A Hausdorff Heuristic for Efficient Computation of Graph Edit Distance

Andreas Fischer[1], Réjean Plamondon[1], Yvon Savaria[1], Kaspar Riesen[2], and Horst Bunke[3]

[1] Département de Génie Électrique, École Polytechnique de Montréal
2900 boul. Édouard-Montpetit, Montréal, Québec H3T 1J4, Canada
{andreas.fischer,rejean.plamondon,yvon.savaria}@polymtl.ca

[2] Institute for Informations Systems, University of Applied Sciences and Arts Northwestern Switzerland, Riggenbachstrasse 16, 4600 Olten, Switzerland
kaspar.riesen@fhnw.ch

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

Abstract. Graph edit distance is a flexible and powerful measure of dissimilarity between two arbitrarily labeled graphs. Yet its application is limited by the exponential time complexity involved when matching unconstrained graphs. We have recently proposed a quadratic-time approximation of graph edit distance based on Hausdorff matching, which underestimates the true distance. In order to implement verification systems for the approximation algorithm, efficiency improvements are needed for the computation of the true distance. In this paper, we propose a Hausdorff heuristic that employs the approximation algorithm itself as a heuristic function for efficient A* computation of the graph edit distance. In an experimental evaluation on several data sets of the IAM graph database, substantial search space reductions and runtime speedups of one order of magnitude are reported when compared with plain A* search.

Keywords: Graph matching, graph edit distance, A* search, Hausdorff distance.

1 Introduction

Graphs are one of the most general data structures in pattern recognition for representing objects. Individual parts of the objects are represented with nodes which are linked with edges to represent binary relationships. Both nodes and edges can be labeled with attributes, for instance in form of feature vectors. This high representational power of graphs has proven successful in pattern recognition and led to widespread applications [1, 2], for example in bioinformatics [3], image classification [4], and computer network analysis [5].

The complexity of the graph data structure usually leads to a high computational complexity when matching two objects. Therefore, many graph matching algorithms impose certain constraints on the graphs. For example spectral methods [6, 7], which are based on efficient eigendecomposition of the adjacency or Laplacian matrix of a graph, primarily target unlabeled graphs or allow only severely constrained label alphabets. Other examples include restrictions to ordered graphs [8] and graphs with unique node labels [9].

Graph edit distance (GED) [10] is a flexible measure of dissimilarity between two graphs, which is able to cope with unconstrained graphs. In particular, arbitrary labels are allowed on both nodes and edges. Originally proposed for string matching [11], the concept of edit distance is to apply a series of edit operations to one object in order to transform it into the other. The edit distance then corresponds with the minimum cost among all possible edit paths.

However, the flexibility of GED comes at the cost of an exponential time complexity with respect to the graph size. The search space of all possible edit paths is usually traversed with a best-first A* search [12]. By using a heuristic function to estimate the future cost of an incomplete edit path, the efficiency of the search procedure can be greatly improved [13–15] but the computational complexity remains the same.

In order to overcome the limitation of exponential time complexity, polynomial approximation of GED is a promising line of current research. In [16], the Hungarian algorithm [17] is used to obtain a cubic-time approximation of GED by assigning nodes and their local edge structure of one graph to nodes and their local edge structure of the other graph. Although only local structure is considered, a high approximation quality is achieved and a strong performance is reported for the task of pattern classification on different graph data sets [16].

Following the same idea of matching nodes and their local edge structure, we have recently proposed an even faster quadratic-time approximation of GED in [18, 19] based on Hausdorff matching [20]. Similar to the comparison of finite subsets of a metric space by means of Hausdorff distance, each node of one graph is compared with each node of the other graph only once to determine its best matching cost, hence the quadratic time complexity. As expected, the deviation from the true edit distance has proven to be larger when compared with the cubic-time approximation. Still, the proposed Hausdorff edit distance (HED) has achieved promising results for the task of pattern classification on diverse graph data sets [18, 19]. It combines the high flexibility of GED to cope with unconstrained graphs with a low quadratic time complexity, which makes HED applicable to a wide range of real-world applications.

So far, a direct comparison of the proposed approximation algorithms with the true edit distance could only be performed for relatively small graphs due to the exponential time complexity of GED. There is a need to improve the efficiency of GED for verification experiments, which can measure the approximation quality in the case of larger graphs observed in many real-world applications. As mentioned above, a common approach to improve the efficiency is the development of accurate heuristic functions for A* computation of GED, which can greatly reduce runtime and memory usage by avoiding a complete traversal of the exponential search space of all possible edit paths. Note that only admissible heuristic functions, which underestimate the true edit distance, can be used for exact computation of GED. Suboptimal variants of A* search like Bayesian A* search [21, 22] do not guarantee a globally optimal solution. Instead, they are interesting for approximating GED as suggested in [23] for beam search and weighted path length search.

In this paper, we propose a Hausdorff heuristic based on HED to improve the efficiency of GED. Since HED underestimates the true edit distance, it is an admissible heuristic function for A* search. The performance of the proposed heuristic is experimentally evaluated on several data sets from the IAM graph database [24] and is compared with plain A* search. Substantial search space reductions and runtime speedups of one order of magnitude are reported.

The remainder of this paper is organized as follows. First, HED is reviewed in Section 2. Afterwards, the proposed Hausdorff heuristic is presented in Section 3 and experimental results are reported in Section 4. Finally, we draw conclusions in Section 5.

2 Hausdorff Edit Distance

In this section, we review the Hausdorff edit distance (HED) [18, 19]. After providing some basic definitions in Section 2.1, HED is defined in Section 2.2.

2.1 Basic Definitions

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, and $\nu : E \to L_E$ is the edge labeling function. L_V and L_E are label sets for nodes and edges, for instance symbolic labels $\{\alpha, \beta, \gamma, \ldots\}$ or the vector space $\mathbb{R}^n$.

Given two graphs $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$, *edit operations* transform nodes and edges of g_1 into nodes and edges of g_2. Three node edit operations are usually considered for $u \in V_1$ and $v \in V_2$, namely *substitutions* $(u \to v)$, *deletions* $(u \to \epsilon)$, and *insertions* $(\epsilon \to v)$. The same set of edit operations is considered for edges $p \in E_1$ and $q \in E_2$, *i.e.* substitutions $(p \to q)$, deletions $(p \to \epsilon)$, and insertions $(\epsilon \to q)$.

A *cost function* $\mathcal{C}$ assigns non-negative costs to node and edge edit operations. An example for Euclidean labels is the Euclidean cost function with substitution cost $\mathcal{C}(u \to v) = ||\mu_1(u) - \mu_2(v)||$ and $\mathcal{C}(p \to q) = ||\nu_1(p) - \nu_2(q)||$. The cost for deletion and insertion is often set to a constant value. Without loss of generality, we will assume $\mathcal{C}(u \to \epsilon) = \mathcal{C}(\epsilon \to v) = C_n$ and $\mathcal{C}(p \to \epsilon) = \mathcal{C}(\epsilon \to q) = C_e$ in the following for all types of cost functions.

2.2 HED Definition

The Hausdorff edit distance $HED(g_1, g_2, \mathcal{C})$ between two graphs g_1 and g_2 is defined with respect to the cost function $\mathcal{C}$ as follows:

$$HED(g_1, g_2, \mathcal{C}) = \sum_{u \in V_1} \min_{v \in V_2 \cup \{\epsilon\}} f(u, v, \mathcal{C}) + \sum_{v \in V_2} \min_{u \in V_1 \cup \{\epsilon\}} f(u, v, \mathcal{C}) \quad (1)$$

It consists of two summation terms that each calculate nearest neighbor distances between the two node sets similar to the Hausdorff distance between finite subsets of a metric space. Nearest neighbors are determined with respect to the node function $f(u, v, \mathcal{C})$, which is defined as

$$f(u, v, \mathcal{C}) = \begin{cases} C_n + \sum_{i=1}^{|P|} \frac{C_e}{2} & \text{for node deletion } (u \rightarrow \epsilon) \\ C_n + \sum_{i=1}^{|Q|} \frac{C_e}{2} & \text{for node insertion } (\epsilon \rightarrow v) \\ \frac{\mathcal{C}(u \rightarrow v) + \frac{HED(P,Q,\mathcal{C})}{2}}{2} & \text{for node substitution } (u \rightarrow v) \end{cases} \tag{2}$$

where P is the set of edges adjacent to u and Q is the set of edges adjacent to v. In the case of deletions, the node deletion cost C_n and half of the implied edge deletion cost is accumulated. Node insertion costs are obtained accordingly. In the case of substitution, half of the substitution cost is considered, which itself consists of the node substitution cost $\mathcal{C}(u \rightarrow v)$ and half of the implied edge cost.

In order to obtain an estimate of the implied edge cost, the edge sets P and Q are matched in the same manner as the node sets, *i.e.* based on a Hausdorff edit distance

$$HED(P, Q, \mathcal{C}) = \sum_{p \in P} \min_{q \in Q \cup \{\epsilon\}} g(p, q, \mathcal{C}) + \sum_{q \in Q} \min_{p \in P \cup \{\epsilon\}} g(p, q, \mathcal{C}) \tag{3}$$

with the corresponding edge function

$$g(p, q, \mathcal{C}) = \begin{cases} C_e & \text{for edge deletion } (p \rightarrow \epsilon) \\ C_e & \text{for edge insertion } (\epsilon \rightarrow q) \\ \frac{\mathcal{C}(p \rightarrow q)}{2} & \text{for edge substitution } (p \rightarrow q) \end{cases} \tag{4}$$

The two divisions by 2 for node substitutions in Equation 2 ensure that HED approximates GED. Only half of the substitution cost is considered because the substitutions, which are not required to be bidirectional, appear in both summation terms in Equation 1. Only half of the implied edge cost is considered because each edge edit operation is implied by exactly two nodes. In effect, an optimal edit cost is assigned to each node without taking the assignments of the other nodes into account. Therefore HED is always less than or equal to GED.

In order to limit the underestimation, lower bounds are used with respect to the number of elements in the set. For $HED(g_1, g_2, \mathcal{C})$, we use a lower bound of $||V_1| - |V_2|| \cdot C_n$ and for $HED(P, Q, \mathcal{C})$, we use a lower bound of $||P| - |Q|| \cdot C_e$. For more details on HED, we refer to [18, 19].

3 Hausdorff Heuristic

In this section, a Hausdorff heuristic based on HED is presented for efficient A* computation of GED. First, GED computation is discussed in Section 3.1. Afterwards, the integration of HED as a heuristic into the A* search algorithm is detailed in Section 3.2.

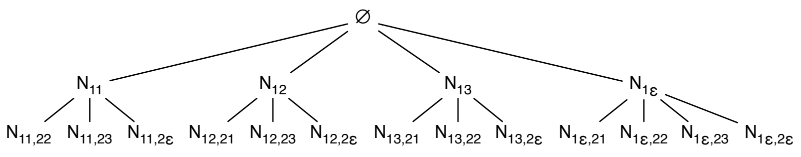

Fig. 1. GED search tree

3.1 GED Computation

The search space for GED is usually spanned by all possible node edit operations that transform V_1 into V_2. Edge edit operations are implied by the node edit operations as soon as both nodes of an edge in g_1 have been assigned to two nodes in g_2.

An example is shown in Figure 1 for $V_1 = \{u_1, u_2\}$ and $V_2 = \{v_1, v_2, v_3\}$. The root of the search tree is an empty node assignment $\emptyset$. At the first level, the first node of V_1 is either assigned to one of the nodes in V_2 or it is deleted. At the second level, the second node of V_1 is assigned in the same way considering all remaining nodes in V_2. At this leaf level, all remaining node insertions are added to the node assignment N. Clearly, the number of entries in the tree is exponential with respect to the number of nodes of the graphs.

Using A* best-first search, the non-expanded node assignments are kept in a sorted *open* list, which is ordered by the cost function

$$f(N) = g(N) + h(N) \tag{5}$$

where $g(N)$ is the cost of all current node edit operations and implied edge edit operations, and $h(N)$ is a heuristic function that estimates the future cost of the node assignment. Admissible heuristics are less than or equal to the real cost. At each step of the search, the currently best node assignment from *open* with the lowest cost function $f(N)$ is removed and its successors are added to *open*. As soon as the currently best node assignment is complete, *i.e.* it transforms V_1 into V_2, the cost of the assignment is returned as GED.

3.2 HED Heuristic

The proposed Hausdorff heuristic estimates the future cost of a node assignment N by means of HED. We consider the subgraph g_1' of g_1 that contains all free nodes $F_1 \subseteq V_1$ according to N and the subgraph g_2' of g_2 that contains all free nodes $F_2 \subseteq V_2$. Then, we calculate the heuristic function

$$h(N) = HED(g_1', g_2', \mathcal{C}) \tag{6}$$

with respect to the underlying cost function $\mathcal{C}$. Because HED underestimates the edit distance between g_1' and g_2', the heuristic function $h(N)$ underestimates the future cost of the node assignment N and is therefore an admissible heuristic for A* computation of GED.

Algorithm 1. Hausdorff heuristic

```
Require: graphs $g_1, g_2$, cost function $\mathcal{C}$, node assignment $N$
Ensure: minimum future cost $c$
 1: for all free nodes $u \in F_1 \subseteq V_1$ according to $N$ do
 2:     $c_1(u) \leftarrow f(u, \epsilon, \mathcal{C})$
 3: end for
 4: for all free nodes $v \in F_2 \subseteq V_2$ according to $N$ do
 5:     $c_2(v) \leftarrow f(\epsilon, v, \mathcal{C})$
 6: end for
 7: for all nodes $u$ in $F_1$ do
 8:     for all nodes $v$ in $F_2$ do
 9:         $cost \leftarrow f(u, v, \mathcal{C})$
10:         $c_1(u) \leftarrow \min(cost, c_1(u))$
11:         $c_2(v) \leftarrow \min(cost, c_2(v))$
12:     end for
13: end for
14: $cost \leftarrow \sum_{u \in F_1} c_1(u) + \sum_{v \in F_2} c_2(v)$
15: return $\max(cost, ||F_1| - |F_2|| \cdot C_n)$
```

A straight-forward computation of Equation 6 is detailed in Algorithm 1. First, the minimum edit costs of each free node $u \in F_1$ and $v \in F_2$ are calculated in lines 1-13. Then, line 14 performs the summation according to Equation 1 and line 15 applies the lower bound (see Section 2.2).

We would like to point out two implementation details, which are important for efficiency. First, the sorted *open* list of the A* search is implemented as a binary search tree that allows to add new elements with $O(\log(n))$ time complexity. Secondly, the function $f(u, v, \mathcal{C})$ from Equation 2 is independent from the node assignment N. Therefore it is pre-computed before the A* search is executed, which leads to an efficient quadratic time complexity of $O(|F_1| \cdot |F_2|)$ for Algorithm 1 with a low constant factor. In particular, the time complexity is independent from the number of edges adjacent to the nodes.

4 Experimental Evaluation

In this section, we report experimental results achieved with the proposed Hausdorff heuristic. The results are compared with plain A* search as a reference, *i.e.* GED computation with the trivial heuristic $h(N) = 0$.

In the following, the selection of suitable graphs is discussed in Section 4.1 and performance results are provided in Section 4.2.

4.1 Graph Selection

Several data sets from the IAM graph database [24] are considered that differ in object domain and graph structure. First, the *Letter I-III* data sets containing graphs of letter drawings, which are artificially distorted with three distortion degrees. Secondly, the *Fingerprints* data set containing graphs of fingerprints from

Table 1. Data set statistics. Number of selected graphs, median number of nodes and edges, minimum and maximum number of nodes.

Data Set	Graphs	$\lvert V\rvert_{med}$	$\lvert E\rvert_{med}$	$\lvert V\rvert_{min}$	$\lvert V\rvert_{max}$
Letters I	200	6	4	4	8
Letters II	200	6	4	4	9
Letters III	200	5	5	4	8
Fingerprints	186	4	6	2	8
Molecules I	95	8	7	4	9
Molecules II	103	9	8	3	9

Table 2. Search space reduction. Average size of the *open* list after A* search.

Data Set	Reference	Hausdorff Heuristic	Reduction Factor
Letters I	568.2	27.3	20.8
Letters II	2,829.0	300.3	9.4
Letters III	2,955.1	386.0	7.7
Fingerprints	6,282.3	2,104.8	3.0
Molecules I	61,994.3	6,094.3	10.2
Molecules II	111,852.1	18,911.7	5.9

the NIST-4 reference database [25]. Thirdly, the *Molecules I* data set based on molecular compounds from the Chemical Carcinogenesis Research Information System (CCRIS) database [26]. And finally, the *Molecules II* data set which is based on molecular compounds from the AIDS Antiviral Screen Database of Active Compounds [27]. Cost functions and their parameter values are adopted from previous work [16, 19].

Due to the exponential time complexity of GED, only relatively small graphs can be included in the evaluation. Besides the runtime, the required memory space is also a limiting factor since the size of the *open* list may grow exponentially during A* search. For each data set, we have selected the first n graphs with less than 10 nodes such that each computation of the $n \cdot n$ edit distances with plain A* search is feasible with less than one million node assignments. If possible $n = 200$ was chosen. Table 1 lists the resulting data set statistics.

4.2 Results

The performance results for search space reduction are listed in Table 2. The proposed Hausdorff heuristic is compared with plain A* search as a reference on the six graph data sets. The average number of elements in the *open* list after A* search is indicated. This number is directly related to the memory space required for GED. Despite the choice of rather small graphs for experimental evaluation, the average size of $111,852.1$ for the Molecules II data set is already very high with respect to the imposed limit of one million node assignments. The experiment was performed with 4GB RAM.

Table 3. Computational speedup. CPU runtime in seconds.

Data Set	Reference	Hausdorff Heuristic	Speedup Factor
Letters I	18.5	1.1	17.1
Letters II	109.7	11.2	9.8
Letters III	110.5	14.6	7.6
Fingerprints	196.4	61.3	3.2
Molecules I	523.5	36.8	14.2
Molecules II	1,224.6	165.3	7.4

The search space reduction achieved with the Hausdorff heuristic is about one order of magnitude in all cases. The best result is obtained on the Letters I data set, where 20.8 times less memory is required to compute GED. We assume that this large reduction factor is related to the suitability of the label domain for the Hausdorff heuristic. The Letters I data set contains weakly distorted drawings of letters whose line endings are labeled with their Cartesian coordinates. Using an Euclidean cost function, this type of node label is effective for selecting the nearest neighbor of $u \in V_1$ in V_2 and vice versa in Equation 1, even if the local edge structure is similar. Furthermore, we report strong results for molecular compounds using a Dirac cost function for matching chemical symbols. On the Molecules I data set, a reduction factor of 10.2 is reported.

The performance results for runtime reduction are provided in Table 3. Experiments were conducted with an Intel Core i7 processor with 2.0GHz CPU using a Java implementation. The runtime is indicated in seconds for matching all $n \cdot n$ selected graphs. The speedups are closely related to the search space reductions, which indicates that the computation of the Hausdorff heuristic does not lead to a significant overhead. In all cases, a CPU runtime reduction of one order of magnitude is obtained. The best result is achieved for the Letters I data set, where 17.1 times less CPU time is required to compute GED when using the Hausdorff heuristic.

5 Conclusions

In this paper, we have proposed a Hausdorff heuristic for efficient A* computation of graph edit distance (GED). The heuristic is based on the Hausdorff edit distance (HED), a quadratic-time approximation of GED, which underestimates the true distance and is hence admissible as a heuristic function for A* search. Based on a domain-specific cost function, the proposed Hausdorff heuristic can cope with unconstrained graphs. In particular, arbitrary labels are allowed on both nodes and edges.

An experimental evaluation is reported for six data sets from the IAM graph database. In all cases, the proposed heuristic has achieved substantial reductions of one order of magnitude in memory space and runtime. The best performance is reported for graphs from the Letters I data set, where 20.8 times less memory and 17.1 times less CPU time were required to compute GED when using the Hausdorff heuristic instead of plain A* search.

In future work, we aim to include larger graphs in the experimental evaluation and to compare and combine different heuristic functions. Our overall aim is to implement an efficient verification system that is able to calculate the true edit distance for large graphs in order to evaluate the approximation quality of HED and other GED approximations. We expect that in addition to software acceleration, a massive hardware acceleration will be necessary to compute GED for larger graphs.

Acknowledgments. This work has been supported by the SNSF grant P300P2-151279 to A. Fischer, the NSERC grant RGPIN-915 to R. Plamondon, a Canada Research Chair grant to Y. Savaria, and a grant from the Hasler Foundation Switzerland to K. Riesen.

References

1. Conte, D., Foggia, P., Sansone, C., Vento, M.: Thirty years of graph matching in pattern recognition. Int. Journal of Pattern Recognition and Artificial Intelligence 18(3), 265–298 (2004)
2. Vento, M.: A one hour trip in the world of graphs, looking at the papers of the last ten years. In: Kropatsch, W.G., Artner, N.M., Haxhimusa, Y., Jiang, X. (eds.) GbRPR 2013. LNCS, vol. 7877, pp. 1–10. Springer, Heidelberg (2013)
3. Borgwardt, K., Ong, C., Schönauer, S., Vishwanathan, S., Smola, A., Kriegel, H.P.: Protein function prediction via graph kernels. Bioinformatics 21(1), 47–56 (2005)
4. Harchaoui, Z., Bach, F.: Image classification with segmentation graph kernels. In: Proc. Int. Conf. on Computer Vision and Pattern Recognition, pp. 1–8 (2007)
5. Shoubridge, P., Kraetzl, M., Wallis, W.D., Bunke, H.: Detection of abnormal change in time series of graphs. Journal of Interconnection Networks 3(1-2), 85–101 (2002)
6. Umeyama, S.: An eigendecomposition approach to weighted graph matching problems. IEEE Trans. on Pattern Analysis and Machine Intelligence 10(5), 695–703 (1988)
7. Wilson, R., Hancock, E., Luo, B.: Pattern vectors from algebraic graph theory. IEEE Trans. on Pattern Analysis and Machine Intelligence 27(7), 1112–1124 (2005)
8. Jiang, X., Bunke, H.: Optimal quadratic-time isomorphism of ordered graphs. Pattern Recognition 32(17), 1273–1283 (1999)
9. Dickinson, P., Bunke, H., Dadej, A., Kraetzl, M.: Matching graphs with unique node labels. Pattern Analysis and Applications 7(3), 243–254 (2004)
10. Sanfeliu, A., Fu, K.S.: A distance measure between attributed relational graphs for pattern recognition. IEEE Trans. on Systems, Man, and Cybernetics 13(3), 353–363 (1983)
11. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. Journal of the Association for Computing Machinery 21(1), 168–173 (1974)
12. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. on Systems, Science, and Cybernetics 4(2), 100–107 (1968)

13. Berretti, S., Del Bimbo, A., Vicario, E.: Efficient matching and indexing of graph models in content-based retrieval. IEEE Trans. on Pattern Analysis and Machine Intelligence 23(10), 1089–1105 (2001)
14. Gregory, L., Kittler, J.: Using graph search techniques for contextual colour retrieval. In: Caelli, T.M., Amin, A., Duin, R.P.W., Kamel, M.S., de Ridder, D. (eds.) SSPR&SPR 2002. LNCS, vol. 2396, pp. 186–194. Springer, Heidelberg (2002)
15. Riesen, K., Fankhauser, S., Bunke, H.: Speeding up graph edit distance computation with a bipartite heuristic. In: Proc. Int. Workshop on Mining and Learning with Graphs, pp. 21–24 (2007)
16. Riesen, K., Bunke, H.: Approximate graph edit distance computation by means of bipartite graph matching. Image and Vision Computing 27(4), 950–959 (2009)
17. Munkres, J.: Algorithms for the assignment and transportation problems. Journal of the Society for Industrial and Applied Mathematics 5(1), 32–38 (1957)
18. Fischer, A., Suen, C.Y., Frinken, V., Riesen, K., Bunke, H.: A fast matching algorithm for graph-based handwriting recognition. In: Kropatsch, W.G., Artner, N.M., Haxhimusa, Y., Jiang, X. (eds.) GbRPR 2013. LNCS, vol. 7877, pp. 194–203. Springer, Heidelberg (2013)
19. Fischer, A., Suen, C., Frinken, V., Riesen, K., Bunke, H.: Approximation of graph edit distance based on Hausdorff matching. Pattern Recognition (submitted)
20. Huttenlocher, D.P., Klanderman, G.A., Kl, G.A., Rucklidge, W.J.: Comparing images using the Hausdorff distance. IEEE Trans. on Pattern Analysis and Machine Intelligence 15, 850–863 (1993)
21. Coughlan, J.M., Yuille, A.L.: Bayesian A* tree search with expected O(N) node expansions: applications to road tracking. Neural Computation 14(8), 1929–1958 (2002)
22. Cazorla, M., Escolano, F., Gallardo, D., Rizo, R.: Junction detection and grouping with probabilistic edge models and Bayesian A*. Pattern Recognition 35(9), 1869–1881 (2002)
23. Neuhaus, M., Riesen, K., Bunke, H.: Fast suboptimal algorithms for the computation of graph edit distance. In: Yeung, D.-Y., Kwok, J.T., Fred, A., Roli, F., de Ridder, D. (eds.) SSPR&SPR 2006. LNCS, vol. 4109, pp. 163–172. Springer, Heidelberg (2006)
24. Riesen, K., Bunke, H.: IAM graph database repository for graph based pattern recognition and machine learning. In: da Vitoria Lobo, N., Kasparis, T., Roli, F., Kwok, J.T., Georgiopoulos, M., Anagnostopoulos, G.C., Loog, M. (eds.) SSPR&SPR 2008. LNCS, vol. 5342, pp. 287–297. Springer, Heidelberg (2008)
25. Watson, C., Wilson, C.: NIST Special Database 4, Fingerprint Database. National Institute of Standards and Technology (1992)
26. Kazius, J., McGuire, R., Bursi, R.: Derivation and validation of toxicophores for mutagenicity prediction. Journal of Medicinal Chemistry 48(1), 312–320 (2005)
27. DTP: AIDS antiviral screen (2004), `http://dtp.nci.nih.gov/docs/aids/aids_data.html`