Combining Bipartite Graph Matching and Beam Search for Graph Edit Distance Approximation

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. Graph edit distance (GED) is a powerful and flexible graph dissimilarity model. Yet, exact computation of GED is an instance of a quadratic assignment problem and can thus be solved in exponential time complexity only. A previously introduced approximation framework reduces the computation of GED to an instance of a linear sum assignment problem. Major benefit of this reduction is that an optimal assignment of nodes (including local structures) can be computed in polynomial time. Given this assignment an approximate value of GED can be immediately derived. Yet, the primary optimization process of this approximation framework is able to consider local edge structures only, and thus, the observed speed up is at the expense of approximative, rather than exact, distance values. In order to improve the overall approximation quality, the present paper combines the original approximation framework with a fast tree search procedure. More precisely, we regard the assignment from the original approximation as a starting point for a subsequent beam search. In an experimental evaluation on three real world data sets a substantial gain of assignment accuracy can be observed while the run time remains remarkable low.

1 Introduction

Graphs, which consist of a finite set of nodes connected by edges, are the most general data structure in computer science. Due to the ability of graphs to represent properties of entities and binary relations at the same time, a growing interest in graph-based object representation can be observed in various fields. In bio- and chemoinformatics, for instance, graph based representations are intensively used [1–3]. Another field of research where graphs have been studied with emerging interest is that of web content and data mining [4, 5]. Image classification [6, 7], graphical symbol and character recognition [8, 9], and computer network analysis [10] are further areas of research where graph based representations draw the attention.

Various procedures for evaluating the similarity or dissimilarity of graphs – known as *graph matching* – have been proposed in the literature [11]. The present paper addresses the issue of processing arbitrarily structured and arbitrarily labeled graphs. Hence, the graph matching method actually employed has to be able to cope with directed and undirected, as well as with labeled and unlabeled graphs. If there are labels on nodes, edges, or both, no constraints on the label alphabet should compromise the representational power of the employed graphs. Anyhow, the matching framework should in any case be flexible enough to be adopted and tailored to certain problem specifications. As it turns out, *graph edit distance* [12, 13] meets both requirements, viz. flexibility and expressiveness.

The major drawback of graph edit distance is its computational complexity which is 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. [14–17]). 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 [18]. 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 core of our framework is able to consider only local, rather than global, edge structure. The main objective of the present paper is to significantly reduce the overestimation of edit distances. To this end, the distance approximation procedure of [18] is combined with a fast (but suboptimal) tree search algorithm, namely *beam search*. Beam search has been employed before as a stand-alone approximation scheme for graph edit distance computation [17]. The present paper adapts this search algorithm for the task of systemically improving the original node assignment and the corresponding edit distance approximation.

The remainder of this paper is organized as follows. Next, in Sect. 2 the concept and computation of graph edit distance as well as the original framework for graph edit distance approximation [18] are summarized. In Sect. 3 the combination of this framework with a beam 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.

2 Graph Edit Distance Computation

2.1 Exact Computation Based on A*

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 $\upsilon = (e_1, \ldots, e_k)$ of k edit operations that transform g_1 completely into g_2 is called an *edit path* between g_1 and g_2.

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 $c(e_i)$ for each edit operation $e_i \in \upsilon$, 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. Clearly, between two similar graphs, there should exist an inexpensive edit path, representing low cost operations, while for dissimilar graphs an edit path with high cost is needed. Consequently, the *edit distance* of two graphs is defined by the minimum cost edit path between two graphs:

$$d(g_1, g_2) = \min_{(e_1,\ldots,e_k)\in\Upsilon(g_1,g_2)} \sum_{i=1}^{k} c(e_i)$$

The exact computation of 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 [19]. The basic idea is to organize the underlying search space as an ordered tree. The root node of the search tree represents the starting point of our search procedure, inner nodes of the search tree correspond to partial edit paths, and leaf nodes represent complete – not necessarily optimal – edit paths.

Such a search tree is constructed dynamically at runtime as follows. The nodes of the source graph g_1 are processed in a fixed order $u_1, u_2, \ldots, u_n$. The deletion $(u_i \rightarrow \varepsilon)$ and all available substitutions $\{(u_i \rightarrow v_{(1)}), \ldots, (u_i \rightarrow v_{(t)})\}$ of a node u_i are thereby considered simultaneously. This produces $(t+1)$ successor nodes in the search tree. If all nodes of the first graph have been processed in an inner node of the tree, the remaining nodes of the second graph are inserted in a single step (which completes the edit path).

A set *open* of partial edit paths contains the search tree nodes to be processed in the next steps. The most promising partial edit path $\upsilon \in$ *open*, i.e. the one with minimal cost so far, is always chosen first (best-first search algorithm). This procedure guarantees that the complete edit path found by the algorithm first is always optimal in the sense of providing minimal cost among all possible competing paths.

2.2 Bipartite Graph Edit Distance Approximation

A major drawback of the procedure described in the last section is its computational complexity. In fact, the problem of graph edit distance can be reformulated as an instance of a *Quadratic Assignment Problem* (*QAP*) [20]. QAPs have been

[1] For edges we use a similar notation.

introduced in [21] and belong to the most difficult combinatorial optimization problems for which only exponential run time algorithms are known to date[2].

The graph edit distance approximation framework introduced in [18] reduces the QAP of graph edit distance computation to an instance of a *Linear Sum Assignment Problem* (*LSAP*) which can be – in contrast with QAPs – efficiently solved.

In order to translate the problem of graph edit distance computation to an instance of an LSAP, the graphs to be matched are subdivided into individual nodes plus local structures in a first step. Next, these independent sets of nodes including local structures are optimally assigned to each other. Finally, an approximate graph edit distance value is derived from this optimal node assignment. In the next paragraphs of this section, these three major steps of our framework are discussed in greater detail.

Assume that the graphs to be matched consists of node sets $V_1 = \{u_1, \ldots, u_n\}$ and $V_2 = \{v_1, \ldots, v_m\}$, respectively. A cost matrix $\mathbf{C}$ is then defined 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.

Note that the described extension of cost matrix $\mathbf{C}$ to dimension $(n+m) \times (n+m)$ is necessary since assignment algorithms for LSAPs expect every entry of the first set to be assigned with exactly one entry of the second set (and vice versa), and we want the optimal matching to be able to possibly include several node deletions and/or insertions. Moreover, matrix $\mathbf{C}$ is by definition quadratic. Consequently, standard algorithms for LSAPs can be used to find the minimum cost assignment.

In order to integrate knowledge about the graph's edge structure, to each cost of a node edit operation c_{ij} the minimum sum of edge edit operation costs,

[2] QAPs belong to the class of $\mathcal{NP}$*-complete* problems. That is, an exact and efficient algorithm for the graph edit distance problem can not be developed unless $\mathcal{P} = \mathcal{NP}$.

implied by the corresponding node operation, is added. That is, we encode the matching cost arising from the local edge structure in the individual entries of matrix $\mathbf{C}$.

The second step of our framework consists in applying an assignment algorithm to the square cost matrix $\mathbf{C}$ in order to find the minimum cost assignment of the nodes and their local edge structure of g_1 to the nodes and their local edge structure of g_2. Note that this task exactly corresponds to an instance of an LSAP and can thus be solved in polynomial time by means of Munkres' algorithm [22], the algorithm of Volgenant-Jonker [23], or others [24][3].

Formally, LSAP optimization procedures operate on a cost matrix $\mathbf{C} = (c_{ij})$ and find a permutation $(\varphi_1, \ldots, \varphi_{n+m})$ of the integers $(1, 2, \ldots, (n+m))$ that minimizes the overall mapping cost $\sum_{i=1}^{(n+m)} c_{i\varphi_i}$. In our scenario, this permutation corresponds to a mapping

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

of nodes. Note that mapping ψ includes node assignments of the form $(u_i \rightarrow v_j)$, $(u_i \rightarrow \varepsilon)$, $(\varepsilon \rightarrow v_j)$, and $(\varepsilon \rightarrow \varepsilon)$ (the latter can be dismissed, of course). Mapping ψ can also be interpreted as partial edit path considering edit operations on nodes only.

In the third step of our framework the partial edit path ψ is completed according to the node edit operations. 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 all of its adjacent nodes. Hence, given the set of node operations in ψ the global edge structures from g_1 and g_2 can be edited accordingly. The cost of the complete edit path is finally returned as an approximate graph edit distance. We denote the approximated distance value between graphs g_1 and g_2 according to mapping ψ with $d_{\langle\psi\rangle}(g_1, g_2)$ (or $d_{\langle\psi\rangle}$ for short).

Note that the edit path corresponding to $d_{\langle\psi\rangle}(g_1, g_2)$ considers the edge structure of g_1 and g_2 in a global and consistent way while the optimal node mapping ψ from step 2 is able to consider the structural information in an isolated way only (single nodes and their adjacent edges). Hence, the distances found by this approximation framework are – in the optimal case – equal to, or – in a suboptimal case – larger than the exact graph edit distance. Yet, the proposed reduction of graph edit distance to an LSAP allows the approximate graph edit distance computation in polynomial time complexity. For the remainder of this paper we denote this graph edit distance approximation algorithm with *BP* (*Bipartite*).

3 Improving the Node Assignment Using Beam Search

In an experimental evaluation in [18] we observed that the overestimation of BP is very often due to a few incorrectly assigned nodes in ψ. That is, only few node

[3] In [18] Munkres' algorithm is deployed, while in [25] also other algorithms have been tested for graph edit distance approximation.

assignments from the second step are responsible for additional (unnecessary) edge operations in the third step (and the resulting overestimation of the true edit distance). Our novel procedure ties in at this observation. That is, the node assignment ψ of our framework is used as a starting point for a subsequent search in order to improve the quality of the distance approximation (rather than using the assignment for graph edit distance approximation directly).

The basic idea of our search procedure is that the original node assignment ψ is systematically varied by swapping the target nodes v_{φ_i} and v_{φ_j} of two node assignments $(u_i \rightarrow v_{\varphi_i}) \in \psi$ and $(u_j \rightarrow v_{\varphi_j}) \in \psi$. For each swap it is verified whether (and to what extent) the derived distance approximation stagnates, increases or decreases. For a systematic variation of mapping ψ a tree search with ψ as the starting point is carried out.

The tree nodes in our search procedure correspond to triples $(\psi, q, d_{\langle\psi\rangle})$, where ψ is a certain node assignment, q denotes the depth of the tree node in the search tree and $d_{\langle\psi\rangle}$ is the approximate distance value corresponding to ψ. The root node of the search tree refers to the optimal node assignment

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

found by our former algorithm BP. Hence, the root node (with depth $= 0$) is given by the triple $(\psi, 0, d_{\langle\psi\rangle})$. Subsequent tree nodes $(\psi', q, d_{\langle\psi'\rangle})$ with depth $q = 1, \ldots, (m+n)$ contain node assignments ψ' with swapped element $(u_q \rightarrow v_{\varphi_q})$.

As usual in tree search based methods, a set *open* is employed that holds all of the unprocessed tree nodes. We keep the tree nodes in *open* sorted in ascending order according to their depth in the search tree (known as *breadth-first search*). Thus, at position 1 of *open* the tree node with smallest depth among all unprocessed tree nodes can be found. As a second order criterion the approximate edit distance $d_{\langle\psi\rangle}$ is used. That is, if two tree nodes have same depth in the search tree, they are queued in *open* according to ascending distance values.

Note that a best-first search algorithm, where *open* is sorted in ascending order according to the cost of the respective solution, would not be suitable for the present task. Best-first search algorithms expect that the cost of a solution increases monotonically with the increase of the depth in the search tree. Obviously, this is not the case in our scenario since for two tree nodes $(\psi', q', d_{\langle\psi'\rangle})$ and $(\psi'', q'', d_{\langle\psi''\rangle})$ with $q' < q''$, it must not necessarily hold that $d_{\langle\psi'\rangle} < d_{\langle\psi''\rangle}$. This is due to the fact that each tree node in the search tree represents a complete node mapping with the corresponding graph edit distance approximation value (in contrast with exact computations of graph edit distance, where inner tree nodes always refer to incomplete mappings).

The extended framework BP with the tree search based improvement is given in Alg. 1 (the first three lines correspond to the three major steps of the original approximation). Before the main loop of the search procedure starts, *open* is initialized with the root node (line 4). As long as *open* is not empty, we retrieve (and remove) the triple $(\psi, q, d_{\langle\psi\rangle})$ at the first position in *open* (the one with

Algorithm 1. BP-Beam(g_1, g_2) (Meta Parameter: b)

```
1.  Build cost matrix $\mathbf{C} = (c_{ij})$ according to the input graphs $g_1$ and $g_2$
2.  Compute optimal node assignment $\psi = \{u_1 \rightarrow v_{\varphi_1}, u_2 \rightarrow v_{\varphi_2}, \ldots, u_{m+n} \rightarrow v_{\varphi_{m+n}}\}$ on $\mathbf{C}$
3.  $d_{best} = d_{\langle\psi\rangle}(g_1, g_2)$
4.  Initialize $open = \{(\psi, 0, d_{\langle\psi\rangle}(g_1, g_2))\}$
5.  while $open$ is not empty do
6.      Remove first tree node in $open$: $(\psi, q, d_{\langle\psi\rangle}(g_1, g_2))$
7.      for $j = (q+1), \ldots, (m+n)$ do
8.          $\psi' = \psi \setminus \{u_{q+1} \rightarrow v_{\varphi_{q+1}}, u_j \rightarrow v_{\varphi_j}\} \cup \{u_{q+1} \rightarrow v_{\varphi_j}, u_j \rightarrow v_{\varphi_{q+1}}\}$
9.          Derive approximate edit distance $d_{\langle\psi'\rangle}(g_1, g_2)$
10.         $open = open \cup \{(\psi', q+1, d_{\langle\psi'\rangle}(g_1, g_2))\}$
11.         if $d_{\langle\psi'\rangle}(g_1, g_2) < d_{best}$ then
12.             $d_{best} = d_{\langle\psi'\rangle}(g_1, g_2)$
13.         end if
14.     end for
15.     while size of $open > b$ do
16.         Remove tree node with highest approximation value $d_{\langle\psi\rangle}$ from $open$
17.     end while
18. end while
19. return $d_{best}$
```

minimal depth and minimal distance value), generate the successors of this specific tree node and add them to *open* (line 6 – 10). That is, similarly to exact computation of the graph edit distance the search tree is dynamically built at run time.

The successors of tree node $(\psi, q, d_{\langle\psi\rangle})$ are generated as follows. The assignments of our original node matching ψ are processed according to the depth q of the current search tree node. That is, at depth q the assignment $u_q \rightarrow v_{\varphi_q}$ is processed and swapped with other assignments. Formally, in order to build the set of successor of node $(\psi, q, d_{\langle\psi\rangle})$ all pairs of node assignments $(u_{q+1} \rightarrow v_{\varphi_{q+1}})$ and $(u_j \rightarrow v_{\varphi_j})$ with $j = (q+1), \ldots, (n+m)$ are individually regarded. For each of these pairs, the target nodes $v_{\varphi_{q+1}}$ and v_{φ_j} are swapped resulting in two new assignments $(u_{q+1} \rightarrow v_{\varphi_j})$ and $(u_j \rightarrow v_{\varphi_{q+1}})$. In order to derive node mapping ψ' from ψ, the original node assignment pair is removed from ψ and the swapped node assignment is added to ψ (see line 8). On line 9 the corresponding distance value $d_{\langle\psi'\rangle}$ is derived and finally, the triple $(\psi', q+1, d_{\langle\psi'\rangle})$ is added to *open* (line 10). Since index j starts at $(q+1)$ we also allow that a certain assignment $u_{q+1} \rightarrow v_{\varphi_{q+1}}$ remains unaltered at depth $(q+1)$ in the search tree.

Since every tree node in our search procedure corresponds to a complete solution and the cost of these solutions neither monotonically decrease nor increase with growing depth in the search tree, we need to buffer the best possible distance approximation found during the tree search (lines 11 – 13 take care of that by checking the distance value of every successor node that has been created).

Note that the algorithmic procedure described so far exactly corresponds to a breadth-first search. That is, the procedure described above explores the space of all possible variations of ψ through pairwise swaps and return the best possible approximation (which corresponds to the exact edit distance, of course). However, such an exhaustive search is both unreasonable and intractable.

In [17] a variant of an A*-algorithm, referred to as *beam search*, has been used in order to approximate graph edit distance from scratch. The basic idea of beam search is that only a fixed number b of nodes to be processed are kept in *open*. This idea can be easily integrated in our search procedure as outlined above. Whenever the **for**-loop on lines $7-14$ has added altered assignments to *open*, only the b assignments with the lowest approximate distance values are kept, and the remaining tree nodes in *open* are removed. This means that not the full search space is explored, but only those nodes are expanded that belong to the most promising assignments (line $15-17$). Note that parameter b can be used as trade-off parameter between run time and approximation quality. That is, it can be expected that larger values of b lead to both better approximations and increased run time (and vice versa).

From now on we refer to this variant of our framework as *BP-Beam* with parameter b.

4 Experimental Evaluation

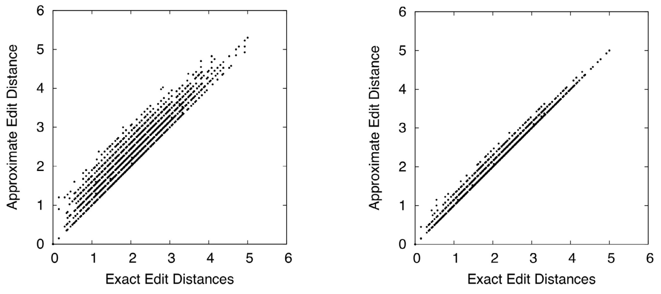
For experimental evaluations three data sets from the IAM graph database repository[4] for graph based pattern recognition and machine learning are used. The first graph data set involves graphs that represent molecular compounds (AIDS), the second graph data set consists of graphs representing fingerprint images (FP), and the third data set consists of graphs representing symbols from architectural and electronic drawings (GREC). For details about the underlying data and/or the graph extraction processes on all data sets we refer to [26].

In Table 1 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 six differently parametrized versions of our novel procedure BP-Beam ($b \in \{5, 10, 15, 20, 50, 100\}$).

First we focus on the degree of overestimation. The original framework (BP) overestimates the graph distance by 12.68% on average on the AIDS data, while on the Fingerprint and GREC data the overestimations of the true distances amount to 6.38% and 2.98%, respectively. These values can be reduced with our extended framework on all data sets. For instance on the AIDS data, the mean relative overestimation can be reduced to 1.93% with $b = 5$. With $b = 5$ also on the other data sets a substantial reduction of $\varnothing o$ can be reported (from 6.38% to 0.61% and from 2.98% to 0.49% on the FP and GREC data set, respectively). Increasing the values of parameter b allows to further decrease the relative overestimation. That is, with $b = 100$ the mean relative overestimation amounts to only 0.87% on the AIDS data set. On the Fingerprint data the overestimation can be heavily reduced from 6.38% to 0.32% with $b = 100$ and on the GREC data set the mean relative overestimation is reduced from 2.98% to 0.27% with this parametrization.

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

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 AIDS data set[5]. We plot for each pair of graphs their 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 our extended framework.



(a) BP (b) BP-Beam(5))

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

Table 1. The mean relative overestimation of the exact graph edit distance ($\varnothing o$) and the mean run time for one matching ($\varnothing t$ in ms) 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)	-	5629.53	-	5000.85	-	3103.76
BP	12.68	0.44	6.38	0.56	2.98	0.43
BP-Beam(5)	1.93	3.98	0.61	2.91	0.49	5.83
BP-Beam(10)	1.79	7.27	0.56	5.17	0.47	10.97
BP-Beam(15)	1.68	10.51	0.51	7.32	0.41	15.90
BP-Beam(20)	1.28	13.48	0.46	9.41	0.33	20.71
BP-Beam(50)	0.95	31.39	0.35	21.58	0.29	46.49
BP-Beam(100)	0.87	60.40	0.32	41.87	0.27	86.00

As expected, the run time of BP-Beam is clearly affected by parameter b. That is, doubling the values for parameter b (from 5 to 10, 10 to 20, or 50 to 100)

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

approximately doubles the run time of our procedure. Comparing the mean run time of BP-Beam(5) with the original framework, we observe that our extension increases run time approximately by factor 9, 6, and 13 on the three data sets. Yet, on all data sets the run time remains remarkable low (a few milliseconds per matching on average only). Furthermore, even with $b = 100$ the average run time lies below 0.1s per matching on every data set. Compared to the huge run time for exact computation (3 or more seconds per matching), the increase of the run time through our extension remains very small.

5 Conclusions

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 combine the bipartite approximation algorithm with a fast tree search algorithm. Formally, given the optimal assignments of nodes and local structures returned by our approximation scheme, variations of this assignment are explored by means of a fast, suboptimal tree search procedure (an exact tree search would be unreasonable, of course). Hence, the present work brings together two different approximation paradigms for graph edit distance, viz. bipartite optimization of local structures and fast beam search. With several experimental results we show that this combination is clearly beneficial as it 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.

In the current version of our extension the node assignment $(u_q \rightarrow v_{\varphi_q})$ to be swapped at search step q are selected in fixed order. In future work we plan, among other activities, to use heuristics for a more elaborated selection order of the node operations to be swapped.

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

References

1. Mahé, P., Ueda, N., Akutsu, T.: Graph kernels for molecular structures – activity relationship analysis with support vector machines. Journal of Chemical Information and Modeling 45(4), 939–951 (2005)
2. Borgwardt, K.: Graph Kernels. PhD thesis, Ludwig-Maximilians-University Munich (2007)
3. Ralaivola, L., Swamidass, S., Saigo, H., Baldi, P.: Graph kernels for chemical informatics. Neural Networks 18(8), 1093–1110 (2005)
4. Schenker, A., Bunke, H., Last, M., Kandel, A.: Graph-Theoretic Techniques for Web Content Mining. World Scientific (2005)
5. Cook, D., Holder, L. (eds.): Mining Graph Data. Wiley-Interscience (2007)

6. Harchaoui, Z., Bach, F.: Image classification with segmentation graph kernels. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–8 (2007)
7. Luo, B., Wilson, R., Hancock, E.: Spectral embedding of graphs. Pattern Recognition 36(10), 2213–2223 (2003)
8. Lladós, J., Sánchez, G.: Graph matching versus graph parsing in graphics recognition. Int. Journal of Pattern Recognition and Artificial Intelligence 18(3), 455–475 (2004)
9. Rocha, J., Pavlidis, T.: A shape analysis model with applications to a character recognition system. IEEE Transactions on Pattern Analysis and Machine Intelligence 16(4), 393–404 (1994)
10. Dickinson, P., Bunke, H., Dadej, A., Kraetzl, M.: Matching graphs with unique node labels. Pattern Analysis and Applications 7(3), 243–254 (2004)
11. 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)
12. 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)
13. Bunke, H., Allermann, G.: Inexact graph matching for structural pattern recognition. Pattern Recognition Letters 1, 245–253 (1983)
14. 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)
15. 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)
16. 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)
17. 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 2006 and SPR 2006. LNCS, vol. 4109, pp. 163–172. Springer, Heidelberg (2006)
18. Riesen, K., Bunke, H.: Approximate graph edit distance computation by means of bipartite graph matching. Image and Vision Computing 27(4), 950–959 (2009)
19. 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)
20. Cortés, X., Serratosa, F., Solé-Ribalta, A.: Active graph matching based on pairwise probabilities between nodes. In: Gimel'farb, G., Hancock, E., Imiya, A., Kuijper, A., Kudo, M., Omachi, S., Windeatt, T., Yamada, K. (eds.) SSPR&SPR 2012. LNCS, vol. 7626, pp. 98–106. Springer, Heidelberg (2012)
21. Koopmans, T., Beckmann, M.: Assignment problems and the location of economic activities. Econometrica 25, 53–76 (1975)
22. Munkres, J.: Algorithms for the assignment and transportation problems. Journal of the Society for Industrial and Applied Mathematics 5, 32–38 (1957)
23. Jonker, R., Volgenant, T.: A shortest augmenting path algorithm for dense and sparse linear assignment problems. Computing 38, 325–340 (1987)
24. Burkard, R., Dell'Amico, M., Martello, S.: Assignment Problems. Society for Industrial and Applied Mathematics, Philadelphia (2009)

25. Fankhauser, S., Riesen, K., Bunke, H.: Speeding up graph edit distance computation through fast bipartite matching. In: Jiang, X., Ferrer, M., Torsello, A. (eds.) GbRPR 2011. LNCS, vol. 6658, pp. 102–111. Springer, Heidelberg (2011)
26. 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.) S+SSPR 2008. LNCS, vol. 5342, pp. 287–297. Springer, Heidelberg (2008)