Learning Heuristics to Reduce the Overestimation of Bipartite Graph Edit Distance Approximation

Miquel Ferrer[1(✉)], Francesc Serratosa[2], and Kaspar Riesen[1]

[1] Institute for Information Systems, University of Applied Sciences and Arts Northwestern Switzerland, Riggenbachstrasse 16, 4600 Olten, Switzerland
{miquel.ferrer,kaspar.riesen}@fhnw.ch
[2] Departament D'Enginyeria Informàtica I Matemàtiques, Universitat Rovira I Virgili, Avda. Països Catalans 26, 43007 Tarragoa, Spain
francesc.serratosa@urv.cat

Abstract. In data mining systems, which operate on complex data with structural relationships, graphs are often used to represent the basic objects under study. Yet, the high representational power of graphs is also accompanied by an increased complexity of the associated algorithms. Exact graph similarity or distance, for instance, can be computed in exponential time only. Recently, an algorithmic framework that allows graph dissimilarity computation in cubic time with respect to the number of nodes has been presented. This fast computation is at the expense, however, of generally overestimating the true distance. The present paper introduces six different post-processing algorithms that can be integrated in this suboptimal graph distance framework. These novel extensions aim at improving the overall distance quality while keeping the low computation time of the approximation. An experimental evaluation clearly shows that the proposed heuristics substantially reduce the overestimation in the existing approximation framework while the computation time remains remarkably low.

1 Introduction

One of the basic objectives in pattern recognition, data mining, and related fields is the development of systems for the analysis or classification of objects [1,2]. These objects (or patterns) can be of any kind [3,4]. Feature vectors are one of the most common and widely used data structure for object representation. That is, for each object a set of relevant properties, or features, is extracted and arranged in a vector. One of the main advantages of this representation is that a large number of algorithms for pattern analysis and classification is available for feature vectors [2]. However, some disadvantages arise from the rather simple structure of feature vectors. First, vectors have to preserve the same length regardless of the size or complexity of the object. Second, vectors are not able to represent binary relations among different parts of the object. As a consequence, for the representation of complex objects where relations

P. Perner (Ed.): MLDM 2015, LNAI 9166, pp. 17–31, 2015.
DOI: 10.1007/978-3-319-21024-7_2

between different subparts play an important role, graphs appear as an appealing alternative to vectorial descriptions.

In particular, graphs can explicitly model the relations between different parts of an object, whereas feature vectors are able to describe the object's properties only. Furthermore, the dimensionality of graphs, i.e., the number of nodes and edges, can be different for every object. Due to these substantial advantages a growing interest in graph-based object representation in machine learning and data mining can be observed in recent years [5,6].

Evaluating the dissimilarity between graphs, commonly referred to as graph matching, is a crucial task in many graph based classification frameworks. Extensive surveys about graph matching in pattern recognition, data mining, and related fields can be found in [7,8]. Graph edit distance [9,10] is one of the most flexible and versatile approaches to error-tolerant graph matching. In particular, graph edit distance is able to cope with directed and undirected, as well as with labeled and unlabeled graphs. In addition, no constraints have to be considered on the alphabets for node and/or edge labels. Moreover, through the concept of cost functions, graph edit distance can be adapted and tailored to diverse applications [11,12].

The major drawback of graph edit distance is, however, its high computational complexity that restricts its applicability to graphs of rather small size. In fact, graph edit distance belongs to the family of *quadratic assignment problems* (QAPs), which in turn belong to the class of $\mathcal{NP}$*-complete* problems. Therefore, exact computation of graph edit distance can be solved in exponential time complexity only. In recent years, a number of methods addressing the high computational complexity of graph edit distance have been proposed (e.g. [13,14]). Beyond these works, an algorithmic framework based on bipartite graph matching has been introduced recently [15]. The main idea behind this approach is to convert the difficult problem of graph edit distance to a *linear sum assignment problem* (LSAP). LSAPs basically constitute the problem of finding an optimal assignment between two independent sets of entities, for which a collection of polynomial algorithms exists [16]. In [15] the LSAP is formulated on the sets of nodes including local edge information. The main advantage of this approach is that it allows the approximate computation of graph edit distance in a substantially faster way than traditional methods. However, it generally overestimates the true edit distance due to some incorrectly assigned nodes. These incorrect assignments are mainly because the framework is able to consider local rather than global edge information only.

In order to overcome this problem and reduce the overestimation of the true graph edit distance, a variation of the original framework [15] has been proposed in [17]. Given the initial assignment found by the bipartite framework, the main idea is to introduce a post-processing step such that the number of incorrect assignments is decreased (which in turn reduces the overestimation). The proposed post-processing varies the original overall node assignment by systematically swapping the target nodes of two individual node assignments. In order to search the space of assignment variations a *beam search* (i.e. a tree search

with pruning) is used. One of the most important observations derived from [17] is that given an initial node assignment, one can substantially reduce the overestimation using this local search method. Yet, the post-processing beam search still produces sub-optimal distances. The reason for this is that beam search possibly prunes the optimal solution in an early stage of the search process.

Now the crucial question arises, how the space of assignment variations could be explored such that promising parts of the search tree are not (or at least not too early) pruned. In [17] the initial assignment is systematically varied without using any kind of heuristic or additional information. In particular, it is not taken into account that certain nodes and/or local assignments have greater impact or are easier to be conducted than others, and should thus be considered first in the beam search process. Considering more important or more evident node assignments in an early stage of the beam search process might reduce the risk of pruning the optimal assignment.

In this paper we propose six different heuristics that modify the mapping order given by the original framework [15]. These heuristics can be used to influence the order in which the assignments are eventually varied during the beam search. With other words, prior to run the beam search strategy proposed in [17], the order of the assignments is varied according to these heuristics.

The remainder of this paper is organized as follows. Next, in Sect. 2, the original bipartite framework for graph edit distance approximation [15] as well as its recent extension [17], named *BP-Beam*, are summarized. In Sect. 3, our novel version of *BP-Beam* is described. An experimental evaluation on diverse data sets is carried out in Sect. 4. Finally, in Sect. 5 we draw conclusions and outline some possible tasks and extensions for future work.

2 Graph Edit Distance Computation

In this Section we start with our basic notation of graphs and then review the concept of graph edit distance. Eventually, the approximate graph edit distance algorithm (which builds the basis of the present work) is described.

2.1 Graph Edit Distance

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, $\mu : V \longrightarrow L_V$ is the node labeling function, and $\nu : E \longrightarrow L_E$ 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 a special case 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 is to transform g_1 into g_2 using edit operations, namely, *insertions*, *deletions*, and *substitutions* of both nodes and edges. The substitution of two nodes u and v is denoted by $(u \rightarrow v)$, the deletion of node

u by $(u \rightarrow \epsilon)$, and the insertion of node v by $(\epsilon \rightarrow v)$[1]. A sequence of edit operations $e_1, \ldots, e_k$ that transform g_1 completely into g_2 is called an edit path between g_1 and g_2.

To find the most suitable edit path out of all possible edit paths between two graphs, a cost measuring the strength of the corresponding operation is commonly introduced (if applicable, one can also merely count the number of edit operations, i.e., the cost for every edit operation amounts to 1). The edit distance between two graphs g_1 and g_2 is then defined by the minimum cost edit path between them. Exact computation of graph edit distance is usually carried out by means of a tree search algorithm (e.g. A* [18]) 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.

2.2 Bipartite Graph Edit Distance Approximation

The computational complexity of exact graph edit distance is exponential in the number of nodes of the involved graphs. That is considering n nodes in g_1 and m nodes in g_2, the set of all possible edit paths contains $O(n^m)$ solutions to be explored. This means that for large graphs the computation of edit distance is intractable. In order to reduce its computational complexity, in [15], the graph edit distance problem is transformed into a linear sum assignment problem (LSAP).

To this end, based on the node sets $V_1 = \{u_1, \ldots, u_n\}$ and $V_2 = \{v_1, \ldots, v_m\}$ of g_1 and g_2, respectively, a cost matrix C is first established as follows:

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

Entry c_{ij} denotes the cost of a node substitution $(u_i \rightarrow v_j)$, $c_{i\epsilon}$ denotes the cost of a node deletion $(u_i \rightarrow \epsilon)$, and $c_{\epsilon j}$ denotes the cost of a node insertion $(\epsilon \rightarrow v_j)$. 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. In every entry $c_{ij} \in C$, not only the cost of the node operation, but also the minimum sum of edge edit operation costs implied by the corresponding node operation is taken into account. That is, the matching cost of the local edge structure is encoded in the individual entries $c_{ij} \in C$.

[1] Similar notation is used for edges.

Note that in [19] another definition of matrix $C = (c_{ij})$ has been proposed. The major idea of [19] is to define a smaller square cost matrix in combination with some (weak) conditions on the cost function. This particular redefinition of C is able to further speed up the assignment process while not affecting the distance accuracy. In the present paper we make use of the original cost matrix definition without any constraints on the cost function.

In the second step of [15], an assignment algorithm is applied to the square cost matrix $C = (c_{ij})$ 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 optimally solved in polynomial time by several algorithms [16].

Any of the LSAP algorithms will return a permutation $(\varphi_1, \ldots, \varphi_{n+m})$ of the integers $(1, 2, \ldots, (n+m))$, which minimizes the overall mapping cost $\sum_{i=1}^{(n+m)} c_{i\varphi_i}$. This permutation corresponds to the mapping

$$\psi = \{u_1 \rightarrow v_{\varphi 1}, u_2 \rightarrow v_{\varphi 2}, \ldots, u_{m+n} \rightarrow v_{\varphi_{m+n}}\}$$

of the nodes of g_1 to the nodes of g_2. Note that ψ does not only include node substitutions $(u_i \rightarrow v_j)$, but also deletions and insertions $(u_i \rightarrow \epsilon)$, $(\epsilon \rightarrow v_j)$ and thus perfectly reflects the definition of graph edit distance (substitutions of the form $(\epsilon \rightarrow \epsilon)$ can be dismissed, of course). Hence, mapping ψ can be interpreted as partial edit path between g_1 and g_2, which considers operations on nodes only.

In the third step of [15], the partial edit path ψ between g_1 and g_2 is completed with respect to the edges. This can be accomplished since edge edit operations are uniquely implied by the adjacent node operations. That is, whether an edge is substituted, deleted, or inserted, depends on the edit operations performed on its adjacent nodes. The total cost of the completed edit path between graphs g_1 and g_2 is finally returned as approximate graph edit distance $d_{\langle\psi\rangle}(g_1, g_2)$. We refer to this graph edit distance approximation algorithm as *BP*(g_1, g_2) from now on[2]. The three major steps of *BP* are summarized in Algorithm 1.

Algorithm 1. *BP*(g_1, g_2)

1: Build cost matrix $C = (c_{ij})$ according to the input graphs g_1 and g_2
2: Compute optimal node assignment $\psi = \{u_1 \rightarrow v_{\varphi 1}, \ldots, u_{m+n} \rightarrow v_{\varphi_{m+n}}\}$ on C using any LSAP solver algorithm
3: **return** Complete edit path according to ψ and $d_{\langle\psi\rangle}(g_1, g_2)$

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 is able to consider the structural information in an isolated way only (single nodes and their adjacent edges). This is due to the fact that during the optimization process of the specific LSAP no information about neighboring node

[2] *BP* stands for *Bipartite*. The assignment problem can also be formulated as finding a matching in a *complete bipartite graph* and is therefore also referred to as *bipartite graph matching problem*.

assignments is available. Hence, in comparison with optimal search methods for graph edit distance, this algorithmic framework might cause additional edge operations in the third step, which would not be necessary in a globally optimal graph matching. Hence, the distances found by this specific framework are – in the best case – equal to, or – in general – 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.

2.3 Beam Search Graph Edit Distance Approximation

Several experimental evaluations indicate that the suboptimality of *BP*, i.e. the overestimation of the true edit distance, is very often due to a few incorrectly assigned nodes in ψ with respect to the optimal edit path [15]. An extension of *BP* presented in [17] ties in at this observation. In particular, the node mapping ψ is used as a starting point for a subsequent search in order to improve the quality of the distance approximation.

Algorithm 2 gives an overview of this process (named *BP-Beam* from now on). First, *BP* is executed using graphs g_1 and g_2 as input. As a result, both the approximate distance $d_{\langle\psi\rangle}(g_1, g_2)$ and the node mapping ψ are available. In a second step, the swapping procedure *BeamSwap* (Algorithm 3) is executed. *BeamSwap* takes the input graphs g_1 and g_2, distance $d_{\langle\psi\rangle}$, mapping ψ, and a meta-parameter b as parameters. The swapping procedure of Algorithm 3 basically varies mapping ψ 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 $(u_j \rightarrow v_{\varphi_j}) \in \psi$, resulting in two new assignments $(u_i \rightarrow v_{\varphi_j})$ and $(u_j \rightarrow v_{\varphi_i})$. For each swap it is verified whether (and to what extent) the derived distance approximation stagnates, increases or decreases.

Algorithm 3 (*BeamSwap*) systematically processes the space of possible swappings by means of a tree search. The tree nodes in the search procedure correspond to triples $(\psi, q, d_{\langle\psi\rangle})$, where ψ is a certain node mapping, 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 mapping ψ found by *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 mappings ψ' where the individual node assignment $(u_q \rightarrow v_{\varphi_q})$ is swapped with any other node assignment of ψ.

Algorithm 2. *BP-Beam*(g_1, g_2, b)

1: $d_{\langle\psi\rangle}(g_1, g_2) = BP(g_1, g_2)$
2: $d_{Beam}(g_1, g_2) = BeamSwap(g_1, g_2, d_{\langle\psi\rangle}, \psi, b)$
3: **return** $d_{Beam}(g_1, g_2)$

As usual in tree search based methods, a set *open* is employed that holds all of the unprocessed tree nodes. Initially, *open* holds the root node $(\psi, 0, d_{\langle\psi\rangle})$ only. The tree nodes in *open* are kept sorted in ascending order according to their depth in the search tree (known as *breadth-first search*). As a second order criterion the approximate edit distance $d_{\langle\psi\rangle}$ is used. As long as *open* is not empty, we retrieve (and remove) the triple $(\psi, q, d_{\langle\psi\rangle})$ at the first position in

Algorithm 3. *BeamSwap*$(g_1, g_2, d_{\langle\psi\rangle}, \psi, b)$

```
1: d_best = d_⟨ψ⟩
2: Initialize open = {(ψ, 0, d_⟨ψ⟩)}
3: while open is not empty do
4:    Remove first tree node in open: (ψ, q, d_⟨ψ⟩)
5:    for j = (q + 1), ..., (m + n) do
6:       ψ' = ψ \ {u_{q+1} → v_{φ_{q+1}}, u_j → v_{φ_j}} ∪ {u_{q+1} → v_{φ_j}, u_j → v_{φ_{q+1}}}
7:       Derive approximate edit distance d_⟨ψ'⟩(g1, g2)
8:       open = open ∪ {(ψ', q + 1, d_⟨ψ'⟩)}
9:       if d_⟨ψ'⟩(g1, g2) < d_best then
10:         d_best = d_⟨ψ'⟩(g1, g2)
11:      end if
12:   end for
13:   while size of open > b do
14:      Remove tree node with highest approximation value d_⟨ψ⟩ from open
15:   end while
16: end while
17: return d_best
```

open, generate the successors of this specific tree node and add them to *open*. To this end 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 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'. 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.

Note that the search space of all possible permutations of ψ contains $(n+m)!$ possibilities, making an exhaustive search (starting with ψ) both unreasonable and intractable. Therefore, only the b assignments with the lowest approximate distance values are kept in *open* at all time (known as *beam search*). 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).

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 in d_{best} (which is finally returned by *BeamSwap* to the main procedure *BP-Beam*).

As stated before, given a mapping ψ from *BP*, the derived edit distance $d_{\langle\psi\rangle}$ overestimates the true edit distance in general. Hence, the objective of any post-processing should be to find a variation ψ' of the original mapping ψ such that $d_{\langle\psi'\rangle} < d_{\langle\psi\rangle}$. Note that the distance d_{best} returned by *BeamSwap* (Algorithm 3) is smaller than, or equal to, the original approximation $d_{\langle\psi\rangle}$ (since $d_{\langle\psi\rangle}$ is initially taken as best distance approximation). Hence, the distance d_{Beam} (finally returned by *BP-Beam* (Algorithm 2))is in any case smaller than, or equal to $d_{\langle\psi\rangle}$.

3 Sorted BP-Beam

Note that the successors of tree node $(\psi, q, d_{\langle\psi\rangle})$ are generated in an arbitrary yet fixed order in *BP-Beam* (or rather in its subprocess *BeamSwap*). In particular, the assignments of the original node mapping 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. Note that beam search prunes quite large parts of the tree during the search process. Hence, the fixed order processing, which does not take any information about the individual node assignments into account, is a clear drawback of the procedure described in [17].

Clearly, it would be highly favorable to process important or evident node assignments as early as possible in the tree search. To this end, we propose six different sorting strategies that modify the order of mapping ψ obtained from BP and feed this reordered mapping into *BeamSwap*. Using these sorting strategies we aim at verifying whether we can learn about the strengths and weaknesses of a given assignment returned by BP before the post processing is carried out. In particular, we want to distinguish assignments from ψ that are most probably incorrect from assignments from ψ that are correct with a high probability (and should thus be considered early in the post processing step).

Algorithm 4. SBP-Beam(g_1,g_2,b)

1: $d_{\langle\psi\rangle}(g_1, g_2) = BP(g_1, g_2)$
2: $\psi' = SortMatching(\psi)$
3: $d_{SortedBeam}(g_1, g_2) = BeamSwap(g_1, g_2, d_{\langle\psi\rangle}, \psi', b)$
4: **return** $d_{SortedBeam}(g_1, g_2)$

The proposed algorithm, referred to as *SBP-Beam* (the initial S stands for *Sorted*), is given in Algorithm 4. Note that *SBP-Beam* is a slightly modified version of *BP-Beam*. That is, the sole difference to *BP-Beam* is that the original mapping ψ (returned by BP) is reordered according to a specific sorting strategy (line 2). Eventually, *BeamSwap* is called using mapping ψ' (rather than ψ) as parameter. Similar to d_{Beam}, $d_{SortedBeam}$ is always lower than, or equal to, $d_{\langle\psi\rangle}(g_1, g_2)$, and can thus be securely returned. Note that the reordering of nodes does not influence the corresponding edit distances and thus $d_{\langle\psi\rangle} = d_{\langle\psi'\rangle}$.

3.1 Sorting Strategies

On line 2 of Algorithm 4 (*SBP-Beam*), the original mapping ψ returned by BP is altered by reordering the individual node assignments according to a certain criterion. Note that the resulting mapping ψ' contains the same node assignments as ψ but in a different order. That is, the order of the assignments is varied but the individual assignments are not modified. Note that this differs from the swapping procedure given in Algorithm 3, where the original node assignments in ψ are modified, changing the target nodes of two individual assignments.

In the following, we propose six different strategies to reorder the original mapping ψ. The overall aim of these sorting strategies is to put more evident assignments (i.e. those to be supposed that are correct) at the beginning of

ψ' such that they are processed first in the tree search. More formally, for each strategy we first give a weight (or rank) to each source node u_i of the assignments in mapping ψ. Then, the order of the assignments $(u_i \rightarrow v_{\varphi i}) \in \psi'$ is set either in ascending or descending order according to the corresponding weight of u_i.

Confident: The source nodes u_i of the assignments $(u_i \rightarrow v_{\varphi i}) \in \psi$ are weighted according to $c_{i\varphi_i}$. That is, for a given assignment $(u_i \rightarrow v_{\varphi i})$, the corresponding value $c_{i\varphi_i}$ in the cost matrix C is assigned to u_i as a weight. The assignments of the new mapping ψ' are then sorted in ascending order according to the weights of u_i. Thus, with this sorting strategy assignments with low costs, i.e. assignments which are somehow evident, appear first in ψ'.

Unique: The source nodes u_i of the assignments $(u_i \rightarrow v_{\varphi i}) \in \psi$ are given a weight according to the following function

$$\max_{\forall j=1,\ldots,m} c_{ij} - c_{i\varphi_i}$$

That is, the weight given to a certain source node u_i corresponds to the maximum difference between the cost $c_{i\varphi_i}$ of the actual assignment $(u_i \rightarrow v_{\varphi i})$ and the cost of a possible alternative assignment for u_i. Note that this difference can be negative, which means that the current assignment $(u_i \rightarrow v_{\varphi i}) \in \psi$ is rather suboptimal (since there is at least one other assignment for u_i with lower cost than $c_{i\varphi_i}$). Assignments in ψ' are sorted in descending order with respect to this weighting criteria. That is, assignments with a higher degree of confidence are processed first.

Divergent: The aim of this sorting strategy is to prioritize nodes u_i that have a high divergence among all possible node assignment costs. That is, for each row i we sum up the absolute values of cost differences between all pairs of assignments

$$\sum_{j=1}^{m-1} \sum_{k=j+1}^{m} |c_{ij} - c_{ik}|$$

Rows with a high divergence correspond to local assignments that are somehow easier to be conducted than rows with low sums. Hence we sort assignments $(u_i \rightarrow v_{\varphi i})$ in descending order with respect to the corresponding divergence in row i.

Leader: With this strategy, nodes u_i are weighted according to the maximum difference between the minimum cost assignment of node u_i and the second minimum cost assignment of u_i. Assume we have $\text{min1}_i = \min_{j=1,\ldots,m} c_{ij}$ and $\text{min2}_i = \min_{j=1,\ldots,m,j\neq k} c_{ij}$ (k refers to the column index of the minimum cost entry min1_i). The weight for node u_i amounts to (the denominator normalizes the weight)

$$\frac{\text{min1}_i - \text{min2}_i}{\text{min2}_i}$$

The higher this difference, the easier is the local assignment to be conducted. Hence, assignments $(u_i \rightarrow v_{\varphi i})$ are sorted in descending order with respect to the weight of u_i.

Interval: First we compute the interval for each row i and each column j of the upper left corner of C, denoted as δ_{r_i} and δ_{c_j} respectively. Given a row i (or column j), the interval is defined as the absolute difference between the maximum and the minimum entry in row i (or column j). We also compute the mean of all row and column intervals, denoted by $\overline{\delta}_r$ and $\overline{\delta}_c$ respectively. The weight assigned to a given assignment $(u_i \rightarrow v_{\varphi i})$ is then
- 1, if $\delta_{r_i} > \overline{\delta}_r$ and $\delta_{c_{\varphi_i}} > \overline{\delta}_c$
- 0, if $\delta_{r_i} < \overline{\delta}_r$ and $\delta_{c_{\varphi_i}} < \overline{\delta}_c$
- 0.5, otherwise

That is, if the intervals of both row and column are greater than the corresponding means, the weight is 1. Likewise, if both intervals are lower than the mean intervals, the weight is 0. For any other case the weight is 0.5.

If the intervals of row i and column φi are larger than the mean intervals, the row and column of the assignment $(u_i \rightarrow v_{\varphi i})$ are in general easier to handle than others. On the other hand, if the row and column intervals of a certain assignment are below the corresponding mean intervals, the individual values in the row and column are close to each other making an assignment rather difficult. Hence, we reorder the assignments $(u_i \rightarrow v_{\varphi i})$ of the original mapping ψ in decreasing order according to these weights.

Deviation: For each row i and each column j of the left upper corner of the cost matrix C we compute the mean $\overline{\theta}_{r_i}$ and $\overline{\theta}_{c_j}$ and the deviation $\overline{\sigma}_{r_i}$ and $\overline{\sigma}_{c_j}$. Then, for each assignment $(u_i \rightarrow v_{\varphi_i}) \in \psi$ we compute its corresponding weight according to the following rule:
- Initially, the weight is 0.
- If $c_{i\varphi_i} < \overline{\theta}_{r_i} - \overline{\sigma}_{r_i}$ we add 0.25 to the weight and compute the total number p of assignments in row i that also fulfill this condition. Note that p is always greater than, or equal to, 1 ($p = 1$ when $c_{i\varphi_i}$ is the sole cost that fulfills the condition in row i). We add $0.5/p$ to the weight.
- Repeat the previous step for column $j = \varphi_i$ using $\overline{\theta}_{c_{\varphi_i}}$ and $\overline{\sigma}_{c_{\varphi_i}}$.

Given an assignment $(u_i \rightarrow v_{\varphi i}) \in \psi$ with cost $c_{i\varphi_i}$ that is lower than the mean minus the deviation, we assume the assignment cost is low enough to be considered as evident (and thus we add 0.25 to the weight). The weight increases if in the corresponding row only few (or no) other evident assignments are available (this is why we add $0.5/p$ to the the weight, being p the number of evident assignments). The same applies for the columns. So at the end, assignments with small weights correspond to rather difficult assignments, while assignments with higher weights refer to assignments which are more evident than others. Hence, we reorder the original mapping ψ in decreasing order according to this weight.

4 Experimental Evaluation

The goal of the experimental evaluation is to verify whether the proposed extension *SBP-Beam* is able to reduce the overestimation of graph edit distance approximation returned by *BP* and in particular *BP-Beam* and how the different sorting strategies affect the computation time. Three data sets from the IAM graph database repository involving molecular compounds (AIDS), fingerprint images (Fingerprint), and symbols from architectural and electronic drawings (GREC) are used to carry out this experimental evaluation. For details about these data sets we refer to [20]. For all data sets, small subsets of 100 graphs are randomly selected on which 10,000 pairwise graph edit distance computations are performed. Three algorithms will be used as reference systems, namely A^* which computes the true edit distance, BP as it is the starting point of our extension, and *BP-Beam* (the system to be further improved).

In a first experiment we aim at researching whether there is a predominant sorting strategy that generally leads to better approximations than the others. To this end we run *SBP-Beam* six times, each using an individual sorting strategy. Parameter b is set to 5 for these and the following experiments.

Table 1 shows the mean relative overestimation ϕo for each dataset and for each sorting strategy. The mean relative overestimation of a certain approximation is computed as the relative difference to the sum of exact distances returned by A^*. The relative overestimation of A^* is therefore zero and the value of ϕo for BP is taken as reference value and corresponds to 100 % (not shown in the table). In addition, given a dataset we rank each sorting strategy in ascending order according to the amount of relative overestimation (in brackets after ϕo). Thus, lower ranks mean lower overestimations and therefore better results. The last row of Table 1 shows the aggregation of ranks for a given sorting strategy, which is a measure of how a particular sorting strategy globally behaves. Table 1 also shows the mean computation time ϕt of every sorting strategy.

Major observations can be made in Table 1. First, there are substantial differences in the overestimation among all sorting strategies for a given dataset.

Table 1. The mean relative overestimation of the exact distance (ϕo) in % together with rank (1-6) of the sorting strategies (in brackets), and the mean computation time ϕt. Last row sums all the ranks for a given sorting strategy.

		Confident	*Unique*	*Divergent*	*Leader*	*Interval*	*Deviation*
AIDS	ϕo [%]	15.81 (3)	15.91 (4)	14.03 (1)	15.38 (2)	18.31 (5)	20.94 (6)
	ϕt [ms]	1.82	1.83	1.80	1.81	1.80	1.82
Fingerprint	ϕo [%]	11.99 (1)	18.06 (6)	16.29 (5)	14.44 (4)	13.64 (3)	13.23 (2)
	ϕt [ms]	1.48	1.50	1.49	1.49	1.50	1.50
GREC	ϕo [%]	20.57 (4)	15.74 (2)	16.67 (3)	21.41 (5)	13.73 (1)	21.65 (6)
	ϕt [ms]	2.63	2.63	2.63	2.61	2.65	2.58
Sum of ranks		8	12	9	11	9	14

For instance, on the AIDS dataset, 5 % of difference between the minimum and the maximum overestimation can be observed (on GREC and Fingerprint the differences amount to about 8 % and 6 %, respectively). This variability indicates that not all of the sorting strategies are able to reduce the overestimation with the same degree. Second, focusing on the sum of ranks we observe that there is not a clear winner among the six strategies. That is, we cannot say that there is a sorting strategy that clearly outperforms the others in general. However, the sum of ranks suggests there are two different clusters, one composed by *Confident*, *Divergent* and *Interval* with a sum of ranks between 8 and 9, and a second composed by the rest of the sorting strategies with (slightly) higher sums. Third, regarding the computation time of the various sorting strategies almost no differences can be observed. That is, all sorting strategies show approximately the same computation time.

In a second experiment, we use the ranks obtained in the previous experiment to measure the impact on the overestimation by running *SBP-Beam* with subsets of sorting strategies. To this end, the following experimental setup is defined. First, for each dataset we run *SBP-Beam* using the sorting strategy with rank 1 (referenced to as *SBP-Beam(1)*). That is, for AIDS *SBP-Beam(1)* employs the *Divergent* strategy, while *Confident* and *Interval* are used for Fingerprint and GREC, respectively. Then *SBP-Beam* is carried out on every dataset using the two best sorting strategies, and return the minimum distance obtained by both algorithms (referred to as *SBP-Beam(2)*). We continue adding sorting strategies in ascending order with respect to their rank until all six strategies are employed at once. Table 2 shows the mean relative overestimation ϕo and the mean computation time ϕt for this evaluation.

Regarding the overestimation ϕo we observe a substantial improvement of the distance quality using *BP-Beam* rather than *BP*. For instance, on the AIDS data the overestimation is reduced by 85 % (similar results are obtained on the other data sets). But more important is the comparison between *BP-Beam* and *SBP-Beam(1)*. Note that *SBP-Beam(1)* is identical to *BP-Beam* but uses reordered

Table 2. The mean relative overestimation of the exact distance (ϕo) in %, and the mean run time for one matcing (ϕt) in ms. The beam size b for *BP-Beam* and *SBP-Beam* is set to 5.

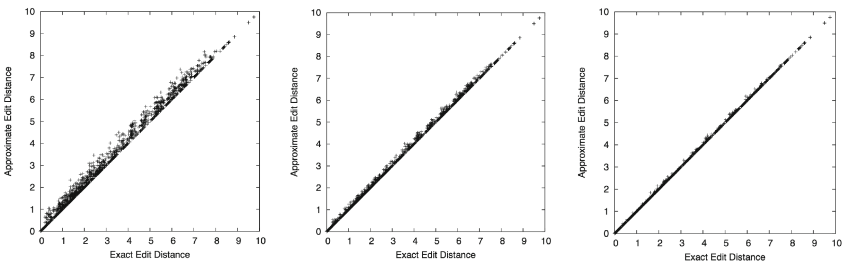
	AIDS		Fingerprint		GREC	
Algorithm	ϕo	ϕt	ϕo	ϕt	ϕo	ϕt
A^*	0.00	25750.22	0.00	2535.42	0.00	7770.81
BP	100.00	0.28	100.00	0.35	100.00	0.27
BP-Beam	15.09	1.82	19.64	1.45	16.98	2.65
SBP-Beam(1)	14.03	1.81	11.99	1.49	13.73	2.64
SBP-Beam(2)	9.83	3.34	8.81	2.61	9.14	5.01
SBP-Beam(3)	8.18	4.85	5.01	3.73	7.33	7.37
SBP-Beam(4)	7.28	6.38	4.42	4.84	6.89	9.74
SBP-Beam(5)	6.06	7.86	3.81	5.95	6.55	12.05
SBP-Beam(6)	5.75	9.41	3.57	7.09	6.25	14.36

mappings ψ' according to the best individual sorting strategy. Thus, by comparing these two algorithms we can assess the true impact of the reordering of mapping ψ. *SBP-Beam(1)* always obtains lower overestimations than *BP-Beam*. The improvement on the overestimation ranges from around 1 % in the case of AIDS dataset to near 8 % in the case of the Fingerprint dataset, which refers to a substantial improvement of the distance quality. This result confirms the hypothesis that the order of the mapping ψ being fed into *BeamSwap* is important and that further improvements of the distance quality can be achieved by means of intelligently ordered assignments in ψ.

Moreover, we observe further substantial reductions of the overestimation as we increase the number of sorting strategies. Note that the overestimation monotonically decreases as the number of sorting strategies is increased, reaching very low overestimations with *SBP-Beam(6)* (between 3.57 % and 6.25 % only). This means that we are able to obtain very accurate mappings (with very few incorrect node assignments) that lead to results very close to the exact distance with our novel procedure.

These substantial reductions of the overestimation from *BP* to *BP-Beam* and *SBP-Beam(6)* can also be seen in Fig. 1 where for each pair of graphs in the Fingerprint dataset, the exact distance (x-axis) is plotted against the distance obtained by an approximation algorithm (y-axis). In fact, for *SBP-Beam(6)* the line-like scatter plot along the diagonal suggests that the approximation is in almost every case equal to the to the optimal distance.

Regarding the computation time ϕt we can report that *BP* provides the lowest computation time on all data sets (approximately 0.3ms per matching on all data sets). *BP-Beam* increases the computation time to approximately 2ms per matching. Note that there is no substantial difference in computation time between *BP-Beam* and *SBP-Beam(1)* (which differ only in the reordering of the individual node assignments). As expected, the run time of *SBP-Beam* linearly grows with the number of sorting strategies. For instance, on the AIDS data set every additional search strategy increases the run time by approximately 1.5ms (on the other data sets a similar run time increase can be observed). However, it



(a) *BP* (b) *BP-Beam* (c) *SBP-Beam(6)*

Fig. 1. Exact (x-axis) vs. approximate (y-axis) edit distance on the Fingerprint dataset computed with (a) *BP*, (b) *BP-Beam*, (c) *SBP-Beam(6)*.

is important to remark that in any case the computation time remains far below the one provided by A^*.

5 Conclusions and Future Work

In recent years a bipartite matching framework for approximating graph edit distance has been presented. In its original version this algorithm suffers from a high overestimation of the computed distance with respect to the true edit distance. Several post-processing approaches, based on node assignment variations, have been presented in order to reduce this overestimation. In this paper, we propose six different heuristics to sort the order of the individual node assignments before the mapping is post-processed. These novel strategies sort the individual node assignments with respect to a weight that measures the confidence of the assignments. The overall aim of the paper is to empirically demonstrate that the order in which the assignments are explored has a great impact on the resulting distance quality. The experimental evaluation on three different databases supports this hypothesis. Despite the results show that there is not a sorting strategy that clearly outperforms the others, we see that all of them are able to reduce the overestimation with respect to the unsorted version. 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.

These results encourage pursuing several research lines in future work. First, in order to verify whether processing non-evident assignments first could also be beneficial, the proposed strategies can be used sorting in reverse order. Second, new and more elaborated sorting strategies based on complex learning algorithms can be developed. Finally, modified versions of *BeamSwap* using different criteria to sort and process the tree nodes could be tested.

Acknowledgements. This work has been supported by the Swiss National Science Foundation (SNSF) project Nr. 200021_153249, the Hasler Foundation Switzerland, and by the Spanish CICYT project DPI2013–42458–P and TIN2013–47245–C2–2–R.

References

1. Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag New York Inc, Secaucus (2006)
2. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification, 2nd edn. Wiley-Interscience, New York (2000)
3. Silva, A., Antunes, C.: Finding multi-dimensional patterns in healthcare. In: Perner, P. (ed.) MLDM 2014. LNCS, vol. 8556, pp. 361–375. Springer, Heidelberg (2014)
4. Dittakan, K., Coenen, F., Christley, R.: Satellite image mining for census collection: a comparative study with respect to the ethiopian hinterland. In: Perner, P. (ed.) MLDM 2013. LNCS, vol. 7988, pp. 260–274. Springer, Heidelberg (2013)
5. Schenker, A., Bunke, H., Last, M., Kandel, A.: Graph-Theoretic Techniques for Web Content Mining. World Scientific, Singapore (2005)

6. Cook, D.J., Holder, L.B.: Mining Graph Data. John Wiley and Sons, New York (2006)
7. Foggia, P., Percannella, G., Vento, M.: Graph matching and learning in pattern recognition in the last 10 years. IJPRAI **28**(1), 1450001 (2014)
8. Conte, D., Foggia, P., Sansone, C., Vento, M.: Thirty years of graph matching in pattern recognition. IJPRAI **18**(3), 265–298 (2004)
9. Sanfeliu, A., Fu, K.-S.: A distance measure between attributed relational graphs for pattern recognition. IEEE Trans. Syst. Man Cybern. SMC-13 (3), 353–362 (1983)
10. Bunke, H., Allermann, G.: Inexact graph matching for structural pattern recognition. Pattern Recogn. Lett. **1**(4), 245–253 (1983)
11. Neuhaus, M., Bunke, H.: A graph matching based approach to fingerprint classification using directional variance. In: Kanade, T., Jain, A., Ratha, N.K. (eds.) AVBPA 2005. LNCS, vol. 3546, pp. 191–200. Springer, Heidelberg (2005)
12. Robles-Kelly, A., Hancock, E.R.: Graph edit distance from spectral seriation. IEEE Trans. Pattern Anal. Mach. Intell. **27**(3), 365–378 (2005)
13. 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)
14. Justice, D., Hero, A.O.: A binary linear programming formulation of the graph edit distance. IEEE Trans. PAMI **28**(8), 1200–1214 (2006)
15. Riesen, K., Bunke, H.: Approximate graph edit distance computation by means of bipartite graph matching. Image Vis. Comput. **27**(7), 950–959 (2009)
16. Burkard, R.E., Dell'Amico, M., Martello, S.: Assignment problems. SIAM **157**(1), 183–190 (2009)
17. Riesen, K., Fischer, A., Bunke, H.: Combining bipartite graph matching and beam search for graph edit distance approximation. In: El Gayar, N., Schwenker, F., Suen, C. (eds.) ANNPR 2014. LNCS, vol. 8774, pp. 117–128. Springer, Heidelberg (2014)
18. Nilsson, N.J., Hart, P.E., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. **SSC–4**(2), 100–107 (1968)
19. Serratosa, F.: Fast computation of bipartite graph matching. Pattern Recogn. Lett. **45**, 244–250 (2014)
20. 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.-Y., Georgiopoulos, M., Anagnostopoulos, G.C., Loog, M. (eds.) SSPR/SPR. LNCS, vol. 5342, pp. 287–297. Springer, Heidelberg (2008)

http://www.springer.com/978-3-319-21023-0

Machine Learning and Data Mining in Pattern Recognition
11th International Conference, MLDM 2015, Hamburg, Germany, July 20-21, 2015, Proceedings
Perner, P. (Ed.)
2015, IX, 454 p. 132 illus., Softcover
ISBN: 978-3-319-21023-0