Fast Suboptimal Algorithms for the Computation of Graph Edit Distance

Michel Neuhaus*, Kaspar Riesen, and Horst Bunke

Institute of Computer Science and Applied Mathematics, University of Bern
Neubrückstrasse 10, CH-3012 Bern, Switzerland
{mneuhaus, riesen, bunke}@iam.unibe.ch

Abstract. Graph edit distance is one of the most flexible mechanisms for error-tolerant graph matching. Its key advantage is that edit distance is applicable to unconstrained attributed graphs and can be tailored to a wide variety of applications by means of specific edit cost functions. Its computational complexity, however, is exponential in the number of vertices, which means that edit distance is feasible for small graphs only. In this paper, we propose two simple, but effective modifications of a standard edit distance algorithm that allow us to suboptimally compute edit distance in a faster way. In experiments on real data, we demonstrate the resulting speedup and show that classification accuracy is mostly not affected. The suboptimality of our methods mainly results in larger inter-class distances, while intra-class distances remain low, which makes the proposed methods very well applicable to distance-based graph classification.

1 Introduction

Graph matching refers to the process of evaluating the structural similarity of graphs. 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, vertices and edges are labeled with arbitrary attributes. One of the most flexible error-tolerant graph matching methods applicable to unconstrained graphs is based on graph edit distance [1]. However, the error-tolerant nature of edit distance — unlike exact graph matching methods such as subgraph isomorphism or maximum common subgraph — potentially allows every vertex of a graph to be mapped to every vertex of another graph. 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. A common way

* Supported by the Swiss National Science Foundation NCCR program *Interactive Multimodal Information Management (IM)2* in the Individual Project *Multimedia Information Access and Content Protection*.

D.-Y. Yeung et al. (Eds.): SSPR&SPR 2006, LNCS 4109, pp. 163–172, 2006.

to make graph matching more efficient is to restrict considerations to special classes of graphs. Examples include the classes of planar graphs [2], bounded-valence graphs [3], trees [4], and graphs with unique vertex labels [5]. A number of graph matching methods based on genetic algorithms have been proposed [6]. Genetic algorithms offer an efficient way to cope with large search spaces, but are non-deterministic and suboptimal. If the structural matching problem is formulated as a vertex labeling problem, relaxation labeling techniques can be used for graph matching [7]. While in some cases such graph matching methods may perform efficiently, it seems to be rather difficult to apply them to strongly distorted data. Recently, a suboptimal edit distance algorithm has been proposed [8] that requires the vertices of graphs to be planarly embedded, which is satisfied in many, but not all computer vision applications of graph matching. In [9], the authors propose an edit distance method based on bipartite matching. The main drawback of their method is that no edge information is used in the bipartite matching step of the algorithm.

In this paper, we address the issue of efficient edit distance computation in a different way. We exploit the fact that exact edit distance algorithms typically explore large areas of the search space that are not relevant for certain classification tasks. We propose simple variants of a standard edit distance algorithm that make the computation substantially faster, but keep the resulting suboptimal distances sufficiently accurate.

2 Graph Edit Distance

The key idea of graph edit distance is to define the dissimilarity of two graphs by the minimal amount of distortion that is needed to transform one graph into the other. The distortion model is defined by a number of underlying vertex and edge edit operations. The most common set of graph edit operations consists of an insertion, a deletion, and a substitution operation on vertices and edges. Given a source and a target graph, the idea is to remove some vertices and edges from the source graph, relabel some of the remaining vertices and edges, and possibly insert some vertices and edges such that eventually the target graph is obtained. A sequence of edit operations that transform the source graph into the target graph is called an *edit path* between source and target graph. Moreover, cost functions are introduced measuring the strength of the distortion caused by each edit operation. These cost functions are used to decide whether an edit path represents weak modifications only or a significant amount of structural distortion. If there exists an inexpensive edit path between two graphs, these graphs are considered structurally similar in terms of the underlying edit operation model and edit cost functions; if no such edit path exists, the graphs are considered dissimilar. Consequently, the *edit distance* of two graphs is defined by the minimum cost edit path between the two graphs [1]. In the following, we denote a graph by $g = (V, E, \mu, \nu)$, where V denotes a finite set of vertices, $E \subseteq V \times V$ a set of directed edges, $\mu : V \to L$ a vertex labeling function assigning each vertex an attribute from L, and $\nu : E \to L$ an edge labeling function. The

substitution of a vertex u by a vertex v is denoted by $u \to v$, the insertion of u by $\varepsilon \to u$, and the deletion of u by $u \to \varepsilon$.

The computation of edit distance is usually carried out by means of a tree search algorithm. Provided that a few weak conditions are satisfied in the definition of edit costs, it is sufficient to consider only a finite number of edit paths to find one with minimum costs. The most widely used method for edit distance computation is based on the A* algorithm [10]. The A* algorithm is a best-first algorithm that attempts to retrieve an optimal path from a search tree based on heuristic information. The idea is to use a search tree to represent the considered optimization problem in a tree data structure, such that the root node represents the starting point, inner nodes correspond to partial solutions, and leaf nodes to complete solutions. A search tree is dynamically constructed at runtime by iteratively creating successor nodes linked by edges to the currently considered node. The A* search algorithm is characterized by a heuristic function that estimates the expected costs of the best route from the root through the current node to a leaf node. At each step during tree traversal, the most promising node — the one with the lowest heuristic cost value — from the set of nodes to be processed is chosen. Formally, for a node of the search tree p, we use $g(p)$ to denote the costs of the optimal path from the root node to the current node p found by A* so far and $h(p)$ to denote the estimated costs from p to a leaf node. The sum $g(p) + h(p)$ gives the heuristic assessment of node p. If the estimated costs $h(p)$ are always lower than, or equal to, the real costs, the algorithm is known to be admissible, that is, an optimal path from the root node to a leaf node is guaranteed to be found by this procedure [10].

In graph edit distance, unlike exact graph matching algorithms, vertices of the source graph can potentially be mapped to any vertex of the target graph. Given two graphs, the A* search tree for edit distance is constructed by considering vertices of the first graph one after the other. An A* algorithm for the computation of graph edit distance is given in Alg. 1. Let us assume that the vertices of the first graph are processed in the order $(u_1, u_2, \ldots)$. All possible edit operations are constructed simultaneously for each vertex, that is, the removal of the vertex (line 12) or the substitution of the vertex by any unprocessed vertex of the second graph (line 11), which produces a number of successor nodes in the search tree. Note that edit operations on edges are implied by edit operations on their adjacent vertices. If all vertices of the first graph have been processed, the remaining vertices of the second graph can be inserted into the graph in a single step (line 14). The set of partial edit paths OPEN consists of the search tree nodes to be considered in the next step. The currently most promising node p of the search tree, or partial edit path, is the one minimizing the A* search costs $g(p)+h(p)$ (line 5). When a complete edit path is obtained in this way, it is guaranteed to be an optimal one and is returned as the solution (line 7). In cases where the edit distance computation takes longer than a predefined threshold, the corresponding distance is set to infinity.

The function $g(p)$ measuring the costs from the root node to the current node p is simply set equal to the cost of the partial edit path accumulated so far. In

Algorithm 1. Computation of graph edit distance by A* 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. $p_{min} = \{u_1 \to v_3, u_2 \to \varepsilon, \ldots, \varepsilon \to v_6\}$

1: Initialize OPEN to the empty set
2: For each vertex $w \in V_2$, insert the substitution $\{u_1 \to w\}$ into OPEN
3: Insert the deletion $\{u_1 \to \varepsilon\}$ into OPEN
4: **loop**
5: Remove $p_{min} = \arg\min_{p \in \text{OPEN}} \{g(p) + h(p)\}$ from OPEN
6: **if** p_{min} is a complete edit path **then**
7: Return p_{min} as the solution
8: **else**
9: Let $p_{min} = \{u_1 \to v_{i_1}, \ldots, u_k \to v_{i_k}\}$
10: **if** $k < |V_1|$ **then**
11: For each $w \in V_2 \setminus \{v_{i_1}, \ldots, v_{i_k}\}$, 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 $p_{min} \cup \bigcup_{w \in V_2 \setminus \{v_{i_1}, \ldots, v_{i_k}\}} \{\varepsilon \to w\}$ into OPEN
15: **end if**
16: **end if**
17: **end loop**

the simplest scenario, the estimated lower bound $h(p)$ of the costs from p to a leaf node is set to zero for all p. This means that no heuristic information of the potentially best search direction is used at all, and one actually performs a breadth-first search. In the remainder of this paper, this method will be referred to as PLAIN-A*. The other extreme would be to compute for a partial edit path the actual optimal path to a leaf node, that is, 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 the least [11]. The method we use in this paper is very intuitive and can be computed efficiently. In the following, assume that a partial edit path at a position in the search tree is given, and let the number of unprocessed vertices of the first graph g_1 and second graph g_2 be n_1 and n_2, respectively. For an efficient estimation of the optimal remaining edit operations, we first attempt to perform as many vertex substitutions as possible, since a substitution is often less expensive than a deletion followed by an insertion. To this end, we potentially substitute each of the n_1 vertices from g_1 with any of the n_2 vertices from g_2. To obtain a lower bound of the exact edit costs, we accumulate the costs of the $\min\{n_1, n_2\}$ least expensive of these vertex substitutions and the costs of $\max\{0, n_1 - n_2\}$ vertex deletions or $\max\{0, n_2 - n_1\}$ vertex insertions. Any of

the selected substitutions that is more expensive than a deletion followed by an insertion operation is replaced by the latter. This procedure only considers the most optimistic way to edit the remaining part of g_1 into the remaining part of g_2, and the estimated costs therefore constitute a lower bound of the exact cost. In the following, we refer to this method as HEURISTIC-A*.

3 Fast Suboptimal Edit Distance Algorithms

The methods described in the previous section find an optimal edit path between two graphs. Unfortunately, the computational complexity of the edit distance algorithm, whether or not heuristics are used to govern the tree traversal process, is exponential in the number of vertices of involved graphs. This means that the running time and space complexity may be huge even for reasonably small graphs. In practice we are able to compute the edit distance of graphs typically containing 12 vertices at most. In this paper, we therefore propose two edit distance variants that are conceptually very simple, but lead to a significant speedup of the computation. These methods do not generally return the optimal edit path, but only a suboptimal one.

3.1 A*-Beamsearch

The first method is based on beam search. Instead of expanding all sucessor 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 the OPEN set in Alg. 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. This means that not the full search space is explored, but only those nodes are expanded that belong to the most promising partial matches. For similar graphs, it is clear that edit operations of an optimal path have low costs. Therefore if only the partial edit paths with lowest costs are considered, we will obtain an edit path that is nearly optimal, which will result in a suboptimal distance close to the exact distance. For dissimilar graphs, the suboptimal distance will remain large. In the following, this method with parameter s is referred to as PLAIN-A*-BEAMSEARCH(s) or HEURISTIC-A*-BEAMSEARCH(s), respectively, depending on whether or not heuristic information is used in the tree search procedure.

3.2 A*-Pathlength

In the second variant, we exploit an observation from edit distance systems in practice. If graphs with a rather large number of vertices are given, it may very well be that a considerable part of an optimal edit path is constructed in the first few steps of the tree traversal, because most substitutions between similar graphs have small costs. Whenever the first significantly more expensive edit operation occurs (in the optimal edit path), this node will prevent the tree search algorithm from quickly reaching a leaf node and unnecessarily make it expand

a large part of the search tree. We therefore propose an additional weighting factor favoring long partial edit paths over shorter ones. Formally, instead of evaluating $g(p) + h(p)$ in Alg. 1 (line 5), we use

$$\frac{g(p) + h(p)}{t^{|p|}} \quad ,$$

with parameter $t > 1$. The term $|p|$ denotes the number of edit operations in partial edit path p. We refer to this method as PLAIN-A*-PATHLENGTH(t) or HEURISTIC-A*-PATHLENGTH(t), respectively.

4 Experimental Results

The methods we propose for speeding up the computation of graph edit distance are suboptimal in the sense that only an approximate edit distance value is obtained. In fact, from the description above it is clear that the approximate distance value will be equal to, or larger than, the exact distance value, since the suboptimal methods find an optimal solution in a subspace of the complete search space. In this section, we measure the speedup of the suboptimal methods and analyze the accuracy of the suboptimal distance.

To address the classification problems considered in this paper, we apply k-nearest-neighbor classifiers in conjunction with edit distance. Given a labeled set of training graphs, an unknown graph is assigned to the class that occurs most frequently among the k closest graphs (in terms of edit distance) from the training set. Hence, we assume that graphs belonging to the same class should be similar. In the experiments, insertion and deletion costs are set to constant values, and substitution costs are set proportional to the Euclidean distance of involved labels. To optimize these edit cost parameters, we first determine a set of parameters that is optimal on a validation set. The validated parameters are then applied to the independent test set. Note that the parameters are optimized once for the exact distance and then used throughout all optimal and suboptimal computations.

We first evaluate the distances on a graph database representing distorted letter drawings. In this experiment, we consider the 15 capital letters that consist of straight lines only (*A, E, F,* ...). For each class, a prototype line drawing is manually constructed. We then apply distortion operators to the prototype line drawings, resulting in randomly shifted, removed, or added lines. Using this procedure, we are able to generate arbitrarily large sample sets of drawings with arbitrarily strong distortions. These drawings are then converted into graphs by representing ending points of lines by vertices and lines by edges. Each vertex is labeled with a two-dimensional attribute giving its position. The graph database used in our experiments consists of a training set, a validation set, and a test set, each of size 150. The letter graphs consist of 4.6 nodes and 4.4 edges on the average.

To obtain a visual representation of the accuracy of the suboptimal methods, we plot for each pair of test and training pattern its exact (horizontal axis)

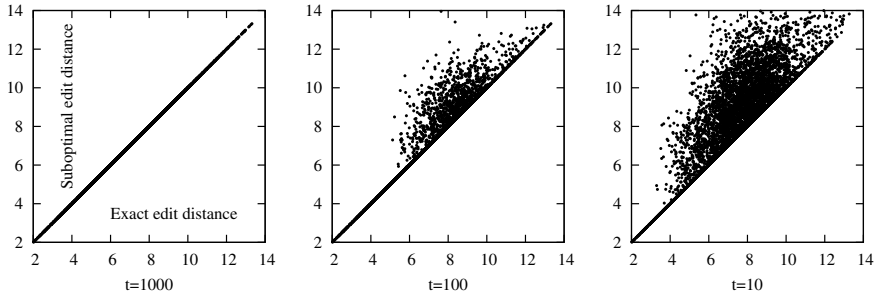

Fig. 1. Distance accuracy of PLAIN-A*-BEAMSEARCH(t) for $t = 1000, 100, 10$

and suboptimal (vertical axis) distance value. The respective illustrations are shown in Fig. 1. For $t = 1000$, we find that the suboptimal method does not differ considerably from the exact method in terms of distance. If the suboptimal method is constrained to $t = 100$ or $t = 10$ items in the OPEN list, however, it often results in larger distances. Additionally evaluating the running time of the edit distance computation, we observe that the suboptimal methods (for $t = 10, 100$) are faster than the exact method by several orders of magnitude.

The crucial question is whether the larger distances belong to graphs of the same class or graphs from different classes. In the latter case, the increased inter-class distance will not negatively affect the classification accuracy. In Table 1 we give the classification accuracy of three nearest-neighbor classifiers and the average time it takes to compute a single edit distance. The traditional edit distance algorithms are denoted by PLAIN-A* and HEURISTIC-A*, and the suboptimal methods proposed in this paper are referred to as PLAIN-A*-BEAMSEARCH, PLAIN-A*-PATHLENGTH, HEURISTIC-A*-BEAMSEARCH, and HEURISTIC-A*-PATHLENGTH. It turns out that the speedup of the suboptimal methods is significant, while the accuracy remains high for most configurations. The speedup of PLAIN-A*-BEAMSEARCH for decreasing parameter is clearly visible. Concerning the accuracy, suboptimal methods can even be observed to outperform the two exact methods in some cases. This means that a suboptimal algorithm may be able to correct misclassifications by assigning higher costs to pairs of graphs from different classes than the exact algorithm. The suboptimal method PLAIN-A*-PATHLENGTH(1.05) achieves the best classification accuracy of 86.7% among all methods and is more than 3 times, or 16 times, respectively, faster than the exact methods. Note that the performance of the two exact methods need not be identical, since in some cases the running time of the faster HEURISTIC-A* may be below and that of the slower PLAIN-A* above the predefined timeout threshold.

For a more thorough evaluation of the classification accuracy, we apply the proposed methods to the problem of image classification. Images are converted into attributed graphs by segmenting them into regions, eliminating regions that are irrelevant for classification, and representing the remaining regions by ver-

Table 1. Letter Database: Classification accuracy and average running time

Method	1-NN	3-NN	5-NN	Time (ms)
PLAIN-A*	82.0	80.7	81.3	2200
PLAIN-A*-BEAMSEARCH(1000)	82.0	80.7	82.7	620
PLAIN-A*-BEAMSEARCH(100)	81.3	79.3	81.3	40
PLAIN-A*-BEAMSEARCH(10)	76.7 ∘	74.7 ∘	72.0 ∘	13
PLAIN-A*-PATHLENGTH(1.05)	79.3	80.0	86.7	132
PLAIN-A*-PATHLENGTH(1.1)	77.3	79.3	82.7	2
HEURISTIC-A*	82.0	80.7	82.7	468
HEURISTIC-A*-BEAMSEARCH(100)	82.0	80.7	82.0	18
HEURISTIC-A*-PATHLENGTH(1.1)	79.3	82.7	84.0	8

∘ Statistically significantly worse than PLAIN-A* and HEURISTIC-A* ($\alpha = 0.05$)

Table 2. Image Database: Classification accuracy and average running time

Method	1-NN	3-NN	5-NN	Time (ms)
PLAIN-A*	46.3	48.2	44.4	10
PLAIN-A*-BEAMSEARCH(10)	46.3	48.2	48.2	8
PLAIN-A*-BEAMSEARCH(5)	48.2	50.0	44.4	6
PLAIN-A*-PATHLENGTH(1.1)	48.2	50.0	46.3	5
HEURISTIC-A*	46.3	48.2	44.4	20
HEURISTIC-A*-BEAMSEARCH(10)	46.3	44.4	48.2	16
HEURISTIC-A*-PATHLENGTH(1.1)	50.0	48.2	51.9	15

Table 3. Fingerprint Database: Classification accuracy and average running time

Method	1-NN	3-NN	5-NN	Time (ms)	
Approximate method [13]	82.6	83.8	84.4	11	
PLAIN-A*	—	—	—	—	[1]
PLAIN-A*-BEAMSEARCH(50)	87.4 •	87.8 •	87.6 •	167	
PLAIN-A*-BEAMSEARCH(40)	85.6 •	88.2 •	88.0 •	74	
PLAIN-A*-BEAMSEARCH(10)	72.0 ∘	72.8 ∘	72.4 ∘	9	
PLAIN-A*-PATHLENGTH(...)	—	—	—	—	[1]
HEURISTIC-A*	—	—	—	—	[1]
HEURISTIC-A*-BEAMSEARCH(50)	87.4 •	87.8 •	87.6 •	218	
HEURISTIC-A*-PATHLENGTH(...)	—	—	—	—	[1]

∘ Statistically significantly worse than reference method [13] ($\alpha = 0.05$)
• Statistically significantly better than reference method [13] ($\alpha = 0.05$)
[1] Empty entries indicate computation failure due to lack of memory

tices and the adjacency of regions by edges [12]. Our image database consists of 5 classes (*city, countryside, people, snowy, streets*) and is split into a training set, a validation set, and a test set of size 54. On the average, the graphs consist of 2.8 nodes and 2.5 edges. The nearest-neighbor classification performance and the running time of the edit distance computation using the exact algorithms and the proposed suboptimal algorithms are given in Table 2. Note that in this application HEURISTIC-A* takes significantly longer for the edit distance computation than PLAIN-A*. This means that the computational overhead of the heuristic evaluation of future costs in the search tree cannot be compensated for by a faster tree traversal, mostly because the graphs under consideration, and hence also the constructed search tree, are rather small. Generally, the decrease of the running time is not massive, but the accuracy of the suboptimal methods is at least as high as that of the exact methods. Particularly PLAIN-A*-PATHLENGTH(1.1) outperforms the exact methods PLAIN-A* and HEURISTIC-A* and is at least twice as fast.

Finally, we apply the proposed methods to the difficult problem of fingerprint classification. To this end, we construct graphs from fingerprint images of the NIST-4 database by extracting characteristic regions in fingerprints and converting the result into attributed graphs [13]. We use a validation set of size 300 and a training set and test set both of size 500. On the average, the fingerprint graphs consist of 5.2 nodes and 8.6 edges. In our experiment, we address the 4-class problem (classes *arch, left loop, right loop, whorl*). In Table 3, in addition to the systems described in this paper, we also give the results of another method [13]. Note that for this dataset, the exact edit distance PLAIN-A* and HEURISTIC-A* cannot be computed because the search tree grows too large. The results clearly demonstrate that the classification accuracy of the suboptimal methods, for moderate running times, is very high.

Summarizing we conclude that although the edit distance computed by the proposed suboptimal methods is not always close to the exact edit distance, this problem mainly pertains to pairs of graphs from different classes and therefore does not negatively affect the classification performance. The suboptimal methods offer more flexibility in terms of tradeoff between speed and accuracy than the exact edit distance.

5 Conclusions

One of the main problems of graph edit distance is its exponential computational complexity, which makes its application feasible for small graphs only. In this paper, we propose two simple variants of a standard tree search algorithm for edit distance. The idea is to explore not the full search space, but only a subspace of promising candidates. The two proposed methods are related to beam search and to a re-weighting of edit operation costs. With these simple modifications, it turns out that a significant speedup of the edit distance computation can be achieved. At the same time, the classification accuracy of the suboptimal methods remains high on all datasets — and is sometimes even higher than the one of the exact method. This means that the suboptimality mainly leads to an increase of inter-class distances, while intra-class distances, which are highly relevant for classification, are not strongly affected. We provide an experimental evaluation and demonstrate the usefulness of our methods on semi-artificial line drawings, on scenery images, and fingerprints.

References

1. Sanfeliu, A., Fu, K.: A distance measure between attributed relational graphs for pattern recognition. IEEE Transactions on Systems, Man, and Cybernetics (Part B) **13** (1983) 353–363
2. Hopcroft, J., Wong, J.: Linear time algorithm for isomorphism of planar graphs. In: Proc. 6th Annual ACM Symposium on Theory of Computing. (1974) 172–184
3. Luks, E.: Isomorphism of graphs of bounded valence can be tested in ploynomial time. Journal of Computer and Systems Sciences **25** (1982) 42–65

4. Torsello, A., Hidovic-Rowe, D., Pelillo, M.: Polynomial-time metrics for attributed trees. IEEE Transactions on Pattern Analysis and Machine Intelligence **27** (2005) 1087–1099
5. Dickinson, P., Bunke, H., Dadej, A., Kraetzl, M.: On graphs with unique node labels. In: Proc. 4th Int. Workshop on Graph Based Representations in Pattern Recognition. LNCS 2726, Springer (2003) 13–23
6. Cross, A., Wilson, R., Hancock, E.: Inexact graph matching using genetic search. Pattern Recognition **30** (1997) 953–970
7. Christmas, W., Kittler, J., Petrou, M.: Structural matching in computer vision using probabilistic relaxation. IEEE Transactions on Pattern Analysis and Machine Intelligence **17** (1995) 749–764
8. Neuhaus, M., Bunke, H.: An error-tolerant approximate matching algorithm for attributed planar graphs and its application to fingerprint classification. In: Proc. 10th Int. Workshop on Structural and Syntactic Pattern Recognition. LNCS 3138, Springer (2004) 180–189
9. Hlaoui, A., Wang, S.: A node-mapping-based algorithm for graph matching (2006) To appear.
10. Hart, P., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions of Systems, Science, and Cybernetics **4** (1968) 100–107
11. Bunke, H., Allermann, G.: Inexact graph matching for structural pattern recognition. Pattern Recognition Letters **1** (1983) 245–253
12. Le Saux, B., Bunke, H.: Feature selection for graph-based image classifiers. In: Proc. 2nd Iberian Conf. on Pattern Recognition and Image Analysis. LNCS 3523, Springer (2005) 147–154
13. Neuhaus, M., Bunke, H.: A graph matching based approach to fingerprint classification using directional variance. In: Proc. 5th Int. Conf. on Audio- and Video-Based Biometric Person Authentication. LNCS 3546, Springer (2005) 191–200