Image and Vision Computing 27 (2009) 950–959

Contents lists available at ScienceDirect

Image and Vision Computing

journal homepage: www.elsevier.com/locate/imavis

Approximate graph edit distance computation by means of bipartite graph matching

Kaspar Riesen *, Horst Bunke

Institute of Computer Science and Applied Mathematics, University of Bern, Neubrückstrasse 10, CH-3012 Bern, Switzerland

A R T I C L E   I N F O

Article history:
Received 15 October 2007
Received in revised form 1 February 2008
Accepted 9 April 2008

Keywords:
Graph based representation
Graph edit distance
Bipartite graph matching

A B S T R A C T

In recent years, the use of graph based object representation has gained popularity. Simultaneously, graph edit distance emerged as a powerful and flexible graph matching paradigm that can be used to address different tasks in pattern recognition, machine learning, and data mining. The key advantages of graph edit distance are its high degree of flexibility, which makes it applicable to any type of graph, and the fact that one can integrate domain specific knowledge about object similarity by means of specific edit cost functions. Its computational complexity, however, is exponential in the number of nodes of the involved graphs. Consequently, exact graph edit distance is feasible for graphs of rather small size only. In the present paper we introduce a novel algorithm which allows us to approximately, or suboptimally, compute edit distance in a substantially faster way. The proposed algorithm considers only local, rather than global, edge structure during the optimization process. In experiments on different datasets we demonstrate a substantial speed-up of our proposed method over two reference systems. Moreover, it is emprically verified that the accuracy of the suboptimal distance remains sufficiently accurate for various pattern recognition applications.

1. Introduction

Graph matching refers to the process of evaluating the structural similarity of graphs. A large number of methods for graph matching have been proposed in recent years [1]. The main advantage of a description of patterns by graphs instead of vectors is that graphs allow for a more powerful representation of structural relations. In the most general case, nodes and edges are labeled with arbitrary attributes. One of the most flexible methods for error-tolerant graph matching that is applicable to various kinds of graphs is based on the edit distance of graphs [2,3]. The idea of graph edit distance is to define the dissimilarity of graphs by the amount of distortion that is needed to transform one graph into another. Using the edit distance, an input graph to be classified can be analyzed by computing its dissimilarity to a number of training graphs. For classification, the resulting distance values may be fed, for instance, into a nearest-neighbor classifier. Alternatively, the edit distance of graphs can also be interpreted as a pattern similarity measure in the context of kernel machines, which makes a large number of powerful methods applicable to graphs [4], including support vector machines for classification and kernel principal component analysis for pattern analysis. There are various applications where the edit distance has proved to be suitable for error-tolerant graph matching [5,6].

Yet, the error-tolerant nature of edit distance potentially allows every node of a graph to be mapped to every node of another graph (unlike exact graph matching methods such as subgraph isomorphism or maximum common subgraph). The time and space complexity of edit distance computation is therefore very high. Consequently, the edit distance can be computed for graphs of a rather small size only.

In recent years, a number of methods addressing the high computational complexity of graph edit distance computation have been proposed. In some approaches, the basic idea is to perform a local search to solve the graph matching problem, that is, to optimize local criteria instead of global, or optimal ones [7,8]. In [9], a linear programming method for computing the edit distance of graphs with unlabeled edges is proposed. The method can be used to derive lower and upper edit distance bounds in polynomial time. Two fast but suboptimal algorithms for graph edit distance computation are proposed in [10]. The authors propose simple variants of an optimal edit distance algorithm that make the computation substantially faster. A number of graph matching methods based on genetic algorithms have been proposed [11]. Genetic algorithms offer an efficient way to cope with large search spaces, but are non-deterministic and suboptimal. A common way to make graph matching more efficient is to restrict considerations to special classes of graphs. Examples include the classes of planar graphs [12], bounded-valence graphs [13], trees [14], and graphs with unique vertex labels [15]. Recently, a suboptimal edit distance algorithm has been proposed [5] that requires the nodes of graphs to be

* Corresponding author. Tel./fax: +41 316318699.
E-mail addresses: riesen@iam.unibe.ch (K. Riesen), bunke@iam.unibe.ch (H. Bunke).

planarly embedded, which is satisfied in many, but not all computer vision applications of graph matching.

In this paper, we propose a new efficient algorithm for edit distance computation for general graphs. The method is based on an (optimal) fast bipartite optimization procedure mapping nodes and their local structure of one graph to nodes and their local structure of another graph. This procedure is somewhat similar in spirit to the method proposed in [16,17]. However, rather than using dynamic programming for finding an optimal match between the sets of local structure, we make use of Munkres' algorithm [18]. Originally, this algorithm has been proposed to solve the assignment problem in polynomial time. However, in the present paper we generalize the original algorithm to the computation of graph edit distance. In experiments on semi-artificial and real-world data we demonstrate that the proposed method results in a substantial speed-up of the computation of graph edit distance, while at the same time the accuracy of the approximated distances is not much affected.

A preliminary version of the current paper appeared in [19]. The current paper has been significantly extended with respect to the underlying methodology and the experimental evaluation.

2. Graph edit distance computation

In this section we first define our basic notation and then introduce graph edit distance and its computation. Let L be a finite or infinite set of labels for nodes and edges.

Definition 1. (Graph) 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,
- $\mu : V \rightarrow L$ is the node labeling function, and
- $\nu : E \rightarrow L$ is the edge labeling function.

This definition allows us to handle arbitrary graphs with unconstrained labeling functions. For example, the labels can be given by the set of integers, the vector space $\mathbb{R}^n$, or a set of symbolic labels $L = \{\alpha, \beta, \gamma, \ldots\}$. Moreover, unlabeled graphs are obtained as a special case by assigning the same label l to all nodes and edges. Edges are given by pairs of nodes (u, v), where $u \in V$ denotes the source node and $v \in V$ the target node of a directed edge. Undirected graphs can be modeled by inserting a reverse edge $(v, u) \in E$ for each edge $(u, v) \in E$ with $\nu(u, v) = \nu(v, u)$.

Graph matching refers to the task of evaluating the dissimilarity of graphs. One of the most flexible methods for measuring the dissimilarity of graphs is the edit distance [2,3]. Originally, the edit distance has been proposed in the context of string matching [20]. Procedures for edit distance computation aim at deriving a dissimilarity measure from the number of distortions one has to apply to transform one pattern into the other. The concept of edit distance has been extended from strings to trees and eventually to graphs [2,3]. Similarly to string edit distance, the key idea of graph edit distance is to define the dissimilarity, or distance, of graphs by the minimum amount of distortion that is needed to transform one graph into another. Compared to other approaches to graph matching, graph edit distance is known to be very flexible since it can handle arbitrary graphs and any type of node and edge labels. Furthermore, by defining costs for edit operations, the concept of edit distance can be tailored to specific applications.

A standard set of distortion operations is given by insertions, deletions, and substitutions of both nodes and edges. We denote the substitution of two nodes u and v by $(u \rightarrow v)$, the deletion of node u by $(u \rightarrow \varepsilon)$, and the insertion of node v by $(\varepsilon \rightarrow v)$. For edges we use a similar notation. Other operations, such as merging and splitting of nodes [21], can be useful in certain applications but are not considered in this paper. Given two graphs, the source graph g_1 and the target graph g_2, the idea of graph edit distance is to delete some nodes and edges from g_1, relabel (substitute) some of the remaining nodes and edges, and insert some nodes and edges in g_2, such that g_1 is finally transformed into g_2. A sequence of edit operations $e_1, \ldots, e_k$ that transform g_1 completely into g_2 is called an edit path between g_1 and g_2. In Fig. 1 an example of an edit path between two graphs g_1 and g_2 is given. This edit path consists of three edge deletions, one node deletion, one node insertion, two edge insertions, and two node substitutions.

Obviously, for every pair of graphs (g_1, g_2), there exist a number of different edit paths transforming g_1 into g_2. Let $\Upsilon(g_1, g_2)$ denote the set of all such edit paths. To find the most suitable edit path out of $\Upsilon(g_1, g_2)$, one introduces a cost for each edit operation, measuring the strength of the corresponding operation. The idea of such a cost function is to define whether or not an edit operation represents a strong modification of the graph. Obviously, the cost function is defined with respect to the underlying node or edge labels.

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 costs is needed. Consequently, the edit distance of two graphs is defined by the minimum cost edit path between two graphs.

Definition 2. (Graph Edit Distance) Let $g_1 = (V_1, E_1, \mu_1, \nu_1)$ be the source and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ the target graph. The graph edit distance between g_1 and g_2 is defined by

$$d(g_1, g_2) = \min_{(e_1, \ldots, e_k) \in \Upsilon(g_1, g_2)} \sum_{i=1}^{k} c(e_i),$$

where $\Upsilon(g_1, g_2)$ denotes the set of edit paths transforming g_1 into g_2, and c denotes the cost function measuring the strength $c(e_i)$ of edit operation e_i.

The computation of the edit distance is usually carried out by means of a tree search algorithm which explores the space of all possible mappings of the nodes and edges of the first graph to the nodes and edges of the second graph. A widely used method is based on the A* algorithm [22] which is a best-first search algorithm. The basic idea is to organize the underlying search space as an ordered tree. The root node of the search tree represents the starting point of our search procedure, inner nodes of the search tree correspond to partial solutions, and leaf nodes represent complete – not necessarily optimal – solutions. Such a search tree is constructed dynamically at runtime by iteratively creating successor nodes linked by edges to the currently considered node in the search tree. In order to determine the most promising node in the current search tree, i.e. the node which will be used for further expansion of the desired mapping in the next iteration, a heuristic function is usually used. Formally, for a node p in the search tree, we use $g(p)$ to denote the cost of the optimal path from the root node to the current node p, i.e. $g(p)$ is set equal to the cost of the partial edit path accumulated so far, and we use $h(p)$ for denoting

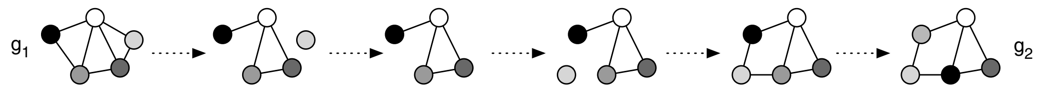

Fig. 1. A possible edit path between graph g_1 and g_2 (node labels are represented by different shades of grey).

the estimated cost from p to a leaf node. The sum $g(p) + h(p)$ gives the total cost assigned to an open node in the search tree. One can show that, given that the estimation of the future costs $h(p)$ is lower than, or equal to, the real costs, the algorithm is admissible, i.e. an optimal path from the root node to a leaf node is guaranteed to be found [22].

Algorithm 1. Graph edit distance algorithm

```
Input:  Non-empty graphs $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and
        $g_2 = (V_2, E_2, \mu_2 \nu_2)$, where $V_1 = \{u_1, \ldots, u_{|V_1|}\}$ and
        $V_2 = \{v_1, \ldots, v_{|V_2|}\}$
Output: A minimum cost edit path from $g_1$ to $g_2$
        e.g. $p_{\min} = \{u_1 \to v_3, u_2 \to \varepsilon, \ldots, \varepsilon \to v_2\}$
 1: initialize OPEN to the empty set {}
 2: For each node $w \in V_2$, insert the substitution $\{u_1 \to w\}$ into OPEN
 3: Insert the deletion $\{u_1 \to \varepsilon\}$ into OPEN
 4: loop
 5:   Remove $p_{\min} = \text{argmin}_{p \in OPEN}\{g(p) + h(p)\}$ from OPEN
 6:   if $p_{\min}$ is a complete edit path then
 7:     Return $p_{\min}$ as the solution
 8:   else
 9:     Let $p_{\min} = \{u_1 \to v_{i1}, \cdots, u_k \to v_{ik}\}$
10:     if $k < |V_1|$ then
11:       For each $w \in V_2 \setminus \{v_{i1}, \cdots, v_{ik}\}$, insert $p_{\min} \cup \{u_{k+1} \to w\}$
          into OPEN
12:       Insert $p_{\min} \cup \{u_{k+1} \to \varepsilon\}$ into OPEN
13:     else
14:       Insert $p_{\min} \cup \bigcup_{w \in V_2 \setminus \{v_{i1}, \cdots, v_{ik}\}} \{\varepsilon \to w\}$ into OPEN
15:     end if
16:   end if
17: end loop
```

In Algorithm 1 the A*-based method for optimal graph edit distance computation is given. The nodes of the source graph are processed in the order $(u_1, u_2, \ldots)$. The deletion (line 12) or the substitution of a node (line 11) are considered simultaneously, which produces a number of successor nodes in the search tree. If all nodes of the first graph have been processed, the remaining nodes of the second graph are inserted in a single step (line 14). The set *OPEN* of partial edit paths contains the search tree nodes to be processed in the next steps. The most promising partial edit path $p \in OPEN$, i.e. the one that minimizes $g(p) + h(p)$, is always chosen first (line 5). This procedure guarantees that the complete edit path found by the algorithm first is always optimal, i.e. has minimal costs among all possible competing paths (line 7).

Note that edit operations on edges are implied by edit operations on their adjacent nodes, i.e. whether an edge is substituted, deleted, or inserted, depends on the edit operations performed on its adjacent nodes. Formally, let $u, u' \in V_1 \cup \{\varepsilon\}$ and $v, v' \in V_2 \cup \{\varepsilon\}$, and assume that the two node operations $(u \to v)$ and $(u' \to v')$ have been executed. We distinguish three cases.

(1) Assume there are edges $e_1 = (u, u') \in E_1$ and $e_2 = (v, v') \in E_2$ in the corresponding graphs g_1 and g_2. Then the edge substitution $(e_1 \to e_2)$ is implied by the node operations given above.

(2) Assume there is an edge $e_1 = (u, u') \in E_1$ but there is no edge $e_2 = (v, v') \in E_2$. Then the edge deletion $(e_1 \to \varepsilon)$ is implied by the node operations given above. Obviously, if $v = \varepsilon$ or $v' = \varepsilon$ there cannot be any edge $(v, v') \in E_2$ and thus an edge deletion $(e_1 \to \varepsilon)$ has to be performed.

(3) Assume there is no edge $e_1 = (u, u') \in E_1$ but an edge $e_2 = (v, v') \in E_2$. Then the edge insertion $(\varepsilon \to e_2)$ is implied by the node operations given above. Obviously, if $u = \varepsilon$ or $u' = \varepsilon$ there cannot be any edge $(u, u') \in E_1$. Consequently, an edge insertion $(\varepsilon \to e_2)$ has to be performed.

Obviously, the implied edge operations can be derived from every partial or complete edit path during the search procedure given in Algorithm 1. The costs of these implied edge operations are dynamically added to the corresponding paths in *OPEN*.

In order to integrate more knowledge about partial solutions in the search tree, it has been proposed to use heuristics [22]. Basically, such heuristics for a tree search algorithm aim at the estimation of a lower bound $h(p)$ of the future costs. In the simplest scenario this lower bound estimation $h(p)$ for the current node p is set to zero for all p, which is equivalent to using no heuristic information about the present situation at all. The other extreme would be to compute for a partial edit path the actual optimal path to a leaf node, i.e. perform a complete edit distance computation for each node of the search tree. In this case, the function $h(p)$ is not a lower bound, but the exact value of the optimal costs. Of course, the computation of such a perfect heuristic is both unreasonable and untractable. Somewhere in between the two extremes, one can define a function $h(p)$ evaluating how many edit operations have to be performed in a complete edit path at least [2]. One possible function of this type is described in the next paragraph.

Let us assume that a partial edit path at a position in the search tree is given, and let the number of unprocessed nodes of the first graph g_1 and second graph g_2 be n_1 and n_2, respectively. For an efficient estimation of the remaining optimal edit operations, we first attempt to perform as many node substitutions as possible. To this end, we potentially substitute each of the n_1 nodes from g_1 with any of the n_2 nodes from g_2. To obtain a lower bound of the exact edit cost, we accumulate the costs of the $\min\{n_1, n_2\}$ least expensive of these node substitutions, and the costs of $\max\{0, n_1 - n_2\}$ node deletions and $\max\{0, n_2 - n_1\}$ node insertions. Any of the selected substitutions that is more expensive than a deletion followed by an insertion operation is replaced by the latter. The unprocessed edges of both graphs are handled analogously. Obviously, this procedure allows multiple substitutions involving the same node or edge and, therefore, it possibly represents an invalid way to edit the remaining part of g_1 into the remaining part of g_2. However, the estimated cost certainly constitutes a lower bound of the exact cost and thus an optimal edit path is guaranteed to be found [22]. In the following, we refer to this method as Heuristic-A*.

3. Fast suboptimal edit distance algorithms

The method described in the previous section finds an optimal edit path between two graphs. Unfortunately, the computational complexity of the edit distance algorithm, whether or not a heuristic function $h(p)$ is used to govern the tree traversal process, is exponential in the number of nodes of the involved graphs. This means that the running time and space complexity may be huge even for reasonably small graphs. In order to cope better with the problem of graph distance, in [16,17] a suboptimal method has been proposed. The basic idea is to decompose graphs into sets of subgraphs. These subgraphs consist of a node and its adjacent structure, i.e. edges including their nodes. The graph matching problem is then reduced to the problem of finding an optimal match between the sets of subgraphs by means of dynamic programming.

In the present paper we introduce a novel algorithm for solving the problem of graph edit distance computation. This approach is somewhat similar to the method described in [16,17], i.e. we approximate the graph edit distance by finding an optimal match between nodes and their local structure of two graphs. However, we use a bipartite matching procedure to find the optimal match rather than dynamic programming. Because of its suboptimal nature, this method does not generally return the optimal edit path, but only an approximate one.

3.1. The assignment problem

Our novel approach for graph edit distance computation is based on the assignment problem. The assignment problem considers the task of finding an optimal assignment of the elements of a set A to the elements of a set B, where A and B have the same cardinality. Assuming that numerical costs are given for each assignment pair, an optimal assignment is one which minimizes the sum of the assignment costs. Formally, the assignment problem can be defined as follows.

Definition 3. (The Assignment Problem) Let us assume there are two sets A and B together with an $n \times n$ cost matrix $\mathbf{C}$ of real numbers given, where $|A| = |B| = n$. The matrix elements $\mathbf{C}_{ij}$ correspond to the costs of assigning the i-th element of A to the j-th element of B. The assignment problem can be stated as finding a permutation $p = p_1, \ldots, p_n$ of the integers $1, 2, \ldots, n$ that minimizes $\sum_{i=1}^{n} \mathbf{C}_{ip_i}$.

The assignment problem can be reformulated as finding an optimal matching in a complete bipartite graph and is therefore also referred to as bipartite graph matching problem. Solving the assignment problem in a brute force manner by enumerating all permutations and selecting the one that minimizes the objective function leads to an exponential complexity which is unreasonable, of course. However, there exists an algorithm which is known as Munkres' algorithm [18][1] that solves the bipartite matching problem in polynomial time. In Algorithm 2 Munkres' method is described in detail. The assignment cost matrix $\mathbf{C}$ given in Definition 3 is the algorithms' input, and the output corresponds to the optimal permutation, i.e. the assignment pairs resulting in the minimum cost. In the description of Munkres' method in Algorithm 2 some lines (rows or columns) of the cost matrix $\mathbf{C}$ and some zero elements are distinguished. They are termed covered or uncovered lines and starred or primed zeros, respectively.

Algorithm 2. Munkres' algorithm for the assignment problem

```
Input:   A cost matrix $C$ with dimensionality $n$
Output:  The minimum cost node or edge assignment
 1: For each row $r$ in $C$, subtract its smallest element from every element in $r$
 2: For each column $c$ in $C$, subtract its smallest element from every element in $c$
 3: For all zeros $z_i$ in $C$, mark $z_i$ with a star if there is no starred zero in its row or column
 4: STEP 1:
 5: for Each column containing a starred zero do
 6:    cover this column
 7: end for
 8: if $n$ columns are covered then GOTO DONE else GOTO STEP 2 end if
 9: STEP 2:
10: if $C$ contains an uncovered zero then
11:    Find an arbitrary uncovered zero $Z_0$ and prime it
12:    if There is no starred zero in the row of $Z_0$ then
13:       GOTO STEP 3
14:    else
15:       Cover this row, and uncover the column containing the starred zero GOTO STEP 2
16:    end if
17: else
18:    Save the smallest uncovered element $e_{\min}$ GOTO STEP 4
19: end if
20: STEP 3: Construct a series $S$ of alternating primed and starred zeros as follows:
21: Insert $Z_0$ into $S$
22: while In the column of $Z_0$ exists a starred zero $Z_1$ do
23:    Insert $Z_1$ into $S$
24:    Replace $Z_0$ with the primed zero in the row of $Z_1$. Insert $Z_0$ into $S$
25: end while
26: Unstar each starred zero in $S$ and replace all primes with stars. Erase all other primes and uncover every line in $C$ GOTO STEP 1
27: STEP 4: Add $e_{\min}$ to every element in covered rows and subtract it from every element in uncovered columns. GOTO STEP 2
28: DONE: Assignment pairs are indicated by the positions of starred zeros in the cost matrix
```

Munkres' algorithm is based on the following theorem.

Theorem 1. (Equivalent Matrices) Given a cost matrix $\mathbf{C}$ as defined in Definition 3, a column vector $\mathbf{c} = (c_1, \ldots, c_n)$, and a row vector $\mathbf{r} = (r_1, \ldots, r_n)$, the square matrix $\mathbf{C}'$ with the elements $\mathbf{C}'_{ij} = \mathbf{C}_{ij} - c_i - r_j$ has the same optimal assignment solution as the matrix $\mathbf{C}$. $\mathbf{C}$ and $\mathbf{C}'$ are said to be equivalent.

Proof. [24] Let p be a permutation of the integers $1, 2, \ldots, n$ minimizing $\sum_{i=1}^{n} \mathbf{C}_{ip_i}$, then

$$\sum_{i=1}^{n} \mathbf{C}'_{ip_i} = \sum_{i=1}^{n} \mathbf{C}_{ip_i} - \sum_{i=1}^{n} c_i - \sum_{j=1}^{n} r_j$$

The values of the last two terms are independent of permutation p so that if p minimizes $\sum_{i=1}^{n} \mathbf{C}_{ip_i}$, it also minimizes $\sum_{i=1}^{n} \mathbf{C}'_{ip_i}$.

Consequently, if we find a new matrix $\mathbf{C}'$ equivalent to the initial cost matrix $\mathbf{C}$, and a permutation p with all $\mathbf{C}'_{ip_i} = 0$, then p also minimizes $\sum_{i=1}^{n} \mathbf{C}_{ip_i}$. Intuitively, Munkres' algorithm transforms the original cost matrix $\mathbf{C}$ into an equivalent matrix $\mathbf{C}'$ having n independent zero elements. This independent set of zero elements exactly corresponds to the optimal assignment pairs.

The operations executed in lines 1 and 2, and STEP 4 of Algorithm 2 find a matrix equivalent to the initial cost matrix (Theorem 1). In lines 1 and 2 the column vector $\mathbf{c} = (c_1, \ldots, c_n)$ is constructed by $c_i = \min\{\mathbf{C_{ij}}\}_{j=1,\ldots,n}$, and the row vector $\mathbf{r} = (r_1, \ldots, r_n)'$ by $r_j = \min\{\mathbf{C}_{ij}\}_{i=1,\ldots,n}$. In STEP 4 the vectors $\mathbf{c}$ and $\mathbf{r}$ are defined by the rules

$$c_i = \begin{cases} e_{\min} & \text{if row } i \text{ is covered} \\ 0 & \text{otherwise} \end{cases} \qquad r_j = \begin{cases} 0 & \text{if column } j \text{ is covered} \\ e_{\min} & \text{otherwise} \end{cases}$$

where $e_{\min}$ is the smallest uncovered element in the cost matrix $\mathbf{C}$.

STEP 1, 2, and 3 of Algorithm 2 are procedures to find a maximum set of independent zeros which mark the optimal assignment. In the worst case the maximum number of operations needed by the algorithm is $O(n^3)$. Note that the $O(n^3)$ complexity is much smaller than the $O(n!)$ complexity required by a brute force algorithm.

3.2. Graph edit distance computation by means of Munkres' algorithm

Munkres' algorithm as introduced in the last section provides us with an optimal solution to the assignment problem in $O(n^3)$ time.

[1] Munkres' algorithm is a refinement of an earlier version by Kuhn [23] and is also referred to as Kuhn-Munkres, or Hungarian algorithm.

The same algorithm can be used to derive a suboptimal solution to the graph edit distance problem as described below.

Let us assume a source graph $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and a target graph $g_2 = (V_2, E_2, \mu_2, \nu_2)$ of equal size, i.e. $|V_1| = |V_2|$, are given. One can use Munkres' algorithm in order to map the nodes of V_1 to the nodes of V_2 such that the resulting node substitution costs are minimal, i.e. we solve the assignment problem of Definition 3 with $A = V_1$ and $B = V_2$. In our solution we define the cost matrix $\mathbf{C}$ such that entry $\mathbf{C}_{i,j}$ corresponds to the cost of substituting the i-th node of V_1 with the j-th node of V_2. Formally, $\mathbf{C}_{i,j} = c(u_i \rightarrow v_j)$, where $u_i \in V_1$ and $v_j \in V_2$, for $i \cdot j = 1, \ldots, |V_1|$.

The constraint that both graphs to be matched are of equal size is too restrictive since it cannot be expected that all graphs in a specific problem domain always have the same number of nodes. However, Munkres' algorithm can be applied to non-quadratic matrices, i.e. to sets with unequal size, as well [24]. In a preliminary version of the present paper Munkres' algorithm has been applied to rectangular cost matrices [19]. In this approach Munkres' algorithm first finds the $\min\{|V_1|, |V_2|\}$ node substitutions which minimize the total costs. Then the costs of $\max\{0, |V_1| - |V_2|\}$ node deletions and $\max\{0, |V_2| - |V_1|\}$ node insertions are added to the minimum cost node assignment such that all nodes of both graphs are processed. Using this approach, all nodes of the smaller graph are substituted and the nodes remaining in the larger graph are either deleted (if they belong to g_1) or inserted (if they belong to g_2).

In the present paper we extend the idea proposed in [19] by defining a new cost matrix $\mathbf{C}$ which is more general in the sense that we allow insertions or deletions to occur not only in the larger, but also in the smaller of the two graphs under consideration. In contrast with the scenario of [19], this new setting now perfectly reflects the edit model of Definition 2. Moreover, matrix $\mathbf{C}$ is by definition quadratic. Consequently, the original method as described in Algorithm 2 can be used to find the minimum cost assignment.

Definition 4. (Cost Matrix) Let $g_1 = (V_1, E_1, \mu_1, \nu_1)$ be the source and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ be the target graph with $V_1 = \{u_1, \ldots, u_n\}$ and $V_2 = \{v_1, \ldots, v_m\}$, respectively. The cost matrix $\mathbf{C}$ is defined as

$$\mathbf{C} = \left[\begin{array}{cccc|cccc}
c_{1,1} & c_{1,2} & \cdots & c_{1,m} & c_{1,\varepsilon} & \infty & \cdots & \infty \\
c_{2,1} & c_{2,2} & \cdots & c_{2,m} & \infty & c_{2,\varepsilon} & \ddots & \vdots \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \ddots & \infty \\
c_{n,1} & c_{n,2} & \cdots & c_{n,m} & \infty & \cdots & \infty & c_{n,\varepsilon} \\
\hline
c_{\varepsilon,1} & \infty & \cdots & \infty & 0 & 0 & \cdots & 0 \\
\infty & c_{\varepsilon,2} & \ddots & \vdots & 0 & 0 & \ddots & \vdots \\
\vdots & \ddots & \ddots & \infty & \vdots & \ddots & \ddots & 0 \\
\infty & \cdots & \infty & c_{\varepsilon,m} & 0 & \cdots & 0 & 0
\end{array}\right]$$

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

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

On the basis of the new cost matrix $\mathbf{C}$ defined above, Munkres' algorithm [18] can be executed (Algorithm 2). This algorithm finds the optimal, i.e. the minimum cost, permutation $p = p_1, \ldots, p_{n+m}$ of the integers $1, 2, \ldots, n+m$ that minimizes $\sum_{i=1}^{n+m} \mathbf{C}_{ip_i}$. Obviously, this is equivalent to the minimum cost assignment of the nodes of g_1 represented by the rows to the nodes of g_2 represented by the columns of matrix $\mathbf{C}$. That is, Munkres' algorithm indicates the minimum cost assignment pairs with starred zeros in the transformed cost matrix $\mathbf{C}'$. These starred zeros are independent, i.e. each row and each column of $\mathbf{C}'$ contains exactly one starred zero. Consequently, each node of graph g_1 is either uniquely assigned to a node of g_2 (left upper corner of $\mathbf{C}'$), or to the deletion node ε (right upper corner of $\mathbf{C}'$). Vice versa, each node of graph g_2 is either uniquely assigned to a node of g_1 (left upper corner of $\mathbf{C}'$), or to the insertion node ε (bottom left corner of $\mathbf{C}'$). The ε-nodes in g_1 and g_2 corresponding to rows $n+1, \ldots, n+m$ and columns $m+1, \ldots, m+n$ in $\mathbf{C}'$ that are not used cancel each other out without any costs (bottom right corner of $\mathbf{C}'$).

So far the proposed algorithm considers the nodes only and takes no information about the edges into account. In order to achieve a better approximation of the true edit distance, it would be highly desirable to involve edge operations and their costs in the node assignment process as well. In order to achieve this goal, an extension of the cost matrix is needed. To each entry $c_{i,j}$, i.e. to each cost of a node substitution $c(u_i \rightarrow v_j)$, the minimum sum of edge edit operation costs, implied by node substitution $u_i \rightarrow v_j$, is added. Formally, assume that node u_i has adjacent edges E_{ui} and node v_j has adjacent edges E_{vj}. With these two sets of edges, E_{ui} and E_{vj}, an individual cost matrix similarly to Definition 4 can be established and an optimal assignment of the elements E_{ui} to the elements E_{vj} according to Algorithm 2 can be performed. Clearly, this procedure leads to the minimum sum of edge edit costs implied by the given node substitution $u_i \rightarrow v_j$. These edge edit costs are added to the entry $c_{i,j}$. Clearly, to the entry $c_{i,\varepsilon}$, which denotes the cost of a node deletion, the cost of the deletion of all adjacent edges of u_i is added, and to the entry $c_{\varepsilon,j}$, which denotes the cost of a node insertion, the cost of all insertions of the adjacent edges of v_j is added.

Note that Munkres' algorithm used in its original form is optimal for solving the assignment problem, but it provides us with a suboptimal solution for the graph edit distance problem only. This is due to the fact that each node edit operation is considered individually (considering the local structure only), such that no implied operations on the edges can be inferred dynamically. The result returned by Munkres' algorithm corresponds to the minimum cost mapping, according to matrix $\mathbf{C}$, of the nodes of g_1 to the nodes of g_2. Given this mapping, the implied edit operations of the edges are inferred, and the accumulated costs of the individual edit operations on both nodes and edges can be computed. These costs serve us as an approximate graph edit distance. The approximate edit distance values obtained by this procedure are equal to, or larger than, the exact distance values, since our suboptimal approach finds an optimal solution in a subspace of the complete search space. In the following we refer to our novel suboptimal graph edit distance algorithm as Bipartite, or BP for short.

4. Experimental results

The purpose of the experiments presented in this section is to empirically verify the feasibility of the proposed suboptimal graph edit distance algorithm. First of all, we compare the runtime of suboptimal graph edit distance with the optimal version. Secondly, we aim at answering the question whether the resulting suboptimal graph edit distances remain sufficiently accurate for pattern recognition tasks. To this end we consider several classification tasks. There are various approaches to graph classification that make use of graph edit distance in some form, including vector space embedding classifiers [25] and graph kernels [26]. In the present paper we make use of a 1-NN classifier to assess the quality of the resulting edit distances because it directly uses these distances without any additional classifier training. Obviously, if a suboptimal algorithm leads to edit distances with poor quality, the classification accuracy of the 1-NN classifier is expected to decrease.

For comparison we use two reference systems to compute graph edit distance. The first reference system is given by the optimal tree search algorithm (Heuristic-A*) described in Section 2, while the second reference system is a modification of Heuristic-A* [10]. Instead of expanding all successor nodes in the search tree, only a fixed number s of nodes to be processed are kept in the *OPEN* set at all times. Whenever a new partial edit path is added to *OPEN* in Algorithm 1, only the s partial edit paths p with the lowest costs $g(p) + h(p)$ are kept, and the remaining partial edit paths in *OPEN* are removed. Obviously, this procedure, which is also known as beam search, corresponds to a pruning process of the search tree during the search procedure. It explores not the full search space, but only those nodes are expanded that belong to the most promising partial matches. We refer to this suboptimal method as Beam(s). Obviously there is a trade-off between running time and suboptimality of the edit distances implied by parameter s.

4.1. Data sets

The classification tasks considered in this paper include the recognition of line drawings (letters and symbols), images, fingerprints, molecules, and proteins involving a total of seven different graph datasets.

4.1.1. Letter database

The first graph dataset used in our experiments involves graphs that represent distorted letter drawings. We consider the 15 capital letters of the Roman alphabet that consist of straight lines only (*A, E, F, H, I, K, L, M, N, T, V, W, X, Y, Z*). For each class, a prototype line drawing is manually constructed. These prototype drawings are then converted into prototype graphs by representing lines by edges and ending points of lines by nodes. Each node is labeled with a two-dimensional attribute giving its position. To obtain large sample sets of graphs with arbitrarily strong distortions, distortion operators are applied to the prototype graphs. This results in randomly translated, deleted, and inserted nodes and edges.

The graph database used in our experiments consists of a training set and a validation set of size 750 each, and a test set of size 1500. In order to test the classifier under different conditions, the distortions are applied on the prototype graphs with three different levels of strength, viz. low, medium and high. Hence, our experimental data set comprises 9000 graphs altogether. In Fig. 2 the prototype graph and a graph instance for each distortion level representing the letters *A*, *M*, *K*, and *Z* are illustrated.

4.1.2. COIL-100 database

The COIL-100 database [27] consists of images of 100 different objects. The objects were placed on a motorized turntable against black background. The turntable was rotated through 360° to vary object pose with respect to a fixed color camera. Images of the objects were taken at pose intervals of 5°. This corresponds to 72 poses per image. Fig. 3 shows an example image of each class. To convert images into attributed graphs the images are segmented into regions of homogeneous color using a mean shift algorithm [28]. Segmented images are transformed into region adjacency graphs by representing regions by nodes, labeled with attributes specifying the color histogram of the corresponding segment, and the adjacency of regions by unlabeled edges.

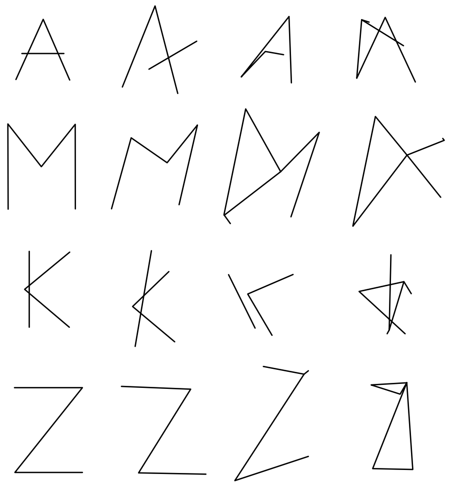
Fig. 2. Instances of letters *A*, *M*, *K*, and *Z*: Original and distortion levels low, medium and high (from left to right).

Fig. 3. COIL images of 100 different objects.

This database consists of 7200 images totally, i.e. 72 images per class. However, for our experiments we make not use of all available images. The training set is composed of 12 images per object, acquired every 30° of rotation. From the remaining images we randomly select five images per object for the validation set, and ten images per object for the test set. This results in a training set of size 1200, a validation set of size 500, and a test set of size 1000.

4.1.3. GREC database

The GREC database used in our experiments consists of a subset of the symbol database underlying the GREC 2005 competition [29]. The images represent symbols from architecture, electronics, and other technical fields. Distortion operators are applied to the original images in order to simulate handwritten symbols. To this end the primitive lines of the symbols are divided into subparts. The ending points of these subparts are then randomly shifted within a certain distance, maintaining connectivity. Each node represents a subpart of a line and is attributed with its relative length (ratio of the length of the actual line to the length of the longest line in the symbol). Connection points of lines are represented by

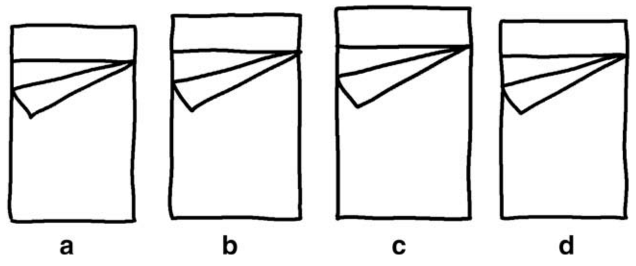

Fig. 4. Four disturbed instances of a technical symbol.

edges attributed with the angle between the corresponding lines. In Fig. 4 four disturbed instances of the same symbol are given. Note that the disturbed images look quite similar at first glance. However, their corresponding graph based representations vary in terms of the number and attribute values of both nodes and edges. Our dataset consists of 20 classes and 30 instances per class (600 graphs totally). We use a training set of size 200, a validation set of size 100, and a test set of size 300.

4.1.4. Fingerprint database

Fingerprints are converted into graphs by filtering the images and extracting regions that are relevant for classification. Next the extracted regions are binarized and a noise removal and thinning procedure is applied [30]. This results in a skeletonized representation of the extracted regions. Ending points and bifurcation points of the skeletonized regions are represented by nodes. Additional nodes are inserted in regular intervals between ending points and bifurcation points. Finally, undirected edges are inserted to link nodes that are directly connected through a ridge in the skeleton. The edges are attributed with an angle denoting the orientation of the edge with respect to the horizontal direction. The procedure of converting fingerprint images into attributed graphs is described in more detail in [31].

The fingerprint database used in our experiments is based on the NIST-4 reference database of fingerprints [32]. It consists of a training set of size 500, a validation set of size 300, and a test set of size 1500. Thus, there are 2300 fingerprint images totally out of the four classes arch, left, right, and whorl from the Galton–Henry classification system [33]. For examples of these fingerprint classes, see Fig. 5.

4.1.5. Molecule database

The molecule dataset consists of graphs representing molecular compounds. We construct graphs from the AIDS Antiviral Screen Database of Active Compounds [34]. Our molecule database consists of two classes (active, inactive), which represent molecules with activity against HIV or not. The molecules are converted into graphs in a straightforward manner by representing atoms as nodes and the covalent bonds as edges. Nodes are labeled with the number of the corresponding chemical symbol and edges by the valence of the linkage. In Fig. 6 some molecular compounds of both classes are illustrated. Note that different shades of grey represent different chemical symbols, i.e. node labels. We use a training and validation set of size 250 each, and a test set of size 1500. Thus, there are 2000 elements totally (1600 inactive elements and 400 active elements).

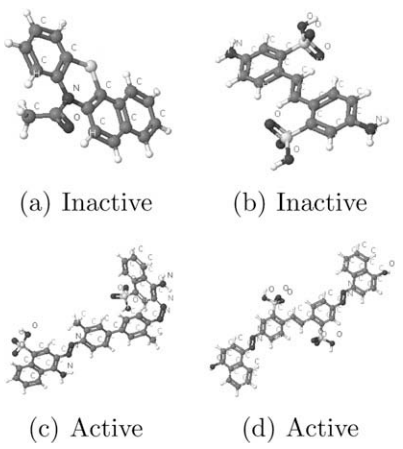

Fig. 6. Some molecule examples of both classes, inactive and active.

4.1.6. Proteins database

The proteins dataset consists of graphs representing proteins. The graphs are constructed from the Protein Data Bank [35] and labeled with their corresponding enzyme class labels from the BRENDA enzyme database [36]. The proteins database consists of six classes (*EC 1, EC 2, EC 3, EC 4, EC 5, EC 6*), which represent proteins out of the six enzyme commission top level hierarchy (EC classes). The proteins are converted into graphs by representing the secondary structure elements of a protein with nodes and edges of an attributed graph. Nodes are labeled with their type (helix, sheet, or loop) and their amino acid sequence (e.g. TFKEVVRLT). Every node is connected with an edge to its three nearest neighbors in space. Edges are labeled with their type and the distance they represent in angstroms. In Fig. 7 six images of proteins of all six classes are given. There are 600 proteins totally, 100 per class. We use a training, validation and test set of equal size (200). The classification task on this dataset consists in predicting the enzyme class membership. To this end, we perform classification as "one-class vs. rest" and repeat this for all six EC top level classes. Consequently, the classification result reported in the next subsection is the average across all EC classes.

Note that the graph datasets used in our experiments are of quite different nature, coming from a variety of applications. Furthermore, the graph sets differ in their characteristics, such as the number of available graphs (|*G*|), the number of different clas-

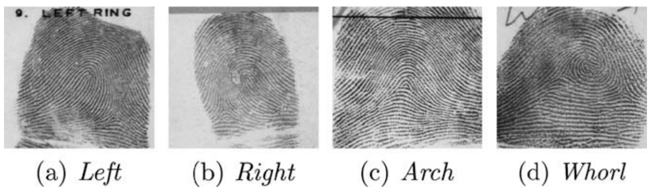

Fig. 5. Fingerprint examples from the four classes.

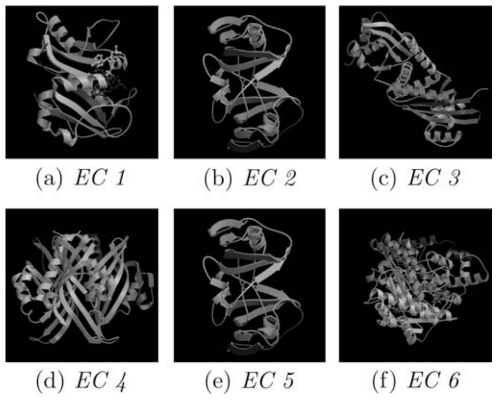

Fig. 7. Protein examples of all top level classes.

Table 1
Graph dataset characteristics

Database	$\vert G\vert$	$\vert\Omega\vert$	$\varnothing\vert V\vert$	$\varnothing\vert E\vert$	Max$\vert V\vert$	Max$\vert E\vert$
Letter (L)	3000	15	4.5	3.1	8	9
Letter (M)	3000	15	4.6	3.3	10	10
Letter (H)	3000	15	4.7	3.4	9	10
COIL	2700	100	3.0	3.0	11	12
GREC	600	20	12.9	18.4	39	59
Fingerprints	2300	4	5.4	4.4	26	24
Molecules	2000	2	9.5	10.0	85	328
Proteins	600	6	32.6	62.1	126	149

ses ($|\Omega|$), and the average and maximum number of nodes and edges per graph ($\varnothing|V|, \varnothing|E|, \max|V|, \max|E|$). In Table 1 a summary of all graph datasets and their corresponding characteristics is given.

4.2. Experimental setup and validation of the meta parameters

In our experiments we divide each database into three disjoint subsets, viz. the training, the validation, and the test set. The elements of the training set are used as prototypes in the NN classifier. The validation set is used to determine the values of the meta parameters τ_{node}, which corresponds to the cost of a node deletion or insertion, and τ_{edge}, which corresponds to the costs of an edge deletion or insertion. For all considered graph data sets node and edge labels are integer numbers, real numbers, real vectors, or strings, and the substitution cost of a pair of labels is given by a suitable distance measure (Euclidean distance or string edit distance [37]).

The validation procedure on the Molecule database is illustrated in Fig. 8. Here the classification results on the validation set are plotted as a function of the two edit cost parameters τ_{node} and τ_{edge}. Finally, the parameter pair that leads to the highest classification accuracy on the validation set is used on the independent test set to perform graph edit distance computation and nearest-neighbor classification.

Besides the computation time (t) and classification accuracy (Acc) we are interested in other indicators. In particular, we compute the correlation (ρ) between exact and suboptimal distances. Clearly, the correlation coefficient between exact and suboptimal distances is an indicator how good a suboptimal method approximates the exact edit distance.

4.3. Results and discussion

In Table 2 the results achieved on all datasets are given. The computation time in Table 2 corresponds to the total time elapsed

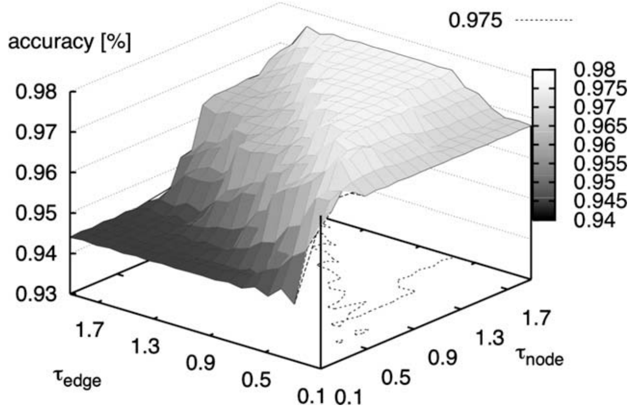

Fig. 8. Validation of edit cost parameters τ_{node} and τ_{edge}.

while performing all graph edit distance computations on a given data set. Missing table entries correspond to cases where a time limit was exceed and the computation was aborted. We observe that exact edit distance computation by means of Heuristic-A* is feasible for the Letter and COIL graphs only. The graphs of the remaining datasets are too complex, large, or dense such that no exact graph edit distances are available.

Comparing the runtime of our suboptimal method, BP, with both reference systems we observe a massive speed-up. On the Letter data at the lowest distortion level, for instance, the novel bipartite edit distance algorithm is about 81 times faster than the exact algorithm (Heuristic-A*) and about 13 times faster than the second reference system (Beam). On the Letter graphs at medium and high distortion level the corresponding speed-ups of our novel algorithm are even higher. On the COIL data our algorithm runs eight times faster than Heuristic-A* and six times faster than Beam. On the three datasets GREC, Fingerprint, and Molecules exact computation of the edit distance is not possible within reasonable time. The runtime of our novel method on these three datasets is about 152 (GREC), 83 (Fingerprint), and 260 (Molecules) times faster than the other suboptimal algorithm Beam. Moreover, we note that on the last dataset (Proteins) all methods other than BP fail due to the lack of memory.

From Table 2 a significant speed-up of the novel bipartite method for graph edit distance computation compared to the exact procedure is evident. However, the question remains whether the approximate edit distances found by BP are accurate enough for pattern recognition tasks. As mentioned before, the distances found by BP are equal to, or larger than, the exact graph edit distance. In fact, this can be seen in the correlation scatter plots in Fig. 9. These scatter plots give us a visual representation of the accuracy of the suboptimal methods Beam and BP on the Letter data at the lowest distortion level. We plot for each pair consisting of one test and one training graph its exact (horizontal axis) and approximate (vertical axis) distance value.

Based on the scatter plots given in Fig. 9 we find that Beam approximates small distance values accurately, i.e. all small approximate distances are equal to the exact distances. On the other hand, large distance values are overestimated quite strongly. The mean and the standard deviation of the difference between the approximate and exact distances are 0.23 and 0.59, respectively. Based on the fact that graphs within the same class usually have a smaller distance than graphs belonging to two different classes this means that the suboptimality of Beam mainly increases interclass distances, while intra-class distances are not strongly affected.

A similar conclusion can be drawn for our novel suboptimal algorithm BP. Many of the small distance values are not overestimated, while higher distance values are increased due to the suboptimal nature of our novel approach. In contrast with the suboptimal method Beam, where the level of overestimation increases with larger distance values, the distance values are better bounded by BP, i.e. also large distance values are not strongly overestimated. That is, both the mean (0.16) and the standard deviation (0.27) of the difference between the approximate and exact distances are smaller than with Beam. On all other datasets, we obtain scatter plots similar to those in Fig. 9 and can draw the same conclusions.

BP considers the local edge structure of the graphs only. Hence, in comparison with Heuristic-A* our novel algorithm BP might find an optimal node mapping which eventually causes additional edge operations. These additional edge edit operations are often deletions or insertions. This leads to additional costs of a multiple of the edit cost parameter τ_{edge}. Obviously, this explains the accumulation of points in the two line-like areas parallel to the diagonal in the distance scatter plot in Fig. 9b.

Table 2
Accuracy (Acc), correlation (ρ), time (t) on all datasets

Database	Heuristic-A*			Beam(10)			BP		
	Acc	ρ	t	Acc	ρ	t	Acc	ρ	t
Letter (L)	91.0	1.00	649′05″	91.1	0.95	107′52″	91.1	0.98	7′52″
Letter (M)	77.9	1.00	2061′29″	78.5	0.93	120′04″	77.6	0.93	6′11″
Letter (H)	63.0	1.00	4914′45″	63.9	0.93	149′47″	61.6	0.97	8′48″
COIL	93.3	1.00	199′19″	93.3	0.99	156′46″	93.3	0.99	24′27″
GREC	–	–	–	76.7	–	1826′46″	86.3	–	12′11″
Fingerprint	–	–	–	84.6	–	166′05″	78.7	–	2′09″
Molecules	–	–	–	96.2	–	1047′55″	97.0	–	4′17′
Proteins	–	–	–	–	–	–	89.3	–	322′21″

Based on the scatter plots and the high correlation coefficient ρ reported in Table 2, one can presume that the classification results of an NN-classifier will not be negatively affected when we substitute the exact edit distances by approximate ones. In fact, this can be observed in Table 2 for all datasets. On Letter (L) the classification accuracy is improved, while on Letter (M) and Letter (H) it drops compared to the exact algorithm. On the COIL data, all algorithms achieve the same classification result. Note that the only difference that is statistically significant (using a Z-test at the 95% level) is the deterioration on Letter (H) from 63.0% to 61.6%. On the datasets GREC and Molecules we observe that our algorithm outperforms the second reference system (with statistical significance), while on Fingerprints the accuracy drops statistically significantly. On Proteins we have only one result. This result is comparable to results published in [38], where a random walk graph kernel was used.

From the results reported in Table 2, we can conclude that in general the classification accuracy of the 1-NN classifier is not negatively affected by using the approximate rather than the exact edit distances. This is due to the fact that most of the overestimated distances belong to inter-class pairs of graphs, while intra-class distances are not strongly affected. Obviously, intra-class distances are of much higher importance for a distance based classifier than inter-class distances. In other words, through the approximation of the edit distances, the graphs are rearranged with respect to each other such that a better classification becomes possible. Graphs which belong to the same class (according to the ground truth) often remain near, while graphs from different classes are pulled apart from each other. Obviously, if the approximation is too inaccurate, the similarity measure and the underlying classifier will be unfavorably disturbed.

5. Conclusions

In the present paper, we have considered the problem of graph edit distance computation. The edit distance of graphs is a powerful and flexible concept that has found various applications in pattern recognition and related areas. A serious drawback is, however, the exponential complexity of graph edit distance computation. Hence using optimal, or exact, algorithms restricts the applicability of the edit distance to graphs of rather small size.

In the current paper, we propose a suboptimal approach to graph edit distance computation, which is based on Munkres algorithm for solving the assignment problem. The assignment problem consists in finding an assignment of the elements of two sets with each other such that a cost function is minimized. In the current paper we show how the graph edit distance problem can be transformed into the assignment problem. Our proposed solution allows for the insertion, deletion, and substitution of both nodes and edges, but considers these edit operations in a rather independent fashion from each. Therefore, while Munkres algorithm returns the optimal solution to the assignment problem, our proposed solution yields only a suboptimal, or approximate, solution to the graph edit distance problem. However, the time complexity is only cubic in the number of nodes of the two underlying graphs.

In the experimental section of this paper we compute edit distance on seven different graph datasets. The graph based representations of our sets cover line drawings, object and fingerprint images, and molecular compounds. On four of our datasets the exact graph edit distance is not applicable since the graphs are too large. However, the proposed suboptimal method can cope with the task of graph edit distance on all datasets. The first finding of our work is that although Beam, which is another suboptimal algo-

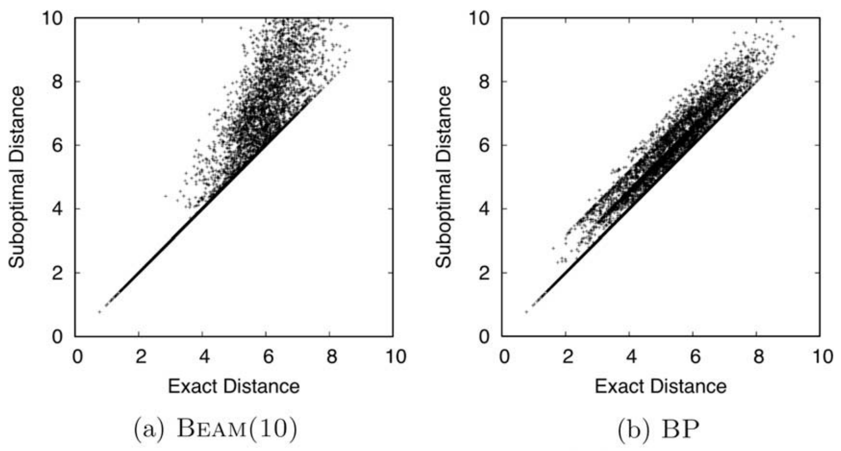

Fig. 9. Scatter plots of the exact edit distances (x-axis) and the suboptimal edit distances (y-axis).

rithm developed previously, achieves remarkable speed-ups compared to the exact method, our novel algorithm is still much faster. Furthermore, our new approach makes graph edit distance feasible for graphs with up to 130 nodes.

The second finding is that suboptimal graph edit distance need not necessarily lead to a deterioration of the classification accuracy of a distance based classifier. On two out of four datasets, the classification accuracy of a nearest-neighbor classifier remains the same or even improves when the exact edit distances are replaced by the suboptimal ones returned by our novel algorithm. This can be explained by the fact that all distances computed by the suboptimal method are equal to, or larger than, the true distances. Moreover, an experimental analysis has shown that the larger the true distances are the larger is their overestimation. In other words, smaller distances are computed more accurately than larger distances by our suboptimal algorithm. This means that inter-class distances are more affected than intra-class distances by the suboptimal algorithm. However, for a nearest-neighbor classifier, small distances have more influence on the decision than large distances. Hence no serious deterioration of the classification accuracy occurs when our novel suboptimal algorithm is used instead of an exact method.

In future work we will study the problem of graph clustering using the proposed graph edit distance method, and integrate the suboptimal distances into more sophisticated graph classifiers-like graph kernels. Also the comparison with other graph matching approaches could be a rewarding avenue to be explored.

References

[1] D. Conte, P. Foggia, C. Sansone, M. Vento, Thirty years of graph matching in pattern recognition, International Journal of Pattern Recognition and Artificial Intelligence 18 (3) (2004) 265–298.
[2] H. Bunke, G. Allermann, Inexact graph matching for structural pattern recognition, Pattern Recognition Letters 1 (1983) 245–253.
[3] A. Sanfeliu, K. Fu, A distance measure between attributed relational graphs for pattern recognition, IEEE Transactions on Systems, Man, and Cybernetics (Part B) 13 (3) (1983) 353–363.
[4] M. Neuhaus, H. Bunke, Edit distance based kernel functions for structural pattern classification, Pattern Recognition 39 (10) (2006) 1852–1863.
[5] M. Neuhaus, H. Bunke, An error-tolerant approximate matching algorithm for attributed planar graphs and its application to fingerprint classification, in: A. Fred, T. Caelli, R. Duin, A. Campilho, D. de Ridder (Eds.), Proceedings of 10th International Workshop on Structural and Syntactic Pattern Recognition, LNCS, 3138, Springer, 2004, pp. 180–189.
[6] A. Robles-Kelly, E. Hancock, Graph edit distance from spectral seriation, IEEE Transactions on Pattern Analysis and Machine Intelligence 27 (3) (2005) 365–378.
[7] M. Boeres, C. Ribeiro, I. Bloch, A randomized heuristic for scene recognition by graph matching, in: C. Ribeiro, S. Martins (Eds.), Proceedings of 3rd Workshop on Efficient and Experimental Algorithms, LNCS, 3059, Springer, 2004, pp. 100–113.
[8] S. Sorlin, C. Solnon, Reactive tabu search for measuring graph similarity, in: L. Brun, M. Vento (Eds.), Proceedings of 5th International Workshop on Graph-based Representations in Pattern Recognition, LNCS, 3434, Springer, 2005, pp. 172–182.
[9] D. Justice, A. Hero, A binary linear programming formulation of the graph edit distance, IEEE Transactions on Pattern Analysis and Machine Intelligence 28 (8) (2006) 1200–1214.
[10] M. Neuhaus, K. Riesen, H. Bunke, Fast suboptimal algorithms for the computation of graph edit distance, in: C. Ribeiro, S. Martins (Eds.), Proceedings of 11th International Workshop on Structural and Syntactic Pattern Recognition, LNCS, 3059, Springer, 2006, pp. 163–172.
[11] A. Cross, R. Wilson, E. Hancock, Inexact graph matching using genetic search, Pattern Recognition 30 (6) (1997) 953–970.
[12] J. Wong, J. Hopcroft, Linear time algorithm for isomorphism of planar graphs, in: Proceedings of 6th Annual ACM Symposium on Theory of Computing, 1974, pp. 172–184.
[13] E. Luks, Isomorphism of graphs of bounded valence can be tested in polynomial time, Journal of Computer and Systems Sciences 25 (1982) 42–65.
[14] A. Torsello, D. Hidovic-Rowe, M. Pelillo, Polynomial-time metrics for attributed trees, IEEE Transactions on Pattern Analysis and Machine Intelligence 27 (7) (2005) 1087–1099.
[15] P. Dickinson, H. Bunke, A. Dadej, M. Kraetzl, On graphs with unique node labels, in: E. Hancock, M. Vento (Eds.), Proceedings of 4th International Workshop on Graph Based Representations in Pattern Recognition, LNCS, 2726, Springer, 2003, pp. 13–23.
[16] M. Eshera, K. Fu, A graph distance measure for image analysis, IEEE Transactions on Systems, Man, and Cybernetics (Part B) 14 (3) (1984) 398–408.
[17] M. Eshera, K. Fu, A similarity measure between attributed relational graphs for image analysis, in: Proceedings of 7th International Conference on Pattern Recognition, 1984, pp. 75–77.
[18] J. Munkres, Algorithms for the assignment and transportation problems, Journal of the Society for Industrial and Applied Mathematics 5 (1957) 32–38.
[19] K. Riesen, M. Neuhaus, H. Bunke, Bipartite graph matching for computing the edit distance of graphs, in: F. Escolano, M. Vento (Eds.), in: Proceedings of 6th International Workshop on Graph Based Representations in Pattern Recognition, LNCS 4538, 2007, pp. 1–12.
[20] R. Wagner, M. Fischer, The string-to-string correction problem, Journal of the Association for Computing Machinery 21 (1) (1974) 168–173.
[21] R. Ambauen, S. Fischer, H. Bunke, Graph edit distance with node splitting and merging and its application to diatom identification, in: E. Hancock, M. Vento (Eds.), Proceedings of 4th International Workshop on Graph Based Representations in Pattern Recognition, LNCS, 2726, Springer, 2003, pp. 95–106.
[22] P. Hart, N. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Transactions of Systems, Science, and Cybernetics 4 (2) (1968) 100–107.
[23] H. Kuhn, The Hungarian method for the assignment problem, Naval Research Logistic Quarterly 2 (1955) 83–97.
[24] F. Bourgeois, J. Lassalle, An extension of the Munkres algorithm for the assignment problem to rectangular matrices, Communications of the ACM 14 (12) (1971) 802–804.
[25] K. Riesen, M. Neuhaus, H. Bunke, Graph embedding in vector spaces by means of prototype selection, in: F. Escolano, M. Vento (Eds.), in: Proceedings of 6th International Workshop on Graph Based Representations in Pattern Recognition, LNCS 4538, 2007, pp. 383–393.
[26] M. Neuhaus, H. Bunke, Bridging the Gap between Graph Edit Distance and Kernel Machines, World Scientific, 2007.
[27] S. Nene, S. Nayar, H. Murase, Columbia object image library: coil-100, Technical Report, Department of Computer Science, Columbia University, New York, 1996.
[28] D. Comaniciu, P. Meer, Robust analysis of feature spaces: color image segmentation, in: IEEE Conference on Computer Vision and Pattern Recognition, 1997, pp. 750–755.
[29] P. Dosch, E. Valveny, Report on the second symbol recognition contest, in: L. Wenyin, J. Llados (Eds.), Graphics Recognition. Ten years review and future perspectives, Proceedings of 6th International Workshop on Graphics Recognition (GREC'05), LNCS, 3926, Springer, 2005, pp. 381–397.
[30] R. Zhou, C. Quek, G. Ng, A novel single-pass thinning algorithm and an effective set of performance criteria, Pattern Recognition Letters 16 (12) (1995) 1267–1275.
[31] M. Neuhaus, H. Bunke, A graph matching based approach to fingerprint classification using directional variance, in: T. Kanade, A. Jain, N. Ratha (Eds.), Proceedings of 5th International Conference on Audio- and Video-based Biometric Person Authentication, LNCS, 3546, Springer, 2005, pp. 191–200.
[32] C. Watson, C. Wilson, NIST Special Database 4, Fingerprint Database, National Institute of Standards and Technology, 1992.
[33] E. Henry, Classification and Uses of Finger Prints, Routledge, London, 1900.
[34] D.T.P. DTP, AIDS antiviral screen, 2004. Available from: <http://dtp.nci.nih.gov/docs/aids/aids-data.html>.
[35] H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. Shidyalov, P. Bourne, The protein data bank, Nucleic Acids Research 28 (2000) 235–242.
[36] I. Schomburg, A. Chang, C. Ebeling, M. Gremse, C. Heldt, G. Huhn, D. abd Schomburg, Brenda, the enzyme database: updates and major new developments, Nucleic Acids Research 32 (2004). Database issue: D431–D433.
[37] V. Levenshtein, Binary codes capable of correcting deletions, insertions and reversals, Soviet Physics Doklady 10 (8) (1966) 707–710.
[38] K. Borgwardt, H.P. Kriegel, Shortest-path kernels on graphs, in: Proceedings of 5th International Conference on Data Mining, 2005, pp. 74–81.