A Quadratic Programming Approach to the Graph Edit Distance Problem

Michel Neuhaus[1] and Horst Bunke[2]

[1] LIP6, Université Pierre et Marie Curie
104 avenue du Président Kennedy, F-75016 Paris, France
mneuhaus@iam.unibe.ch
[2] Institute of Computer Science, University of Bern
Neubrückstrasse 10, CH-3012 Bern, Switzerland
bunke@iam.unibe.ch

Abstract. In this paper we propose a quadratic programming approach to computing the edit distance of graphs. Whereas the standard edit distance is defined with respect to a minimum-cost edit path between graphs, we introduce the notion of fuzzy edit paths between graphs and provide a quadratic programming formulation for the minimization of fuzzy edit costs. Experiments on real-world graph data demonstrate that our proposed method is able to outperform the standard edit distance method in terms of recognition accuracy on two out of three data sets.

1 Introduction

In structural pattern recognition, the edit distance measure has been widely used for error-tolerant graph matching. The successful application of graph edit distance is mainly due to its intuitive and universal definition. Based on a node and edge distortion model, the edit distance is defined as the minimum amount of distortion that is needed to transform a given graph into another one [1,2], which follows the intuitive understanding that the more dissimilar two graphs are, the more transformation operations have to be performed. Graph edit distance is applicable to arbitrarily labeled and arbitrarily structured graphs — and other data structures such as strings [3], trees [4], and hyper-graphs [5] — and can therefore be considered a universal matching scheme for complex patterns. In practice, the flexibility of graph edit distance, which allows us to assign weights to individual distortion operations based on the type of distortion and the involved nodes and edges, renders edit distance applicable to various practical graph matching tasks.

Computing the edit distance of two graphs results in a time and space complexity that is exponential in the number of nodes of the two graphs. Particularly in the presence of large graphs, the edit distance problem is computationally very demanding. In recent years, a number of methods have been proposed to render the computation of graph edit distance feasible. In [6], an approximate edit distance algorithm for planarly embedded nodes is introduced. The algorithm

F. Escolano and M. Vento (Eds.): GbRPR 2007, LNCS 4538, pp. 92–102, 2007.

exploits the position node information that is available in many graph representations in pattern recognition. The approximate edit distance is computed in an iterative procedure by successively optimizing local matching criteria. Two fast suboptimal variants of a standard edit distance algorithms are proposed in [7]. The idea is to restrict the matching process to promising candidates by applying a technique for search tree pruning and a re-weighting of edit costs. These approaches have in common that they attempt to refine the standard tree search algorithm for edit distance to speed up the computation.

In the present paper, we propose to circumvent the standard inefficient algorithm altogether by addressing the edit distance problem by means of quadratic programming. The basic idea is to formulate the minimum-cost optimization problem of edit distance in the well-known mathematical framework of quadratic programming [8], which allows us to tackle the complex graph matching problem using standard optimization methods. In the longer term, it would be desirable to develop fast (possibly suboptimal) optimizers for the particular edit distance quadratic programming formulation, which is not covered in this paper. Our main contribution is an alternative method for the computation of edit distance.

The quadratic programming approach leads us to the notion of fuzzy edit paths. The result of our method is either a minimum-cost fuzzy edit path or, after defuzzification, a standard edit path between two graphs. In this respect, the method we propose in this paper is loosely related to relaxation labeling techniques for graph matching [9,10], where the idea is to define the matching problem as a node labeling problem and to apply iterative procedures refining the labeling until a sufficiently accurate matching is obtained. Unlike these relaxation labeling techniques, which are sometimes defined for numerically labeled or weighted graphs only, the method we propose is applicable to arbitrarily labeled graphs and is closely related to the standard edit distance measure. In [11], a linear programming method for computing the edit distance of graphs with unlabeled edges that is somehow related to our approach is introduced.

This paper is structured as follows. In Section 2, we briefly introduce graph edit distance. The proposed quadratic programming formulation of edit distance is described in Section 3. Experimental results on three real-world graph data sets are given in Section 4. Finally, in Section 5 a few summarizing conclusions are drawn.

2 Graph Edit Distance

Graph edit distance is an error-tolerant dissimilarity measure on graphs. The edit distance method is applicable to arbitrarily labeled graphs, that is, graphs with any kind of labels attached to nodes and edges. A graph is commonly defined by a four-tuple $g = (V, E, \mu, \nu)$, where V denotes a finite set of nodes, $E \subseteq V \times V$ is a set of directed edges, $\mu : V \rightarrow L$ is a node labeling function assigning each node a label from alphabet L, and $\nu : E \rightarrow L$ is an edge labeling function. Note that in practical applications, numerical labels (attribute vectors) usually prevail. The idea of edit distance is to define a set of basic graph distortion operations, or

edit operations, and define the dissimilarity of two given graphs by the minimal amount of edit operations that are needed to transform one graph into the other one [1,2]. While the edit distance concept theoretically allows for a wide range of edit operations, for most applications it is sufficient to consider the insertion, deletion, and substitution of nodes and edges only. A node deletion operation, for instance, refers to the removal of a node and its adjacent edges, and an edge substitution operation is equivalent to changing the label of an edge. The edit distance method can be tailored to specific application by assigning each edit operation a cost value reflecting the strength of the corresponding distortion. For instance, changing an edge label by a small amount might often be considered a weaker distortion than the removal of a node together with all edges connected to this node. In this particular case, the edge substitution would be assigned a lower cost than the node deletion. The total edit costs of a given sequence of edit operations transforming one graph into another one, or edit path between the two graphs, is obtained by summing up the costs of the individual edit operations. Finally, the edit distance of two graphs is defined as the minimum cost edit path between them, that is, the least expensive way to edit one graph into the other one, given an edit operation model and an edit cost function. If we denote by $P(g, g')$ the set of edit paths transforming a graph g into a graph g' and by C the function assigning costs to edit operations, the edit distance of g and g' is defined by

$$d(g, g') = \min_{(w_1,\ldots,w_k)\in P(g,g')} \sum_{i=1}^{k} C(w_i) \ , \tag{1}$$

where $(w_1, \ldots, w_k)$ represents an edit path consisting of k edit operations.

The simplest way to compute edit distance is obviously to generate all edit paths between two graphs and determine the one with minimum costs. In more sophisticated approaches, lookahead techniques or heuristics are used to determine which edit paths seem to be promising candidates for exploration. A standard edit distance computation algorithm is based on an A* tree search algorithm with efficient heuristics [1,7,12]. The idea is to systematically explore all relevant edit paths by traversing, in a best-first fashion, a search tree with inner nodes representing partial edit paths and leaf nodes representing complete edit paths. The flexibility of edit distance, potentially allowing any node of one graph to be mapped to any node of the second graph, results in exponential computational costs in terms of time and space complexity. That is, the edit distance of graphs is typically tractable for graphs with up to about a dozen of nodes only.

3 Quadratic Programming for Graph Edit Distance

Quadratic programming is a particular type of mathematical optimization problem [8]. It turns out that the graph edit distance problem needs only a few slight adaptations to fit into the quadratic programming framework, which makes a new class of algorithms available for the computation of graph edit distance.

3.1 Quadratic Programming

Quadratic programming refers to a range of optimization problems satisfying a general mathematical form. In the following, the quadratic programming problem will be described and briefly discussed. First, let the set of real matrices of dimension $a \times b$ be denoted by $\mathbb{R}^{a \times b}$. For a given dimension $n \geq 1$, let us assume that a symmetric matrix $Q \in \mathbb{R}^{n \times n}$ and a vector $c \in \mathbb{R}^n$ are given. Furthermore for $l, m \geq 1$, let matrices $R \in \mathbb{R}^{l \times n}$ and $S \in \mathbb{R}^{m \times n}$ as well as vectors $u \in \mathbb{R}^l$ and $v \in \mathbb{R}^m$ be given. The general quadratic programming problem can then be formulated as [8]

$$\begin{aligned} &\text{Minimize } f(x) = \frac{1}{2} x' Q x + c' x \qquad \text{for } x \in \mathbb{R}^{\mathrm{n}} \\ &\text{such that} \\ &\qquad Rx = u \\ &\qquad Sx \geq v \ . \end{aligned} \tag{2}$$

Note that the vector inequality constraint in the last line means that all components of the two vectors must satisfy the inequality. Solving the quadratic programming problem consists of finding an $x \in \mathbb{R}^n$ that minimizes $f(x)$ such that the given equality and inequality conditions are satisfied. The expression *quadratic programming* is due to the fact that the target function $f(x)$ is a quadratic function of the argument x. The equality constraint can be seen as a compact representation of l independent equality conditions (one per line of matrix R), and similarly the inequality constraint is equivalent to m inequality conditions.

Quadratic programming problems can always be solved, or shown to be unfeasible, in a finite amount of time. However, the actual complexity of the computation depends strongly on the characteristics of the problem, in particular on the matrix Q and the number of relevant inequality constraints [8]. If Q is positive definite, for instance, the quadratic programming problem can typically be solved as efficiently as linear programming problems. Furthermore, it is also known in this case that there exists a globally optimal solution, provided that the equality and inequality constraints are satisfied for at least one vector. The methods commonly used to solve quadratic programming problems can roughly be divided into interior point methods, active set methods, and conjugate gradient methods [8]. In our experiments, we use the interior point algorithm from the Computational Optimization Program Library [13].

A classic example of a quadratic programming problem is the management of investment portfolios [8]. The idea is to model the tradeoff between risk and expected return for a collection of investments. Quadratic programming can be used to derive an investment strategy that predicts high returns with low variance. The popular support vector machine method for classification and regression is another example. The maximum-margin hyperplane separating two classes can be found by solving a quadratic programming problem [14], namely by minimizing the squared norm of the hyperplane weight vector given a number

of linear constraints. In the following, we will apply quadratic programming to the graph edit distance problem.

3.2 Fuzzy Edit Path

The standard graph edit distance is defined by the minimum-cost edit path between two graphs. A common interpretation of substitutions in an optimal edit path is that they indicate which parts of one graph can be identified in the other graph. That is, a set of node substitutions can be seen as a mapping of nodes of one graph to nodes of another graph. Analogously, deleted (or inserted) nodes and edges can be interpreted as those nodes and edges of the first graph (second graph) that cannot be matched, with sufficient accuracy, to nodes and edges of the second graph (first graph). Hence, given an edit path between two graphs, each node and edge is either substituted with another node and edge, or deleted or inserted.

The basic idea of fuzzy edit paths is to allow nodes and edges of one graph to be simultaneously assigned to several nodes and edges of another graph. In the following, let us assume that two graphs $g = (V, E, \mu, \nu)$ and $g' = (V', E', \mu', \nu')$ with $|V| = n$ and $|V'| = n'$ are given. Clearly, there exist $n \cdot n'$ distinct substitutions of a node $u \in V$ with a node $v \in V'$, such a substitution being denoted by $u \rightarrow v$. A fuzzy edit path is defined by assigning a weight to each possible node substitution. Formally, a fuzzy edit path between g and g' is a function $w : V \times V' \rightarrow [0, 1]$ satisfying the conditions

$$\sum_{v \in V'} w(u, v) = 1 \text{ for each } u \in V \quad \text{and} \quad \sum_{u \in V} w(u, v) = 1 \text{ for each } v \in V' \ . \quad (3)$$

This weighting function w can be understood as a kind of membership function reflecting how well a node substitution conforms to, or how strongly it violates, the structure and labels of the two graphs. The interpretation of a fuzzy edit path that is optimal with respect to some matching criterion is that two nodes u, v with a large value of $w(u, v)$ are likely to correspond to a good structural match, while nodes with small values of $w(u, v)$ should rather be considered unmatchable. The advantage of fuzzy edit paths over standard edit paths is that they allow us to integrate ambiguity directly in the definition of edit paths, instead of being forced to settle for one edit transformation for each node and edge.

In order to construct a standard edit path from a fuzzy edit path, a defuzzification procedure can be carried out. A straight-forward defuzzification method consists in selecting from all fuzzy node substitutions those with large fuzzy weights. The first node substitution to be inserted into the standard edit path is obtained by selecting from all fuzzy node substitutions the one with largest fuzzy weight, say the substitution $u \rightarrow v$. In the following steps, all fuzzy node substitutions involving u and v will no longer be considered. The second node substitution of the standard edit path is obtained by selecting from the remaining fuzzy node substitutions the one with largest fuzzy weight. Again, all fuzzy node

substitutions containing either one of the two nodes of the selected substitution are ignored in successive steps. This iterative procedure is continued until no more node substitutions can be extracted. The remaining nodes are considered equivalent to node deletions and insertions. Finally, edge operations are inferred from node operations. Note that in the computation of fuzzy edit paths, edge edit operation costs will be included in the definition of fuzzy weights attached to node substitutions. That is, not only the substitution of nodes, but also the edge structure plays a role in the defuzzification procedure outlined above.

3.3 Quadratic Programming Formulation

In the preceding paragraphs, fuzzy edit paths have been introduced as an extension to standard edit paths. The remaining question is how to compute a fuzzy edit path between two graphs that is optimal with respect to some node and edge matching criterion. The method we propose in this paper is based on a quadratic programming formulation of the graph matching problem. The basic idea is to encode node and edge edit costs in a cost matrix and minimize the overall costs corresponding to a fuzzy edit path.

Again, let the two graphs under consideration be denoted by $g = (V, E, \mu, \nu)$ and $g' = (V', E', \mu', \nu')$, and let $|V| = n$ and $|V'| = n'$. It is clear that there exist $n \cdot n'$ substitutions between g and g'. In view of this, we construct a real matrix $Q \in \mathbb{R}^{nn' \times nn'}$ where rows and columns are indexed by substitutions $u \to v$, where $u \in V, v \in V'$. That is, each row, and the corresponding column, of the matrix is associated with one distinct node substitution. The matrix Q is then constructed in such a way that diagonal entries hold the costs of node substitutions, while off-diagonal entries correspond to edge edit costs. The entry at position $(u \to v, u \to v)$ is set to the node substitution costs of $u \to v$; the entry at position $(u \to v, p \to q)$ is set to the edge edit costs resulting from substituting $u \to v$ and $p \to q$, depending on the existence of edges between u and p as well as between v and q. It should be noted that edit costs can be defined for any kind of node and edge labels, including symbols from a finite alphabet and complex labels such as strings. The proposed approach is thus not limited to graphs with numerical labels, but applicable to arbitrarily labeled graphs, which is one of the strengths of graph edit distance.

An example of two graphs with $n = n' = 3$ is provided in Fig. 1, where nodes are labeled with a two-dimensional position attribute and edges are unlabeled. It is clear that in this example the nine possible distinct node substitutions are $A \to a, A \to b, A \to c, B \to a, B \to b, B \to c, C \to a, C \to b, C \to c$. When constructing the 9×9 matrix Q, each row and column is associated with one of these substitutions. In Fig. 2, an example cost matrix Q is shown for the two graphs in Fig. 1. Note that in this example, node substitution costs are set equal to the squared Euclidean distance of the two node labels, and node and edge insertion and deletion costs are set to a constant value of 10. The substitution of unlabeled edges can be carried out for free. For example, since node A is labeled with $(1, 1)$ and node a with $(1, 6)$, the substitution $A \to a$ results in costs $Q_{A \to a, A \to a} = 25$. Since there exists an edge between A and B as well as

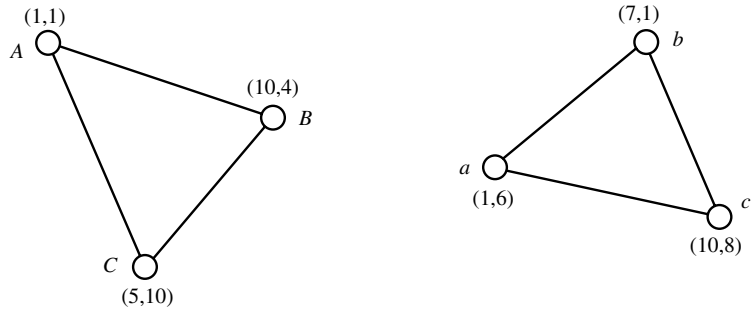

Fig. 1. Two example graphs g (left) and g' (right)

an edge between a and b, the substitutions $A \rightarrow a$ and $B \rightarrow b$ involve no edge operations costs, hence $Q_{A\rightarrow a,B\rightarrow b} = 0$. As node A is not connected to itself by an edge, the substitutions $A \rightarrow a$ and $A \rightarrow b$ involve the insertion of an edge, which leads to $Q_{A\rightarrow a,A\rightarrow b} = 10$.

Recall that fuzzy edit paths are defined in Sect. 3.2 as functions assigning weights to all possible node substitutions between two graphs. Also, the matrix Q consists of one row, and column, per node substitution. In view of this, we define a fuzzy cost function assigning each row of Q a weight according to the rules stated in Sect. 3.2. That is, each row, and the corresponding column, is associated with a node substitution and a fuzzy weight. It should be noted that these fuzzy weights are not pre-defined, but to be determined in the optimization process. Hence, the idea of the reformulated graph matching problem is to find fuzzy weights that satisfy the conditions of a fuzzy edit path and minimize the structural error. To this end, we propose to minimize the expression $x'Qx$, where x denotes the $n \cdot n'$-dimensional vector of fuzzy weights, one for each row of Q. The minimization is carried out over all fuzzy weights x satisfying the conditions defined in Sect. 3.2. In this optimization formulation, the weight associated with a node substitution $u \rightarrow v$ will influence not only the weighting of the node substitution costs of $u \rightarrow v$ (in the diagonal entry $Q_{u\rightarrow v,u\rightarrow v}$), but also all edge edit costs involving the substitution $u \rightarrow v$ (in off-diagonal entries $Q_{u\rightarrow v,p\rightarrow q}$ and $Q_{p\rightarrow q,u\rightarrow v}$). Clearly, this optimization process aims at assigning large weights to node substitutions that involve low node and edge costs, and assigning small weights to node substitutions that result in high costs. Note that this optimization principle is not identical to the minimum cost edit path concept in standard graph edit distance, but the intuitive interpretation of minimizing penalty costs for structural errors is comparable.

The optimization problem described above can be formulated in the standard quadratic programming framework. To this end, the matrix Q in Eq. 2 is defined as the cost matrix Q described above, the solution vector x in Eq. 2 is the weight vector x mentioned above, and vector c in Eq. 2 is the zero vector. Furthermore, it is easy to see that the conditions of consistent fuzzy weights can be formulated in terms of equality and inequality conditions — $Rx = u$ stating that fuzzy weights sum up to 1 as shown in Eq. 3, and $Sx \geq v$ restricting considerations to fuzzy

$$Q = \begin{pmatrix} 25 & 10 & 10 & 10 & 0 & 0 & 10 & 0 & 0 \\ 10 & 36 & 10 & 0 & 10 & 0 & 0 & 10 & 0 \\ 10 & 10 & 130 & 0 & 0 & 10 & 0 & 0 & 10 \\ 10 & 0 & 0 & 85 & 10 & 10 & 10 & 0 & 0 \\ 0 & 10 & 0 & 10 & 18 & 10 & 0 & 10 & 0 \\ 0 & 0 & 10 & 10 & 10 & 16 & 0 & 0 & 10 \\ 10 & 0 & 0 & 10 & 0 & 0 & 32 & 10 & 10 \\ 0 & 10 & 0 & 0 & 10 & 0 & 10 & 85 & 10 \\ 0 & 0 & 10 & 0 & 0 & 10 & 10 & 10 & 29 \end{pmatrix}$$

$A \to a$	0.662	Solution
$A \to b$	0.297	
$A \to c$	0.041	
$B \to a$	0.000	
$B \to b$	0.619	
$B \to c$	0.381	
$C \to a$	0.338	
$C \to b$	0.084	
$C \to c$	0.578	

$A \to \ldots$:	$0.662 + 0.297 + 0.041 = 1$	Constraints satisfied
$B \to \ldots$:	$0.000 + 0.619 + 0.381 = 1$	
$C \to \ldots$:	$0.338 + 0.084 + 0.578 = 1$	
$\ldots \to a$:	$0.662 + 0.000 + 0.338 = 1$	
$\ldots \to b$:	$0.297 + 0.619 + 0.084 = 1$	
$\ldots \to c$:	$0.041 + 0.381 + 0.578 = 1$	

Fig. 2. Example quadratic programming problem matrix Q (corresponding to the graphs in Fig. 1) and solution weight vector satisfying fuzzy edit path constraints

weights between 0 and 1. In Fig. 2, the result of the quadratic programming approach to matching the two graphs in Fig. 1 is shown. The solution vector clearly satisfies the fuzzy edit path constraints. After defuzzification, we obtain the same optimal edit path as the standard edit distance algorithm, $\{A \to a, B \to b, C \to c\}$. Note that from the solution vector, it is not only possible to extract the most likely edit path, but also other edit paths that seem to be rather likely, such as the one with edit operations $\{A \to b, B \to c, C \to a\}$ in our case.

4 Experimental Results

In this section, we evaluate how the proposed method performs in comparison to the standard edit distance method on three graph data sets representing letters, images, and diatoms. These data sets are considered difficult because of non-compact and overlapping classes.

The letter data set consists of line drawings of 15 capital letters. Nodes are labeled with a position attribute, and edges are unlabeled. The graphs are split into a training set and validation set each of size 150 and a test set of size 750. The image data set consists of 5 classes of region adjacency graphs representing images after processing and filtering [15]. Nodes contain a color histogram attribute, and edges contain a region adjacency attribute. The diatom data set contains a total of 162 patterns split into a training set, validation set, and test set of equal size. The diatom data set is derived from microscopic images of 22 diatom classes. These images first undergo a segmentation process and are then transformed into graphs [16]. Nodes contain attributes describing region

Table 1. Recognition accuracy on validation set (VS) and test set (TS)

Data set	Method	Accuracy VS	Accuracy TS	Running time
Letter	Standard	67.3	69.3	12.4'
	Proposed	76.7	74.9 •	19.5'
Image	Standard	64.8	48.1	9s
	Proposed	72.2	59.3 •	18s
Diatoms	Standard	86.5	66.7 •	8s
	Proposed	54.1	47.2	15s

• Improvement over other method statistically significant ($\alpha = 0.05$).

features, and edges contain a common boundary attribute. This data set is split into a training set and validation set each of size 37 and a test set of size 36.

The recognition accuracy of a k-nearest-neighbor classifier based on the standard edit distance method [1,6] and the quadratic programming method proposed in this paper are given in Table 1. Note that the relevant classification accuracy is the one on the test set. In two out of three cases, the proposed method outperforms the standard method significantly, while on the third data set, the proposed method is clearly inferior. Note that the test set results marked with a dot are significantly better than the other ones on a statistical significance level of $\alpha = 0.05$. These classification results show that the proposed method based on quadratic programming constitutes a viable alternative to the standard edit distance method on certain data sets. As far as the running time is concerned, the proposed method seems to require typically twice the running time of the standard algorithm. It should be noted, however, that the efficiency of the proposed method heavily depends on the implementation of the quadratic programming algorithm at hand.

5 Conclusions

In this paper we propose a novel approach to computing graph edit distance. The idea is based on fuzzy edit paths between graphs. In contrast to standard edit distance, the result of the graph matching process is not a transformation of one graph into the other one (an edit path), but rather for each possible node substitution a computed fuzzy weight. The higher this fuzzy weight, the less the corresponding node substitution violates the node and edge structure and labels of the two graphs. For the computation of fuzzy edit paths from standard edit operation costs, a quadratic programming algorithm can be used. The aim is to compute a fuzzy edit path that minimizes the structural error in a manner similar to the minimization of edit costs in graph edit distance. The resulting fuzzy edit path can then easily be turned into a standard edit path by means of a defuzzification procedure. An experimental evaluation on graphs representing line drawings, images, and diatoms demonstrates that the proposed method significantly outperforms the standard edit distance method on two out

of three data sets, although the standard edit distance algorithm is typically twice as fast.

In the future, we intend to further study the applicability of quadratic programming principles to graph matching. We would like to investigate whether it may be advantageous to use other quadratic programming formulations of the minimum-cost edit path problem than the one presented in this paper. Also, while applying a defuzzification procedure to fuzzy edit path turns out to be advantageous to directly using fuzzy edit paths for classification, there does not exist a unique way to turn fuzzy edit paths into standard edit paths. For instance, applying error-minimization techniques such as Munkres' algorithm for defuzzification might be a viable alternative to our proposed iterative procedure.

Acknowledgments

The first author was supported by the Swiss National Science Foundation under grant no. PBBE2-113362.

References

1. Bunke, H., Allermann, G.: Inexact graph matching for structural pattern recognition. Pattern Recognition Letters 1, 245–253 (1983)
2. Sanfeliu, A., Fu, K.: A distance measure between attributed relational graphs for pattern recognition. IEEE Transactions on Systems, Man, and Cybernetics (Part B) 13(3), 353–363 (1983)
3. Wagner, R., Fischer, M.: The string-to-string correction problem. Journal of the Association for Computing Machinery 21(1), 168–173 (1974)
4. Selkow, S.: The tree-to-tree editing problem. Information Processing Letters 6(6), 184–186 (1977)
5. Bunke, H., Dickinson, P., Kraetzl, M.: Theoretical and algorithmic framework for hypergraph matching. In: Roli, F., Vitulano, S. (eds.) ICIAP 2005. LNCS, vol. 3617, pp. 463–470. Springer, Heidelberg (2005)
6. 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., de Ridder, D. (eds.) Structural, Syntactic, and Statistical Pattern Recognition. LNCS, vol. 3138, pp. 180–189. Springer, Heidelberg (2004)
7. 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.) Structural, Syntactic, and Statistical Pattern Recognition. LNCS, vol. 4109, pp. 163–172. Springer, Heidelberg (2006)
8. Nocedal, J., Wright, S.: Numerical Optimization. Springer, Heidelberg (2000)
9. Christmas, W., Kittler, J., Petrou, M.: Structural matching in computer vision using probabilistic relaxation. IEEE Transactions on Pattern Analysis and Machine Intelligence 17(8), 749–764 (1995)
10. Wilson, R., Hancock, E.: Structural matching by discrete relaxation. IEEE Transactions on Pattern Analysis and Machine Intelligence 19(6), 634–648 (1997)

11. Justice, D., Hero, A.: A binary linear programming formulation of the graph edit distance. IEEE Trans. on Pattern Analysis and Machine Intelligence 28(8), 1200–1214 (2006)
12. Tsai, W., Fu, K.: Error-correcting isomorphism of attributed relational graphs for pattern analysis. IEEE Transactions on Systems, Man, and Cybernetics (Part B) 9(12), 757–768 (1979)
13. Zhang, X., Ye, Y.: Computational Optimization Program Library: Convex Quadratic Programming. University of Iowa (1998)
14. Schölkopf, B., Smola, A.: Learning with Kernels. MIT Press, Cambridge, MA (2002)
15. Le Saux, B., Bunke, H.: Feature selection for graph-based image classifiers. In: Marques, J.S., Pérez de la Blanca, N., Pina, P. (eds.) IbPRIA 2005. LNCS, vol. 3523, pp. 147–154. Springer, Heidelberg (2005)
16. 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)