additional edit operations). We denote the substitution of two nodes $u \in V_1$ and $v \in V_2$ by $(u \rightarrow v)$, the deletion of node $u \in V_1$ by $(u \rightarrow \varepsilon)$, and the insertion of node $v \in V_2$ by $(\varepsilon \rightarrow v)$, where ε refers to the empty node. For edge edit operations we use a similar notation.

Definition 2 (*Edit Path*) A set $\{e_1, \ldots, e_k\}$ of k edit operations e_i that transform g_1 completely into g_2 is called an *edit path* $\lambda(g_1, g_2)$ between g_1 and g_2.

Note that edge edit operations are unambiguously defined via node edit operations. That is, whether an edge (u, v) is substituted, deleted, or inserted, depends on the edit operations actually performed on both adjacent nodes u and v. Formally, let $u, u' \in V_1 \cup \{\varepsilon\}$ and $v, v' \in V_2 \cup \{\varepsilon\}$, and assume that both edit operations $(u \rightarrow v)$ and $(u' \rightarrow v')$ are present in the edit path $\lambda(g_1, g_2)$ under consideration. Depending on whether or not there is an edge $(u, u') \in E_1$ and/or an edge $(v, v') \in E_2$, the following three cases can be distinguished.

1. If there are edges $e_1 = (u, u') \in E_1$ and $e_2 = (v, v') \in E_2$, the edge substitution $(e_1 \rightarrow e_2)$ is implied by $(u \rightarrow v)$ and $(u' \rightarrow v')$.
2. If there is an edge $e_1 = (u, u') \in E_1$ but no edge $e_2 = (v, v') \in E_2$, the edge deletion $(e_1 \rightarrow \varepsilon)$ is implied by $(u \rightarrow v)$ and $(u' \rightarrow v')$. Obviously, if v or v' refers to the empty node ε there cannot be any edge $(v, v') \in E_2$ and thus the edge deletion $(e_1 \rightarrow \varepsilon)$ is necessary.
3. If there is no edge $e_1 = (u, u') \in E_1$ but an edge $e_2 = (v, v') \in E_2$, the edge insertion $(\varepsilon \rightarrow e_2)$ is implied by $(u \rightarrow v)$ and $(u' \rightarrow v')$. Again, if $u = \varepsilon$ or $u' = \varepsilon$ there cannot be any edge $(u, u') \in E_1$.

Thus, it is in general sufficient that an edit path $\lambda(g_1, g_2)$ covers the nodes from V_1 and V_2 only. From now on we assume that an edit path $\lambda(g_1, g_2)$ explicitly describes the correspondences found between the graphs' nodes V_1 and V_2, while the edge edit operations are implicitly given by these node correspondences.

Commonly, one introduces a cost $c(e)$ for every edit operation e, measuring the strength of the corresponding operation. The idea of such a cost is to define whether or not an edit operation e represents a strong modification of the graph. Clearly, between two similar graphs, there should exist an inexpensive edit path, representing low cost operations, while for dissimilar graphs an edit path with high cost is needed. Consequently, the edit distance of two graphs is defined as follows.

Definition 3 (*Graph Edit Distance*) Let $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ be two graphs. The *graph edit distance* $d_{\lambda_{\min}}(g_1, g_2)$, or $d_{\lambda_{\min}}$ for short, between g_1 and g_2 is defined by

$$d_{\lambda_{\min}}(g_1, g_2) = \min_{\lambda \in \Upsilon(g_1, g_2)} \sum_{e_i \in \lambda} c(e_i) \quad , \tag{1}$$

where $\Upsilon(g_1, g_2)$ denotes the set of all complete edit paths transforming g_1 into g_2, c denotes the cost function measuring the strength $c(e_i)$ of node edit operation e_i (including the cost of all edge edit operations implied by the operations applied on the adjacent nodes of the edges), and $\lambda_{\min}$ refers to the minimal cost edit path found in $\Upsilon(g_1, g_2)$.

There might be two (or more) edit paths with equal minimal cost in $\Upsilon(g_1, g_2)$. That is, the minimal cost edit path $\lambda_{\min} \in \Upsilon(g_1, g_2)$ is not necessarily unique. Moreover, graph edit distance does not build a metric function in general. In practice, however, the definition of some weak conditions on the cost function c are sufficient such that the graph edit distance becomes a metric function [8].

An adequate definition of cost functions is not only important for building a metric, but also for the effectiveness of edit distance based pattern recognition (see [52] for an extensive review on different cost functions for graph edit distance). In case of unlabeled graphs, the cost is usually defined via unit cost for all deletions and insertions of both nodes and edges, while substitutions are free of cost. Formally,

$$\begin{aligned} c(u \to \varepsilon) = c(\varepsilon \to u') &= c((u, v) \to \varepsilon) \\ &= c(\varepsilon \to (u', v')) = 1 \\ c(u \to u') &= c((u, v) \to (u', v')) = 0 \end{aligned}$$

for all nodes $u, v \in V_1$ and $u', v' \in V_2$ as well as all edges $(u, v) \in E_1$ and $(u', v') \in E_2$.

In general, however, the cost $c(e)$ of a particular edit operation e is defined with respect to the underlying label alphabets L_V and L_E. For instance, for numerical node and edge labels, i.e. for label alphabets $L_V, L_E = \mathbb{R}^n$, a Minkowski distance can be used to model the cost of a substitution operation on the graphs. The Minkowski cost function defines the substitution cost proportional to the Minkowski distance of the two corresponding labels. The basic intuition behind this approach is that the more dissimilar two labels are, the stronger is the distortion associated with the corresponding substitution.

In other applications the node and/or edge labels might be not numerical and thus non-numerical distance functions have to be employed to measure the cost of a particular substitution operation. For instance, the label alphabet can be given by the set of all strings of arbitrary size over a finite set of symbols. In this case a distance model for strings, as for instance the *string edit distance* [28,58], could be used for measuring the cost of a substitution. In other problem domains, the label alphabet might be given by a finite set of n symbolic labels $L_{V/E} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$. In such case a substitution cost model using a *Dirac function*, which returns zero when the involved labels are identical and a non-negative constant otherwise, could be the method of choice.

The definition of application specific cost functions, which can be adopted to the peculiarity of the underlying label alphabet, accounts for the flexibility of graph edit distance. However, prior knowledge about the labels and their meaning has to be available. If in a particular case this prior knowledge is not available, automatic procedures for learning the cost model from a set of sample graphs are available as well [12,30–32,53].

2.1 Computation of Graph Edit Distance

The exact computation of graph edit distance $d_{\lambda_{\min}}(g_1, g_2)$ is often based on A* [5,16,21,40]. A* is a best-first search algorithm [22] which always finds an optimal solution if the underlying heuristic function is *admissible*.

However, for graphs with m and n nodes the time complexity of these complete and admissible search algorithms is in $O(m^n)$. Hence, the computation of exact edit distance is limited to graphs of rather small size. In fact, graph edit distance belongs to the family of *quadratic assignment problems* (QAPs) [26], which in turn belong to the class of NP-complete problems (see [7,36] for details on formulating the GED problem as QAP). That is, an exact and efficient algorithm for the graph edit distance problem can not be developed unless $P = NP$.[1]

Various methods address the high complexity of graph edit distance computation. Local optimization criteria [6,33,54], for instance, are used to solve the error-tolerant graph matching problem in a more efficient way. Another idea for efficient graph edit distance is to prune the underlying search tree and consequently reduce both the search space and the matching time [34]. Linear programming for computing the edit distance of graphs with unlabeled edges is proposed in [25]. Finding an optimal match between the sets of subgraphs by means of dynamic programming [13,14] is another possibility for speeding up the computation of graph edit distance.

2.1.1 Cubic Time Approximation of Graph Edit Distance

Authors of this paper introduced an algorithmic framework which allows the approximate computation of graph edit distance in a substantially faster way than traditional methods on general graphs [38]. The basic idea of this approach is to reduce the quadratic assignment problem 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. However, in contrast with QAPs, LSAPs are able to optimize the assignment problem with respect to a linear term only.

For solving LSAPs a large number of algorithms exist (see [9] for an exhaustive survey). They range from primal–dual combinatorial algorithms [24,27,29], to simplex-like methods [2,35] and other approaches [1,55]. The time complexity of the best performing exact algorithms for LSAPs is cubic in the size of the problem. Hence, LSAPs can be—in contrast with QAPs—quite efficiently solved.

LSAPs are concerned with the problem of finding the best bijective assignment between the independent entities of two sets $S_1 = \{s_1^{(1)}, \ldots, s_n^{(1)}\}$ and $S_2 = \{s_1^{(2)}, \ldots, s_n^{(2)}\}$ of equal

[1] www.claymath.org/millennium-problems/p-vs-np-problem.

size. In order to assess the quality of an assignment of two entities, a cost c_{ij} is commonly defined that measures the suitability of assigning the i-th element $s_i^{(1)} \in S_1$ to the j-th element $s_j^{(2)} \in S_2$ (resulting in $n \times n$ cost values c_{ij} $(i, j = 1, \dots, n)$).

Definition 4 (*Linear Sum Assignment Problem (LSAP)*) Given two disjoint sets $S_1 = \{s_1^{(1)}, \dots, s_n^{(1)}\}$ and $S_2 = \{s_1^{(2)}, \dots, s_n^{(2)}\}$ and a cost c_{ij} for every pair of entities $(s_i^{(1)}, s_j^{(2)}) \in S_1 \times S_2$, the Linear Sum Assignment Problem (LSAP) is given by finding

$$\min_{(\varphi_1,\dots,\varphi_n)\in\mathscr{S}_n} \sum_{i=1}^{n} c_{i\varphi_i}$$

where $\mathscr{S}_n$ refers to the set of all $n!$ possible permutations of n integers.

By reformulating the graph edit distance problem to an instance of an LSAP, three major issues have to be resolved.

1. First, LSAPs are generally stated on independent sets with equal cardinality. However, in our case the elements to be assigned to each other are given by the sets of nodes (and edges) with unequal cardinality in general.
2. Second, solutions to LSAPs 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 an LSAP corresponds to a bijective assignment of the underlying entities). However, graph edit distance is a more general assignment problem as it explicitly allows both deletions and insertions to occur on the basic entities (rather than only substitutions).
3. Third, graphs do not only consist of independent sets of entities (i.e. nodes) but also of structural relationships between these entities (i.e. edges that connect pairs of nodes). LSAPs are not able to consider these relationships in a global and consistent way.

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

$$V_1^+ = V_1 \cup \overbrace{\{\varepsilon_1, \dots, \varepsilon_m\}}^{m \textit{ empty nodes}}$$

and

$$V_2^+ = V_2 \cup \underbrace{\{\varepsilon_1, \dots, \varepsilon_n\}}_{n \textit{ empty nodes}}.$$

The LSAP can now be carried out on these extended node sets. Formally, based on the extended node sets

$$V_1^+ = \{u_1, \dots, u_n, \varepsilon_1, \dots, \varepsilon_m\} \text{ and } V_2^+ = \{v_1, \dots, v_m, \varepsilon_1, \dots, \varepsilon_n\}$$

of g_1 and g_2, respectively, a *cost matrix* $\mathbf{C}$ can be established as follows.

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

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

Obviously, the left upper part of the cost matrix $\mathbf{C} = (c_{ij})$ represents the costs of all possible node substitutions, the diagonal of the right upper part the costs of all possible node deletions, and the diagonal of the bottom left part the costs of all possible node insertions. Every node can be deleted or inserted at most once. Therefore any non-diagonal element of the right-upper and left-lower part can be set to ∞. The bottom right part of the cost matrix is set to zero since substitutions of the form $(\varepsilon \rightarrow \varepsilon)$ should not cause any cost. In [46,50,51] alternative definitions of a cost matrix $\mathbf{C}$ have been proposed in order to decrease the size of the assignment problem.

The third issue stated above is about the edge structure of both graphs which cannot be considered by LSAPs. In fact, so far the cost matrix $\mathbf{C} = (c_{ij})$ considers the nodes of both graphs only, and thus the assignment algorithm does not take any structural constraints into account. In order to integrate knowledge about the graph structure, to each entry c_{ij}, i.e. to each cost of a node edit operation $(u_i \rightarrow v_j)$, the minimum sum of edge edit operation costs, implied by the corresponding node operation, is added.

This particular encoding of the minimum matching cost arising from the local edge structure enables the LSAP to consider information about the local, yet not global, edge structure of a graph. Hence, this heuristic procedure partially resolves the third issue (a complete solution for this third problem would be equivalent to an exact computation of graph edit distance, which would be unreasonable of course). Several other attempts have been made to include more adjacency information into the assignment process of [38] (e.g. [19,44]). However, in this paper we make use of the original version of the algorithm.

The permutation $(\varphi_1, \dots, \varphi_{(n+m)})$ that minimizes the sum of cost

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

actually corresponds to a bijective assignment

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

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, the permutation $(\varphi_1, \ldots, \varphi_{(n+m)})$ perfectly corresponds to a to an admissible and complete edit path between the graphs under consideration, i.e. $\psi \in \Upsilon(g_1, g_2)$. The sum of costs of ψ gives us an approximation value $d_\psi(g_1, g_2)$, or d_ψ for short, for the graph edit distance.

Note that the edge operations are only added to the edit path ψ after the optimization process has been terminated. This is because LSAP solving algorithms are not able to take information about assignments of adjacent nodes into account during run time. In other words, for finding the edit path $\psi \in \Upsilon(g_1, g_2)$ based on the cost matrix $\mathbf{C} = (c_{ij})$ the structural information of the graphs is considered in an isolated way only (single nodes and their adjacent edges). Hence, $\psi \in \Upsilon(g_1, g_2)$ is a suboptimal edit path with cost greater than, or equal to, the optimal edit path $\lambda_{\min}$ (see [43] for a formal proof).

For the remainder of this paper we denote this graph edit distance approximation algorithm with *BP-GED* (*Bipartite Graph Edit Distance*).[2]

2.1.2 Quadratic Time Approximation of Graph Edit Distance

From a high level perspective, the algorithmic framework presented in [38] consists of the following three major steps.

1. In a first step the graphs to be matched are subdivided into individual nodes including local structural information.
2. Next, in step 2, an algorithm that solves the LSAP is employed in order to find an optimal assignment of the nodes (plus local structures) of both graphs.
3. Finally, in step 3, an approximate graph edit distance is derived from the assignment of step 2.

For the second step of BP-GED Munkres' algorithm [29], also referred to as Kuhn–Munkres, or Hungarian algorithm, is deployed in the existing framework [38]. 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. There exist several studies where different LSAP solving algorithms for this particular graph matching procedure have been compared with each other [15,23,51].

Recently, it has been proposed to solve the LSAP stated on $\mathbf{C}$ with an approximation rather than with an exact algorithm [41,42]. While for the optimal solution of LSAPs quite an arsenal of algorithms is available, only a few works are concerned with the suboptimal solution of general LSAPs (see [4] for an early survey).

A basic greedy algorithm that suboptimally solves the LSAP stated on cost matrix $\mathbf{C}$ is formalized in Algoritm 1. 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. More formally, for each row i in the cost matrix $\mathbf{C} = (c_{ij})$ the minimum cost entry $\varphi_i = \underset{\forall j}{\arg\min}\, c_{ij}$ 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 exactly once (i.e. $\forall j$ refers to available columns in $\mathbf{C}$). Clearly, the complexity of this suboptimal assignment algorithm is $O((n+m)^2)$.

In contrast with optimal LSAP solvers, the quadratic time assignment method operates in a greedy manner. That is, this approach is not able to undo a certain node assignment once it

[2] The assignment problem can also be formulated as finding a matching in a *complete bipartite graph* and is therefore also referred to as *bipartite graph matching problem*.

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

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

has been added to ψ. Hence, this particular method crucially depends on the order in which the nodes are processed. However, the nodes of a graph and thus the first n rows (and m columns) in $\mathbf{C}$ are arbitrarily ordered.

In order to take this observation into account, we reorder the first n rows of $\mathbf{C}$ in ascending order with respect to the minimum cost entry per row (before the assignment is carried out). Formally, we sort the cost matrix $\mathbf{C}$ such that

$$\min_{\forall j} c_{1j} \leq \min_{\forall j} c_{2j} \leq \ldots \leq \min_{\forall j} c_{nj} \quad .$$

That is, in the first row the overall minimum cost assignment can be found, while the second row contains the second smallest row minimum, the third row the third smallest row minimum, and so on.

This heuristic ensures that the most evident assignments, i.e. the assignments with overall lowest cost, are considered first by our greedy assignment. For the remainder of this paper we denote the graph edit distance approximation where the basic node assignment (i.e. step 2 of BP-GED) is computed by means of this greedy procedure with *GR-GED*.

3 Building the Utility Matrix

Major contribution of this paper is the introduction of a novel matrix to solve the graph matching problem. Similar to [41,42] we aim at solving the basic LSAP in $O(n^2)$ time in order to approximate the graph edit distance with a greedy assignment. However, 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.

Basically, for each assignment $(u_i \rightarrow v_j)$ the cost c_{ij} is assessed with respect to all cost entries in the i-th row and j-th column of $\mathbf{C}$. That is, we define a utility of this particular edit operation in relation to all other possible assignments that involve u_i or v_j.

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

$$\textit{row-min}_i = \min_{j=1,\ldots,(n+m)} c_{ij}$$

and

$$\textit{row-max}_i = \max_{j=1,\ldots,(n+m)} c_{ij} \quad .$$

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

$$\begin{aligned}\textit{row-utility}_{ij} &= \textit{row-win}_{ij} - \textit{row-loss}_{ij} \\ &= \frac{\textit{row-max}_i + \textit{row-min}_i - 2c_{ij}}{\textit{row-max}_i - \textit{row-min}_i} \quad .\end{aligned}$$

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

$$\textit{col-min}_j = \min_{i=1,\ldots,(n+m)} c_{ij}$$

and

$$\textit{col-max}_j = \max_{i=1,\ldots,(n+m)} c_{ij} \quad .$$

Eventually, we define

$$\textit{col-win}_{ij} = \frac{\textit{col-max}_j - c_{ij}}{\textit{col-max}_j - \textit{col-min}_j}$$

and

$$\textit{col-loss}_{ij} = \frac{c_{ij} - \textit{col-min}_j}{\textit{col-max}_j - \textit{col-min}_j} \quad .$$

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

$$\begin{aligned}\textit{col-utility}_{ij} &= \textit{col-win}_{ij} - \textit{col-loss}_{ij} \\ &= \frac{\textit{col-max}_j + \textit{col-min}_j - 2c_{ij}}{\textit{col-max}_j - \textit{col-min}_j} \quad .\end{aligned}$$

To finally estimate the utility u_{ij} of a node edit operation $(u_i \rightarrow v_j)$ we apply one of the following three rules.

– Min:

$$u_{ij} = \min(\textit{row-utility}_{ij}, \textit{col-utility}_{ij})$$

– Max:

$$u_{ij} = \max(\textit{row-utility}_{ij}, \textit{col-utility}_{ij})$$

– Sum:

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

Since both *row-utility*$_{ij}$ and *col-utility*$_{ij}$ lie in the interval $[-1, 1]$, we observe $u_{ij} \in [-1, 1]$ for the first and second rule, while $u_{ij} \in [-2, 2]$ accounts for the third rule. We denote the final utility matrix by $\mathbf{U}_{\text{min}}$, $\mathbf{U}_{\text{max}}$, or $\mathbf{U}_{\text{sum}}$ (depending on the rule actually employed) from now on.

We aim at employing the same greedy assignment procedure on **U** as used for GR-GED (see Algorithm 1.[3]) Hence, we also reorder the first n rows of **U**. However, this time in descending order with respect to the maximum utilities in each row. That is, in the first row the overall maximum utility can be found, the second row contains the second highest row utility, and so on. Formally, we sort the utility matrix such that

$$\max_{\forall j} u_{1j} \geq \max_{\forall j} u_{2j} \geq \ldots \geq \max_{\forall j} u_{nj} \quad .$$

This heuristic again ensures that the most evident assignments, i.e. the assignments with highest utility, are considered first by our greedy assignment.

The rationale behind the transformation of **C** to **U** is based on the following observation. When picking the minimum element c_{ij} from cost matrix **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.

Example 1 Let us consider a simple toy example with a 3×3 cost matrix.

$$\mathbf{C} = \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \end{array} \begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \left[\begin{array}{ccc} 1 & 2 & 3 \\ 4 & 3 & 3 \\ 3 & 5 & 10 \end{array}\right] \end{array}$$

With a greedy algorithm we would find the assignment

$$\psi = \{u_1 \rightarrow v_1, u_2 \rightarrow v_2, u_3 \rightarrow v_3\}$$

with a total assignment cost of 14.

The row and column utilities computed on **C** are given by

$$\begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \end{array} \begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \left[\begin{array}{ccc} 1 & 0 & -1 \\ -1 & 1 & 1 \\ 1 & \frac{3}{7} & -1 \end{array}\right] \end{array} \quad \text{and} \quad \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \end{array} \begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \left[\begin{array}{ccc} 1 & 1 & 1 \\ -1 & \frac{1}{3} & 1 \\ -\frac{1}{3} & -1 & -1 \end{array}\right] \end{array}$$

[3] Applying the greedy assignment on utilities rather than costs means that we replace $\arg\min_{\forall j}$ by $\arg\max_{\forall j}$, of course.

Table 1 The mean and max number of nodes and edges in the data set

Data	$\varnothing\lvert V\rvert$	$\varnothing\lvert E\rvert$	max $\lvert V\rvert$	max $\lvert E\rvert$
AIDS	15.7	16.2	95	103
MUTA	30.3	30.8	417	112
PROT	32.6	62.1	126	149
PAH	20.7	24.4	28	34
MAO	18.4	19.6	27	29

Then, the final utility matrix (using the sum rule) is defined as

$$\mathbf{U}_{\text{sum}} = \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \end{array} \begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \begin{bmatrix} 2 & 1 & 0 \\ -2 & \frac{4}{3} & 2 \\ \frac{2}{3} & -\frac{4}{7} & -2 \end{bmatrix} \end{array}$$

resulting in the greedy assignment

$$\psi = \{u_1 \rightarrow v_1, u_2 \rightarrow v_3, u_3 \rightarrow v_2\}$$

with a lower total assignment cost, viz. 9, than directly achieved on **C**.

4 Experimental Evaluation

4.1 Setup and Data Sets

In the experimental evaluation we aim at investigating the benefit of using the utility matrix **U** instead of the cost matrix **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. There are various other approaches to graph classification that make use of graph edit distance in some form. However, the nearest neighbor paradigm is particularly interesting for the present evaluation because it directly uses the distances without any additional classifier training.

For the experimental evaluations three data sets from the IAM graph database repository [37] and two data sets from GREYC's dataset repository[4] are used. In Table 1 the mean and max number of nodes and edges for each data set is given.

Four data sets consist of graphs representing molecular compounds from different applications, viz. AIDS, MUTA, MAO, and PAH (all of these data sets represent two class problems) and one data set consists of graphs that represent proteins stemming from six different classes (PROT).

The molecular structures, which consist of atoms and covalent bonds, are converted into graphs in a very natural and straightforward manner by representing atoms as nodes and the covalent bonds as edges. Nodes are labeled with their corresponding chemical symbol and edges by the valence of the linkage.

The proteins are converted into graphs by representing the structure, the sequence, and chemical properties of a protein by nodes and edges. Nodes represent secondary structure

[4] https://brunl01.users.greyc.fr/CHEMISTRY/index.html.

Table 2 The mean run time for one matching ($\varnothing t$), the relative increase/decrease 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		Reference systems		Novel systems		
		BP-GED(C)	GR-GED(C)	GR-GED($\mathbf{U}_{min}$)	GR-GED($\mathbf{U}_{max}$) (%)	GR-GED($\mathbf{U}_{sum}$) (%)
AIDS	$\varnothing t$	3.6×10^{-3} s	1.3×10^{-3} s	1.3×10^{-3} s		
	sod	–	2.1%	2.3%	2.3	2.4
	rr	99.1%	99.0%	99.0%	98.9	99.0
MUTA	$\varnothing t$	33.9×10^{-3} s	4.8×10^{-3} s	5.1×10^{-3} s		
	sod	–	1.3%	0.7%	0.3	0.7
	rr	70.2%	69.6%	71.7%	71.2	71.6
PROT	$\varnothing t$	25.6×10^{-3} s	14.1×10^{-3} s	14.1×10^{-3} s		
	sod	–	9.9%	3.8%	8.3	2.7
	rr	67.5%	64.5%	66.0%	66.5	66.0
PAH	$\varnothing t$	3.3×10^{-3} s	3.0×10^{-3} s	3.1×10^{-3} s		
	sod	–	−4.5%	−6.6%	−6.1	−6.7
	rr	63.8%	64.9%	64.9%	66.0	64.9
MAO	$\varnothing t$	2.4×10^{-3} s	1.9×10^{-3} s	1.9×10^{-3} s		
	sod	–	18.1%	67.2%	75.7	67.2
	rr	85.3%	75.0%	80.9%	83.8	82.4

elements (SSE) within the protein structure, labeled with their type (helix, sheet, or loop) and their amino acid sequence. Every pair of nodes is connected by an edge if they are neighbors along the amino acid sequence (sequential edges) or if they are neighbors in space within the protein structure (structural edges). Every node is connected to its three nearest spatial neighbors. In case of sequential relationships, the edges are labeled with their length in amino acids, while in case of structural edges a distance measure in Ångstroms is used as a label.

For molecular compounds the following cost model has been employed. For node and edge deletions/insertions constant positive costs τ_{node} and τ_{edge} have been used. The node substitution cost is measured via Dirac function returning 0 if the two symbols are equal, and $2\tau_{node}$ otherwise. Edge substitutions are free of cost.

For the protein graphs a cost model based on the amino acid sequences is used. For node substitutions the type of the involved nodes is compared first. If two types are identical, the amino acid sequences of the nodes to be substituted are compared by means of string edit distance [58]. For edge substitutions, we measure the dissimilarity with a Dirac function returning 0 if the two edge types are equal, and $2\tau_{edge}$ otherwise.

4.2 Results and Discussion

In Table 2 the results obtained with five different graph edit distance approximations are shown. The first algorithm is BP-GED(**C**), which solves the LSAP on **C** in an optimal manner in cubic time [38]. The second algorithm is GR-GED(**C**), which solves the LSAP on **C** in a greedy manner in quadratic time [41,42]. These two systems act as reference systems for our novel approach. The remaining algorithms are GR-GED(**U**), which employ the greedy algorithm on a utility matrix **U** instead of **C** using the three rules for the definition of the utilities (Min, Max, and Sum).

We first focus on the mean run time for one matching in ms ($\varnothing t$) and compare BP-GED with GR-GED that both operate on the original cost matrix **C**. On all data sets substantial speed-ups of the greedy approach can be observed. On the AIDS data set, for instance, the greedy approach GR-GED(**C**) is approximately three times faster than BP-GED. On the MUTA data set the mean matching time is decreased from 33.9 to 4.8 ms (seven times faster) and on the PROT data the greedy approach approximately halves the matching time (25.6 vs. 14.1ms). Also on PAH and MAO slight decreased run times are observed.

Comparing GR-GED(**C**) with GR-GED(**U**) we observe no, or only negligible, increases of the matching time when the latter approach is used (we show only one mean time for all utility matrices as no differences are observable between the three strategies). The slight increase of the run time, which is actually observable on MUTA and PAH only, is due to the computational overhead that is necessary for transforming the cost matrix **C** to the utility matrix **U**.

Next, we focus on the distance quality of the greedy approximation algorithms. 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 or decrease of the sum of distances when compared with BP-GED (*sod*).

For instance, we observe that GR-GED(**C**) increases the sum of distances by 2.1% on the AIDS data when compared with BP-GED. On three other data sets, viz. MUTA, PROT, and MAO, the sum of distances is also increased (by 1.3, 9.9 and 18.1%, respectively). On PAH we observe a decrease of the sum of distance by 4.5%. By using the utility matrix **U** rather than the cost matrix **C** in the greedy assignment algorithm, we observe smaller sums of distances on the MUTA, PAH, and PROT data sets. On the other data sets our novel approach leads to higher approximation errors (on MAO, for instance, the approximation error is substantially increased). Note, however, that increased distances are not necessarily disadvantageous. For instance, increasing the distances between two graph stemming from different classes might be beneficial for distance based classifiers. Thus, we finally 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(**C**) achieves lower recognition rates than the same classifier that uses distances from BP-GED(**C**) (on all data sets but PAH). This loss in recognition accuracy may be attributed to the fact that the approximations in GR-GED are coarser than those in BP-GED. However, our novel procedure, i.e. GR-GED(**U**), improves the recognition accuracy on four out of five data sets when compared to GR-GED(**C**). For instance, on the MUTA data set the recognition rate is increased from 69.6 to 71.7% when $\mathbf{U}_{\text{min}}$ rather than **C** is used. In fact, this result on the MUTA data set refers to the overall best recognition rate among all competing algorithms. Also on the PAH data set our novel approach (using $\mathbf{U}_{\text{max}}$) achieves the overall best result.

Comparing the three different utility matrices $\mathbf{U}_{\text{min}}$, $\mathbf{U}_{\text{max}}$, and $\mathbf{U}_{\text{sum}}$ with each other, we observe that the Max rule achieves the best result on three out of five data sets (PROT, PAH, and MAO), while Min and Sum achieve the best result on two and one data set, respectively.

5 Conclusions and Future Work

In this 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 five 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 (or even decreased). In particular, our novel approach is up to seven times faster than BP-GED(**C**) and improves the classification accuracy on four out of five data sets when compared with the same algorithm operating on cost (rather than utility) matrices [GR-GED(**C**)].

In future work we aim at testing other (greedy) assignment algorithms on the utility matrix **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. Last but not least, we plan to evaluate this as well as other transformations not only on real world but also on artificial graphs in order to better understand the benefits and limitations of matrix transformations in the context of graph matching.

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

References

1. Achatz H, Kleinschmidt P, Paparrizos K (1991) Applied geometry and discrete mathematics, chap. A dual forest algorithm for the assignment problem. AMS 1–11 (1991)
2. Ahuja R, Orlin J (1992) The scaling network simplex algorithm. Oper Res 40(1):5–13
3. Ambauen R, Fischer S, Bunke H (2003) Graph edit distance with node splitting and merging and its application to diatom identification. In: Hancock E, Vento M (eds) Proceedings of 4th International workshop on graph based representations in pattern recognition, LNCS 2726. Springer, pp 95–106
4. Avis D (1983) A survey of heuristics for the weighted matching problem. Networks 13:475–493
5. Berretti S, Del Bimbo A, Vicario E (2001) Efficient matching and indexing of graph models in content-based retrieval. IEEE Trans Pattern Anal Mach Intell 23(10):1089–1105
6. Boeres M, Ribeiro C, Bloch I (2004) A randomized heuristic for scene recognition by graph matching. In: Ribeiro C, Martins S (eds) Proceedings of 3rd workshop on efficient and experimental algorithms, LNCS 3059. Springer, pp 100–113
7. Bougleux S, Brun L, Carletti V, Foggia P, Gauzere B, Vento M (2017) Graph edit distance as a quadratic assignment problem. Pattern Recognit Lett 87(1):38–46
8. Bunke H, Allermann G (1983) Inexact graph matching for structural pattern recognition. Pattern Recognit Lett 1:245–253
9. Burkard R, Dell'Amico M, Martello S (2009) Assignment problems. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA
10. Conte D, Foggia P, Sansone C, Vento M (2004) Thirty years of graph matching in pattern recognition. Int J Pattern Recognit Artif Intell 18(3):265–298
11. Cordella L, Foggia P, Sansone C, Vento M (2004) A (sub)graph isomorphism algorithm for matching large graphs. IEEE Trans Pattern Anal Mach Intell 26(20):1367–1372
12. Cortes X, Serratosa F (2015) Learning graph-matching edit-costs based on the optimality of the oracle's node correspondences. Pattern Recognit Lett 56:22–29
13. Eshera M, Fu K (1984) A graph distance measure for image analysis. IEEE Trans Syst Man Cybern (Part B) 14(3):398–408
14. Eshera M, Fu K (1984) A similarity measure between attributed relational graphs for image analysis. In: Proceedings of 7th international confernece on pattern recognition, pp 75–77
15. Fankhauser S, Riesen K, Bunke H (2011) Speeding up graph edit distance computation through fast bipartite matching. In: Jiang X, Ferrer M, Torsello A (eds) Proceedings of 8th international workshop on graph based representations in pattern recognition, LNCS 6658, pp 102–111

16. Fischer A, Plamandon R, Savaria Y, Riesen K, Bunke H (2014) A Hausdorff heuristic for efficient computation of graph edit distance. In: Fränti P, Brown G, Loog M, Escolano F, Pelillo M (eds) Proceedings of international workshop on structural and syntactic pattern recognition, LNCS 8621, pp 83–92
17. Foggia P, Percannella G, Vento M (2014) Graph matching and learning in pattern recognition in the last 10 years. Int J Pattern Recognit Artif Intell 28(1):1450001
18. Gärtner T, Horvath T, Wrobel S (2010) Graph kernels. Springer, Berlin, pp 467–469
19. Gauzere B, Bougleux S, Riesen K, Brun L (2014) Approximate graph edit distance guided by bipartite matching of bags of walks. In: Fränti P, Brown G, Loog M, Escolano F, Pelillo M (eds) Proceedings of international workshop on structural and syntactic pattern recognition, LNCS 8621, pp 73–82
20. Gauzere B, Brun L, Villemin D (2011) Two new graph kernels and applications to chemoinformatics. In: Jiang X, Ferrer M, Torsello A (eds) Proceedings of 8th international workshop on graph based representations in pattern recognition, pp 112–121
21. Gregory L, Kittler J (2002) Using graph search techniques for contextual colour retrieval. In: Caelli T, Amin A, Duin R, Kamel M, de Ridder D (eds) Proceedings of the joint IAPR international workshop on structural, syntactic, and statistical pattern recognition, LNCS 2396, pp 186–194
22. Hart P, Nilsson N, Raphael B (1968) A formal basis for the heuristic determination of minimum cost paths. IEEE Trans Syst Sci Cybern 4(2):100–107
23. Jones W, Chawdhary A, King A (2015) Revisiting volgenant-jonker for approximating graph edit distance. In: Liu C, Luo B, Kropatsch W, Cheng J (eds.) Proceedings of 10th international workshop on graph based representations in pattern recognition, LNCS 9069, pp 98–107
24. Jonker R, Volgenant A (1987) A shortest augmenting path algorithm for dense and sparse linear assignment problems. Computing 38:325–340
25. Justice D, Hero A (2006) A binary linear programming formulation of the graph edit distance. IEEE Trans Pattern Anal Ans Mach Intell 28(8):1200–1214
26. Koopmans T, Beckmann M (1975) Assignment problems and the location of economic activities. Econometrica 25:53–76
27. Kuhn H (1955) The Hungarian method for the assignment problem. Nav Res Logist Q 2:83–97
28. Levenshtein V (1966) Binary codes capable of correcting deletions, insertions and reversals. Sov Phys Dokl 10(8):707–710
29. Munkres J (1957) Algorithms for the assignment and transportation problems. J Soc Ind Appl Math 5(1):32–38
30. Neuhaus M, Bunke H (2004) A probabilistic approach to learning costs for graph edit distance. In: Kittler J, Petrou M, Nixon M (eds) Proceedings of 17th international conference on pattern recognition, 3, pp 389–393
31. Neuhaus M, Bunke H (2005) Self-organizing maps for learning the edit costs in graph matching. IEEE Trans Syst Man Cybern (Part B) 35(3):503–514
32. Neuhaus M, Bunke H (2007) Automatic learning of cost functions for graph edit distance. Inf Sci 177(1):239–247
33. Neuhaus M, Bunke H (2007) Bridging the gap between graph edit distance and kernel machines. World Scientific, Singapore
34. Neuhaus M, Riesen K, Bunke H (2006) Fast suboptimal algorithms for the computation of graph edit distance. In: Yeung DY, Kwok J, Fred A, Roli F, de Ridder D (eds) Proceedings of 11th international workshop on strucural and syntactic pattern recognition, LNCS 4109, pp 163–172
35. Orlin J (1985) On the simplex algorithm for networks and generalized networks. Math Program Stud 24:166–178
36. Riesen K (2016) Structural pattern recognition with graph edit distance. Springer, Berlin
37. Riesen K, Bunke H (2008) IAM graph database repository for graph based pattern recognition and machine learning. In: da Vitoria Lobo N et al (ed) Structural, syntactic, and statistical pattern recognition, LNCS 5342, pp 287–297
38. Riesen K, Bunke H (2009) Approximate graph edit distance computation by means of bipartite graph matching. Image Vis Comput 27(4):950–959
39. Riesen K, Bunke H (2010) Graph classification and clustering based on vector space embedding. World Scientific, Singapore
40. Riesen K, Fankhauser S, Bunke H (2007) Speeding up graph edit distance computation with a bipartite heuristic. In: Frasconi P, Kersting K, Tsuda K (eds) Proceedings of 5th international workshop on mining and learning with graphs, pp 21–24
41. Riesen K, Ferrer M, Dornberger R, Bunke H (2015) Greedy graph edit distance. In: Perner P (ed) Proceedings 11th international conference on machine learning and data mining in pattern recognition, LNAI 9166, pp 1–14

42. Riesen K, Ferrer M, Fischer A, Bunke H (2015) Approximation of graph edit distance in quadratic time. In: Liu C, Luo B, Kropatsch W, Cheng J (eds) Proceedings of 10th international workshop on graph based representations in pattern recognition, LNCS 9069, pp 3–12
43. Riesen K, Fischer A, Bunke H (2014) Computing upper and lower bounds of graph edit distance in cubic time. In: Gayar N, Schwenker F, Suen C (eds) Proceedings of international workshop on artificial neural networks in pattern recognition, LNAI 8774, pp 129–140
44. Riesen K, Fischer A, Bunke H (2014) Improving graph edit distance approximation by centrality measures. In: Proceedings of 22nd international conference on pattern recognition, pp 3910–3914
45. Riesen K, Fischer A, Bunke H (2016) Approximation of graph edit distance by means of a utility matrix. In: Schwenker F, Abbas H, El Gayar N, Trentin E (eds) Proceedings of 7th IAPR TC3 workshop on artificial neural networks in pattern recognition, LNCS 9896. Springer, pp 185–194
46. Riesen K, Neuhaus M, Bunke H (2007) Bipartite graph matching for computing the edit distance of graphs. In: Escolano F, Vento M (eds) Proceedings of 6th international workshop on graph based representations in pattern recognition, LNCS 4538, pp 1–12
47. Rossi L, Torsello A, Hancock E (2013) A continuous-time quantum walk kernel for unattributed graphs. In: Kropatsch W, Artner N, Haxhimusa Y, Jiang X (eds) Proceedings of 9th international workshop on graph based representations in pattern recognition, pp 101–110
48. Sanfeliu A, Fu K (1983) A distance measure between attributed relational graphs for pattern recognition. IEEE Trans Syst Man Cybern (Part B) 13(3):353–363
49. Selkow S (1977) The tree-to-tree editing problem. Inf Process Lett 6(6):184–186
50. Serratosa F (2014) Fast computation of bipartite graph matching. Pattern Recognit Lett 45:244–250
51. Serratosa F (2015) Speeding up fast bipartite graph matching through a new cost matrix. Int J Pattern Recognit Artif Intell 29(2):1550010
52. Serratosa F, Cortés X, Solé-Ribalta A (2012) On the graph edit distance cost: properties and applications. Int J Pattern Recognit Artif Intell 26(5):126
53. Serratosa F, Solé-Ribalta A, Cortes X (2011) Automatic learning of edit costs based on interactive and adaptive graph recognition. In: Jiang X, Ferrer M, Torsello A (eds) Proceedings of 8th international workshop on graph based representations in pattern recognition, LNCS 6658, pp 152–163
54. Sorlin S, Solnon C (2005) Reactive tabu search for measuring graph similarity. In: Brun L, Vento M (eds) Proceedings of 5th international workshop on graph-based representations in pattern recognition, LNCS 3434. Springer, pp 172–182
55. Srinivasan V, Thompson G (1977) Cost operator algorithms for the transportation problem. Math Program 12:372–391
56. Tsai W, Fu K (1979) Error-correcting isomorphism of attributed relational graphs for pattern analysis. IEEE Trans Syst Man Cybern (Part B) 9(12):757–768
57. Tsai W, Fu K (1983) Subgraph error-correcting isomorphisms for syntactic pattern recognition. IEEE Trans Syst Man Cybern (Part B) 13:48–61
58. Wagner RA, Fischer MJ (1974) The string-to-string correction problem. J Assoc Comput Mach 21(1):168–173