A Hungarian Algorithm for Error-Correcting Graph Matching

Sébastien Bougleux, Benoit Gaüzère, Luc Brun

A Hungarian Algorithm for Error-Correcting Graph Matching*

Sébastien Bougleux[1], Benoit Gaüzère[2], and Luc Brun[1]

[1] Normandie Univ, CNRS - ENSICAEN - UNICAEN, Caen, France
[2] Normandie Univ, INSA de Rouen, France

Abstract. Bipartite graph matching algorithms become more and more popular to solve error-correcting graph matching problems and to approximate the graph edit distance of two graphs. However, the memory requirements and execution times of this method are respectively proportional to $(n+m)^2$ and $(n+m)^3$ where n and m are the order of the graphs. Subsequent developments reduced these complexities. However, these improvements are valid only under some constraints on the parameters of the graph edit distance. We propose in this paper a new formulation of the bipartite graph matching algorithm designed to solve efficiently the associated graph edit distance problem. The resulting algorithm requires $\mathcal{O}(nm)$ memory space and $\mathcal{O}(\min(n,m)^2 \max(n,m))$ execution times.

Keywords: Graph edit distance, Bipartite matching, Error-correcting matching, Hungarian algorithm

1 Introduction

Computing an efficient similarity or dissimilarity measure between graphs is a major problem in structural pattern recognition. The graph edit distance (GED), developed in the context of error-correcting graph matching, provides such a measure. It may be understood as the minimal amount of distortion required to transform one graph into another, by a sequence of edit operations applied on nodes and edges, restricted here to substitutions, insertions and removals. Such a sequence is called an edit path. Each possible edit operation is penalized by a non-negative cost, and the integration of these costs over an edit path defines the length (or the cost) of this path. An edit path having a minimal length, among all edit paths transforming one graph into another one defines the GED between these two graphs. Since computing the GED is NP-complete, it is restricted to rather small graphs. So several approaches have been proposed to approximate the GED efficiently and to process larger graphs.

In this paper, graphs are assumed to be simple (no loop nor multiple edge), and each element of the two graphs can be edited only once (no composition

* Appears in International Workshop on Graph-Based Representations in Pattern Recognition (GbRPR), LNCS 10310, pp 118–127, Springer, 2017 (`http://doi.org/10.1007/978-3-319-58961-9_11`)

of edit operations). Under these hypotheses, each node of a graph G_1 can be either substituted once to a node of another graph G_2, or removed. Similarly, any node of G_2 may be substituted once, or inserted. Since each node of G_1 and G_2 is transformed only once, such operations on nodes can be encoded by a $(n+m) \times (n+m)$ permutation matrix $\mathbf{X}$ [12], where n and m denote the orders of G_1 and G_2. The costs related to these operations can be encoded by a $(n+m) \times (n+m)$ cost matrix $\mathbf{C}$. Using different heuristics [12,6] to design matrix $\mathbf{C}$, an approximation of the GED can be obtained by solving a linear sum assignment problem (LSAP), *i.e.* by computing an optimal permutation matrix $\mathbf{X}$, for instance with the Hungarian algorithm in $O((n+m)^3)$ time complexity.

However, matrix $\mathbf{C}$ contains an important amount of redundant information mainly used to transform the initial graph edit distance problem into a bipartite matching problem (LSAP). The storage of these additional information induces important memory requirements and increases the size of matrix $\mathbf{C}$, which determines the complexity of the algorithm. Moreover, the resulting matrix $\mathbf{X}$ may contain some useless operations. F. Serratosa [13] proposed to reduce the size of matrix $\mathbf{C}$ in the special case where the graph edit distance fulfills all the axioms of a distance. Such an assumption induces several constraints of the elementary edit costs. Assuming these constraints, Serratosa proposed either to store a $n \times m$ rectangular cost matrix whose optimal solution may be found in $\mathcal{O}(\min(n,m)^2 \max(n,m))$ using the Bourgeois' adaption [4] of the Hungarian algorithm or to store a $\max(n,m) \times \max(n,m)$ cost matrix [14] whose optimal solution may be found by combining the Jonker-Volgenant [8] and Hungarian algorithms. The overall complexity of this last approach is $\mathcal{O}(\max(n,m)^3)$.

Following [12], the approach proposed in this paper approximates the graph edit distance by the Hungarian algorithm. However, our method reformulates the basic problem, hence leading to a $(n+1) \times (m+1)$ cost matrix [2]. Note that a similar formulation has been proposed by [7]. However, this formulation is combined with a Jonker-Volegenant matrix reduction and the classical Hungarian algorithm, hence leading to a $\mathcal{O}((n+m)^3)$ overall complexity. In this paper we investigates the basic principles of the Hungarian algorithm in order to adapt it to this new formulation. Such an extension is detailed in Section 3 after a short introduction to the Hungarian algorithm in Section 2. The resulting algorithm has a worst case complexity of $\mathcal{O}(\min(n,m)^2 \max(n,m))$. Conversely to the methods [13] proposed by Serratosa, our method only assumes that the edit costs are non negative. We also provide in Section 4 accuracy and execution times of a previously published quadratic minimizer [2,3] of the GED combined with our new Hungarian algorithm.

2 Bipartite Matching and Hungarian Algorithm

Preliminary definitions. Given a bipartite graph $(U \cup V, E)$, a *matching* M is a subset of E such that each node in $U \cup V$ is incident to at most one edge of M. It defines a bijective mapping between a subset of U and a subset of V. An edge is *matching edge* if it is in M, else it is an *unmatching edge*. A node incident to

an edge of M is *covered* by M, and otherwise *uncovered*. If all nodes of both sets are covered, the two sets have the same size and the matching is called *perfect*. It defines a bijection between U and V, also called an *assignment*.

Consider a matching M with at least two uncovered nodes, one in each set. A path in the bipartite graph is called *alternating* if it alternates between unmatching and matching edges. An alternating path that begins and ends with uncovered nodes is called *augmenting*. If an augmenting path P exists, a new matching is obtained from M by removing the matching edges of P and by inserting the unmatching ones. The new matching augments the number of matching edges by one, and the number of covered nodes by two.

Linear sum assignment problem and its dual. Consider two sets U and V with the same size n. Each assignment of an element $i \in U$ to an element $j \in V$ is penalized by a non-negative[3] cost $c_{i,j}$. All costs are encoded through a $n \times n$ matrix $\mathbf{C} = (c_{i,j})_{(i,j)\in U\times V}$, *i.e.* a node-node cost matrix associated with the complete bipartite graph $(U \cup V, U \times V)$. When the assignment of a node i to a node j is forbidden, the cost of the edge (i,j) is commonly set to a large value ω, larger than all costs. The *linear sum assignment problem* (LSAP), or minimal-cost perfect matching problem, consists in finding a perfect matching having a minimal cost L, among all perfect matchings:

$$\underset{\mathbf{X}}{\operatorname{argmin}} \left\{ L(\mathbf{X},\mathbf{C}) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j}x_{i,j} \;:\; \mathbf{X} \in \{0,1\}^{n\times n},\, \mathbf{X}\mathbf{1} = \mathbf{1},\, \mathbf{X}^T\mathbf{1} = \mathbf{1} \right\} \quad (1)$$

where $\mathbf{X}$ defines the node-node adjacency matrix of a perfect matching M ($x_{i,j} = 1$ if $(i,j) \in M$ and $x_{i,j} = 0$ else), *i. e.* a *permutation matrix*.

Several algorithms have been developed to find a solution to the LSAP [5]. Among them, the Hungarian algorithm is commonly used to compute approximate GED [12,13,14,6,2]. When it is properly implemented, it finds a solution in $O(n^3)$ in time and in $O(n^2)$ in space [9,5], in worst-case.

The Hungarian algorithm uses a primal-dual approach to find a solution to the LSAP and its dual problem, known as the maximum labeling problem:

$$\underset{(\mathbf{u},\mathbf{v})}{\operatorname{argmax}} \left\{ \mathbf{1}^T\mathbf{u} + \mathbf{1}^T\mathbf{v} \;:\; \mathbf{u},\mathbf{v} \geq \mathbf{0},\; \mathbf{u}\mathbf{1}^T + \mathbf{v}\mathbf{1}^T \leq \mathbf{C} \right\} \quad (2)$$

where vectors $\mathbf{u} = (u_i)_{i=1,\ldots,n}$ and $\mathbf{v} = (v_j)_{j=1,\ldots,n}$ associate a label (or capacity) to each node of $U \cup V$. A pair $(\mathbf{u},\mathbf{v})$ satisfying the constraint $\mathbf{u}\mathbf{1}^T + \mathbf{v}\mathbf{1}^T \leq \mathbf{C}$ is called a *feasible node labeling*. A pair $(\mathbf{X},(\mathbf{u},\mathbf{v}))$ solves the LSAP and its dual iff it verifies the complementary slackness condition:

$$\forall (i,j) \in U \times V,\;\; ((x_{i,j} = 1) \wedge (u_i + v_j = c_{i,j})) \vee ((x_{i,j} = 0) \wedge (u_i + v_j \leq c_{i,j})) \quad (3)$$

More generally, given a feasible node labeling, let $E^0 = \{\, (i,j) \in U \times V \;:\; c_{i,j} = u_i + v_j \}$, the graph induced by this set is called the *equality subgraph*. When E^0 contains an optimal perfect matching, it contains also all other ones.

[3] if some costs are negative, all costs are shifted by $-\min_{i,j}\{c_{i,j}\}$ [5]

Hungarian algorithm Given a cost matrix $\mathbf{C}$, an initial feasible node labeling $(\mathbf{u}, \mathbf{v})$ and an associated matching M (included in the equality subgraph), the Hungarian algorithm proceeds by iteratively updating M and $(\mathbf{u}, \mathbf{v})$ such that two more nodes are covered at each iteration. It is realized by growing a tree of alternating paths in the equality subgraph, called *Hungarian tree*, until an augmenting path is found. At each iteration of the growing process, the tree is augmented by a pair of unmatching and matching edges of the equality subgraph. If this is not possible, because the equality subgraph does not contain enough unmatching edges, the feasible node labeling is revised. We describe the efficient version detailed in [9,5]. The tree is represented by matching edges and by a predecessor array, denoted by pred, which encodes the predecessor (a node of U) of each node of V. Nodes encountered in the tree are encoded by the sets $T_U \subset U$ and $T_V \subset V$. The efficiency of the algorithm relies on maintaining slack variables during the search for an augmenting path: $\forall j \in V \setminus T_V,\ \text{slack}_j = \min\{c_{i,j} - u_i - v_j,\ i \in T_U\}$.

1. If all nodes of U are covered by M, a pair of solutions is found. Else, initialize a Hungarian tree rooted in an uncovered node $i \in U$: $T_U = \{i\}$ and $T_V = \emptyset$. Also, initialize all slack values to $+\infty$.
2. Grow the Hungarian tree in the equality subgraph from a leaf node $i \in T_U$:
   (a) Update neighbors of i to add unmatching edges (i, j) to the tree:

$$\forall j \in V \setminus T_V,\ \begin{cases} \text{if } c_{i,j} - u_i - v_j < \text{slack}_j \text{ then} \\ \quad \text{slack}_j \leftarrow c_{i,j} - u_i - v_j \\ \quad \text{pred}_j \leftarrow i \\ \quad \text{if slack}_j = 0 \text{ then } T_V \leftarrow T_V \cup \{j\} \end{cases} \tag{4}$$

   (b) If there is no leaf node in T_V, the tree cannot grow anymore. The dual variables are updated to add at least one unmatching edge in the equality subgraph and in the tree:

$$\delta = \min\{\text{slack}_j,\ j \in V \setminus T_V\} \tag{5}$$

$$\forall i \in T_U,\ u_i \leftarrow u_i + \delta \tag{6}$$

$$\forall j \in T_V,\ v_j \leftarrow v_j - \delta \tag{7}$$

$$\forall j \in V \setminus T_V,\ \begin{cases} \text{slack}_j \leftarrow \text{slack}_j - \delta \\ \text{if slack}_j = 0 \text{ then } T_V \leftarrow T_V \cup \{j\} \end{cases} \tag{8}$$

   (c) If there is an uncovered leaf node $j \in T_V$, an augmenting path is found, go to step 3. Else, the tree is extended with the unmatching edge (i, j) followed by the matching edge (l, j) by inserting l into T_U. Then go to step 2a with $i \leftarrow l$.
3. Update the matching by backtracking in the tree from the node $j \in V$ found in step 2c to the root, *i. e.* by traversing an augmenting path. Along this path, each matching edge is removed from the matching and each unmatching edge is inserted. Then go to step 1.

An initial feasible labeling is usually given by $u_i \leftarrow \min\{c_{i,j},\, \forall j \in V\}\ \forall i \in U$, and $v_j \leftarrow \min\{c_{i,j} - u_i,\, \forall i \in U\}\ \forall j \in V$. A matching is then deduced from this labeling by traversing the equality subgraph. More sophisticated methods, such as the one proposed by Jonker and Volgenant [8,5] can also be used.

3 Proposed Adaptation of the Hungarian Algorithm

Error-correcting matching and minimal-cost problem. An error-correcting matching from a set U to a set V transforms U into V by editing their elements, together with their attributes. Edit operations are restricted here to substitutions, removals and insertions. Let $U^\epsilon = U \cup \{\epsilon\}$ and $V^\epsilon = V \cup \{\epsilon\}$ be the sets extended by the null element ϵ. Consider the complete bipartite graph $(U^\epsilon \cup V^\epsilon, U^\epsilon \times V^\epsilon)$. An *error-correcting matching* in this graph is a subset of edges connecting each node in U to a unique node of V (substituted by) or to ϵ (removed), and similarly, each node in V to a unique node of U (substituted to) or to ϵ (inserted). Null nodes are unconstrained, they can be connected to zero or more nodes. By considering node-node matrices associated to bipartite graphs, all error-correcting matching are represented by the set of binary matrices:

$$\Pi^\epsilon_{n,m} = \{\ \ \mathbf{X} \in \{0,1\}^{(n+1)\times(m+1)}\ :\ x_{n+1,m+1} = 0, \tag{9}$$

$$\forall j = 1,\ldots,m,\ \ \textstyle\sum_{i=1}^{n+1} x_{i,j} = 1,\ \ \forall i = 1,\ldots,n,\ \ \sum_{j=1}^{m+1} x_{i,j} = 1\ \Big\} \tag{10}$$

Null elements correspond to the last row and the last column. As observed in Eq. 10, they are unconstrained.

Let $\mathbf{C}$ be a $(n+1)\times(m+1)$ cost matrix associated to the complete bipartite graph, *i. e.* a non-negative cost[3] for each substitution, removal and insertion:

$$\mathbf{C} = \left(\begin{array}{ccc|c} c_{1,1} & \cdots & c_{1,m} & c_{1,\epsilon} \\ \vdots & \ddots & \vdots & \vdots \\ c_{n,1} & \cdots & c_{n,m} & c_{n,\epsilon} \\ \hline c_{\epsilon,1} & \cdots & c_{\epsilon,m} & 0 \end{array}\right) \tag{11}$$

The cost of an error-correcting bipartite matching is then written as

$$L(\mathbf{X},\mathbf{C}) = \sum_{i=1}^{n+1}\sum_{j=1}^{m+1} c_{i,j}x_{i,j} = \sum_{i=1}^{n}\sum_{j=1}^{m} c_{i,j}x_{i,j} + \sum_{i=1}^{n} c_{i,\epsilon}x_{i,m+1} + \sum_{j=1}^{m} c_{\epsilon,j}x_{n+1,j}$$

Transforming U into V, with minimum cost, consists in finding an error-correcting bipartite matching having a minimal cost:

$$\operatorname*{argmin}_{\mathbf{X}} \left\{ L(\mathbf{X},\mathbf{C}),\ \mathbf{X} \in \Pi^\epsilon_{n,m} \right\} \tag{12}$$

This is a *linear sum assignment problem with error-correction* (LSAPE). Its dual problem, given by $\max\limits_{(\mathbf{u},\mathbf{v})} \left\{ \mathbf{1}^T\mathbf{u} + \mathbf{1}^T\mathbf{v}\ :\ \mathbf{u}\mathbf{1}^T + \mathbf{v}\mathbf{1}^T \leq \mathbf{C},\ u_{n+1} = v_{m+1} = 0 \right\}$,is

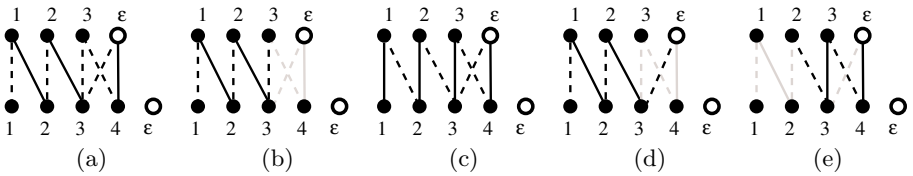


Fig. 1. (a) An incomplete error-correcting matching (solid) and the other edges of the inequality subgraph (dashed). (b) An augmenting path between two uncovered nodes. (c) The new matching obtained by interchanging matching and unmatching edges along this path. (d,e) An augmenting path ending by a null node.

similar to the labeling problem dual to the LSAP, with two elements constrained to be null (the null elements). Based on these formulations of the LSAPE and its dual, it is not difficult to show that the framework used to analyze and solve the LSAP and its dual problem still apply. The Hungarian algorithm can thus be adapted to find a pair of the primal and dual solutions satisfying Eq. 3. The adaptation concerns the processing of null nodes, since they are unconstrained. While the notion of alternating path and Hungarian tree are unchanged, this modifies the notion of augmenting paths as follows.

Augmenting paths. Since null nodes are always unconstrained, any path containing a null node ends by this node. This is equivalent to consider null nodes as never covered. As before (Sec. 2), an augmenting path can end with an uncovered node (Fig. 1(a)), which may thus be a null node (Fig. 1(d)). In this last case, the new matching contains one more covered node and one more matching edge. An augmenting path can also end with a null node incident to a matching edge (Fig. 1(e)). In this case, the new matching augments the number of covered nodes by one while the number of matching edges remains the same. So an augmenting path can be constructed by growing a Hungarian tree until an uncovered node is encountered, including null nodes. Null nodes do not need to be explicitly represented in the tree to find an augmenting path (always leaf nodes). This allows to modify the Hungarian algorithm as follows.

Hungarian algorithm. Given two sets U and V, and a $(n+1)\times(m+1)$ edit cost matrix (Eq. 11) $\mathbf{C}$, consider an initial[4] feasible node labeling $(\mathbf{u},\mathbf{v})$ and an associated incomplete error-correcting matching M (all nodes are not yet covered). We complete the Hungarian algorithm described in Section 2 in order to treat the case of null nodes independently, without altering the global process. To this, the growing of the Hungarian is stopped when a null node is encountered:

[4] The Jonker-Volgenant algorithm proposed in [7] can be used to provide a good initialization. Here we adapt the basic one (Sec. 2): $u_i \leftarrow \min\{c_{i,j},\ \forall j \in V^\epsilon\}\ \forall i \in U$, and $v_j \leftarrow \min\{c_{i,j} - u_i,\ \forall i \in U^\epsilon\}\ \forall j \in V$, with $u_{n+1} = v_{m+1} = 0$. An error-correcting matching is then deduced as in Sec. 2 by traversing the equality subgraph.

– A null node incident to a matching edge (here an insertion) can be detected in Eq. 4 and Eq. 8 of Step 2 by replacing the instruction $T_V \leftarrow T_V \cup \{j\}$ by:

$$\text{if } (\epsilon, j) \in M \text{ go to step 3, else } T_V \leftarrow T_V \cup \{j\}. \tag{13}$$

– A null node incident to an unmatching edge (here a removal) can be detected in Step 2c, when there is an edge $(l, \epsilon) \notin M$ in the equality subgraph, *i. e.* if $c_{l,\epsilon} = u_l$. If this is the case, the algorithm goes to Step 3 instead of going to Step 2a. A null node incident to an unmatching edge can also be detected after the update of the dual variables in Step 2b, as detailed below.

Dual variables are updated (step 2b) such that costs associated to null nodes are also taken into account. Therefore, Eq. 5 is replaced by:

$$\delta = \min\{\, \min\{\, \text{slack}_j,\ j \in V \setminus T_V\},\ \ \min\{\, c_{i,\epsilon} - u_i,\ i \in T_U\}\,\}\,. \tag{14}$$

Then, after Eq. 6 and Eq. 7, and just before Eq. 8, if the minimum δ is obtained from an unmatching edges (i, ϵ), an augmenting path is found and the algorithm goes to Step 3.

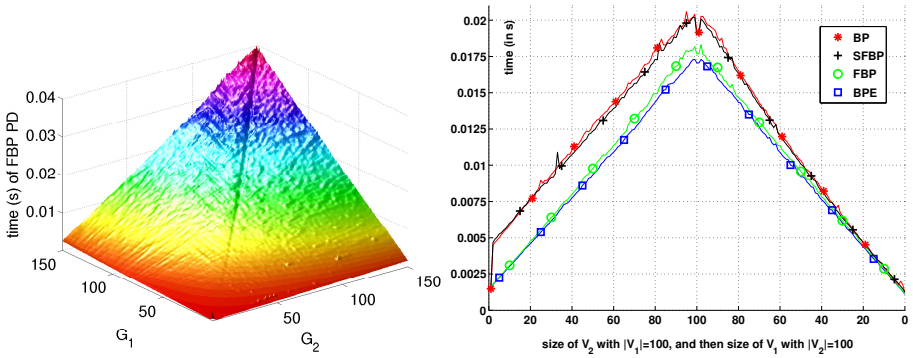
The proposed modifications allow to cover all nodes of U. Some nodes of V may not be covered, which occurs if $n < m$ or if at least one node in U is assigned to a null node. To find a minimal-cost error-correcting matching, the modified Hungarian algorithm is completed by the following step to cover all nodes of V:

4 When all nodes of U are covered, swap the sets U and V, and go to Step 1 with $\mathbf{C}^T$ and $(\mathbf{v}, \mathbf{u})$ as initial feasible node labeling.

The proposed algorithm finds a minimal-cost error-correcting matching in $O(\min\{n, m\}^2 \max\{n, m\})$ in time and $O(nm)$ in space, see [1] for a proof. These complexities are similar to the ones obtained in [4] for solving the LSAP with rectangular cost matrices.

4 Experiments

Bipartite GED. The other formulations of the LSAPE (Sec. 1), transform the problem into a LSAP with a square cost matrix for BP [12] and SFBP [14], or with a rectangular one for FBP [13]. The Hungarian algorithm used in these works [12], differs from the algorithm presented in Section 2 on two aspects: several Hungarian trees are grown at each iteration, and the cost matrix is updated instead of the dual variables. As already discussed [9,5], the version described in this paper has lower execution times. So we have repeated the experiments carried out in [14] on artificially created graphs, with the Hungarian algorithm of Section 2 for solving BP and SFBP. Note that our implementation of the Hungarian algorithm is optimized such that forbidden assignments (with a cost equal to ω) are not treated. As already observed in [14], all the methods lead to a similar approximation of the GED. This is also the case of the approach proposed in this paper (denoted by BPE). A more interesting behavior concerns



(a) (b)

Fig. 2. Computational time of the bipartite GED with respect to the graphs' order.

the computational time. Fig. 2(a) shows the average run time of 10 computations of FBP, with respect to the order of the graphs. Contrary to what was observed in [14], the shape of the run time surface is symmetric. The run time surface of the other algorithms (BP, SFBP and BPE) have a similar pyramidal shape. As illustrated in Fig. 2(b), BP and SFBP have a similar behavior, with an asymmetry, and are less efficient than FBP and BPE. Observe that these two last approaches have also a similar behavior. Contrary to FBP, BPE does not impose any constraint on the costs.

IPFP and GNCCP. As illustrated in [2,3], LSAP methods may also be the core component of different solvers of quadratic programming formulations of the GED. A first method [2] called QAP consists in adapting the IPFP algorithm [10] to the computation of the quadratic formulation of GED. Basically, IPFP iterates over LSAP resolutions to compute a gradient direction leading to an approximate solution of a relaxed version of the quadratic problem. The second proposition [2] uses a convex-concave relaxation of the IPFP approach to tackle drawbacks induced by the influence of initialization and by the final projection step from a stochastic matrix to a mapping one. This approach, denoted GNCCP, iterates over a slightly modified version of IPFP which iterates over LSAP resolutions. Therefore, these two contributions use LSAP as a core component in their respective algorithms. In these experiments, we evaluate the gain obtained by the use of our new algorithm (LSAPE) to resolve LSAP steps in QAP [3] and GNCCP (new in this paper) approaches instead of the classic Hungarian algorithm.

Both algorithms are evaluated on real world chemical datasets[5] composed of different kinds of molecules : Alkane and Acyclic are represented as acyclic graphs of about 8 nodes in average, whereas MAO and PAH are composed of

[5] Datasets are available at `https://iapr-tc15.greyc.fr/links.html`

Table 1. Accuracy and complexity scores. d and t denote respectively the average edit distance and computational time (in seconds).

Algorithm	Alkane		Acyclic		MAO		PAH	
	d	t	d	t	d	t	d	t
A^*	15.47	1.29	17.33	6.02	–	–	–	–
Riesen and Bunke [12]	35.16	0.00135	35.43	0.00109	105	0.00551	138	0.00692
LSAP [6]	34.51	0.00205	32.52	0.00181	56.89	0.02218	123.6	0.03342
LSAPE	34.51	0.00203	32.61	0.00179	56.92	0.02212	123.8	0.03338
QAP [2]	19.28	0.00925	20.51	0.00711	32.97	0.04158	48.5	0.08285
QAPE [3]	19.33	0.00553	20.43	0.00489	32.94	0.03017	48.9	0.04832
Neuhaus [11]	20.5	0.07	25.7	0.0424	59.1	7	52.9	8.2
GNCCP [2]	16.54	0.3474	18.36	0.2481	32.14	4.128	39.2	6.141
GNCCPE	16.83	0.116	19.09	0.07638	32.92	0.4673	38.7	0.8623

larger graphs, with an average size of 20 nodes. As in [2,6], the cost of substituting nodes and edges has been set to 1, and to 3 for insertions and deletions.

Table 1 shows average edit distances and computational times obtained by different approaches on the four chemical datasets. $A^\star$ approach, on the first line, computes the exact graph edit distance and constitutes a reference for approximation methods. However, due to its high complexity, exact graph edit distances have been only computed for Alkane and Acyclic datasets. The first block of three methods, from line 2 to 4, corresponds to methods based on the bipartite approach. The line denoted as Riesen and Bunke corresponds to the original method proposed in [12], while the two others use a different cost matrix [6] using respectively LSAP and LSAPE algorithms. The next block, lines 5 to 7, corresponds to methods based on the quadratic formulation of the graph edit distance. QAP and QAPE [3] use IPFP algorithm with respectively LSAP and LSAPE algorithms. The line denoted as "Neuhaus" corresponds to another quadratic approach [11] which does not handle insertions and removals of nodes during the optimization process. Finally, the last block corresponds to GNCCP approach [2] using LSAP and LSAPE algorithms.

As expected, approximations of graph edit distances are not significantly different using either LSAP or LSAPE approaches. Conversely, as previously observed [2,3], methods based on a quadratic formulation obtain better approximations than the ones based on a linear approximation. From a computational point of view, quadratic approaches require more computational time. However, using LSAPE instead of LSAP algorithm leads to a significant improvement on computational times. This gain almost reaches 10 times with MAO dataset. On MAO and PAH datasets, executions times of LSAP and QAPE methods are comparable. Note that we only observe a very tight improvement using LSAPE instead of LSAP within the original bipartite approach (lines 3 and 4). This lim-

ited gain can be explained by the fact that most of computational time is spent in computing the cost matrix rather than optimizing the mapping problem.

5 Conclusion

We have presented in this paper a new type of linear sum assignment problem designed to solve efficiently the bipartite graph edit distance. The resulting algorithm only supposes that the basic costs are non negative. It requires the storage of an $(n+1)\times(m+1)$ matrix, n and m being the orders of both graphs and has a time complexity of $\mathcal{O}(\min(n,m)^2\max(n,m))$. This algorithm may be applied once to obtain a rough estimate of the edit distance or be integrated into more complex iterative quadratic solvers. The speed-up obtained by our algorithm is significant in this last case and opens the way to the computation of the graph edit distance on larger graphs.

References

1. Bougleux, S., Brun, L.: Linear sum assignment with edition. Tech. rep., Normandie Univ, GREYC UMR 6072, Caen, France (2016)
2. Bougleux, S., Brun, L., Carletti, V., Foggia, P., Gaüzère, B., Vento, M.: Graph edit distance as a quadratic assignment problem. Pattern Recognition Letters 87, 38–46 (2017)
3. Bougleux, S., Gaüzère, B., Brun, L.: Graph edit distance as a quadratic program. In: International Conference on Pattern Recognition. IEEE (2016)
4. Bourgeois, F., Lassalle, J.: An extension of the Munkres algorithm for the assignment problem to rectangular matrices. Commun. ACM 14, 802–804 (1971)
5. Burkard, R., Dell'Amico, M., Martello, S.: Assignment Problems. SIAM (2009)
6. Gaüzère, B., Bougleux, S., Riesen, K., Brun, L.: Approximate graph edit distance guided by bipartite matching of bags of walks. In: Structural, Syntactic, and Statistical Pattern Recognition. LNCS, vol. 8621, pp. 73–82 (2014)
7. Jones, W., Chawdhary, A., King, A.: Revisiting Volgenant-Jonker for approximating graph edit distance. In: Graph-Based Representations in Pattern Recognition. LNCS, vol. 9069, pp. 98–107. Springer Int. Pub. (2015)
8. Jonker, R., Volgenant, A.: Improving the hungarian assignment algorithm. Oper. Res. Lett. (5), 171–175 (1986)
9. Lawler, E.: Combinatorial Optimization: Networks and Matroids. Holt, Rinehart and Winston, New York (1976)
10. Leordeanu, M., Hebert, M., Sukthankar, R.: An integer projected fixed point method for graph matching and map inference. In: Advances in Neural Information Processing Systems, vol. 22, pp. 1114–1122 (2009)
11. Neuhaus, M., Bunke, H.: A quadratic programming approach to the graph edit distance problem. In: Graph-Based Representations in Pattern Recognition. LNCS, vol. 4538, pp. 92–102. Springer Berlin Heidelberg (2007)
12. Riesen, K., Bunke, H.: Approximate graph edit distance computation by means of bipartite graph matching. Image and Vision Computing 27, 950–959 (2009)
13. Serratosa, F.: Fast computation of bipartite graph matching. Pattern Recognition Letters 45, 244–250 (2014)
14. Serratosa, F.: Speeding up fast bipartite graph matching through a new cost matrix. Int. Journal of Pattern Recognition 29(2) (2015)