Improving Approximate Graph Edit Distance Using Genetic Algorithms

Kaspar Riesen[1], Andreas Fischer[2], and Horst Bunke[3]

[1] Institute for Information Systems, University of Applied Sciences and Arts Northwestern Switzerland, Riggenbachstrasse 16, 4600 Olten, Switzerland
kaspar.riesen@fhnw.ch
[2] Biomedical Science and Technologies Research Centre, Polytechnique Montreal 2500 Chemin de Polytechnique, Montreal H3T 1J4, Canada
andreas.fischer@polymtl.ca
[3] Institute of Computer Science and Applied Mathematics, University of Bern, Neubrückstrasse 10, 3012 Bern, Switzerland
bunke@iam.ch

Abstract. Many flexible methods for graph dissimilarity computation are based on the concept of edit distance. A recently developed approximation framework allows one to compute graph edit distances substantially faster than traditional methods. Yet, this novel procedure considers the local edge structure only during the primary optimization process. Hence, the speed up is at the expense of an overestimation of the true graph edit distances in general. The present paper introduces an extension of this approximation framework. Regarding the node assignment from the original approximation as a starting point, we implement a search procedure based on a genetic algorithm in order to improve the approximation quality. In an experimental evaluation on three real world data sets a substantial gain of distance accuracy is empirically verified.

1 Introduction

Graph matching refers to the process of evaluating the structural similarity of graphs. A large number of methods for graph matching have been proposed in recent years (see [1, 2] for exhaustive surveys). Due to its ability to cope with arbitrarily structured graphs with unconstrained label alphabets for both nodes and edges, the concept of graph edit distance [3] can be applied to virtually any kind of graphs. Therefore, graph edit distance has been used in the context of classification and clustering tasks in diverse applications [4–6].

Given two graphs, the source graph g_1 and the target graph g_2, the basic idea of graph edit distance is to transform g_1 into g_2 using some distortion operations. A standard set of distortion operations is given by *insertions*, *deletions*, and *substitutions* of both nodes and edges. We denote the substitution of two nodes u and v by $(u \rightarrow v)$, the deletion of node u by $(u \rightarrow \varepsilon)$, and the insertion of node v by $(\varepsilon \rightarrow v)$[1]. A sequence of edit operations $e_1, \ldots, e_k$ that transform g_1 completely into g_2 is called an *edit path* between g_1 and g_2.

[1] For edges we use a similar notation.

P. Fränti et al. (Eds.): S+SSPR 2014, LNCS 8621, pp. 63–72, 2014.

Let $\Upsilon(g_1, g_2)$ denote the set of all possible edit paths between two graphs g_1 and g_2. To find the most suitable edit path out of $\Upsilon(g_1, g_2)$, one introduces a cost for each edit operation, measuring the strength of the corresponding operation. The *edit distance* of two graphs is then defined by the minimum cost edit path between two graphs.

The computation of exact graph edit distance is usually carried out by means of a tree search algorithm which explores the space of all possible mappings of the nodes and edges of the first graph to the nodes and edges of the second graph. A widely used method is based on the A* algorithm [7] which is a best-first search algorithm. The computational complexity of the exact edit distance algorithm, whether or not heuristic functions are used to govern the tree traversal process, is exponential in the number of nodes of the involved graphs. Consequently, exact 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 (e.g. [8–11]). The authors of the present paper also introduced an algorithmic framework which allows the approximate computation of graph edit distance in a substantially faster way than traditional methods [12]. Yet, the substantial speed-up in computation time is at the expense of an overestimation of the actual graph edit distance. The reason for this overestimation is that the algorithm is able to consider only local, rather than global, edge structure during the optimization process. The main objective of the present paper is to significantly reduce the overestimation of edit distances in our approximation framework. To this end, the distance approximation found by the procedure of [12] is systematically improved using a search procedure based on genetic algorithms.

Genetic algorithms have been proposed in the context of error-tolerant graph matching in various publications [13–15]. The basic idea of this approach is to formalize matchings as states (*chromosomes*) with a corresponding performance (*fitness*). An initial pool of these chromosomes, i.e. matchings, evolves iteratively into other generations of matchings. To this end, different genetic operations are applied to the current matchings. Though the search space is explored in a random fashion, genetic algorithms can be designed so as to favour promising chromosomes, i.e. well fitting matchings, and further improve them by specific genetic operations.

The remainder of this paper is organized as follows. Next, in Sect. 2 the original framework for graph edit distance approximation [12] is summarized. In Sect. 3 the extension of this specific framework using a genetic search procedure is introduced. An experimental evaluation on diverse data sets is carried out in Sect. 4, and in Sect. 5 we draw some conclusions and outline some possible tasks and extensions for future work.

2 Bipartite Graph Edit Distance Approximation

In the framework presented in [12], for matching two graphs g_1 and g_2 with nodes $V_1 = \{u_1, \ldots, u_n\}$ and $V_2 = \{v_1, \ldots, v_m\}$, respectively, a cost matrix $\mathbf{C}$ is first established as follows:

$$\mathbf{C} = \left[\begin{array}{cccc|cccc} c_{11} & c_{12} & \cdots & c_{1m} & c_{1\varepsilon} & \infty & \cdots & \infty \\ c_{21} & c_{22} & \cdots & c_{2m} & \infty & c_{2\varepsilon} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \ddots & \infty \\ c_{n1} & c_{n2} & \cdots & c_{nm} & \infty & \cdots & \infty & c_{n\varepsilon} \\ \hline c_{\varepsilon 1} & \infty & \cdots & \infty & 0 & 0 & \cdots & 0 \\ \infty & c_{\varepsilon 2} & \ddots & \vdots & 0 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \infty & \vdots & \ddots & \ddots & 0 \\ \infty & \cdots & \infty & c_{\varepsilon m} & 0 & \cdots & 0 & 0 \end{array}\right]$$

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

Obviously, the left upper corner of the cost matrix represents the costs of all possible node substitutions, the diagonal of the right upper corner the costs of all possible node deletions, and the diagonal of the bottom left corner the costs of all possible node insertions. Note that each node can be deleted or inserted at most once. Therefore any non-diagonal element of the right-upper and left-lower part is set to ∞. The bottom right corner of the cost matrix is set to zero since substitutions of the form $(\varepsilon \rightarrow \varepsilon)$ should not cause any costs. In the definition of cost matrix $\mathbf{C}$, to each entry c_{ij}, i.e. to each cost of a node edit operation, the minimum sum of edge edit operation costs, implied by the corresponding node operation, is added (i.e. the matching cost arising from the local edge structure is encoded in the individual entries of $\mathbf{C}$).

On the basis of the square cost matrix $\mathbf{C}$ a bipartite assignment algorithm is executed (first step). The result returned by this bipartite optimization procedure corresponds to the minimum cost mapping m of the nodes and their local edge structure of g_1 to the nodes and their local edge structure of g_2. Mapping m can be seen as partial edit path $\pi = e_1, \ldots, e_l$, where each edit operation $e_i \in \pi$ reflects an operation on nodes from V_1 and/or V_2 (deletions, insertions or substitutions). In a second step the edit path π between g_1 and g_2 is completed according to mapping m. Note that edit operations on edges are implied by edit operations on their adjacent nodes, i.e. whether an edge is substituted, deleted, or inserted, depends on the edit operations performed on its adjacent nodes. Hence, given the set of node operations $e_1, \ldots, e_l$, the global edge structures from g_1 and g_2 can be edited accordingly. The cost of the complete edit path π is finally returned as an approximate graph edit distance. We denote the approximated edit distance between graphs g_1 and g_2 according to mapping m with $d(g_1, g_2, m)$.

Note that the edit path corresponding to $d(g_1, g_2, m)$ considers the edge structure of g_1 and g_2 in a global and consistent way while the optimal node mapping m from step 1 is able to consider the structural information in an isolated way only (single nodes and their adjacent edges). This is due to the fact that during the optimization process no information about neighboring node mappings is available. Hence, in comparison with optimal search methods for graph edit distance, our novel algorithmic framework might cause additional edge operations

in the second step, which would not be necessary in a globally optimal graph matching. Hence, the distances found by this specific framework are – in the optimal case – equal to, or – in a suboptimal case – larger than the exact graph edit distance.

For the remainder of this paper we denote this graph edit distance approximation algorithm with *BP* (*Bipartite*).

3 Improving Graph Edit Distance Approximations Using Genetic Algorithms

In several experimental evaluations we observed that the suboptimality of BP is very often due to a few incorrectly assigned nodes in m. That is, only few node assignments from the first step are responsible for the additional edge operations in the second step (and the resulting overestimation of the true edit distance). Our novel procedure ties in at this observation. Rather than returning the approximate edit distance directly, a genetic search procedure based on mapping m is started.

The chromosomes in our genetic search procedure are mappings related to our original node assignment m. In order to build an initial population $P(0)$ containing chromosomes (mappings), we compute N random variations $\{m_1^{(0)}, \ldots, m_N^{(0)}\}$ of m. A single variation $m_i^{(0)} \in P(0)$ of m is computed as follows. Every node assignment $u_i \to v_j$ in m is possibly omitted with a certain probability p (referred to as *mutation probability*). That is, in an alternative mapping we enforce nodes u_i and v_j to be assigned to other nodes than v_j and u_i, respectively. This is ensured by means of an update of the cost matrix $\mathbf{C}$ such that entry $c_{i,j}$ (corresponding to the assignment $u_i \to v_j$) is set to ∞. Given the updated cost matrix (with ∞-entries at certain positions) an optimal node assignment is computed using our former procedure. This results in a new mapping $m_i^{(0)}$ which does not contain $(u_i \to v_j)$ any more. Note that $m_i^{(0)}$ corresponds to an optimal node assignment based on the altered cost matrix. Hence, $m_i^{(0)}$ is consistent, i.e. every node of g_1 is assigned to a single node of g_2 (or deleted) and every node of g_2 is assigned to a single node of g_1 (or inserted).

This mutation procedure is repeated N times to mapping m in order to get N different mappings $P(0) = \{m_1^{(0)}, \ldots, m_N^{(0)}\}$ and thus N different approximations of the true graph edit distance[2]. Note that all of these approximate edit distance values are still equal to, or larger than, the exact distance values. Hence, without knowing the exact graph edit distance, the fitness of every assignment $m_i^{(0)}$ can be rated according to its specific distance value $d(g_1, g_2, m_i^{(0)})$, viz. the lower $d(g_1, g_2, m_i^{(0)})$ the better the fitness of $m_i^{(0)}$.

Given the initial population $P(0)$ the following iterative procedure is carried out next. A new population $P(t+1)$ of mappings is built upon a subset E of $P(t)$,

[2] Note that the original mapping m is initially added to $P(0)$ such that the approximation found by our extension is guaranteed to be at least as accurate as the original approximation $d(g_1, g_2, m)$.

often referred to as *parents*. In order to select the parents from a given population $P(t)$, the $(f \cdot N)$ best approximations, i.e. the approximations in $P(t)$ with lowest distance values, are selected ($f \in]0, 1]$). In our framework, all approximations from E are added without any modifications to the next population $P(t+1)$. This ensures that the best solution found so far will not be lost during the search procedure (known as *survival of the fittest*).

In order to derive the remaining mappings of the new population $P(t+1)$, the following procedure is repeated $(N - |E|)$-times. Two mappings m' and m'' from the pool of parents E are randomly selected and eventually combined to one mapping m. To this end, the cost matrices $\mathbf{C}' = c'_{i,j}$ and $\mathbf{C}'' = c''_{i,j}$ corresponding to mappings m' and m'', respectively, are merged by means of

$$\mathbf{C}_m = \max\{c'_{i,j}, c''_{i,j}\}$$

Based on $\mathbf{C}_m$ an optimal mapping m is computed and eventually added to $P(t+1)$. Due to the definition of $\mathbf{C}_m$ the node mappings omitted in at least one of the mappings m' and m'' will also be prevented in the merged mapping m. The detour via optimal assignment computation on a cost matrix $\mathbf{C}_m$ again ensures that the merged mapping m is consistent with the underlying graphs (nodes of both graphs are uniquely assigned to nodes of the other graph or deleted/inserted).

The two main steps of the genetic algorithm (selection of parents $E \subseteq P(t)$ and computation of a new generation of mappings $P(t+1)$ based on E) are repeated until the best distance approximation has not been improved during the last τ iterations. It is well known that genetic algorithms are not deterministic. Therefore, we repeat the complete search procedure s times from the scratch and return the overall best approximation found in these s runs (which makes the algorithmic procedure more stable and reduces the risk of finding a poor approximation due to a poor random initialization).

Given that the genetic search procedure stops after t iterations on average, the two main steps of our former approximation framework, namely the computation of an optimal mapping m based on a cost matrix and the derivation of the corresponding edit distance, have to be carried out $(s \cdot t \cdot N)$-times. Hence, one can expect that our extended framework increases the run time by the magnitude of $(s \cdot t \cdot N)$ compared to our original framework.

The complete algorithmic procedure is given in Alg. 1. Note that the first three lines of Alg. 1 correspond to the original framework *BP*, while line 4 to 18 describe the proposed extension, denoted by BPGA from now on.

4 Experimental Evaluation

For experimental evaluations, three data sets from the IAM graph database repository[3] for graph based pattern recognition and machine learning are used [16]. The first graph data set involves graphs that represent molecular compounds (AIDS). We construct graphs from the AIDS Antiviral Screen Database

[3] `www.iam.unibe.ch/fki/databases/iam-graph-database`

Algorithm 1. BPGA(g_1, g_2) (Meta Parameters: N, p, τ, f, s)

```
1: Build cost matrix C according to the input graphs g1 and g2
2: Compute optimal node assignment m on C
3: Derive edit path and approximate edit distance based on m
4: for i = 1, ..., s do
5:    build initial population P(0) of mappings {m_1^(0), ..., m_N^(0)} based on m using mutation
      probability p
6:    d_best = min_{i=1,...,N} {d(g1, g2, m_i^(0))}
7:    t = 0; l = 0
8:    while t − l < τ do
9:       select a subset E ⊆ P(t) of parents (|E| = f · N)
10:      build a new population P(t + 1) = {m_1^(t+1), ..., m_N^(t+1)} from E
11:      d = min_{i=1,...,N} {d(g1, g2, m_i^(t+1))}
12:      t = t + 1
13:      if d < d_best then
14:         d_best = d; l = t
15:      end if
16:   end while
17: end for
18: return d_best
```

of Active Compounds [17]. This data set consists of two classes (*active, inactive*), which represent molecules with activity against HIV or not. The molecules are converted into graphs in a straightforward manner by representing atoms as nodes and the covalent bonds as edges. Nodes are labeled with the number of the corresponding chemical symbol and edges by the valence of the linkage.

The second graph data set consists of graphs representing fingerprint images (FP) [18]. In order to obtain graphs from fingerprint images, the relevant regions are binarized and a noise removal and thinning procedure is applied. This results in a skeletonized representation of the extracted regions. Ending points and bifurcation points of the skeletonized regions are represented by nodes. Additional nodes are inserted in regular intervals between ending points and bifurcation points. Finally, undirected edges are inserted to link nodes that are directly connected through a ridge in the skeleton. Each node is labeled with a two-dimensional attribute giving its position. The edges are attributed with an angle denoting the orientation of the edge with respect to the horizontal direction.

The third data set consists of graphs representing symbols from architectural and electronic drawings (GREC) [19]. The images occur at five different distortion levels. Depending on the distortion level, either erosion, dilation, or other morphological operations are applied. The result is thinned to obtain lines of one pixel width. Finally, graphs are extracted from the resulting denoised images by tracing the lines from end to end and detecting intersections as well as corners. Ending points, corners, intersections and circles are represented by nodes and labeled with a two-dimensional attribute giving their position. The nodes are connected by undirected edges which are labeled as *line* or *arc*. An additional attribute specifies the angle with respect to the horizontal direction or the diameter in case of arcs.

Our procedure BPGA has five meta parameters to be defined by the user (see Table 1 for a survey). In the following evaluations only two of them are altered in order to evaluate their impact on the approximation quality, viz. the population size N as well as the mutation probability p. The three remaining parameters (τ, f, s) are freezed to constants. In fact, in preliminary experimental evaluations it turns out that these parameters – given that they do not fall below a certain threshold – do nearly not affect the resulting approximations. We choose minimum values for τ, f, s such that stable and reasonable results on all of the three data sets can be observed. In the Table 1 the meta parameters and their respective values are summarized.

Table 1. Meta Parameters of BPGA

Parameter	Meaning	Value
N	Population Size	$\{50, 100\}$
p	Mutation probability that a given node mapping in m is prevented (needed to build $P(0)$)	$\{0.1, 0.3, 0.5, 0.7\}$
τ	Termination when best solution has not been improved during the last τ iterations	6
f	Percentage of chromosomes selected from $P(t)$ as parents to build $P(t+1)$	0.25
s	Number of runs	3

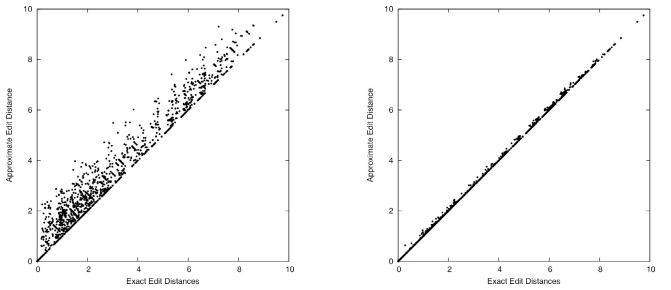
In Table 2 the achieved results are shown. On each data set and for each graph edit distance algorithm two characteristic numbers are computed, viz. the mean relative overestimation of the exact graph edit distance ($\varnothing o$) and the mean run time to carry out one graph matching ($\varnothing t$). The algorithms employed are A* and BP (reference systems) and eight differently parametrized versions of our novel procedure BPGA ($N \in \{50, 100\}$; $p \in \{0.1, 0.3, 0.5, 0.7\}$).

First we focus on the degree of overestimations and regard the results of BPGA with $N = 50$ only. The original framework (BP) overestimates the graph distance by 12.68% in average on the AIDS data. On the Fingerprint and GREC data the overestimations amount to 6.38% and 2.98%, respectively. These values can be substantially reduced with our extended framework. For instance on the AIDS data, the mean relative overestimation can be reduced to 2.01% in the best case ($p = 0.5$). That is, the mean relative overestimation of our novel framework is approximately six times smaller than the one of the original approximation framework. On the GREC data set the mean relative overestimation is reduced from 2.98% to 0.83% in the best case ($p = 0.3$) and on the Fingerprint data the overestimation can be heavily reduced from 6.38% to 0.13% ($p = 0.3$ or $p = 0.5$). We observe that a mutation probability between 0.3 and 0.5 works well on all three data sets.

Comparing the mean run time of our novel procedure with the original framework on the AIDS data, we observe that our extension takes approximately 300 times longer for one matching in average with $N = 50$ (approx. 167- and 200-times longer matching times in average on the Fingerprint and GREC data,

Table 2. The mean relative overestimation of the exact graph edit distance ($\varnothing o$) and the mean run time for one matching ($\varnothing t$) using a specific graph edit distance algorithm

Algorithm	Data Set					
	AIDS		FP		GREC	
	$\varnothing$ o	$\varnothing$ t	$\varnothing$ o	$\varnothing$ t	$\varnothing$ o	$\varnothing$ t
A* (Exact)	-	5.63	-	5.00	-	3.10
BP	12.68	0.0004	6.38	0.0006	2.98	0.0004
BPGA(50, 0.1)	2.96	0.12	0.20	0.10	1.00	0.08
BPGA(50, 0.3)	2.18	0.11	0.13	0.10	0.83	0.08
BPGA(50, 0.5)	2.01	0.12	0.14	0.10	0.83	0.08
BPGA(50, 0.7)	2.12	0.11	0.15	0.10	0.89	0.08
BPGA(100, 0.1)	2.33	0.23	0.14	0.20	0.82	0.16
BPGA(100, 0.3)	1.53	0.22	0.09	0.21	0.66	0.17
BPGA(100, 0.5)	1.42	0.23	0.09	0.20	0.68	0.17
BPGA(100, 0.7)	1.54	0.23	0.11	0.20	0.74	0.16



(a) BP (b) BPGA(100, 0.3)

Fig. 1. Exact (x-axis) vs. approximate (y-axis) graph edit distance

respectively). The observed run time increase perfectly lies within the expected multiplication of the average run time by $(s \cdot t \cdot N)$. However, compared to the exact algorithm our extension is still very fast (approximately 40 to 50 times faster on all data sets with $N = 50$).

Increasing the population size N to 100 allows us to further decrease the overestimation. Yet, the reduction is at the prize of approximately doubling the mean runtime when compared to $N = 50$ on all data sets. Also with $N = 100$ a mutation probability between 0.3 and 0.5 seems to be the best choice on all data sets.

The substantial improvement of the approximation accuracy can also be observed in the scatter plots in Fig. 1. These scatter plots give us a visual representation of the accuracy of the suboptimal methods on the Fingerprint data set[4]. We plot for each pair of graphs their exact (horizontal axis) and approximate

[4] On the other data sets similar results can be observed.

(vertical axis) distance value. The reduction of the overestimation using our proposed extension is clearly observable and illustrates the power of our extended framework.

5 Conclusion and Future Work

In the present paper we propose an extension of our previous graph edit distance approximation algorithm (BP). The major idea of our work is to use the suboptimal graph edit distance and the underlying node assignment in a genetic search procedure to improve the approximation accuracy. With several experimental results we show that this extension leads to a substantial reduction of the overestimations typical for BP. Though the run times are increased when compared to our former framework (as expected), they are still far below the run times of the exact algorithm.

We see three important lines of research for future work. First, we want to implement other search methods than genetic algorithms (e.g. floating search [20]). Second, there seems to be room for developing other merging methods to build a new assignment based on two given assignments (without the need to compute optimal assignments based on $\mathbf{C}$). Finally, the experimental evaluation will be extended (more data sets, more exhaustive evaluations of the meta parameters, etc.).

Acknowledgements. This work has been supported by the *Hasler Foundation* Switzerland and the *Swiss National Science Foundation* project P300P2-151279.

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. Bunke, H., Allermann, G.: Inexact graph matching for structural pattern recognition. Pattern Recognition Letters 1, 245–253 (1983)
4. Neuhaus, M., Bunke, H.: An error-tolerant approximate matching algorithm for attributed planar graphs and its application to fingerprint classification. In: Fred, A., Caelli, T.M., Duin, R.P.W., Campilho, A.C., de Ridder, D. (eds.) SSPR&SPR 2004. LNCS, vol. 3138, pp. 180–189. Springer, Heidelberg (2004)
5. Ambauen, R., Fischer, S., Bunke, H.: Graph edit distance with node splitting and merging and its application to diatom identification. In: Hancock, E., Vento, M. (eds.) GbRPR 2003. LNCS, vol. 2726, pp. 95–106. Springer, Heidelberg (2003)
6. Robles-Kelly, A., Hancock, E.: Graph edit distance from spectral seriation. IEEE Transactions on Pattern Analysis and Machine Intelligence 27(3), 365–378 (2005)
7. 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(2), 100–107 (1968)

8. Boeres, M.C., Ribeiro, C.C., Bloch, I.: A randomized heuristic for scene recognition by graph matching. In: Ribeiro, C.C., Martins, S.L. (eds.) WEA 2004. LNCS, vol. 3059, pp. 100–113. Springer, Heidelberg (2004)
9. Sorlin, S., Solnon, C.: Reactive tabu search for measuring graph similarity. In: Brun, L., Vento, M. (eds.) GbRPR 2005. LNCS, vol. 3434, pp. 172–182. Springer, Heidelberg (2005)
10. Justice, D., Hero, A.: A binary linear programming formulation of the graph edit distance. IEEE Trans. on Pattern Analysis ans Machine Intelligence 28(8), 1200–1214 (2006)
11. 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)
12. Riesen, K., Bunke, H.: Approximate graph edit distance computation by means of bipartite graph matching. Image and Vision Computing 27(4), 950–959 (2009)
13. Cross, A., Wilson, R., Hancock, E.: Inexact graph matching using genetic search. Pattern Recognition 30(6), 953–970 (1997)
14. Wang, I., Fan, K.C., Horng, J.T.: Genetic-based search for error-correcting graph isomorphism. IEEE Transactions on Systems, Man, and Cybernetics (Part B) 27(4), 588–597 (1997)
15. Suganthan, P.: Structural pattern recognition using genetic algorithms. Pattern Recognition 35(9), 1883–1893 (2002)
16. 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)
17. DTP, AIDS antiviral screen (2004), `http://dtp.nci.nih.gov/docs/aids/aids_data.html`
18. Watson, C., Wilson, C.: NIST Special Database 4, Fingerprint Database. National Institute of Standards and Technology (March 1992)
19. Dosch, P., Valveny, E.: Report on the second symbol recognition contest. In: Liu, W., Lladós, J. (eds.) GREC 2005. LNCS, vol. 3926, pp. 381–397. Springer, Heidelberg (2006)
20. Pudil, P., Novovicova, J., Kittler, J.: Floating search methods in feature-selection. Pattern Recognition Letters 15(11), 1119–1125 (1994)