Improved Graph Edit Distance Approximation with Simulated Annealing

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

[1] Institute for Information Systems, University of Applied Sciences FHNW, Riggenbachstrasse 16, 4600 Olten, Switzerland
kaspar.riesen@fhnw.ch
[2] Department of Informatics, University of Fribourg and HES-SO, 1700 Fribourg, Switzerland
andreas.fischer@unifr.ch
[3] Institute of Computer Science and Applied Mathematics, University of Bern, Neubrückstrasse 10, 3012 Bern, Switzerland
bunke@iam.ch

Abstract. The present paper is concerned with graph edit distance, which is widely accepted as one of the most flexible graph dissimilarity measures available. A recent algorithmic framework for approximating the graph edit distance overcomes the major drawback of this distance model, viz. its exponential time complexity. Yet, this particular approximation suffers from an overestimation of the true edit distance in general. Overall aim of the present paper is to improve the distance quality of this approximation by means of a post-processing search procedure. The employed search procedure is based on the idea of simulated annealing, which turns out to be particularly suitable for complex optimization problems. In an experimental evaluation on several graph data sets the benefit of this extension is empirically confirmed.

1 Introduction

Due to their power and flexibility, graphs have found widespread application in pattern recognition and related fields [1,2]. Prominent examples of a classes of patterns, which can be formally represented in a more suitable and natural way by means of graphs rather than with feature vectors, are chemical compounds [3], binary executables [4], or networks [5].

The problem of computing graph dissimilarity is commonly solved via a particular *graph matching* algorithm. Graph matching has been the topic of numerous studies in pattern recognition over the last decades [1,2], resulting in powerful methods such as, for instance, *spectral methods* [6] or *graph kernels* [3]. *Graph edit distance* [7], introduced about 30 years ago, is still one of the most flexible graph distance models available. Yet, the run time of exact graph edit distance computation is exponential in the number of nodes of the involved graphs, which limits its applicability to rather small graphs.

In [8] the authors of the present paper introduced an algorithmic framework for the approximation of graph edit distance in cubic time. Yet, one of the major

P. Foggia et al. (Eds.): GbRPR 2017, LNCS 10310, pp. 222–231, 2017.
DOI: 10.1007/978-3-319-58961-9_20

problems of this particular approximation framework is that it overestimates the true edit distance quite often. The present paper is concerned with an extension of this approximation that aims at making the distance approximation more accurate. The idea of this extension is based on a post-processing procedure that takes the result of the original approximation as a starting point for a (non-exhaustive) search process.

Note that in [9] several search procedures for the improvement of the approximation accuracy have already been proposed (amongst others, greedy forward search procedures). The novelty of the present paper is twofold. First, it presents a search strategy which takes into account both a lower and an upper bound on the true edit distance (rather than only the upper bound as proposed in [9]). Second, we make use of a different search method which is based on *simulated annealing* [10,11].

The basic idea of simulated annealing is to explore the search space in a random fashion and accepting solutions as long as they are getting better than the previous solution. Yet, in contrast with pure greedy algorithms, simulated annealing also accepts worse solutions with a certain probability (which slowly decreases during run time). The property of accepting worse solutions is fundamental as this allows to escape local minima during the search process.

The remainder of this paper is organized as follows. Next, in Sect. 2, the approximation framework for graph edit distance is reviewed. In Sect. 3, the novel search procedure based on simulated annealing is described in detail. Eventually, in Sect. 4, we empirically confirm the benefit of this extension on three graph data sets. Finally, in Sect. 5, we conclude the paper.

2 Graph Edit Distance (GED)

2.1 Basic Definition of GED

A graph g is a four-tuple $g = (V, E, \mu, \nu)$, where V is the finite set of nodes, $E \subseteq V \times V$ is the set of edges, $\mu : V \to L_V$ is the node labeling function, and $\nu : E \to L_E$ is the edge labeling function. 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 various label alphabets from different domains. Unlabeled graphs are obtained by assigning the same (empty) label $\varnothing$ to all nodes and edges, i.e. $L_V = L_E = \{\varnothing\}$.

Given two graphs, $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$, the basic idea of *graph edit distance* (*GED*) [7] is to transform g_1 into g_2 using edit operations, viz. *insertions*, *deletions*, and *substitutions* of both nodes and edges. The substitution of two nodes u and v is denoted by $(u \to v)$, the deletion of node u by $(u \to \varepsilon)$, and the insertion of node v by $(\varepsilon \to v)$[1]. A set of edit operations $\lambda(g_1, g_2) = \{e_1, \ldots, e_k\}$ that completely transform g_1 into g_2 is called an edit path between g_1 and g_2.

[1] A similar notation is used for edges.

Let $\Upsilon(g_1, g_2)$ denote the set of all admissible 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 $c(e_i)$ for every 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. The *graph edit distance* $d_{\lambda_{\min}}$ between two graphs $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ is then defined by

$$d_{\lambda_{\min}}(g_1, g_2) = \min_{\lambda \in \Upsilon(g_1, g_2)} \sum_{e_i \in \lambda} c(e_i).$$

2.2 Approximate Computation of GED

The problem of minimizing the graph edit distance can be reformulated as an instance of a *Quadratic Assignment Problem* (*QAP*) which in turn belong to the class of $\mathcal{NP}$*-complete* problems. QAPs basically consist of a linear and a quadratic term which have to be simultaneously optimized. In case of graph edit distance, the linear term of QAPs can be used to model the sum of node edit costs, while the latter is commonly used to represent the sum of edge edit costs (see [12] for further details).

The graph edit distance approximation framework introduced in [8] reduces the QAP of graph edit distance computation to an instance of a *Linear Sum Assignment Problem* (*LSAP*). Similar to QAPs, LSAPs deal with the question how the entities of two sets can be optimally assigned to each other. We formally represent assignments by means of permutations $(\varphi_1, \ldots, \varphi_n)$ of the integers $(1, 2, \ldots, n)$. Such a permutation refers to the assignment where the i-th entity of the first set is mapped to the entity at position φ_i in the second set $(i = 1, \ldots, n)$.

For solving LSAPs, which cope with a linear term only, a large number of efficient algorithms exist (see [13] for an exhaustive survey). The time complexity of the best performing exact algorithms for LSAPs is cubic in the size of the problem. Hence, LSAPs can be – in contrast with QAPs – quite efficiently solved.

In order to reformulate the graph edit distance problem to an instance of an LSAP, the use of a square $(n + m) \times (n + m)$ cost matrix $\mathbf{C}$ has been proposed in [8]. This particular cost matrix represents the costs of all possible node substitutions as well as all possible node deletions and node insertions. The framework proposed in [8] optimizes the linear term of the LSAP stated on $\mathbf{C}$.

By omitting the quadratic term during the assignment process, we neglect the structural relationships between the nodes (i.e. the edges between the nodes). In order to integrate knowledge about the graph structure, to each entry $c_{ij} \in \mathbf{C}$, i.e. to each cost of a node edit operation $(u_i \rightarrow v_j)$, the minimum sum of edge edit operation costs, implied by the corresponding node operation, is added. This particular encoding of the minimum matching cost arising from the local edge structure enables the LSAP to consider information about the local, yet not global, edge structure of a graph.

A minimum cost permutation $(\varphi_1, \ldots, \varphi_{n+m})$ derived on $\mathbf{C} = (c_{ij})$ via LSAP solving algorithm corresponds to the assignment of all nodes of g_1 to all nodes of g_2. Assignment ψ includes edit operations of the form $(u_i \rightarrow v_j)$, $(u_i \rightarrow \varepsilon)$, and

$(\varepsilon \rightarrow v_j)$[2]. Two different distance approximations can now be instantly derived from this node assignment, viz. an upper and a lower bound on the true graph edit distance.

$$\psi = ((u_1 \rightarrow v_{\varphi_1}), (u_2 \rightarrow v_{\varphi_2}), \ldots, (u_{m+n} \rightarrow v_{\varphi_{m+n}}))$$

For the upper bound we observe that edit operations on edges are uniquely defined by the edit operations on their adjacent nodes. That is, whether an edge (u, v) is substituted with an existing edge from the other graph, deleted, or inserted actually depends on the operations performed on both adjacent nodes u and v (and whether or not there is an edge between the matching nodes of the other graph). Hence, we can use the node assignment ψ to infer the complete set of globally consistent edge edit operations. The sum of costs of the node edit operations plus the costs of the implied edge operations gives us a first approximation value for the graph edit distance. Note that this approximation generally overestimates the true edit distance and actually builds an upper bound on the exact distance [14]. Thus, we denote this approximation with $d_{up}(g_1, g_2)$, or d_{up} for short.

The second approximation, which actually provides a lower bound d_{low} on the true edit distance [14], can be additionally inferred from the optimal assignment $(\varphi_1, \ldots, \varphi_n)$. Remember that every entry $c_{ij} \in \mathbf{C}$ reflects the cost of the corresponding node edit operation $(u_i \rightarrow v_j)$ plus the minimal cost of editing the incident edges of u_i to the incident edges of v_j. Hence, given an optimal permutation $(\varphi_1, \ldots, \varphi_{(n+m)})$, the minimal sum $\sum_{i=1}^{(n+m)} c_{i\varphi_i}$ can be subdivided into costs for node edit operations and costs for edge edit operations. Since every edge (u_i, u_j) is adjacent with two individual nodes u_i and u_j, every edge is considered twice in two independent entries in the optimal sum $\sum_{i=1}^{(n+m)} c_{i\varphi_i}$ (viz. once in entry $c_{i\varphi_i}$ and once in entry $c_{j\varphi_j}$). In order to derive a suitable approximation for the true edit distance, the cost of edge edit operations encoded in the sum $\sum_{i=1}^{(n+m)} c_{i\varphi_i}$ has thus to be multiplied by $\frac{1}{2}$. In summary, we obtain a lower bound on the true edit distance by summing up the cost of all node and half the cost of all edge edit operations, given the optimal assignment.

It is important to note that the permutation $(\varphi_1, \ldots, \varphi_{n+m})$ can be arbitrarily permuted and the resulting approximation d_{up} remains an admissible upper bound on the true edit distance. Yet, this does not account for the lower bound as defined above. That is, d_{low} constitutes a lower bound on the exact edit distance, if, and only if, the underlying permutation $(\varphi_1, \ldots, \varphi_{n+m})$ refers to the optimal solution of the LSAP stated on $\mathbf{C}$.

3 Improving the Accuracy with Simulated Annealing

It has been observed that both bounds d_{up} and d_{low} might introduce a (substantial) approximation error compared to the exact edit distance $d_{\lambda_{\min}}$.

[2] Edit operations of the form $(\varepsilon \rightarrow \varepsilon)$ can be dismissed, of course.

The present work aims at improving the overall distance quality of the approximation by means of a post processing procedure which searches within the interval $[d_{up}, d_{low}]$. The proposed search procedure is based on *simulated annealing*, which emulates a phenomenon in material science, viz. the annealing of solids. Simulated annealing has been originally proposed to obtain a state of minimum energy of a multiparticle physical system [10] and has later been adopted to solve difficult optimization problems [11].

The basic idea of solving optimization problems with simulated annealing is to start with a (random) initial solution and then randomly disturb it. As long as the resulting solution is better than the previous one, it is accepted and used in the following step. If the resulting solution is worse than the previous one, it may still be accepted with a certain probability. This probability is typically reciprocally proportional to the quality difference of the current and the previous solution and proportional to the current temperature. Usually, one starts with a high temperature in order to rather frequently allow deteriorations in the first iterations. Yet, during the running process the temperature is gradually decreased, and thus the probability that a worse solution is accepted becomes smaller. This reflects the idea of initially sampling the search space in larger steps and then gradually focusing on smaller, promising areas for the final solution.

The detailed algorithmic procedure for the improvement of the distance accuracy is given in Algorithm 1. As input parameters the algorithm takes the cost matrix $\mathbf{C}$, the upper and lower bound of the true edit distance d_{up} and d_{low}, the maximum number of iterations N, the starting temperature T, as well as the temperature decrease factor F.

On line 1 of Algorithm 1 two counters ($counter_1$ and $counter_2$) are initialized with zero. The former controls the number of iterations, while the latter is used to compute the probability of resetting the current search to a new random starting point (details follow below). Next, on line 2 and 3, a list with the first $(n + m)$ integers is initialized (in ascending order) and d_{min} as well as $d_{current}$ are initialized with the original upper bound d_{up}.

On line 4 the main loop of the search procedure starts. In every iteration we aim at improving, i.e. decreasing, the current upper bound $d_{current}$ by means of slightly changing the assignment ψ. In any case $d_{current}$ remains a valid upper bound on the exact edit distance. However, remember that the lower bound d_{low} cannot be improved, i.e. increased, during the proposed search process.

The main loop of Algorithm 1 is repeated until the current upper bound $d_{current}$ becomes equal to d_{low}. In this case we have found the optimal edit distance and can stop the procedure. Yet, this can only occur when the lower bound is equal to the true edit distance, of course (i.e. when $d_{low} = d_{\lambda_{\min}}$). Otherwise, the maximum number of iterations N have to be carried out. In either case, d_{min}, which corresponds to the minimal upper bound that has been found during the search process, is finally returned by the algorithm.

In every iteration of the main loop a new candidate for the upper bound is generated by means of the sub-procedure *Candidate-Generator*, which takes $order_{current}$ and $\mathbf{C}$ as parameters (see line 6). This sub-procedure, outlined in

Algorithm 1. Compute-Improvement($\mathbf{C}, d_{up}, d_{low}, N, T, F$)

1: $counter_1$=0 and $counter_2$=0
2: $order_{current} = (1, 2, \ldots, (n+m))$
3: $d_{min} = d_{up}$ and $d_{current} = d_{up}$
4: **while** $((d_{min} - d_{low}) > 0$ and $counter_1 < N)$ **do**
5: $counter_1$++
6: $(d_{cand}, order_{cand})$ = Candidate-Generator($order_{current}$, $\mathbf{C}$)
7: $\Delta = |d_{cand} - d_{current}|$
8: select random number r from $[0, 1]$
9: **if** $(d_{cand} < d_{current})$ or $\left(r < \exp\left(\frac{-\Delta}{\Delta_{avg} \times T}\right)\right)$ **then**
10: $d_{current} = d_{cand}$
11: $order_{current} = order_{cand}$
12: **end if**
13: **if** $(d_{current} < d_{min})$ **then**
14: $d_{min} = d_{current}$
15: $counter_2$=0
16: **else**
17: $counter_2$++
18: **end if**
19: select random number r from $[0, 1]$
20: **if** $\left(r < \frac{counter_2}{N}\right)$ **then**
21: $order_{current}$ = random permutation of $(1, 2, \ldots, (n+m))$
22: **end if**
23: $T = F \times T$
24: **end while**
25: **return** d_{min}

Algorithm 2, randomly changes the current order on one position. Formally, the integer at position r in $order_{current}$ is moved to the head of the current list (the remaining parts remain unaltered). Next, the LSAP stated on $\mathbf{C}$ is solved with a suboptimal assignment algorithm in $O((n+m)^2)$ time [15]. This algorithm iterates through the rows of $\mathbf{C}$ and assigns every node to the minimum unused node in the respective row in a greedy manner. By removing column φ_i in $\mathbf{C}$ it is ensured that every column of the cost matrix is considered exactly once (i.e. $\forall j$ refers to available columns in $\mathbf{C}$). This assignment procedure crucially depends on the order in which the rows are processed (actually defined in $order_{current}$). Due to the (slight) change of the processing order introduced at the beginning of Algorithm 2, an alternative assignment ψ and thus an alternative upper bound can be expected. Finally, we return both the candidate processing order $order_{cand}$ and the corresponding distance approximation d_{cand} to the main procedure.

Both $order_{cand}$ and d_{cand} are accepted when d_{cand} is lower than $d_{current}$ (i.e. we observe an improvement of the current upper bound) – see line 9 to 12 of Algorithm 1. If the distance approximation d_{cand} is greater than (or equal to) the current upper bound $d_{current}$, it may still be accepted with probability

$$P = \exp\left(\frac{-\Delta}{\Delta_{avg} \times T}\right),$$

where Δ refers to the absolute difference between d_{cand} and $d_{current}$, the normalizing factor Δ_{avg} corresponds to the running average of all values of Δ at that time, and T is the current temperature. Note the influence of Δ and T on the probability P. The greater the deterioration Δ, the smaller is P. Vice versa, the

greater the current temperature T, the greater is P (yet, note that temperature T is gradually lowered by factor F at the end of every iteration – see line 23).

On line 13 to line 18 we verify whether the current distance $d_{current}$ is smaller than the minimal upper bound d_{min} that has been found so far. Whenever a new minimal distance has been found, $counter_2$ is reset to zero, otherwise $counter_2$ is increased by one (i.e. we count the number of iterations without improvements of the minimal upper bound). This counter is eventually used to control whether or not the current solution is reset to a new random starting point. Formally, the probability that the current processing order ($order_{current}$) is randomly disturbed on all positions increases with $counter_2$ (see line 20 to 22). The rationale behind this resetting is that whenever the number of iterations without improvements exceeds a certain limit, a restart of the search procedure from another point in the search domain might be beneficial.

Algorithm 2. Candidate-Generator($order = (i_{(1)}, i_{(2)}, \ldots, i_{((n+m))})$, $\mathbf{C}$)

1: select random integer r from $[0, (n+m)]$
2: $order = (i_{(r)}, i_{(1)}, i_{(2)}, \ldots, i_{(r-1)}, i_{(r+1)}, \ldots, i_{((n+m))})$
3: $\psi = \{\}$
4: **for** $i \in order$ **do**
5: $\quad \varphi_i = \arg\min_{\forall j} c_{ij}$
6: $\quad$ Remove column φ_i from $\mathbf{C}$
7: $\quad \psi = \psi \cup \{(u_i \rightarrow v_{\varphi_i})\}$
8: **end for**
9: **return** $(d_\psi, order)$

4 Experimental Evaluation

4.1 Experimental Setup

The experimental evaluation aims at investigating the benefit of the post processing search procedure proposed in the present paper in a graph matching scenario. In particular, we measure the approximation error and the computation time on three different real world data sets from the IAM graph database repository [16][3]. The first graph data set involves graphs that represent molecular compounds (AIDS). The graphs from the second and third data set represent images of fingerprints (FP) and images of symbols from architectural and electronic drawings (GREC). For details on the graph extraction methods and the graph characteristics we refer to [16]. From all data sets, subsets of 1,000 graphs are randomly selected on which 1,000,000 pairwise graph edit distance computations are conducted.

Rather than choosing an appropriate starting temperature T it might be more intuitive to define the probabilities P_s and P_e of accepting a worse solution at the

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

beginning and at the end of the optimization process, respectively. Eventually, one can define the starting and end temperature according to

$$T_s = \frac{1}{\ln(P_s)} \quad \text{and} \quad T_e = \frac{1}{\ln(P_e)}.$$

The end temperature T_e is merely used for a proper definition of the decrease factor F for the temperature $T_{(n+1)}$ in iteration $(n+1)$ with respect to the current temperature T_n. Formally, given the the total number of iterations N, the decrease factor F can be defined as

$$F = \left(\frac{T_e}{T_s}\right)^{1/(N-1)}.$$

In our evaluation we set the starting and end probability to $P_s = 0.8$ and $P_e = 0.01$ and we test the novel algorithm with $N = 1000$ and $N = 10,000$ iterations (referred to as BP-SA(1) and BP-SA(10), respectively).

4.2 Empirical Investigation

In Table 1 the mean computation time per graph pair (t) as well as the approximation error, i.e. the degree of overestimation (o), is indicated for the different edit distance algorithms on all data sets. Exact-GED and BP-GED refer to an exact computation via tree search algorithm and the original approximation framework presented in [8] (these two algorithms are the reference systems). The first reference system is mainly used to control whether our novel method's computation time remains below the computation time of an exact algorithm, while the second reference system is mainly used to investigate the impact of the novel method on the approximation quality.

We first focus on the computation time. We note that BP-GED needs some fractions of a millisecond on average for one graph matching. With the proposed extension we observe an increase of the mean computation time to 1–3 ms and 10–25 ms on average with BP-SA(1) and BP-SA(10), respectively. Yet, comparing these matching times with the matching times of the exact algorithm (which takes 3–5 s per matching on average), the increase of the run time seems to be acceptable.

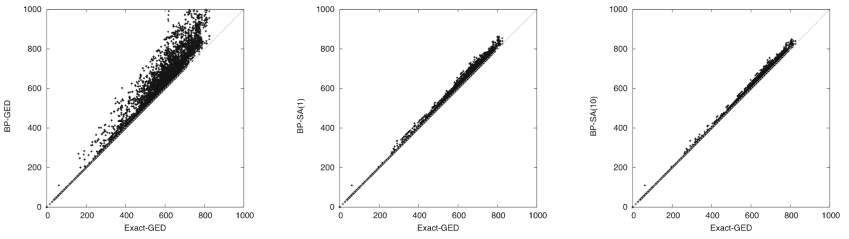
Taking the sum of distances of BP-GED as reference point for the overestimation (i.e. we take the sum of distances returned by BP-GED as 100%), we observe reductions of the approximation error of approximately 77%, 98%, and 85% on the three data sets (using BP-SA(1)). The approximation error can be further reduced by increasing the number of iterations from $N = 1000$ to $N = 10,000$. That is, with 10,000 iterations we can report reductions of the approximation error of approximately 89%, 99%, and 93%.

The substantial improvement of the approximation accuracy can be also observed in the scatter plots in Fig. 1 (on the GREC data set[4]). These scatter

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

Table 1. The mean run time for one matching (t) and overestimation error (o) using a specific graph edit distance algorithm.

Data Set	Algorithm							
	Exact-GED		BP-GED		BP-SA(1)		BP-SA(10)	
	t	o	t	o	t	o	t	o
AIDS	5.63 s	0.00	0.07 ms	100.00	2.95 ms	23.42	25.44 ms	10.82
FP	5.00 s	0.00	0.29 ms	100.00	1.24 ms	1.91	9.46 ms	0.44
GREC	3.10 s	0.00	0.20 ms	100.00	2.21 ms	14.23	17.84 ms	6.40



(a) BP-GED (b) BP-SA(1) (c) BP-SA(10)

Fig. 1. Exact (x-axis) vs. approximate (y-axis) graph edit distance on the GREC data computed with (a) original framework BP-GED, (b) BP-SA(1), and (c) BP-SA(10).

plots give us a visual representation of the accuracy of our approximations. We plot for each pair of graphs its exact (horizontal axis) and approximate (vertical axis) distance value. The reduction of the overestimation using our proposed extension is clearly observable and illustrates the power of BP-SA.

5 Conclusions

In the present paper we propose to improve the graph edit distance quality of a recent approximation framework by means of simulated annealing. The basic idea of this search process is to start with the upper- and lower bound on the true edit distance and then randomly search in the neighborhood of the current solution. As long as we improve the current distance, the new solution is accepted. Yet, also a deterioration of the solution might be accepted by the algorithm with a certain probability (that depends on both the level of deterioration and the search progress). This allows the search procedure to overcome local minima and possibly find the globally optimal solution. With an empirical investigation on three data sets we observe that substantial improvements of the approximation quality can be made with our novel extension.

Acknowledgements. This work has been supported by the *Hasler Foundation* Switzerland.

References

1. Conte, D., Foggia, P., Sansone, C., Vento, M.: Thirty years of graph matching in pattern recognition. Int. J. Pattern Recognit. Artif. Intell. **18**(3), 265–298 (2004)
2. Foggia, P., Percannella, G., Vento, M.: Graph matching and learning in pattern recognition in the last 10 years. Int. J. Pattern Recognit. Artif. Intell. **28**(1), 1450001 (2014)
3. Gaüzère, B., Brun, L., Villemin, D.: Two new graphs kernels in chemoinformatics. Pattern Recognit. Lett. **33**(15), 2038–2047 (2012)
4. Kinable, J., Kostakis, O.: Malware classification based on call graph clustering. J. Comput. Virol. **7**(4), 233–245 (2011)
5. Dickinson, P.J., Bunke, H., Dadej, A., Kraetzl, M.: Matching graphs with unique node labels. Pattern Anal. Appl. **7**(3), 243–254 (2004)
6. Wilson, R.C., Hancock, E.R., Luo, B.: Pattern vectors from algebraic graph theory. IEEE Trans. Pattern Anal. Mach. Intell. **27**(7), 1112–1124 (2005)
7. Bunke, H., Allermann, G.: Inexact graph matching for structural pattern recognition. Pattern Recognit. Lett. **1**, 245–253 (1983)
8. Riesen, K., Bunke, H.: Approximate graph edit distance computation by means of bipartite graph matching. Image Vis. Comput. **27**(4), 950–959 (2009)
9. Riesen, K., Bunke, H.: Improving bipartite graph edit distance approximation using various search strategies. Pattern Recognit. **48**(4), 1349–1363 (2015)
10. Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Equation of state calculations by fast computing machines. J. Chem. Phys. **21**(6), 1087 (1953)
11. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **4598**, 671–680 (1983)
12. Riesen, K.: Structural Pattern Recognition with Graph Edit Distance: Approximation Algorithms and Applications. Advances in Computer Vision and Pattern Recognition. Springer, Heidelberg (2015)
13. Burkard, R., Dell'Amico, M., Martello, S.: Assignment Problems. Society for Industrial and Applied Mathematics, Philadelphia (2009)
14. Riesen, K., Fischer, A., Bunke, H.: Estimating graph edit distance using lower and upper bounds of bipartite approximations. Int. J. Pattern Recognit. Artif. Intell. **29**(2), 1550011 (2015)
15. Riesen, K., Ferrer, M., Dornberger, R., Bunke, H.: Greedy graph edit distance. In: Perner, P. (ed.) MLDM 2015. LNCS, vol. 9166, pp. 3–16. Springer, Cham (2015). doi:10.1007/978-3-319-21024-7_1
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.) Structural, Syntactic, and Statistical Pattern Recognition. LNCS, vol. 5342, pp. 287–297. Springer, Heidelberg (2008)