Approximation of Graph Edit Distance Based on Hausdorff Matching

Andreas Fischer[1*], Ching Y. Suen[1*],
Volkmar Frinken[2*], Kaspar Riesen[3*], Horst Bunke[4*]

[1] *Concordia University, Centre for Pattern Recognition and Machine Intelligence, 1455 de Maisonneuve Blvd West, H3G 1M8 Montreal, Canada*

[2] *Universitat Autònoma de Barcelona, Computer Vision Center, Edifici O, 08193 Bellaterra, Barcelona, Spain*

[3] *University of Applied Sciences and Arts Northwestern Switzerland, Riggenbachstrasse 16, 4600 Olten, Switzerland*

[4] *University of Bern, Institute of Computer Science and Applied Mathematics, Neubrückstrasse 10, 3012 Bern, Switzerland*

Abstract

Graph edit distance is a powerful and flexible method for error-tolerant graph matching. Yet it can only be calculated for small graphs in practice due to its exponential time complexity when considering unconstrained graphs. In this paper we propose a quadratic time approximation of graph edit distance based on Hausdorff matching. In a series of experiments we analyze the performance of the proposed Hausdorff edit distance in the context of graph classification and compare it with a cubic time algorithm based on the assignment problem. Investigated applications include nearest neighbor classification of graphs representing letter drawings, fingerprints, and molecular compounds as well as hidden Markov model classification of vector space embedded graphs representing handwriting. In many cases, a substantial speedup is achieved with only a minor loss in accuracy or, in one case, even with a gain in accuracy. Overall, the proposed Hausdorff edit distance shows a promising potential in terms of flexibility, efficiency, and accuracy.

Keywords: graph edit distance, Hausdorff distance, approximation algorithms, graph classification, graph embedding, handwriting recognition

1. Introduction

Due to their ability to represent properties of objects and binary relationships at the same time, graphs have found widespread applications in pattern

*Corresponding author. Phone: +1 514 848-2424 Ext. 7950, Fax: +1 514 848 2830
Email address: andreas.fischer@polymtl.ca (Andreas Fischer[1])

recognition [1, 2, 3]. Graph-based representations have been successfully used in various fields of research including bioinformatics [4], web content mining [5], image classification [6], graphical symbol recognition [7], character recognition [8], and computer network analysis [9] to name just a few.

Yet, after the initial enthusiasm induced by the representational power and flexibility of graphs in the late seventies, a number of problems became evident. First, working with graphs is unequally more challenging than working with feature vectors, as even basic mathematical operations cannot be defined in a standard way, but must be provided depending on the specific application. Secondly, graphs suffer from their own flexibility. For instance, computing the distance of a pair of objects, which is an important task in many areas, is linear in the number of data items in the case where vectors are employed. The same task for graphs, however, is much more complex, since one cannot simply compare the sets of nodes and edges, which are generally unordered and of different sizes.

In the last decades various procedures for evaluating proximity, that is similarity or dissimilarity, of graphs have been proposed in the literature [1]. The process of evaluating the similarity of two graphs is commonly referred to as *graph matching*. Roughly speaking, there are two categories of tasks in graph matching, viz. *exact graph matching* and *error-tolerant graph matching*. In the former case, for a matching to be successful, it is required that a strict correspondence is found between the two graphs being matched, or at least among their subparts. Prominent examples include methods for graph and subgraph isomorphism [10, 11].

Due to the intrinsic variability of the objects under consideration and the noise resulting from the graph extraction process, it cannot be expected that two graphs representing the same class of objects are completely, or at least to a large part, identical in their structure. Especially if the nodes and edges of a graph are attributed with real-valued labels, it is most probable that the actual graphs differ somewhat from their ideal model. Obviously, such noise crucially hampers the applicability of exact matching techniques, and consequently exact graph matching is rarely used in real-world pattern recognition applications. In order to overcome this drawback, various approaches to error-tolerant graph matching have been proposed [1].

Graph edit distance (GED) offers an intuitive way to integrate tolerance to errors into the graph matching process and is applicable to virtually all types of graphs. Originally, edit distance has been developed for string matching [12]. A generalization to graphs first emerged in [13] in the context of error-correcting isomorphism and has been extended to a general graph distance subsequently [14, 15]. The key idea is to model structural variation by edit operations reflecting modifications in structure and labeling. A standard set of edit operations is given by *deletions*, *insertions*, and *substitutions* of both nodes and edges.[1] Given two graphs, the idea of graph edit distance is to delete

[1] Note that other operations, such as *merging* and *splitting* of nodes have been proposed [16].

some nodes and edges in the first graph, substitute (relabel) some of the remaining nodes and edges, and insert some new nodes and edges such that the first graph is transformed into the second graph. An edit cost is assigned to each edit operation and the graph edit distance corresponds with the minimum cost among all valid graph transformations. It is possible to integrate domain specific knowledge about object similarity, if available, when defining the costs of the elementary edit operations. Furthermore, if in a particular case prior domain knowledge is not available, automatic procedures for learning the edit costs from a set of sample graphs exist [17, 18, 19].

The flexibility of graph edit distance to cope with any kind of graph structure, node labels, and edge labels is an advantage over other graph matching methods that often impose constraints on the graphs. For example spectral methods [20, 21], which are based on the eigendecomposition of the adjacency or Laplacian matrix of a graph, primarily target unlabeled graphs or allow only severely constrained label alphabets. Other common constraints include restrictions to ordered graphs [22], planar graphs [23, 24], bounded-valence graphs [25], trees [26], and graphs with unique node labels [9]. The absence of such constraints makes graph edit distance applicable to a wide range of real-world applications.

However, the flexibility comes at the cost of high computational complexity. Optimal algorithms for computing the graph edit distance are typically based on combinatorial search procedures that explore the space of all valid graph transformations. Often A* search techniques using some heuristics are employed [27, 28, 29, 30]. A* is a best-first search algorithm [31] which is *complete* and *admissible*, that is it always finds a solution if there is one and it never overestimates the cost of reaching the goal. The time complexity is exponential with respect to the number of nodes of the involved graphs, thus constraining graph edit distance to small graphs in practice.

In recent years, a number of methods addressing the high computational complexity of graph edit distance computation have been proposed. Probabilistic relaxation labeling [32, 33] adopts a Bayesian perspective on graph edit distance and iteratively applies edit operations to improve a maximum *a posteriori* criterion. As an alternative to this hill climbing approach, genetic algorithms have been proposed for optimization in [34]. In either case, the method is usually more efficient than A* search but is prone to finding only local optima in the search space. The same holds true for the methods proposed in [35, 36] where a randomized construction of initial mappings is followed by a local search procedure. For A* search, efficiency improvements by means of suboptimal beam search techniques are proposed in [37]. In [38], a linear programming method for computing the edit distance of graphs with unlabeled edges is reported. Based on the assignment problem, a polynomial time calculation of upper and lower bounds is suggested.

Assignment edit distance (AED) is a general method to approximate graph edit distance for unconstrained graphs in cubic time with respect to the number of graph nodes. Originally, it has been introduced as a novel heuristic for optimal graph edit distance computation based on fast node assignments [30]. This

heuristic has eventually been used in [39] as a stand alone graph edit distance approximation. The method is based on a fast optimization procedure mapping nodes and their local structure of one graph to nodes and their local structure of another graph. In order to find an optimal match between the sets of local structure the Hungarian algorithm [40][2] is deployed. Since only local structures are matched the resulting assignment edit distance is not optimal. Yet a series of graph classification experiments empirically demonstrated a substantial speedup without significant loss in accuracy when using this approximation algorithm [39]. Further speedups were reported in [42] with a different cubic time assignment algorithm, viz. the algorithm of Volgenant-Jonker [43]. In recent years, the algorithmic framework of assignment edit distance has been successfully employed for several applications where graphs have been embedded in vector spaces [44]. In [45] the method has been adopted to the problem of exact graph matching, namely graph isomorphism and subgraph isomorphism. The graph matching framework has been made publicly available as a stand-alone software tool [46].[3]

In this paper, we continue this line of research by introducing the Hausdorff edit distance (HED), which approximates the graph edit distance more efficiently in quadratic rather than cubic time. It combines the idea of assignment edit distance, that is to find a match between nodes and their local structure, with a more efficient pairwise node matching. In a straight-forward fashion, each node of one graph is compared with each node of the other graph similar to comparing subsets of a metric space using the Hausdorff distance [47]. Taking into account costs for deletion, substitution, and insertion of a node and its adjacent edges, an optimal match is found for each node individually. The sum of all matching costs is then considered as a distance measure which is less than or equal to the graph edit distance.

Despite the fact that an optimal match is found for each node individually and not for all nodes conjointly, the proposed Hausdorff edit distance performs astonishingly well in a series of graph classification experiments. They empirically demonstrate a substantial speedup when compared with graph edit distance and assignment edit distance. In many cases, the speedup is achieved with only a minor loss in accuracy or, in one case, even with a gain in accuracy. We report results for nearest neighbor classification of graphs representing letter drawings, fingerprints, and molecular compounds. Furthermore, we have investigated the Hausdorff edit distance in the context of a more complex document analysis system for automatic handwriting recognition in historical manuscripts.

The remainder of this paper is organized as follows. First, a description of graph edit distance and its cubic time approximation is provided in Section 2. Afterwards, the proposed Hausdorff edit distance is detailed in Section 3. Graph classification experiments are first presented for k-nearest neighbor recognition

[2]The Hungarian algorithm is a refinement of an earlier version by Kuhn [41] and is also referred to as Munkres or Kuhn-Munkres algorithm.

[3]`http://www.fhnw.ch/wirtschaft/iwi/gmt`

in Section 4, followed by a report on handwriting recognition using graph embedding and hidden Markov models in Section 5. Finally, we draw some conclusions in Section 6.

Note that this paper is an extended version of an earlier conference publication [48]. While the previous publication was focused on the handwriting recognition application, this paper provides a generalization of the algorithm that is able to cope with virtually all types of graphs. The detailed description of the proposed Hausdorff edit distance in Section 3 is completely new as well as the graph classification experiments with different types of graphs in Section 4.

2. Graph Edit Distance

In the following, we describe the graph edit distance (GED) and its cubic time approximation, called assignment edit distance (AED) in this paper. Some basic notations and definitions are introduced in Sections 2.1 and 2.2, the optimal computation of GED is discussed in Section 2.3, and AED is presented in Section 2.4.

2.1. Graphs

A *graph* g is a four-tuple $g = (V, E, \mu, \nu)$. V is the finite set of nodes, $E \subseteq V \times V$ is the set of edges, $\mu : V \to L_V$ is the node labeling function, $\nu : E \to L_E$ is the edge labeling function, and L_V and L_E are finite or infinite label sets for nodes and edges, respectively.

In case of *labeled graphs* the labels for both nodes and edges can be given by the set of integers $L = \{1, 2, 3, \ldots\}$, the vector space $L = \mathbb{R}^n$, a set of symbolic labels $L = \{\alpha, \beta, \gamma, \ldots\}$, or a combination of two or more label alphabets. *Unlabeled graphs* are obtained as a special case by assigning the same label ε to all nodes and edges, that is $L_V = L_E = \{\varepsilon\}$.

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. *Directed graphs* directly correspond to the definition above. However, the class of *undirected graphs* can be modeled as well, inserting a reverse edge $(v, u) \in E$ for each edge $(u, v) \in E$ with an identical label, that is $\nu(u, v) = \nu(v, u)$. In this case, the direction of an edge can be ignored, since there are always edges in both directions.

The *edge degree* $|u|$ denotes the number of edges adjacent to u including incoming as well as outgoing edges in the case of directed graphs.

2.2. GED Definition

Graph edit distance (GED) is a generalization of string edit distance, also known as Levenshtein distance [12, 49]. Strings can be regarded as a special case of graphs where the order of the characters allows to establish a string-to-string correspondence efficiently by means of dynamic programming. For unconstrained graphs, however, the graph-to-graph correspondence has to take nodes as well as edges into account and cannot rely on the order of the nodes, which dramatically increases the complexity of edit distance computation.

Given two graphs, the source graph g_1 and the target graph g_2, the idea of GED is to delete some nodes and edges from g_1, substitute some of the remaining nodes and edges by changing their labels, 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 into g_2 is called an *edit path* between g_1 and g_2.

Let $\Upsilon(g_1, g_2)$ denote the set of all possible edit paths between two graphs g_1 and g_2. Clearly, every edit path between two graphs g_1 and g_2 is a model describing the correspondences found between the graphs' substructures. That is, each node of g_1 is either deleted or uniquely substituted with a node in g_2, and analogously, each node in g_2 is either inserted or matched with a unique node in g_1. The same applies for the edges.

To find the most suitable edit path out of $\Upsilon(g_1, g_2)$, one introduces a cost $\mathcal{C}(e_i)$ for each edit operation e_i, measuring the strength of the corresponding operation. The idea of such a cost is to define whether or not an edit operation 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 costs is needed. Consequently, GED is defined as the minimum cost edit path between two graphs:

$$\mathrm{GED}(g_1, g_2, \mathcal{C}) = \min_{(e_1,\ldots,e_k)\in\Upsilon(g_1,g_2)} \sum_{i=1}^{k} \mathcal{C}(e_i) \tag{1}$$

That is, dissimilar graphs have large edit distances and similar graphs have small edit distances.

The flexibility of GED is based on the cost function $\mathcal{C}(e_i)$ which can be defined for any kind of graph taking into account any kind of node and edge labels. The cost function only needs to satisfy weak conditions such that $\Upsilon(g_1, g_2)$ becomes computable, that is *non-negativity* $\mathcal{C}(e_i) \geq 0$ and some *triangle inequality* conditions of the form $\mathcal{C}(x \to y) \leq \mathcal{C}(x \to z) + \mathcal{C}(z \to y)$ to exclude unnecessary substitutions $(x \to z)$ of nodes and edges from the edit path [44]. If, additionally, the substitution cost for nodes and edges is *symmetric* $\mathcal{C}(x \to y) = \mathcal{C}(y \to x)$ and only identical substitutions have zero cost $\mathcal{C}(x \to x) = 0$ the corresponding GED is metric. For more details on the conditions and properties of valid cost functions, we refer to [44]. In general, some domain knowledge is needed to define suitable cost functions. Yet there are also automatic procedures to learn the edit costs from a set of sample graphs [17, 18, 19].

For example, a frequently used cost function for nodes and edges with vectorial labels is the *Euclidean cost function* defined as follows:

$$\begin{aligned}
&\mathcal{C}(u \to v) = \alpha \cdot ||\mu_1(u) - \mu_2(v)|| \\
&\mathcal{C}(u \to \epsilon) = \alpha \cdot \tau_n,\ \mathcal{C}(\epsilon \to v) = \alpha \cdot \tau_n \\
&\mathcal{C}(p \to q) = (1-\alpha) \cdot ||\nu_1(p) - \nu_2(q)|| \\
&\mathcal{C}(p \to \epsilon) = (1-\alpha) \cdot \tau_e,\ \mathcal{C}(\epsilon \to q) = (1-\alpha) \cdot \tau_e
\end{aligned} \tag{2}$$

for nodes $u \in V_1, v \in V_2$ and edges $p \in E_1, q \in E_2$. Substitutions correspond with the Euclidean distance of the vectorial labels, deletions and insertions have

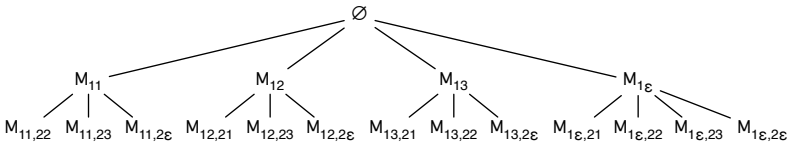


Figure 1: GED search tree.

fixed costs $\tau_n, \tau_e > 0$, and the node and edge costs are balanced with the factor $0 \leq \alpha \leq 1$. The parameters τ_n, τ_e, and α are user-defined and need to be optimized on a validation set for a given graph matching task.

2.3. GED Computation

The computation of GED is usually based on the A* algorithm [31] to explore all possible edit paths in $\Upsilon(g_1, g_2)$ by means of a tree search over all possible mappings of the nodes of the source graph $V_1 = \{u_1, \ldots, u_{|V_1|}\}$ to the nodes of the target graph $V_2 = \{v_1, \ldots, v_{|V_2|}\}$.

In the following, we express a valid mapping of the nodes by means of a *node map* $M = \{e_1, \ldots, e_{|M|}\}$ which contains node edit operations including substitutions $(u \rightarrow v)$, deletions $(u \rightarrow \epsilon)$, and insertions $(\epsilon \rightarrow v)$ such that each node $u \in V_1$ and $v \in V_2$ appears at most once in M. That is, multiple mappings of the same node are not allowed. The node map is called *complete* if all nodes of V_1 and V_2 appear in M, and *partial* otherwise.

The search tree is constructed as follows. At the root level, an empty partial map $\emptyset$ is inserted. At the first level, partial maps $\{e_1\}$ are inserted as children for each possible edit operation for the first node u_1.[4] There are $|V_2| + 1$ possibilities including all node substitutions $(u_1 \rightarrow v_1), \ldots, (u_1 \rightarrow v_{|V_2|})$ and the node deletion $(u_1 \rightarrow \epsilon)$. At the second level, partial maps $\{e_1, e_2\}$ are inserted as children of $\{e_1\}$ for each possible edit operation for the second node u_2. This procedure is continued until all nodes of the source graph are processed at the final level $|V_1|$ of the search tree. Clearly, the number of entries in the tree is exponential with respect to the number of nodes of the involved graphs.

An example is shown in Figure 1 for $V_1 = \{u_1, u_2\}$ and $V_2 = \{v_1, v_2, v_3\}$ with $M_{11} = \{(u_1 \rightarrow v_1)\}$, $M_{11,2\epsilon} = \{(u_1 \rightarrow v_1), (u_2 \rightarrow \epsilon)\}$, *etc.* Note that all node maps at the leaf level 2 are still partial since there is at least one node in V_2 which does not appear in M.

From the leaf nodes at level $|V_1|$ of the search tree, complete edit paths can be derived. First, a complete node map M is obtained by adding an insertion $(\epsilon \rightarrow v)$ for each unmapped node $v \in V_2$. Then, three cases are distinguished to imply edge edit operations from node edit operations:

- Node deletion. $(u \rightarrow \epsilon)$ implies the deletion of all edges $\{p_1, \ldots, p_{|u|}\}$ adjacent to $u \in V_1$, that is $(p_1 \rightarrow \epsilon), \ldots, (p_{|u|} \rightarrow \epsilon)$.

[4] The order of the nodes in the source graph is random and does not affect the result.

Algorithm 1 EPC($M, \mathcal{C}$) – Edit Path Cost

Require: complete node map M, cost function $\mathcal{C}$
Ensure: edit path cost c
1: $c \leftarrow 0$
2: **for all** node deletions $(u \rightarrow \epsilon)$ in M **do**
3: $\quad c \leftarrow c + \mathcal{C}(u \rightarrow \epsilon)$
4: $\quad c \leftarrow c + \frac{\mathcal{C}(p \rightarrow \epsilon)}{2}$ $\forall$ implied edge deletions
5: **end for**
6: **for all** node insertions $(\epsilon \rightarrow v)$ in M **do**
7: $\quad c \leftarrow c + \mathcal{C}(\epsilon \rightarrow v)$
8: $\quad c \leftarrow c + \frac{\mathcal{C}(\epsilon \rightarrow q)}{2}$ $\forall$ implied edge insertions
9: **end for**
10: **for all** node substitutions $(u \rightarrow v)$ in M **do**
11: $\quad c \leftarrow c + \mathcal{C}(u \rightarrow v)$
12: $\quad c \leftarrow c + \frac{\mathcal{C}(p \rightarrow q)}{2}$ $\forall$ implied edge substitutions
13: $\quad c \leftarrow c + \frac{\mathcal{C}(p \rightarrow \epsilon)}{2}$ $\forall$ implied edge deletions
14: $\quad c \leftarrow c + \frac{\mathcal{C}(\epsilon \rightarrow q)}{2}$ $\forall$ implied edge insertions
15: **end for**
16: **return** c

- Node insertion. $(\epsilon \rightarrow v)$ implies the insertion of all edges $\{q_1, \ldots, q_{|v|}\}$ adjacent to $v \in V_2$, that is $(\epsilon \rightarrow q_1), \ldots, (\epsilon \rightarrow q_{|v|})$.
- Node substitution. $(u \rightarrow v)$ implies edge substitions $(p_i \rightarrow q_j)$ only if both nodes of p_i are mapped to the nodes of q_j.[5] Otherwise, edge deletions $(p_i \rightarrow \epsilon)$ and insertions $(\epsilon \rightarrow q_j)$ are implied.

Algorithm 1 details the calculation of the edit path cost EPC($M, \mathcal{C}$) based on a complete node map. Only half of the edge costs are added in Lines 4, 8, 12, 13, and 14 because each edge edit operation is implied by exactly two node edit operations.

The A* algorithm does not necessarily expand the whole search tree. At each time step, all partial node maps P, which correspond with non-leaf entries in the search tree, are sorted according to the function

$$f(P) = g(P) + h(P) \tag{3}$$

where $g(P)$ are the current costs of the node edit operations and implied edge edit operations[6] and $h(P)$ is a heuristic that estimates the future costs taking into account all unmapped nodes. Then, the best partial map P with the smallest value of $f(P)$ is expanded for the next time step. If $h(P)$ is less than

[5] In case of directed graphs the edge direction needs to be preserved, too.
[6] Implied edge costs are only added if both nodes of the edge are already part of P.

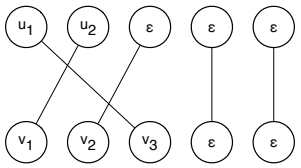


Figure 2: AED assignment graph.

or equal to the real future costs, the search procedure can be stopped as soon as a complete node map M^* is found with $f(M^*) \leq f(P)\ \forall P$. In this case it is guaranteed that M^* is the optimal node map with respect to GED and $f(M^*) = \mathrm{EPC}(M^*, \mathcal{C}) = \mathrm{GED}(g_1, g_2, \mathcal{C})$ is the optimal solution for GED.

Although good heuristics can speedup the search [27, 29, 28, 30], the time complexity of the A* algorithm remains exponential with respect to the number of nodes of the involved graphs and hence GED can often be calculated only for small graphs in practice.

2.4. Assignment Edit Distance

AED [39] has been proposed to approximate GED in cubic time with respect to the number of nodes of the involved graphs. The idea is to reduce the GED problem to an assignment problem between nodes and their local structure.

In general, the assignment problem can be stated as follows. Given two sets $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$ of equal size and a predefined assignment cost $C(a_i, b_j) \in \mathbb{R}$, the goal of the assignment problem is to find a bijection from A to B such that the sum of the assignment costs is minimal. A well-known solution for this problem is provided by the Hungarian algorithm [40] that has a cubic time complexity of $O(n^3)$ with respect to the size of the sets, n. For a detailed description of the algorithm, we refer to [39].

GED is reduced to an assignment problem by setting $A = V_1 \cup \{\epsilon_1, \ldots, \epsilon_{|V_2|}\}$ and $B = V_2 \cup \{\epsilon_1, \ldots, \epsilon_{|V_1|}\}$. That is, each assignment between the sets of size $n = |V_1| + |V_2|$ corresponds either with a node substitution $(u \to v)$, deletion $(u \to \epsilon)$, insertion $(\epsilon \to v)$, or an *empty substitution* $(\epsilon \to \epsilon)$.[7] Based on the node matching cost $C_n(u, v) \in \mathbb{R}$ discussed below, the Hungarian algorithm is then applied to find an optimal solution for the assignment problem. After removing all empty substitutions, the solution found by the algorithm corresponds with a complete node map M^a which is optimal with respect to the assignment problem. Finally, the *assignment edit distance* (AED) is defined as the edit path cost according to the node map M^a and Algorithm 1, that is

$$\mathrm{AED}(g_1, g_2, \mathcal{C}) = \mathrm{EPC}(M^a, \mathcal{C}) \tag{4}$$

[7] Note that we do not distinguish between different ϵ nodes for the description of AED in this paper. A distinction can be made in order to assure that each regular node can only be mapped to one specific ϵ node in order to speedup the assignment [39].

An example is shown in Figure 2 for $V_1 = \{u_1, u_2\}$ and $V_2 = \{v_1, v_2, v_3\}$ with $M^a = \{(u_1 \to v_3), (u_2 \to v_1), (\epsilon \to v_2)\}$. The undirected assignment graph is *bipartite*, that is it can be divided into two disjoint sets of nodes such that every edge connects the two sets. For AED it is required that each node of one set is connected with exactly one node from the other set.

The *node matching cost* $C_n(u, v)$ for $u \in V_1 \cup \{\epsilon\}$ and $v \in V_2 \cup \{\epsilon\}$ is defined with respect to the cost function $\mathcal{C}$ and with a view to Algorithm 1 in order to approximate GED. Three cases are distinguished:

- Node deletion. $C_n(u, \epsilon) = \mathcal{C}(u \to \epsilon) + \sum_{i=1}^{|u|} \frac{\mathcal{C}(p_i \to \epsilon)}{2}$ with respect to the edges $\{p_1, \ldots, p_{|u|}\}$ adjacent to u.
- Node insertion. $C_n(\epsilon, v) = \mathcal{C}(\epsilon \to v) + \sum_{i=1}^{|v|} \frac{\mathcal{C}(\epsilon \to q_i)}{2}$ with respect to the edges $\{q_1, \ldots, q_{|v|}\}$ adjacent to v.
- Node substitution. $C_n(u, v) = \mathcal{C}(u \to v) + \frac{C_e(u,v)}{2}$ with respect to the edge matching cost $C_e(u, v)$ detailed below.

The cost for empty substitutions $C_n(\epsilon, \epsilon)$ is zero. For node deletion and insertion, the exact node matching cost is known *a priori*, that is before the assignment problem is solved. For node substitution, however, the implied edge costs are not known *a priori* because the node map M is missing that would be needed to determine valid edge substitutions (see Section 2.3). At this point, AED has to estimate the implied edge costs and therefore becomes suboptimal.

The *edge matching cost* $C_e(u, v)$ for $u \in V_1$ and $v \in V_2$ is calculated without constraints of a node map M. It is assumed that every edge $\{p_1, \ldots, p_{|u|}\}$ adjacent to u can potentially be substituted with any edge $\{q_1, \ldots, q_{|v|}\}$ adjacent to v. The Hungarian algorithm is used to obtain an optimal edge assignment with $A = \{p_1, \ldots, p_{|u|}, \epsilon_1, \ldots, \epsilon_{|v|}\}$, $B = \{q_1, \ldots, q_{|v|}, \epsilon_1, \ldots, \epsilon_{|u|}\}$, $C(\epsilon, \epsilon) = 0$ for empty substitutions, and $C(a_i, b_j) = \mathcal{C}(a_i \to b_j)$ otherwise. Finally, $C_e(u, v)$ is the sum of all edge assignment costs.

The implied edge costs for node substitution are less than or equal to the true costs, because the true edge assignments are additionally constrained by a node map M. Therefore, the solution M^a found by the Hungarian algorithm may not be optimal with respect to GED and the corresponding edit path cost $\mathrm{EPC}(M^a, \mathcal{C})$ returned by Algorithm 1 may be higher than the cost of the optimal edit path $\mathrm{EPC}(M^*, \mathcal{C})$. Hence AED *overestimates* the true edit distance, that is $\mathrm{AED}(g_1, g_2, \mathcal{C}) \geq \mathrm{GED}(g_1, g_2, \mathcal{C})$. Note that if the graph contains no edges, AED is optimal since the estimation error only affects implied edge costs.

3. Hausdorff Edit Distance

AED described in Section 2.4 still has a considerable cubic time complexity of $O((n_1 + n_2)^3)$ in the number of nodes $n_1 = |V_1|$ and $n_2 = |V_2|$ of the involved graphs. This is due to the fact that an optimal solution to the assignment problem for nodes and their local structure is calculated *globally*, taking all node

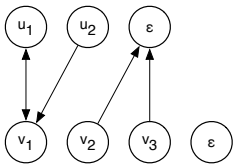


Figure 3: HED assignment graph.

matchings into account conjointly. In this paper, we investigate an even faster approximation of GED that matches nodes and their local structure *locally*, that is independently of the other node assignments. In a straight-forward fashion, each node of the first graph is compared with each node of the second graph similar to comparing subsets of a metric space using the Hausdorff distance [47]. Accordingly, the proposed Hausdorff edit distance (HED) can be calculated in quadratic time, that is $O(n_1 \cdot n_2)$.

In the following, HED is defined in Section 3.1 and its computation is detailed in Section 3.2.

3.1. HED Definition

The Hausdorff distance of two subsets A, B of a metric space is defined as

$$H(A, B) = \max(\sup_{a \in A} \inf_{b \in B} d(a, b), \sup_{b \in B} \inf_{a \in A} d(a, b)) \tag{5}$$

with respect to the metric $d(a, b)$. For finite sets A, B the Hausdorff distance is

$$H(A, B) = \max(\max_{a \in A} \min_{b \in B} d(a, b), \max_{b \in B} \min_{a \in A} d(a, b)) \tag{6}$$

which can be interpreted as the maximum among all nearest neighbor distances between A and B. Because of the maximum operator, the Hausdorff distance is sensitive to outliers. By replacing the maximum operator with the sum

$$H'(A, B) = \sum_{a \in A} \min_{b \in B} d(a, b) + \sum_{b \in B} \min_{a \in A} d(a, b) \tag{7}$$

the modified Hausdorff distance takes into account all nearest neighbor distances and becomes less sensitive to outliers. Derived from Equation 7, we then define the *Hausdorff edit distance* (HED) between two graphs $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ as

$$\mathrm{HED}(g_1, g_2, \mathcal{C}) = \sum_{u \in V_1} \min_{v \in V_2 \cup \{\epsilon\}} C_n^*(u, v) + \sum_{v \in V_2} \min_{u \in V_1 \cup \{\epsilon\}} C_n^*(u, v) \tag{8}$$

with respect to a modified node matching cost $C_n^*(u, v)$ defined below.

The nearest neighbors can be interpreted as node assignments. An example is shown in Figure 3 for $V_1 = \{u_1, u_2\}$ and $V_2 = \{v_1, v_2, v_3\}$ with the assignments

$\{(u_1 \to v_1), (u_2 \to v_1), (v_1 \to u_1), (v_2 \to \epsilon)(v_3 \to \epsilon)\}$. The HED assignment graph is bipartite, directed, and contains one edge for each node in V_1 and V_2. In contrast to the AED assignment graph, multiple assignments to the same node are allowed.

The definition of the modified node matching cost $C_n^*(u, v)$ for HED is based on the definition of $C_n(u, v)$ for AED in Section 2.4

$$C_n^*(u,v) = \begin{cases} \frac{C_n(u,v)}{2}, & \text{if } (u \to v) \text{ is a substitution} \\ C_n(u,v), & \text{otherwise} \end{cases} \tag{9}$$

Only half of the costs are considered for node substitutions because HED does not enforce bidirectional substitutions. In Equation 8, the substitution costs are distributed over two summation terms. Only if a substitution is established in both directions, the full cost is accumulated. Node deletions and insertions, on the other hand, only appear in one of the summation terms and hence their full cost is taken into account.

The edge matching cost $C_e(u, v)$ is defined differently for HED when compared with AED. Instead of solving an assignment problem for the edges, Hausdorff matching is performed in the same way as for the nodes. Given two sets of edges $P = \{p_1, \ldots, p_{|u|}\}$ adjacent to u and $Q = \{q_1, \ldots, q_{|v|}\}$ adjacent to v, the edge matching cost is

$$C_e(u,v) = \sum_{p \in P} \min_{q \in Q \cup \{\epsilon\}} \mathcal{C}^*(p \to q) + \sum_{q \in Q} \min_{p \in P \cup \{\epsilon\}} \mathcal{C}^*(p \to q) \tag{10}$$

with respect to the modified cost function

$$\mathcal{C}^*(p \to q) = \begin{cases} \frac{\mathcal{C}(p \to q)}{2}, & \text{if } (p \to q) \text{ is a substitution} \\ \mathcal{C}(p \to q), & \text{otherwise} \end{cases} \tag{11}$$

Like for AED, $C_e(u, v)$ is less than or equal to the true cost because no constraints of a node map M are taken into account for edge assignments. Furthermore, the Hausdorff matching finds an optimal assignment for each edge individually, without constraints from the other edge assignments.

In summary, while GED finds an optimal assignment of all nodes conjointly, HED finds an optimal assignment for each node individually. If the best assignment for a node is a deletion or an insertion, Equation 8 captures its cost exactly when compared with GED. If the best assignment is a substitution, that is if half the substitution costs are smaller than the costs for deletion or insertion, Equation 8 captures only an estimate of the implied edge costs which may be lower than the true costs. Overall, the HED *underestimates* the true edit distance, that is $\text{HED}(g_1, g_2, \mathcal{C}) \leq \text{GED}(g_1, g_2, \mathcal{C})$.

Finally, in order to constrain the underestimation, the following lower bounds

Algorithm 2 HEC$(A, B, \mathcal{C})$ – Hausdorff Edit Cost

Require: sets A, B, cost function $\mathcal{C}$
Ensure: Hausdorff edit cost c
1: **for all** elements a in A **do**
2: $c_1(a) \leftarrow \mathcal{C}(a \rightarrow \epsilon)$
3: **end for**
4: **for all** elements b in B **do**
5: $c_2(b) \leftarrow \mathcal{C}(\epsilon \rightarrow b)$
6: **end for**
7: **for all** elements a in A **do**
8: **for all** elements b in B **do**
9: $c_1(a) \leftarrow \min(\frac{\mathcal{C}(a \rightarrow b)}{2}, c_1(a))$
10: $c_2(b) \leftarrow \min(\frac{\mathcal{C}(a \rightarrow b)}{2}, c_2(b))$
11: **end for**
12: **end for**
13: $c \leftarrow \sum_{a \in A} c_1(a) + \sum_{b \in B} c_2(b)$
14: **return** c

are used in this paper for HED$(g_1, g_2, \mathcal{C})$ and $C_e(u, v)$, respectively

$$
\begin{aligned}
L(g_1, g_2) &= \begin{cases} (|V_1| - |V_2|) \cdot \min_{u \in V_1} \mathcal{C}(u \rightarrow \epsilon), & \text{if } |V_1| > |V_2| \\ (|V_2| - |V_1|) \cdot \min_{v \in V_2} \mathcal{C}(\epsilon \rightarrow v), & \text{otherwise} \end{cases} \\
L(u, v) &= \begin{cases} (|u| - |v|) \cdot \min_{p \in P} \mathcal{C}(p \rightarrow \epsilon), & \text{if } |u| > |v| \\ (|v| - |u|) \cdot \min_{q \in Q} \mathcal{C}(\epsilon \rightarrow q), & \text{otherwise} \end{cases}
\end{aligned}
\tag{12}
$$

These lower bounds assert a minimum amount of deletion and insertion costs if the two matched sets differ in size.

3.2. HED Computation

Algorithms 2 and 3 compute HED with a straight-forward implementation used in this paper. The core matching algorithm is the Hausdorff edit cost HEC$(A, B, \mathcal{C})$ in Algorithm 2 that compares two sets A, B based on a cost function $\mathcal{C}$. Unlike the common Hausdorff distance, it does not only take substitutions into account (Lines 9 and 10) but also deletions (Line 2) and insertions (Line 5). Half of the substitution costs are considered for each set.

The Hausdorff edit distance HED$(g_1, g_2, \mathcal{C})$ in Algorithm 3 follows the same procedure as Algorithm 2. Node deletions and insertions (Lines 2 and 5) include the cost of the node edit operation as well as half of the (exact) costs of the implied edge edit operations. Node substitutions (Lines 11 and 12) include the cost of the node edit operation as well as half of the estimated costs of the implied edge edit operations c_e. The basic estimation of c_e in Line 9 is established by means of Hausdorff matching HEC$(u.E, v.E, \mathcal{C})$ with respect to the set of edges $u.E$ adjacent to node u, the set of edges $v.E$ adjacent to node

Algorithm 3 HED$(g_1, g_2, \mathcal{C})$ – Hausdorff Edit Distance

Require: graphs g_1, g_2, cost function $\mathcal{C}$
Ensure: Hausdorff edit distance d
1: **for all** $u \in g_1.V$ **do**
2: $\quad d_1(u) \leftarrow \mathcal{C}(u \to \epsilon) + \sum_{p \in u.E} \frac{\mathcal{C}(p \to \epsilon)}{2}$
3: **end for**
4: **for all** $v \in g_2.V$ **do**
5: $\quad d_2(v) \leftarrow \mathcal{C}(\epsilon \to v) + \sum_{q \in v.E} \frac{\mathcal{C}(\epsilon \to q)}{2}$
6: **end for**
7: **for all** $u \in g_1.V$ **do**
8: $\quad$ **for all** $v \in g_2.V$ **do**
9: $\quad\quad c_e \leftarrow \mathrm{HEC}(u.E, v.E, \mathcal{C})$
10: $\quad\quad c_e \leftarrow \max(L(u, v), c_e)$
11: $\quad\quad d_1(u) \leftarrow \min(\frac{\mathcal{C}(u \to v) + \frac{c_e}{2}}{2}, d_1(u))$
12: $\quad\quad d_2(v) \leftarrow \min(\frac{\mathcal{C}(u \to v) + \frac{c_e}{2}}{2}, d_2(v))$
13: $\quad$ **end for**
14: **end for**
15: $d \leftarrow \sum_{u \in g_1.V} d_1(u) + \sum_{v \in g_2.V} d_2(v)$
16: $d \leftarrow \max(L(g_1, g_2), d)$
17: **return** d

v, and the cost function $\mathcal{C}$. This estimate is bounded in Line 10 according to Equation 12. The final HED is summed up in Line 15 and bounded in Line 16.

Considering the two nested *for* loops in Lines 7 and 8, the computational complexity is $O(n_1 \cdot n_2)$ with respect to the number of nodes $n_1 = |V_1|$ and $n_2 = |V_2|$ of the involved graphs. Note that HED also reduces the computational complexity of adjacent edge matching when compared with AED. This is because Hausdorff matching is applied in Line 9 of Algorithm 3 instead of using the Hungarian algorithm.

4. Graph Classification

In this section, an empirical evaluation of the Hausdorff edit distance (HED, Section 3) is presented for the task of graph classification. The proposed graph matching algorithm is compared with assignment edit distance (AED, Section 2.4) with respect to runtime, recognition accuracy, and approximation of graph edit distance (GED, Section 2.3).

In the following, the investigated graph data sets are introduced in Section 4.1, the experimental setup is discussed in Section 4.2, the results are presented in Section 4.3, and a further analysis of the results regarding node correspondence is provided in Section 4.4.

Table 1: Graph data sets.

(a) Graph types of the different data sets. Node labels L_V, edge labels L_E, and a flag indicating whether or not the edges are directed.

Database	L_V	L_E	Directed
Letters	$(x, y) \in \mathbb{R}^2$	none	no
Fingerprints	none	$\varphi \in [-\pi, +\pi]$	yes
Molecules	symbolic	none	no

(b) Graph size. Average number of nodes $|V|_{avg}$, maximum number of nodes $|V|_{max}$, average number of edges $|E|_{avg}$, and maximum number of edges $|E|_{max}$.

Database	$\|V\|_{avg}$	$\|V\|_{max}$	$\|E\|_{avg}$	$\|E\|_{max}$
Letters I	4.7	8	3.1	6
Letters II	4.7	9	3.2	7
Letters III	4.7	9	4.5	9
Fingerprints	5.4	26	4.4	25
Molecules	15.7	95	16.2	103

(c) Graph classification task. Number of classes, a flag indicating whether or not the classes are represented with the same number of samples, and total number of samples for training, validation, and testing.

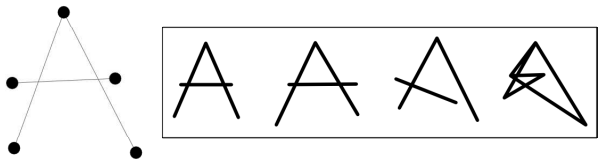
Database	Classes	Balanced	Train	Valid	Test
Letters	15	yes	750	750	750
Fingerprints	4	no	500	300	2000
Molecules	2	no	250	250	1500

4.1. Data Sets

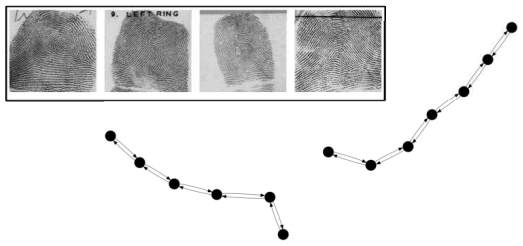
In order to evaluate HED for different types of graphs, three data sets from the IAM graph database [50][8] have been selected, namely the *Letters*, *Fingerprints*, and *Molecules* data sets described below. Three variants of the Letters data set are termed *Letters I*, *Letters II*, and *Letters III* in this paper with increasing levels of distortion. The basic properties of the graphs are listed in Table 1(a) and the typical graph size is shown in Table 1(b). The size ranges from small graphs with under 10 nodes and edges in the Letters data set up to a maximum of about 100 nodes and edges in the Molecules data set. Finally, the characteristics of the graph classification task are summarized in Table 1(c).

The *Letters* data set contains letter drawings of 15 capital letters of the Roman alphabet that can be drawn with straight lines only, that is *A*, *E*, *F*, *etc.* For each class, a prototype is manually created. The drawings are then converted

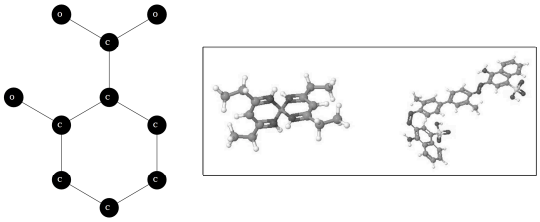
[8] http://www.iam.unibe.ch/fki/databases/iam-graph-database



(a) Letters



(b) Fingerprints



(c) Molecules

Figure 4: Graph data sets.

into graphs by representing lines with undirected edges and line endpoints with nodes that are labeled with coordinates $(x, y) \in \mathbb{R}^2$. Afterwards, the prototypes are artificially modified with different degrees of distortion in order to create the Letters I, II, and III data sets, respectively. Figure 4(a) shows a sample graph of the letter *A*. Next to the graph, the letter prototype and the three distortion degrees are illustrated. Each distortion level contains 750 graphs for training recognition systems, to validate system parameters, and to test the final system performance, respectively, as indicated in Table 1(c). The data set is *balanced*, that is each class is represented with the same number of samples.

The *Fingerprints* data set is based on the NIST-4 reference database of fingerprints [51]. Four classes of the Galton-Henry classification system are considered including *whorl*, *left*, *right*, and *arch*. Relevant image regions are binarized and thinned to skeletons of one pixel width. For graph extraction, endpoints, bifurcation points, as well as equidistant points on the skeleton in between are represented by unlabeled nodes. They are connected with edges if they are di-

rectly connected in the skeleton image. Edges are directed and labeled with the angle $\varphi \in [-\pi, +\pi]$ between the start node and the end node. Figure 4(b) shows a sample graph of the class *whorl*. In this example, the relevant image region contains two disconnected skeleton parts. Above the graph, the four fingerprint classes are illustrated. The data set contains 500 graphs for training, 300 for validation, and 2000 for testing.

Finally, the *Molecules* data set is based on molecular compounds from the AIDS Antiviral Screen Database of Active Compounds [52]. Two classes are distinguished, namely *active* and *inactive* depending on the activity of the compounds against HIV. A straight-forward graph representation is used, in which atoms are nodes labeled with their chemical symbol, and covalent bonds are unlabeled edges. Figure 4(c) shows a sample graph of the class *inactive*. Next to the graph, larger molecules of the two classes are illustrated. For the task of graph classification, the data set includes 250 graphs for training, 250 graphs for validation, and 1500 graphs for testing.

For a more detailed description of the different graph data sets from the IAM graph database, we refer to [50].

4.2. Experimental Setup

Graph classification is performed with respect to the k-nearest neighbor classifier (KNN) based on distances calculated with GED, AED, and HED, respectively. Nearest neighbors are selected in the training set, the classification accuracy achieved on the validation set is used to optimize system parameters, and the independent test set is used to measure the final system performance.

The edit distance and its approximations are based on a cost function $\mathcal{C}$. For all data sets, the cost functions are adopted from previous work [39]. They are similar to the Euclidean cost function defined in Section 2.2. The cost for node deletion and insertion is set to $\tau_n > 0$, the cost for edge deletion and insertion is set to $\tau_e > 0$, the costs of node edit operations are multiplied with balancing factor $0 \leq \alpha \leq 1$, and the costs of edge edit operations are multiplied with factor $(1 - \alpha)$. That is, the cost functions used for the different graph data sets only differ in their substitution costs.

For node substitutions on the Letters data set, the Euclidean distance between the vectorial labels $(x, y) \in \mathbb{R}^2$ is used

$$\mathcal{C}(u \to v) = \alpha \cdot ||(x_1, y_1) - (x_2, y_2)|| \tag{13}$$

For edge substitutions on the Fingerprints data set, a normalized difference between the angles $\varphi \in [-\pi, +\pi]$ is taken into account

$$\mathcal{C}(p \to q) = (1 - \alpha) \cdot \min(|\varphi_1 - \varphi_2|, 2\pi - |\varphi_1 - \varphi_2|) \tag{14}$$

Finally, for node substitutions on the Molecules data set, a Dirac function is employed for comparing chemical symbols s_1 and s_2

$$\mathcal{C}(u \to v) = \begin{cases} \alpha \cdot 2\tau_n, & \text{if } s_1 \neq s_2 \\ 0, & \text{otherwise} \end{cases} \tag{15}$$

Table 2: Experimental results on the test set.

(a) Runtime and accuracy results. Runtime in seconds on a 2GHz processor for recognizing the whole test set, speedup factor × of HED when compared with AED, accuracy in percentage, and accuracy difference Δ of HED when compared with AED.

Database	Runtime [s]			Accuracy [%]		
	AED	HED	×	AED	HED	Δ
Letters I	111.3	18.9	5.9	99.73	97.87	−1.87
Letters II	120.5	18.8	6.4	94.27	86.93	−7.33
Letters III	165.2	20.8	7.9	89.87	79.20	−10.67
Fingerprints	1320.6	116.4	11.3	80.30	82.80	+2.50
Molecules	4203.4	83.2	50.5	99.60	99.20	−0.40

(b) Speedup results of HED when compared with GED.

Database	Runtime [s]		
	GED	HED	×
Letters I	436.2	18.9	23.1
Letters II	3085.4	18.8	164.4
Letters III	2421.3	20.8	116.5

In case of the Letters and Molecules data sets, the edges have no labels. They can be substituted with zero cost. Likewise, unlabeled nodes can be substituted with zero cost in the case of the Fingerprints data set.

System parameters that are optimized on the validation set include the cost for node deletion and insertion τ_n as well as the cost for edge deletion and insertion τ_e. They are optimized over the range $\tau_n, \tau_e \in \{0.1, 0.4, \ldots, 2.5, 10.0\}$ with a grid search. The number of nearest neighbors has been fixed for the experiments to $k = 5$ and the balancing factor has been fixed to $\alpha = 0.5$ according to previous work [39].

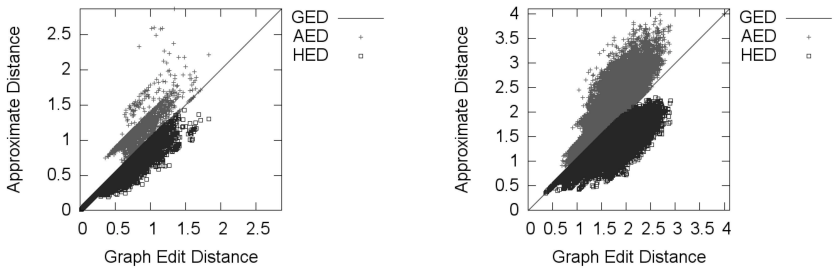
We use a publicly available Java implementation of GED and AED [46][9] and have implemented HED within the same software toolkit. Runtime comparisons are performed on a 2GHz processor.

4.3. Results

The main experimental results are listed in Table 2(a). In general, the proposed quadratic time HED provides a trade-off between speed and accuracy when compared with the cubic time AED. That is, a substantial speedup is achieved at the expense of a certain loss in recognition accuracy.

Speedups between one and two orders of magnitude are achieved ranging from a factor of 5.9 in the case of the relatively small Letters I graphs (maximum

[9] `http://www.fhnw.ch/wirtschaft/iwi/gmt`



(a) Intra-class (b) Inter-class

Figure 5: GED approximation Letters I.

Table 3: Approximation results. Percentage of distances equal to GED as well as the approximation error and its standard deviation.

(a) Intra-class

Database	AED [%]			HED [%]		
	Equal	Δ_{avg}	$\Delta_{\pm sdev}$	Equal	Δ_{avg}	$\Delta_{\pm sdev}$
Letters I	95.2	+1.8	9.4	69.2	−3.8	7.9
Letters II	75.9	+16.9	38.7	26.1	−26.9	20.9
Letters III	71.6	+6.9	16.9	17.3	−24.3	15.3

(b) Inter-class

Database	AED [%]			HED [%]		
	Equal	Δ_{avg}	$\Delta_{\pm sdev}$	Equal	Δ_{avg}	$\Delta_{\pm sdev}$
Letters I	27.2	+12.6	13.9	8.6	−17.4	11.2
Letters II	18.3	+34.2	29.4	0.03	−47.0	10.5
Letters III	28.6	+10.4	14.2	0.01	−31.7	11.1

of 8 nodes, see Table 1(b)) to a factor of 50.5 for the larger Fingerprints graphs (maximum of 95 nodes).

The three Letters data sets are small enough such that the exact GED can be computed as well in reasonable time. Speedups are listed in Table 2(b) with a maximum speedup factor of 164.4 for the Letters II data set (maximum of 9 nodes). In accordance with the findings in [39], the accuracy achieved with GED does not significantly differ form the accuracy achieved with AED for the Letters data set.

The best accuracy result of HED is achieved for the Fingerprints data set, where an increase of +2.5% is reported. Together with a speedup factor of 11.3, this is the best overall result we can report for HED. Furthermore, the relatively small losses in accuracy of −0.4% for the Molecules data set and −1.87% for the Letters I data set can be seen as a success of the proposed method, since a substantial speedup can be achieved at only a small expense in recognition

accuracy. The higher losses of -7.33% for the Letters II and -10.67% for the Letters III data sets, however, might be too restrictive for the trade-off between speed and accuracy and show limitations of the proposed method. All differences in recognition accuracy reported in Table 2(a) are statistically significant (t-test, $\alpha = 0.05$).

Besides runtime and accuracy, we have also investigated the approximation behavior of HED on the Letters data set. Both AED and HED approximate GED with the conditions $\mathrm{HED}(g_1, g_2, \mathcal{C}) \leq \mathrm{GED}(g_1, g_2, \mathcal{C}) \leq \mathrm{AED}(g_1, g_2, \mathcal{C})$ (see Sections 2.4 and 3). Figures 5(a) and 5(b) illustrate this behavior on the Letters I data set for both intra-class and inter-class distances. Note that in the plots the markers for HED occlude the markers for AED in the case of $\mathrm{HED}(g_1, g_2, \mathcal{C}) = \mathrm{AED}(g_1, g_2, \mathcal{C})$, but $\mathrm{HED}(g_1, g_2, \mathcal{C}) \leq \mathrm{AED}(g_1, g_2, \mathcal{C})$ held true in all cases.

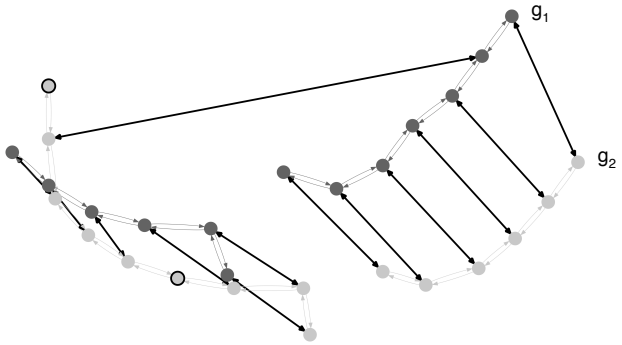
In Tables 3(a) and 3(b) we report the percentage of distances that are equal to GED as well as the approximation error and its standard deviation. The indicated error is the mean difference in percentage between the approximation and GED. As expected, the approximation error of HED is larger than that of AED. The percentage of equal distances is high for intra-class distances and the error is low in this case. A minimum average error of -3.8% is achieved with HED for intra-class distances among the slightly distorted graphs of the Letters I data set. A maximum average error of -47.0% is reported for inter-class distances on the Letters II data set.

4.4. Node Correspondence

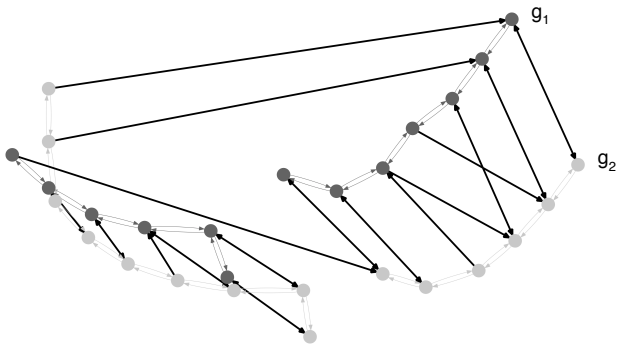
In the previous section, experimental results were reported with respect to the HED dissimilarity measure, which is an approximation of GED. In addition to this dissimilarity measure, HED also provides an interpretation of the graph matching in terms of node correspondence. That is, each node of the first graph (and its local structure) is either substituted with a node of the second graph or it is deleted. Vice versa, each node of the second graph is either substituted with a node of the first graph or it is inserted. However, as pointed out in Section 3.1, substitutions are not required to be bidirectional, which is a major difference to the node correspondence implied by GED and AED. In this section, we analyze the HED node correspondence for an example from the Fingerprints data set and compare it with the AED node correspondence.

In Figures 6a and 6b, we consider two fingerprint graphs g_1 (dark gray) and g_2 (light gray) which belong to the same class. In both cases, the relevant image region consists of two disconnected skeleton parts whose endpoints as well as several equidistant skeleton points are represented by nodes (see Section 4.1). Note that the nodes of the graphs are unlabeled and the Cartesian coordinates of the skeleton points are only used to visualize the graphs. Instead, the (directed) edges are labeled with the angle between two linked skeleton points. Consequently, the graph representation becomes invariant to translation. For more details on the graph representation, we refer to [53].

In Figure 6a, the node correspondence implied by AED is shown. Bidirectional black arrows indicate node substitutions and black circles around two



(a) AED



(b) HED

Figure 6: Node correspondence.

nodes of g_2 indicate node insertions. The node correspondence is established with respect to a cost function that takes angular differences of the edge labels into account (see Section 4.2). Since the Cartesian coordinates of the skeleton points are not used as labels on the nodes, substitutions may also be established between distant parts of the image which occurs once in this example. Because g_2 has two more nodes than g_1, a minimum of two node insertions are required for graph matching.

In contrast, Figure 6b shows the node correspondence implied by HED. Node substitutions are not required to be bidirectional, *that is* multiple correspondences can be established with the same node. With respect to the underlying cost function, whose parameters were optimized with respect to the recognition accuracy on the validation set, node substitutions are preferred over deletion and insertion in all cases in this example. In the case of small differences in the length of the skeleton parts and the resulting small differences in the number of nodes, HED provides a smoother node correspondence than AED.

We assume this property of the HED node correspondence is responsible,

in large parts, for the increase in classification accuracy observed for the Fingerprints data set when using HED instead of AED (see Section 4.3). For the shown example, HED identifies g_2 as the nearest training graph for the test graph g_1 and the KNN classifier assigns g_1 to the correct class. In contrast, AED results in a higher dissimilarity between g_1 and g_2, does not identify g_2 as the nearest neighbor, and eventually assigns g_1 to the wrong class.

5. Handwriting Recognition

In this section we report experimental results for an application of HED in the context of a more complex document analysis system for automatic handwriting recognition in historical manuscripts, for which HED was originally proposed in [48].

Despite the high representational power of graphs for single characters, for instance in the case of Chinese handwriting [8], graphs are only rarely used for unconstrained handwriting recognition. Due to variable character shapes and touching characters, it is usually not possible to segment handwriting into characters before recognition [54]. Only few recognition systems exist that can perform both segmentation and recognition at the same time, most prominently hidden Markov models (HMM) [55, 56]. They are based on a statistical pattern representation by means of feature vectors and thus graphs cannot be used in a straight-forward fashion.

In a recent approach [57], graph-based handwriting recognition is attempted by means of vector space embedding [44] such that a recognition with statistical classifiers becomes feasible. Promising results are reported when compared with standard statistical feature sets, yet the AED used for vector space embedding is computationally prohibitive. Runtimes of about half a minute per word constrain the approach to small collections of historical manuscripts in practice.

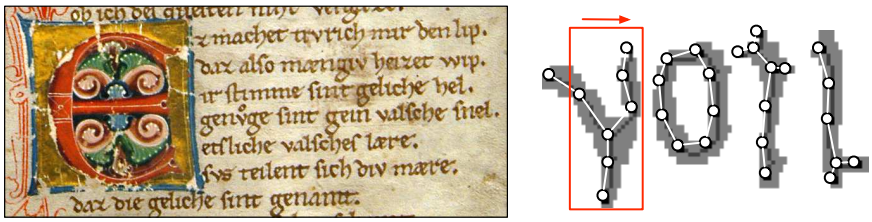
In this section, we investigate the performance of HED as a replacement of AED for the proposed handwriting recognition system. First, the handwriting data set is introduced in Section 5.1 and the recognition system is outlined in Section 5.2. Then, the experimental setup is discussed in Section 5.3 and results are presented in Section 5.4. Finally, a discussion of the experimental results, including the results from Section 4, is provided in Section 5.5.

5.1. Data Set

Experiments are conducted on the Parzival database[10] [58] which includes images from a 13th century manuscript written in Old German. $11,743$ word images are considered that contain $3,177$ word classes and 87 characters. A sample manuscript image is illustrated in Figure 7(a).

Handwriting graphs are extracted from skeleton images by representing endpoints, bifurcation points, and equidistant points on the skeleton with nodes.

[10] http://www.iam.unibe.ch/fki/databases/iam-historical-document-database



(a) Manuscript image (b) Skeleton graph, sliding window

Figure 7: Graph-based handwriting recognition.

Table 4: Handwriting graphs. Median number of nodes $|V|_{med}$, node labels L_V, edge labels L_E, and whether or not the edges are directed.

Database	$\|V\|_{med}$	L_V	L_E	Directed
Parzival	30	$(x, y) \in \mathbb{R}^2$	none	no

Nodes are labeled with their coordinates $(x, y) \in \mathbb{R}^2$ and are linked with unlabeled, undirected edges whenever they are directly connected in the skeleton image. The graphs are very similar to the ones in the Fingerprints data set (see Section 4.1) but instead of labeling the edges with angles, the nodes are labeled with coordinates. A sample handwriting graph is shown in Figure 7(b) and some basic statistics of the graph data set are listed in Table 4.

5.2. Recognition System

Handwriting graphs are transformed into feature vectors by means of dissimilarity space embedding [44]. First, prototype character graphs are selected, either manually or automatically [59]. Then, a sliding window is moved over the handwriting graph from left to right as illustrated in Figure 7(b). At each position, the graph dissimilarity $d(g_1, g_2) \in \mathbb{R}$ between the subgraph g_1 in the window and the prototype graph g_2 is calculated for all prototypes. This results in a sequence of feature vectors $x_1, \ldots, x_N$ with $x_i = (d(g_{1,i}, g_{2,1}), \ldots, d(g_{1,i}, g_{2,n}))$ and a dimensionality $x_i \in \mathbb{R}^n$ equal to the number n of prototypes.

For calculating the dissimilarity $d(g_1, g_2)$, the use of AED based on the Euclidean cost function was proposed in [57] together with a normalization $\hat{d}(g_1, g_2) = 1 - \frac{d(g_1,g_2)}{d_{max}(g_1,g_2)}$ according to the maximum distance $d_{max}(g_1, g_2)$. The maximum distance corresponds with the costs of deleting all nodes and edges of g_1 and inserting all nodes and edges of g_2. A final normalization $n(g_1, g_2) = \frac{\hat{d}(g_1,g_2)^2}{\sum_P \hat{d}(g_1,p)}$ is performed with respect to the prototype graphs P.

Alternatively, HED can be directly applied instead of AED without changing the underlying recognition framework. Furthermore, since the nodes are labeled with vectorial labels, the Hausdorff distance H (Equation 6 in Section 3) and its variant H' (Equation 7) can be employed directly together with the Euclidean distance if the edges of the graphs are ignored. In that case, only the

Table 5: Handwriting recognition results on the test set.

(a) Runtime and accuracy results. Average runtime in seconds on a 2.66GHz processor for recognizing one word, speedup factor $\times$ of HED when compared with AED, word recognition accuracy in percentage, and accuracy difference Δ of HED when compared with AED.

Database	Runtime [s]			Accuracy [%]		
	AED	HED	$\times$	AED	HED	Δ
Parzival	33.24	2.57	12.9	94.00	93.66	-0.34

(b) Accuracy results of Hausdorff distance H and its variant H' compared with HED.

Database	Accuracy [%]		
	H	H'	HED
Parzival	49.78	83.95	93.66

normalization $n(g_1, g_2) = \frac{d(g_1,g_2)}{\sum_P d(g_1,p)}$ with respect to all prototype graphs P is performed.

After embedding, the resulting feature vector sequence can be used for recognition with any statistical classifier. In this study, hidden Markov models (HMM) are employed for word recognition. For further details on the recognition system as well as the graph similarity features in general, we refer to [57].

5.3. Experimental Setup

First, the word images are divided into three distinct sets for training, validation, and testing. Half of the words are used for training and a quarter of the words for validation and testing, respectively. For vector space embedding, 79 manually selected character prototypes are used.

Parameters that are optimized with respect to the validation accuracy include the distance $D \in \{3, 5, 7, 9\}$ of nodes on the skeleton image, the costs $\tau_n, \tau_e \in \{0, 0.4D, 0.6D, \ldots, 1.4D\}$ of the Euclidean cost function, and some HMM parameters adopted from previous work [57].

5.4. Results

The main handwriting recognition results on the test set are listed in Table 5(a). HED achieves a substantial speedup of one order of magnitude with factor 12.9 when compared with AED. The minor loss in accuracy of -0.34% is statistically not significant (t-test, $\alpha = 0.05$). This is an excellent result for HED endowing the proposed graph-based handwriting recognition system with a high efficiency for real-world applications.

An interesting anomaly is observed for the optimal system parameter values $(D, \tau_n, \tau_e) = (3.0, 3.0, 0.0)$. The cost for edge deletions and insertions $\tau_e = 0.0$ means that using graphs without edges has turned out to be optimal for this

application. Instead of using edges, a dense coverage of the skeleton image with $D = 3.0$ is preferred. We assume this anomaly is caused by broken characters and other types of noise present in skeleton images of historical handwritings due to binarization problems. Imposing an edge cost may lead to larger dissimilarities among graphs of the same class in this case.

For graph matching, the anomaly has two implications. First, AED corresponds with GED because the estimation error only affects implied edge costs (see Section 2.4). That is, the accuracy of HED is compared with the accuracy of the exact edit distance in this experiment. Secondly, the common Hausdorff distance H and its variant H' are valid alternatives to AED and HED because without edges, the graphs are reduced to finite subsets of the Euclidean space.

Accuracy results for H and H' are reported in Table 5(b). A remarkable difference between H and H' is observed. Clearly, taking into account only the maximum among all nearest neighbor distances with H is not well-suited for comparing complex character shapes. A second observation is that HED performs significantly better than H', which motivates further studies on HED as a variant of Hausdorff matching for subsets of a metric space, taking into account not only substitutions but also deletions and insertions.

5.5. Discussion

The graph classification results presented in Sections 4.3 and 5.4 support our claim that HED is a promising quadratic time approximation of GED.

First, HED has proven to be *flexible*. It could be demonstrated that HED is applicable to different types of labeled graphs including graphs with vectorial labels on the nodes (Letters, Parzival), graphs with scalar labels on the edges (Fingerprints), and graphs with symbolic labels on the nodes (Molecules) taking into account directed edges (Fingerprints) as well as undirected edges (Letters, Molecules, Parzival). Indeed, HED is by definition applicable to virtually all types of graphs in the same way as GED since it is based on a general cost function. The flexibility of graph matching techniques is important in many real-world pattern recognition applications that employ different kinds of label alphabets.

Secondly, HED was *efficient* in terms of computational costs. Substantial speedups could be achieved with the quadratic time HED when compared with the exponential time GED and the cubic time AED. In the experiments, we have considered graphs up to a maximum number of about 100 nodes (Molecules) and report speedups of up to two orders of magnitude when compared with AED. The quadratic time complexity of HED motivates an application to much larger graphs in the future, which cannot be matched by AED due to computational limitations.

Thirdly, HED was *fairly accurate*. The recognition accuracy was competitive in all cases. In one case (Fingerprints), the accuracy of AED could even be outperformed significantly for matching graphs representing skeleton images. In one case (Parzival), the loss in recognition accuracy was not statistically significant. In two cases (Letters I, Molecules) only a minor loss of accuracy

had to be taken into account for achieving substantial speedups. However, there were also two cases (Letters II, Letters III) for which the loss in accuracy was considerable and might be too restrictive in a real-world scenario for the trade-off between speed and accuracy.

Besides the recognition accuracy, we have also investigated the ability of HED to approximate the edit distance. In general, HED underestimates GED and it might be interesting to investigate the potential of HED as a heuristic of the A* algorithm in order to speedup the exact computation of GED. The measured approximation error was considerable mounting up to an average of about 50% (Letters II) but the approximation was fairly accurate for similar graphs, which is important for the task of classification.

Overall, it is quite surprising that HED has proven to be a much faster and fairly accurate alternative to GED and AED in the conducted experiments. Two steps are performed to reduce the algorithmic complexity of GED. First, instead of optimizing a complete edit path, only node assignments are optimized with respect to estimated implied edge costs similar to AED. Secondly, instead of optimizing a complete node assignment, only individual nodes are matched thus reducing the complexity from cubic to quadratic time. Clearly, there are situations in which this approach is bound to fail, especially in the case of unlabeled graphs. A promising line of future research in this context might be the integration of global information about the graph structure into each node of the graph in order to support the local assignments.

6. Conclusions

In this paper, we have introduced the Hausdorff edit distance (HED). It is a general method for error-tolerant graph matching designed to approximate graph edit distance (GED) in quadratic time with respect to the nodes of the involved graphs. Similar to the assignment edit distance (AED), which approximates GED in cubic time, nodes and their local structure of one graph are matched with nodes and their local structure of the other graph.

In a series of experimental evaluations on graphs representing letter drawings, fingerprints, molecular compounds, and handwriting, three properties of HED could be demonstrated. First, HED is *flexible*. It can match virtually all types of graphs because it is defined with respect to a general cost function for node and edge edit operations. Secondly, HED is *efficient* in terms of computational costs. Due to the reduction in computational complexity, speedups of up to two orders of magnitude could be achieved when compared with AED for graphs with a maximum number of about 100 nodes. Finally, HED is *fairly accurate*. In all cases, the recognition accuracy for graph classification was competitive when compared with AED and enabled at least a trade-off between speedup and loss in recognition accuracy. In many cases, a substantial speedup was achieved with only a minor loss in accuracy or, in one case, even with a gain in accuracy.

Graphs are very attractive for pattern recognition applications due to their high representational power. Yet, graphs often suffer from their own flexibility.

As GED demonstrates, even basic operations such as the computation of a distance between two graphs can lead to an exponential time complexity when no constraints are imposed on the graph structure or the labels on the nodes and edges. In order to make graph-based representations attractive for a wide range of potential real-world applications, efficient graph matching procedures play a central role. HED is a promising contribution in this context. For a handwriting recognition application, we could demonstrate how the combination of vector space embedding and HED allowed us to use graph-based representations flexibly and efficiently in a field where graphs are only rarely used.

There are several promising lines of future research. First, there might be variants of HED, other than its basic formulation, which are more accurate. Then, it might be interesting to evaluate HED as a heuristic for speeding up the computation of GED. Further, HED could be investigated in the context of much larger graphs that cannot be processed by AED due to computational limitations. Finally, HED has shown some promising results as an alternative to the common Hausdorff distance by taking deletions and insertions into account in addition to substitutions.

Acknowledgments

This work has been supported by the Swiss National Science Foundation fellowship project PBBEP2_141453, the Spanish project TIN2009-14633-C03-03, and the Spanish MICINN under the MIPRCV "Consolider Ingenio 2010" CSD2007-00018 project.

[1] D. Conte, P. Foggia, C. Sansone, M. Vento, Thirty years of graph matching in pattern recognition, Int. Journal of Pattern Recognition and Artificial Intelligence 18 (2004) 265–298.

[2] M. Vento, A one hour trip in the world of graphs, looking at the papers of the last ten years, in: Proc. Int. Workshop on Graph-Based Representations, 2013, pp. 1–10.

[3] A. Kandel, H. Bunke, M. Last (Eds.), Applied Graph Theory in Computer Vision and Pattern Recognition, volume 52 of *Studies in Computational Intelligence*, Springer, 2007.

[4] K. Borgwardt, C. Ong, S. Schönauer, S. Vishwanathan, A. Smola, H.-P. Kriegel, Protein function prediction via graph kernels, Bioinformatics 21 (2005) 47–56.

[5] A. Schenker, M. Last, H. Bunke, A. Kandel, Classification of web documents using graph matching, Int. Journal of Pattern Recognition and Artificial Intelligence 18 (2004) 475–496.

[6] Z. Harchaoui, F. Bach, Image classification with segmentation graph kernels, in: Proc. Int. Conf. on Computer Vision and Pattern Recognition, 2007, pp. 1–8.

[7] J. Llados, E. Marti, J. Villanueva, Symbol recognition by error-tolerant subgraph matching between region adjacency graphs, IEEE Trans. on Pattern Analysis and Machine Intelligence 23 (2001) 1137–1143.

[8] S. Lu, Y. Ren, C. Y. Suen, Hierarchical attributed graph representation and recognition of handwritten Chinese characters, Pattern Recognition 24 (1991) 617–632.

[9] P. Dickinson, H. Bunke, A. Dadej, M. Kraetzl, Matching graphs with unique node labels, Pattern Analysis and Applications 7 (2004) 243–254.

[10] J. R. Ullmann, An algorithm for subgraph isomorphism, Journal of the Association for Computing Machinery 23 (1976) 31–42.

[11] L. Cordella, P. Foggia, C. Sansone, M. Vento, A (sub)graph isomorphism algorithm for matching large graphs, IEEE Trans. on Pattern Analysis and Machine Intelligence 26 (2004) 1367–1372.

[12] R. A. Wagner, M. J. Fischer, The string-to-string correction problem, Journal of the Association for Computing Machinery 21 (1974) 168–173.

[13] W. H. Tsai, K. S. Fu, Error-correcting isomorphism of attributed relational graphs for pattern analysis, IEEE Trans. on Systems, Man, and Cybernetics 9 (1979) 757–768.

[14] A. Sanfeliu, K. S. Fu, A distance measure between attributed relational graphs for pattern recognition, IEEE Trans. on Systems, Man, and Cybernetics 13 (1983) 353–363.

[15] H. Bunke, G. Allermann, Inexact graph matching for structural pattern recognition, Pattern Recognition Letters 1 (1983) 245–253.

[16] R. Ambauen, S. Fischer, H. Bunke, Graph edit distance with node splitting and merging and its application to diatom identification, in: Proc. Int. Workshop on Graph-Based Representations, 2003, pp. 95–106.

[17] M. Neuhaus, H. Bunke, Self-organizing maps for learning the edit costs in graph matching, IEEE Trans. on Systems, Man, and Cybernetics 35 (2005) 503–514.

[18] M. Neuhaus, H. Bunke, Automatic learning of cost functions for graph edit distance, Information Sciences 177 (2007) 239–247.

[19] T. S. Caetano, J. J. McAuley, L. Cheng, Q. V. Le, A. J. Smola, Learning graph matching, IEEE Trans. on Pattern Analysis and Machine Intelligence 31 (2009) 1048–1058.

[20] S. Umeyama, An eigendecomposition approach to weighted graph matching problems, IEEE Trans. on Pattern Analysis and Machine Intelligence 10 (1988) 695–703.

[21] R. Wilson, E. Hancock, B. Luo, Pattern vectors from algebraic graph theory, IEEE Trans. on Pattern Analysis and Machine Intelligence 27 (2005) 1112–1124.

[22] X. Jiang, H. Bunke, Optimal quadratic-time isomorphism of ordered graphs, Pattern Recognition 32 (1999) 1273–1283.

[23] J. Hopcroft, J. Wong, Linear time algorithm for isomorphism of planar graphs, in: Proc. 6th Annual ACM Symposium on Theory of Computing, 1974, pp. 172–184.

[24] M. Neuhaus, H. Bunke, An error-tolerant approximate matching algorithm for attributed planar graphs and its application to fingerprint classification, in: Proc. Int. Workshop on Structural, Syntactic, and Statistical Pattern Recognition, 2004, pp. 180–189.

[25] E. Luks, Isomorphism of graphs of bounded valence can be tested in polynomial time, Journal of Computer and Systems Sciences 25 (1982) 42–65.

[26] A. Torsello, D. Hidovic-Rowe, M. Pelillo, Polynomial-time metrics for attributed trees, IEEE Trans. on Pattern Analysis and Machine Intelligence 27 (2005) 1087–1099.

[27] A. Dumay, R. van der Geest, J. Gerbrands, E. Jansen, J. Reiber, Consistent inexact graph matching applied to labelling coronarysegments in arteriograms, in: Proc. Int. Conf. on Pattern Recognition, volume 3, 1992, pp. 439–442.

[28] S. Berretti, A. Del Bimbo, E. Vicario, Efficient matching and indexing of graph models in content-based retrieval, IEEE Trans. on Pattern Analysis and Machine Intelligence 23 (2001) 1089–1105.

[29] L. Gregory, J. Kittler, Using graph search techniques for contextual colour retrieval, in: Proc. Int. Workshop on Structural, Syntactic, and Statistical Pattern Recognition, 2002, pp. 186–194.

[30] K. Riesen, S. Fankhauser, H. Bunke, Speeding up graph edit distance computation with a bipartite heuristic, in: Proc. Int. Workshop on Mining and Learning with Graphs, 2007, pp. 21–24.

[31] P. E. Hart, N. J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Trans. on Systems, Science, and Cybernetics 4 (1968) 100–107.

[32] R. Wilson, E. Hancock, Structural matching by discrete relaxation, IEEE Trans. on Pattern Analysis and Machine Intelligence 19 (1997) 634–648.

[33] R. Myers, R. Wilson, E. Hancock, Bayesian graph edit distance, IEEE Trans. on Pattern Analysis and Machine Intelligence 22 (2000) 628–635.

[34] A. Cross, R. Wilson, E. Hancock, Inexact graph matching using genetic search, Pattern Recognition 30 (1997) 953–970.

[35] M. Boeres, C. Ribeiro, I. Bloch, A randomized heuristic for scene recognition by graph matching, in: Proc. Int. Workshop on Efficient and Experimental Algorithms, 2004, pp. 100–113.

[36] S. Sorlin, C. Solnon, Reactive tabu search for measuring graph similarity, in: Proc. Int. Workshop on Graph-Based Representations, 2005, pp. 172–182.

[37] M. Neuhaus, K. Riesen, H. Bunke, Fast suboptimal algorithms for the computation of graph edit distance, in: Proc. Int. Workshop on Structural, Syntactic, and Statistical Pattern Recognition, 2006, pp. 163–172.

[38] D. Justice, A. Hero, A binary linear programming formulation of the graph edit distance, IEEE Trans. on Pattern Analysis ans Machine Intelligence 28 (2006) 1200–1214.

[39] K. Riesen, H. Bunke, Approximate graph edit distance computation by means of bipartite graph matching, Image and Vision Computing 27 (2009) 950–959.

[40] J. Munkres, Algorithms for the assignment and transportation problems, Journal of the Society for Industrial and Applied Mathematics 5 (1957) 32–38.

[41] H. Kuhn, The Hungarian method for the assignment problem, Naval Research Logistic Quarterly 2 (1955) 83–97.

[42] S. Fankhauser, K. Riesen, H. Bunke, Speeding up graph edit distance computation through fast bipartite matching, in: Proc. Int. Workshop on Graph-Based Representations, 2011, pp. 102–111.

[43] R. Jonker, T. Volgenant, A shortest augmenting path algorithm for dense and sparse linear assignment problems, Computing 38 (1987) 325–340.

[44] K. Riesen, H. Bunke, Graph Classification and Clustering based on Vector Space Embedding, World Scientific, 2010.

[45] S. Fankhauser, K. Riesen, H. Bunke, P. Dickinson, Suboptimal graph isomorphism using bipartite matching, Int. Journal of Pattern Recognition and Artificial Intelligence 26 (2012).

[46] K. Riesen, S. Emmenegger, H. Bunke, A novel software toolkit for graph edit distance computation, in: Proc. Int. Workshop on Graph-Based Representations, 2013, pp. 142–151.

[47] D. P. Huttenlocher, G. A. Klanderman, G. A. Kl, W. J. Rucklidge, Comparing images using the Hausdorff distance, IEEE Trans. on Pattern Analysis and Machine Intelligence 15 (1993) 850–863.

[48] A. Fischer, C. Suen, V. Frinken, K. Riesen, H. Bunke, A fast matching algorithm for graph-based handwriting recognition, in: Proc. Int. Workshop on Graph-Based Representations, 2013, pp. 194–203.

[49] V. Levenshtein, Binary codes capable of correcting deletions, insertions and reversals, Soviet Physics Doklady 10 (1966) 707–710.

[50] K. Riesen, H. Bunke, IAM graph database repository for graph based pattern recognition and machine learning, in: Proc. Int. Workshop on Structural, Syntactic, and Statistical Pattern Recognition, 2008, pp. 287–297.

[51] C. Watson, C. Wilson, NIST Special Database 4, Fingerprint Database, National Institute of Standards and Technology, 1992.

[52] DTP, AIDS antiviral screen, http://dtp.nci.nih.gov/docs/aids/aids_data.html, 2004.

[53] M. Neuhaus, H. Bunke, A graph matching based approach to fingerprint classification using directional variance, in: Proc. 5th Int. Conf. on Audio- and Video-Based Biometric Person Authentication, 2005, pp. 191–200.

[54] H. Bunke, T. Varga, Off-line Roman cursive handwriting recognition, in: B. Chaudhuri (Ed.), Digital Document Processing, Springer, 2007, pp. 165–173.

[55] L. R. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition, Proceedings of the IEEE 77 (1989) 257–285.

[56] T. Ploetz, G. A. Fink, Markov models for offline handwriting recognition: A survey, Int. Journal on Document Analysis and Recognition 12 (2009) 269–298.

[57] A. Fischer, K. Riesen, H. Bunke, Graph similarity features for HMM-based handwriting recognition in historical documents, in: Proc. Int. Conf. on Frontiers in Handwriting Recognition, 2010, pp. 253–258.

[58] A. Fischer, M. Wüthrich, M. Liwicki, V. Frinken, H. Bunke, G. Viehhauser, M. Stolz, Automatic transcription of handwritten medieval documents, in: Proc. Int. Conf. on Virtual Systems and Multimedia, 2009, pp. 137–142.

[59] A. Fischer, H. Bunke, Character prototype selection for handwriting recognition in historical documents with graph similarity features, in: Proc. European Signal Processing Conference, 2011, pp. 1435–1439.