Fast linear sum assignment with error-correction and no cost constraints

Sébastien Bougleux, Benoit Gaüzère, David Blumenthal, Luc Brun

Fast Linear Sum Assignment with Error-Correction and no Cost Constraints

Sébastien Bougleux[†], Benoit Gaüzère[⋆], David B. Blumenthal[‡], and Luc Brun[*]

[†]Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC UMR 6072, 14000 Caen, France
[⋆]Normandie Univ, UNIROUEN, UNIHAVRE, INSA Rouen, LITIS, 76000 Rouen, France
[‡]Free University of Bozen-Bolzano, Faculty of Computer Science, Dominikanerplatz 3, 39100 Bozen, Italy
[*]Normandie Univ, ENSICAEN, UNICAEN, CNRS, GREYC UMR 6072, 14000 Caen, France

July 10, 2019

Abstract

We propose an algorithm that efficiently solves the linear sum assignment problem with error-correction and no cost constraints. This problem is encountered for instance in the approximation of the graph edit distance. The fastest currently available solvers for the linear sum assignment problem require the pairwise costs to respect the triangle inequality. Our algorithm is as fast as these algorithms, but manages to drop the cost constraint. The main technical ingredient of our algorithm is a cost-dependent factorization of the node substitutions.

1 Introduction

Finding correspondences between structured or unstructured data is a fundamental problem in data processing and analysis, in particular in computer vision, pattern recognition and machine learning. In various application scenarios, data is represented by attributed graphs, and the correspondence problem hence translates to the task to compute those graphs G, from a given collection, that "correspond best" to a given graph H.

One way to formalize correspondence between graphs is to say that two attributed graphs G and H correspond well, if and only if they are close to each other w. r. t. some distance measure. A very sensitive and therefore widely used

distance measure is the graph edit distance (GED) [7, 26, 20]. It is defined as the minimum cost of an edit path between G and H. An edit path between G and H is a sequence of elementary edit operations that transform G into H, where the elementary edit operations are removal or insertion of an edge or an isolated node and substitution of a node's or an edge's attribute. Each elementary edit operation comes with an associated non-negative edit cost.

Since it is NP-hard to exactly compute GED [31], algorithms that compute lower and upper bounds are important. Many existing algorithms [21, 31, 20, 12, 32, 9, 3, 2, 30, 10] proceed as follows: In a first step, they lossily transform the problem of computing the graph edit distance between G and H into an instance $\mathbf{C}$ of the linear sum assignment problem with error-correction (LSAPE), where $\mathbf{C}$ is a real-valued matrix. LSAPE is similar to the linear sum assignment problem for bipartite graphs (LSAP), but also allows node removals and insertions in addition to node substitutions. In a second step, a LSAPE solver is called to compute an optimal solution $\mathbf{X}^\star$ for $\mathbf{C}$. Subsequently, $\mathbf{X}^\star$ is interpreted as an edit path between G and H and hence gives an upper bound for GED. By using the optimality of $\mathbf{X}^\star$, some approaches also allow to derive a lower bound of the GED from the cost of $\mathbf{X}^\star$ under $\mathbf{C}$ [31, 32, 3, 2]. In addition to that, LSAPE arises as a subproblem in several approaches that refine the upper bound by local or hybrid greedy search strategies [22, 25, 11]. This is also the case for approaches based on relaxations of quadratic programming formulations of GED and conditional gradient descent [5, 4, 10].

The runtime performance of the approaches mentioned in the previous paragraph crucially depend on the performance of the LSAPE solver they employ. Existing LSAPE solvers fall in two categories. Solvers of the first kind reduce LSAPE to LSAP [23, 21, 27, 29, 28]: Using different strategies, they first transform an instance $\mathbf{C}$ of LSAPE into an instance $\overline{\mathbf{C}}$ of LSAP and then use standard approaches available for LSAP such as the Hungarian Algorithm for computing an optimal solution $\overline{\mathbf{X}}^\star$ for $\overline{\mathbf{C}}$. Finally, $\overline{\mathbf{X}}^\star$ is transformed into an optimal solution $\mathbf{X}^\star$ for the LSAPE instance $\mathbf{C}$. Algorithms of the second kind directly solve LSAPE by adapting existing approaches for LSAP [13, 14, 4]. Furthermore, LSAPE solvers can be separated in general methods [23, 21, 13, 14, 4] and cost-constrained methods that are only applicable to those instances of LSAPE that respect the triangle inequality [27, 29, 28].

The fastest currently available LSAPE solver is the algorithm `FBP` which reduces an instance $\mathbf{C} \in \mathbb{R}^{(m+1)\times(n+1)}$ of LSAPE to an instance $\overline{\mathbf{C}} \in \mathbb{R}^{n\times m}$ of LSAP. However, `FBP` requires $\mathbf{C}$ to respect the triangle inequality. In this paper, we propose the algorithm `FLWC`, a fast solver for LSAPE without cost constraints. Like `FBP`, `FLWC` reduces $\mathbf{C}$ to a $(n \times m)$-sized instance of LSAP. Unlike `FBP`, `FLWC` does not require $\mathbf{C}$ to respect the triangle inequality. `FLWC` builds upon the insight that node substitutions whose costs do not respect the triangle inequality can be factorized into removals and insertions. A thorough experimental evaluation shows that, for instances that do respect the triangle inequality, `FLWC` is as fast as `FBP`, while, for unconstrained instances, `FLWC` clearly outperforms all competitors.

The remainder of the paper is organized as follows: In Section 2, we formally

introduce LSAPE and LSAP. In Section 3, we discuss related work. In Section 4, we present FLWC. In Section 5, we empirically evaluate the performance of FLWC w. r. t. the performance of existing competitors.

2 Preliminaries

2.1 Linear Sum Assignment Problem

The *linear sum assignment problem* (LSAP) is defined on complete bipartite graphs $K_{n,m} = (\mathcal{U}, \mathcal{V}, \mathcal{U} \times \mathcal{V})$, where $\mathcal{U} = \{u_i\}_{i \in I}$ and $\mathcal{V} = \{v_j\}_{j \in J}$ are sets of nodes, and $I = \{1, \ldots, n\}$ and $J = \{1, \ldots, m\}$ are index sets. In the following, we will assume w. l. o. g. that $n \leq m$. This can easily be enforced by exchanging the roles of $\mathcal{U}$ and $\mathcal{V}$, if necessary. An edge set $\mathcal{M} \subset \mathcal{U} \times \mathcal{V}$ is called a *maximum matching* for $K_{n,m}$ if and only if all nodes in $\mathcal{U}$ are incident with exactly one edge in $\mathcal{M}$ and all nodes in $\mathcal{V}$ are incident with at most one edge in $\mathcal{M}$. A maximum matching $\mathcal{M}$ can be encoded with a binary matrix $\overline{\mathbf{X}} = (\overline{x}_{i,j}) \in \{0,1\}^{n \times m}$ by setting $\overline{x}_{i,j} = 1$ just in case $(u_i, v_j) \in \mathcal{M}$. In the following, we will always represent matchings with their induced binary matrices. The set $\Pi_{n,m}$ of all maximum matchings for $K_{n,m}$ is hence given as follows:

$$\Pi_{n,m} = \begin{Bmatrix} \overline{\mathbf{X}} \in \{0,1\}^{n \times m} : \\ \forall j \in J, \ \sum_{i=1}^{n} \overline{x}_{i,j} \leq 1 \\ \forall i \in I, \ \sum_{j=1}^{m} \overline{x}_{i,j} = 1 \end{Bmatrix} \tag{1}$$

When $n = m$, maximum matchings become *perfect matchings*, and $\Pi_{n,n}$ becomes the set of all $n \times n$ permutation matrices.

Given a cost matrix $\overline{\mathbf{C}} = (\overline{c}_{i,j}) \in \mathbb{R}^{n \times m}$ for the edges of $K_{n,m}$, LSAP asks to compute a maximum matching for $K_{n,m}$ which minimizes the induced edge costs.

Problem 1 (LSAP) *Given a cost matrix $\overline{\mathbf{C}} \in \mathbb{R}^{n \times m}$, LSAP consists to find a maximum matching $\overline{\mathbf{X}}^\star \in \Pi_{n,m}$ with $\overline{\mathbf{X}}^\star \in \operatorname{argmin}_{\overline{\mathbf{X}} \in \Pi_{n,m}} L(\overline{\mathbf{X}}, \overline{\mathbf{C}})$, where $L(\overline{\mathbf{X}}, \overline{\mathbf{C}}) = \sum_{i=1}^{n} \sum_{j=1}^{m} \overline{c}_{i,j} \overline{x}_{i,j}$.*

This is a classical combinatorial optimization problem, also known as the *minimum cost maximum (or perfect) matching problem in bipartite graphs*. It can be solved in polynomial time and space complexities with several algorithms. For instance, if the cost matrix $\overline{\mathbf{C}}$ is balanced, i. e., if $n = m$, it can be solved in $O(n^3)$ time and $O(n^2)$ space with the Kuhn-Munkres Hungarian Algorithm [15, 19]. For the unbalanced case, it can be solved in $O(n^2 m)$ time [6]. See [16, 8] for more details.

2.2 Linear Sum Assignment Problem with Error-Correction

Given a maximum matching $\overline{\mathbf{X}}$ for $K_{n,m} = (\mathcal{U}, \mathcal{V}, \mathcal{U} \times \mathcal{V})$ and a cost matrix $\overline{\mathbf{C}} \in \mathbb{R}^{n \times m}$, a cell $\overline{x}_{i,j}$ of $\overline{\mathbf{X}}$ with $\overline{x}_{i,j} = 1$ can be interpreted as a substitution of

the node $u_i \in \mathcal{U}$ by the node $v_j \in \mathcal{V}$ with associated substitution cost $\overline{c}_{i,j}$. LSAP can hence be viewed as the task to substitute all the nodes of $\mathcal{U}$ by pairwise distinct nodes of $\mathcal{V}$ such that the substitution cost is minimized. Note that, under this interpretation, LSAP does not require all nodes in $\mathcal{V}$ to be taken care of, as, if $n < m$, there are always nodes in $\mathcal{V}$ that do not substitute any node in $\mathcal{U}$.

There are scenarios such as approximation of GED, where one is faced with a slightly different matching problem: First, in addition to node substitutions, one also wants to allow node insertions and node removals. Second, one wants to enforce that all the nodes in $\mathcal{V}$ are taken care of. The *linear sum assignment problem with error-correction* (LSAPE) models these settings. To this purpose, the sets $\mathcal{U}$ and $\mathcal{V}$ are extended to $\mathcal{U}_\epsilon = \mathcal{U} \cup \{\epsilon\}$ and $\mathcal{V}_\epsilon = \mathcal{V} \cup \{\epsilon\}$, where ϵ is a dummy node. An edge set $\mathcal{M} = \mathcal{S} \cup \mathcal{R} \cup \mathcal{I} \subset \mathcal{U}_\epsilon \times \mathcal{V}_\epsilon$ is called *error-correcting matching* for $K_{n,m,\epsilon} = (\mathcal{U}_\epsilon, \mathcal{V}_\epsilon, \mathcal{U}_\epsilon \times \mathcal{V}_\epsilon)$ if and only if it does not contain the edge (ϵ, ϵ) and all nodes in $\mathcal{U}$ and $\mathcal{V}$ are incident with exactly one edge in $\mathcal{M}$. The set $\mathcal{S} \subset \mathcal{U} \times \mathcal{V}$ contains all node substitutions, $\mathcal{R} \subset \mathcal{U} \times \{\epsilon\}$ contains all removals, and $\mathcal{I} \subset \{\epsilon\} \times \mathcal{V}$ contains all insertions. The set $\Pi_{n,m,\epsilon}$ of all error-correcting matchings for $K_{n,m,\epsilon}$ is hence given as

$$\Pi_{n,m,\epsilon} = \begin{Bmatrix} \mathbf{X} = \begin{pmatrix} \mathbf{X}_{\text{sub}} & \mathbf{x}_{\text{rem}} \\ \mathbf{x}_{\text{ins}} & 0 \end{pmatrix} \in \{0,1\}^{(n+1)\times(m+1)} : \\ \forall j \in J,\ x_{\epsilon j} + \sum_{i=1}^{n} x_{i,j} = 1 \\ \forall i \in I,\ x_{i\epsilon} + \sum_{j=1}^{m} x_{i,j} = 1 \end{Bmatrix}, \tag{2}$$

where $\mathbf{X}_{\text{sub}} = (x_{i,j}) \in \mathbb{R}^{n\times m}$ encodes node substitutions, $\mathbf{x}_{\text{rem}} = (x_{i,\epsilon}) \in \mathbb{R}^{n\times 1}$ encodes node removals, and $\mathbf{x}_{\text{ins}} = (x_{\epsilon,j}) \in \mathbb{R}^{1\times m}$ encodes node insertions. Assume now that $\mathbf{C} \in \mathbb{R}^{(n+1)\times(m+1)}$ is a cost matrix of the form

$$\mathbf{C} = \begin{pmatrix} \mathbf{C}_{\text{sub}} & \mathbf{c}_{\text{rem}} \\ \mathbf{c}_{\text{ins}} & 0 \end{pmatrix}, \tag{3}$$

where $\mathbf{C}_{\text{sub}} = (c_{i,j}) \in \mathbb{R}^{n\times m}$ contains the substitution costs, $\mathbf{c}_{\text{rem}} = (c_{i,\epsilon}) \in \mathbb{R}^{n\times 1}$ contains the removal costs, and $\mathbf{c}_{\text{ins}} = (c_{\epsilon,j}) \in \mathbb{R}^{1\times m}$ contains the insertion costs. Then LSAPE asks to compute an error-correcting matching for $K_{n,m,\epsilon}$ such that the sum of the induced substitution, removal, and insertion costs is minimized.

Problem 2 (LSAPE) *Given a cost matrix* $\mathbf{C} \in \mathbb{R}^{(n+1)\times(m+1)}$ *of form* (3)*, LSAPE consists to find an error-correcting matching* $\mathbf{X}^\star \in \Pi_{n,m,\epsilon}$ *with* $\mathbf{X}^\star \in \operatorname{argmin}_{\mathbf{X}\in\Pi_{n,m,\epsilon}} L(\mathbf{X}, \mathbf{C})$*, where* $L(\mathbf{X}, \mathbf{C}) = \sum_{i=1}^{n}\sum_{j=1}^{m} c_{i,j}x_{i,j} + \sum_{i=1}^{n} c_{i,\epsilon}x_{i,\epsilon} + \sum_{j=1}^{m} c_{\epsilon,j}x_{\epsilon,j}$*.*

Table 1 summarizes the notations introduced in this section.

Table 1: Frequently used notations

syntax	semantic
$\mathbf{C} \in \mathbb{R}^{(n+1)\times(m+1)}$	instance of LSAPE
$\overline{\mathbf{C}} \in \mathbb{R}^{n\times m}$	instance of LSAP
$\mathbf{X} \in \Pi_{n,m,\epsilon}$	error-correcting matching
$\overline{\mathbf{X}} \in \Pi_{n,m}$	maximum matching
$\mathbf{X}^\star \in \Pi_{n,m,\epsilon}$	optimal error-correcting matching
$\overline{\mathbf{X}}^\star \in \Pi_{n,m}$	optimal maximum matching
$L(\mathbf{X}, \mathbf{C})$	cost of error-correcting matching $\mathbf{X}$ w. r. t. $\mathbf{C}$
$L(\overline{\mathbf{X}}, \overline{\mathbf{C}})$	cost of maximum matching $\overline{\mathbf{X}}$ w. r. t. $\overline{\mathbf{C}}$

3 State of the Art for LSAPE

3.1 Unconstrained Reduction to LSAP

The standard algorithm *extended bipartite matching (EBP)* [20, 21, 23] solves LSAPE by transforming it to LSAP. Given an instance $\mathbf{C}$ of LSAPE, `EBP` constructs an instance $\overline{\mathbf{C}} \in \mathbb{R}^{(n+m)\times(m+n)}$ of LSAP as

$$\overline{\mathbf{C}} = \begin{pmatrix} \mathbf{C}_{\text{sub}} & \omega(\mathbf{1}_{n\times n} - \mathbf{I}_n) + \operatorname{diag}(\mathbf{c}_{\text{rem}}) \\ \omega(\mathbf{1}_{m\times m} - \mathbf{I}_m) + \operatorname{diag}(\mathbf{c}_{\text{ins}}) & \mathbf{0}_{m\times n} \end{pmatrix},$$

where ω denotes a very large value and the operator diag maps a vector $(v_i)_{i=1}^k$ to the diagonal matrix $(d_{i,j})_{i,j=1}^k$ with $d_{i,i} = v_i$ and $d_{i,j} = 0$ for all $i \neq j$. `EBP` then calls a LSAP solver to compute an optimal maximum matching $\overline{\mathbf{X}}^\star \in \Pi_{n+m,m+n}$ for LSAP. By construction, $\overline{\mathbf{X}}^\star$ is of the form

$$\overline{\mathbf{X}}^\star = \begin{pmatrix} \overline{\mathbf{X}}^\star_{\text{sub}} & \overline{\mathbf{X}}^\star_{\text{rem}} \\ \overline{\mathbf{X}}^\star_{\text{ins}} & \overline{\mathbf{X}}^\star_{\text{comp}} \end{pmatrix}, \tag{4}$$

where $\overline{\mathbf{X}}^\star_{\text{sub}} \in \{0,1\}^{n\times m}$, $\overline{\mathbf{X}}^\star_{\text{rem}} \in \{0,1\}^{n\times n}$, and $\overline{\mathbf{X}}^\star_{\text{ins}} \in \{0,1\}^{m\times m}$. The southeast quadrant $\overline{\mathbf{X}}^\star_{\text{comp}} \in \{0,1\}^{m\times n}$ contains assignments from dummy nodes to dummy nodes which are not needed for encoding node removals or insertions but are anyway computed by LSAP algorithms. $\overline{\mathbf{X}}^\star$ is then transformed into an optimal error-correcting matching $\mathbf{X}^\star \in \Pi_{n,m,\epsilon}$ for LSAPE with $L(\mathbf{X}^\star, \mathbf{C}) = L(\overline{\mathbf{X}}^\star, \overline{\mathbf{C}})$ by setting $\mathbf{X}^\star_{\text{sub}} = \overline{\mathbf{X}}^\star_{\text{sub}}$, $\mathbf{x}^\star_{\text{rem}} = \overline{\mathbf{X}}^\star_{\text{rem}}\mathbf{1}_n$, and $\mathbf{x}^\star_{\text{ins}} = \mathbf{1}_m^T\overline{\mathbf{X}}^\star_{\text{ins}}$. The time complexity of `EBP` is dominated by the complexity of solving the LSAP instance $\overline{\mathbf{C}}$. Therefore, `EBP` runs in $O((n+m)^3)$ time.

3.2 Cost-Constrained Reductions to LSAP

The algorithms *fast bipartite matching (FBP)* [27] and *square fast bipartite matching (SFBP)* [28, 29] build upon more compact reductions of LSAPE to LSAP than `EBP`. However, both `FBP` and `SFBP` are applicable only to those instances $\mathbf{C}$ of

LSAPE which respect the following triangle inequalities:

$$\forall (i,j) \in I \times J, \;\; c_{i,j} \leq c_{i,\epsilon} + c_{\epsilon,j} \tag{5}$$

In other terms, FBP and SFBP can be used for instances of LSAPE where substituting a node $u_i \in \mathcal{U}$ by a node $v_j \in \mathcal{V}$ is never more expensive than removing u_i and inserting v_j. The following proposition is the key-ingredient of both FBP and SFBP. Recall that we have assumed w. l. o. g. that $n \leq m$.

Proposition 1 (Cf. [27, 28, 29]) *Let $\mathbf{X}^\star \in \Pi_{n,m,\epsilon}$ be an optimal error-correcting matching for an instance $\mathbf{C} \in \mathbb{R}^{(n+1)\times(m+1)}$ of LSAPE which satisfies* (5)*. Then $\mathbf{X}$ contains no node removal, i. e., satisfies $\mathbf{x}_{\text{rem}} = \mathbf{0}_n$. Note that this implies that $\mathbf{X}^\star$ contains exactly $m - n$ node insertions and hence that $\mathbf{x}_{\text{ins}} = \mathbf{0}_m$, if $n = m$.*

Given an instance $\mathbf{C}$ of LSAPE that satisfies (5), FBP constructs an instance $\overline{\mathbf{C}} \in \mathbb{R}^{n\times m}$ of LSAP by setting $\overline{\mathbf{C}} = \mathbf{C}_{\text{sub}} - \mathbf{c}_{\text{rem}}\mathbf{1}_n^T - \mathbf{1}_m\mathbf{c}_{\text{ins}}$. Subsequently, FBP computes an optimal maximum matching $\overline{\mathbf{X}}^\star \in \Pi_{n,m}$ for $\overline{\mathbf{C}}$. Because of Proposition 1, it then holds that the matrix $\mathbf{X}^\star \in \Pi_{n,m,\epsilon}$ defined by $\mathbf{X}^\star_{\text{sub}} = \overline{\mathbf{X}}^\star$, $\mathbf{x}^\star_{\text{rem}} = \mathbf{0}_n$, and $\mathbf{x}^\star_{\text{ins}} = \mathbf{1}_m^T - \mathbf{1}_n^T\overline{\mathbf{X}}^\star$ is an optimal error-correcting matching for $\mathbf{C}$.

The variant FBP_0 of FBP transforms $\mathbf{C}$ into a balanced instance $\overline{\mathbf{C}}_0 \in \mathbb{R}^{m\times m}$ of LSAP, by adding the matrix $\mathbf{0}_{m-n,n}$ below $\overline{\mathbf{C}}$ defined for FBP. After solving this instance, FBP_0 transforms the resulting optimal maximum matching

$$\begin{pmatrix} \overline{\mathbf{X}}^\star \\ \star \end{pmatrix}$$

for $\overline{\mathbf{C}}_0$ into an optimal error-correcting matching $\mathbf{X}^\star \in \Pi_{n,m,\epsilon}$ for $\mathbf{C}$, by applying the same transformation rules as FBP on $\overline{\mathbf{X}}^\star$.

Like FBP_0, SFBP transforms an instance $\mathbf{C}$ into a balanced instance $\overline{\mathbf{C}} \in \mathbb{R}^{m\times m}$ of LSAP. However, $\overline{\mathbf{C}}$ is now defined as follows:

$$\overline{\mathbf{C}} = \begin{pmatrix} \mathbf{C}_{\text{sub}} \\ \mathbf{1}_{m-n}\mathbf{c}_{\text{ins}} \end{pmatrix}$$

In the next step, SFBP computes an optimal maximum matching

$$\overline{\mathbf{X}}^\star = \begin{pmatrix} \overline{\mathbf{X}}^\star_{\text{sub}} \\ \overline{\mathbf{X}}^\star_{\text{ins}} \end{pmatrix}$$

for $\overline{\mathbf{C}}$. SFBP then constructs the matrix $\mathbf{X}^\star \in \Pi_{n,m,\epsilon}$ by setting $\mathbf{X}^\star_{\text{sub}} = \overline{\mathbf{X}}^\star_{\text{sub}}$, $\mathbf{x}^\star_{\text{rem}} = \mathbf{0}_n$, and $\mathbf{x}^\star_{\text{ins}} = \mathbf{1}_m^T\overline{\mathbf{X}}^\star_{\text{ins}}$. Again, Proposition 1 ensures that $\mathbf{X}^\star$ is indeed an optimal error-correcting matching for LSAPE.

The time complexities of FBP, FBP_0, and SFBP are dominated by the complexities of solving the LSAP instances $\overline{\mathbf{C}}$. Therefore, FBP runs in $O(n^2m)$ time, while FBP_0 and SFBP run in $O(m^3)$ time. These are significant improvements over EBP. However, recall that FBP, FBP_0, and SFBP can be used only if the cost matrix $\mathbf{C}$ respects the triangle inequalities (5).

Adaptations of Classical Algorithms LSAPE can also be solved directly by adapting algorithms originally designed for LSAP. An adaptation of the Jonker-Volgenant Algorithm is proposed in [14]. An adaption of the Hungarian Algorithm, denoted $\texttt{HNG}_\epsilon$ in this paper, has been suggested in [4]. Both modifications lead to an overall time complexity of $O(n^2m)$.

4 A Compact Reduction from LSAPE to LSAPE without Cost Constraints

The main contribution of this paper is to show that LSAPE without cost constraints can be reduced to an instance of LSAP of size $n \times m$. The reduction translates into the algorithm `FLWC` (fast solver for LSAPE without cost constraints), which, like `FBP`, runs in $O(n^2m)$ time, but, unlike `FBP`, $\texttt{FBP}_0$, and `SFBP`, does not assume the costs to respect the triangle inequalities (5). The following Theorem 1 states the reduction principle. It relies on a cost-dependent factorization of substitutions, removals and insertions. In Section 4.1, we discuss special cases and present `FLWC`. In Section 4.2, we present the proof of Theorem 1.

Theorem 1 (Reduction Principle) *Let $\mathbf{C} \in \mathbb{R}^{(n+1)\times(m+1)}$ be an instance of LSAPE and let $\overline{\mathbf{X}}^\star \in \Pi_{n,m}$ be an optimal maximum matching for the instance $\overline{\mathbf{C}}$ of LSAP defined by*

$$\forall (i,j) \in I \times J,\ \ \overline{c}_{i,j} = \delta^{\mathbf{C}}_{i,j,\epsilon} c_{i,j} + (1 - \delta^{\mathbf{C}}_{i,j,\epsilon})(c_{i,\epsilon} + c_{\epsilon,j}) - \delta_{n<m}\, c_{\epsilon,j}, \qquad (6)$$

where

$$\delta_{n<m} = \begin{cases} 1 & \text{if } n < m \\ 0 & \text{else} \end{cases}, \quad \delta^{\mathbf{C}}_{i,j,\epsilon} = \begin{cases} 1 & \text{if } c_{i,j} \leq c_{i,\epsilon} + c_{\epsilon,j} \\ 0 & \text{else} \end{cases}$$

Furthermore, let the function $f_{\mathbf{C}} : \Pi_{n,m} \to \Pi_{n,m,\epsilon}$ be defined as follows:

$$\forall (i,j) \in I \times J, \quad x_{i,j} = \left(f_{\mathbf{C}}(\overline{\mathbf{X}})\right)_{i,j} = \delta^{\mathbf{C}}_{i,j,\epsilon} \overline{x}_{i,j} \qquad (7)$$

$$\forall i \in I, \quad x_{i,\epsilon} = \left(f_{\mathbf{C}}(\overline{\mathbf{X}})\right)_{i,\epsilon} = 1 - \sum_{j=1}^{m} \delta^{\mathbf{C}}_{i,j,\epsilon} \overline{x}_{i,j} \qquad (8)$$

$$\forall j \in J, \quad x_{\epsilon,j} = \left(f_{\mathbf{C}}(\overline{\mathbf{X}})\right)_{\epsilon,j} = 1 - \sum_{i=1}^{n} \delta^{\mathbf{C}}_{i,j,\epsilon} \overline{x}_{i,j} \qquad (9)$$

Then $\mathbf{X}^\star = f_{\mathbf{C}}(\overline{\mathbf{X}}^\star)$ is an optimal error-correcting matching for $\mathbf{C}$ with cost $L(\mathbf{X}^\star, \mathbf{C}) = L(\overline{\mathbf{X}}^\star, \overline{C}) + \delta_{n<m} \sum_{j=1}^{m} c_{\epsilon,j}$.

Example 1 *Assume that $n = 2$, $m = 3$, and consider the instance $\mathbf{C}$ of LSAPE and the induced instance $\overline{\mathbf{C}}$ of LSAP:*

$$\mathbf{C} = \begin{array}{c} i \diagdown j \\ 1 \\ 2 \\ \epsilon \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & \epsilon \end{array} \\ \begin{pmatrix} 3 & 5 & 1 & 4 \\ 8 & 9 & 4 & 4 \\ 2 & 4 & 0 & 0 \end{pmatrix} \end{array} \xrightarrow{\textit{apply (6)}} \overline{\mathbf{C}} = \begin{array}{c} i \diagdown j \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 \\ 4 & 4 & 4 \end{pmatrix} \end{array}$$

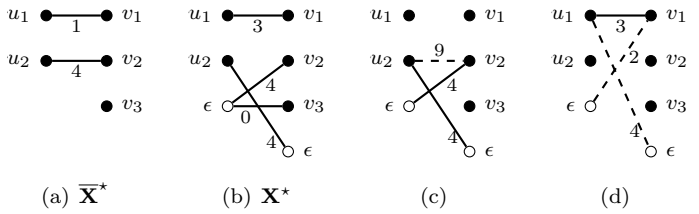


Figure 1: Illustration of the reduction principle stated in Theorem 1

For instance, we have $\bar{c}_{1,1} = \delta^{\mathbf{C}}_{1,1\epsilon}3 + (1 - \delta^{\mathbf{C}}_{1,1\epsilon})(2 + 4) - \delta_{2<3}2 = 3 - 2 = 1$ *and* $\bar{c}_{2,2} = \delta^{\mathbf{C}}_{2,2\epsilon}9 + (1 - \delta^{\mathbf{C}}_{2,2\epsilon})(4 + 4) - \delta_{2<3}4 = 8 - 4 = 4$. *Fig. 1(a) shows an optimal maximum matching* $\overline{\mathbf{X}}^\star \in \Pi_{n,m}$ *for* $\overline{C}$, *and Fig. 1(b) shows an optimal error-correcting matching* $\mathbf{X}^\star = f_{\mathbf{C}}(\overline{X}^\star) \in \Pi_{n,m,\epsilon}$ *for* $\mathbf{C}$. *Note that* $f_{\mathbf{C}}$ *factorizes the substitution* (u_2, v_2) *into the removal* (u_2, ϵ) *and the insertion* (ϵ, v_2), *since* $c_{2,2} > c_{2,\epsilon} + c_{\epsilon,2}$ *(Fig. 1(c)). On the other hand, the substitution* (u_1, v_1) *is not factorized, since* $c_{1,1} \leq c_{1,\epsilon} + c_{\epsilon,1}$ *(Fig. 1(d)). Furthermore, we have* $L(\mathbf{X}^\star, \mathbf{C}) = 11$, $L(\overline{\mathbf{X}}^\star, \overline{C}) = 5$, *and* $\delta_{n<m} \sum_{j=1}^{m} c_{\epsilon,j} = 6$. *Therefore, it holds that* $L(\mathbf{X}^\star, \mathbf{C}) = L(\overline{\mathbf{X}}^\star, \overline{C}) + \delta_{n<m} \sum_{j=1}^{m} c_{\epsilon,j}$, *as stated by our reduction principle.*

4.1 Discussion of Special Cases and Presentation of `FLWC`

Theorem 1 states that a general instance $\mathbf{C} \in \mathbb{R}^{(n+1)\times(m+1)}$ of LSAPE, which is not required to respect the triangle inequalities (5), can be reduced to a $(n \times m)$-sized instance of LSAP. However, there is still room for improvement if $\mathbf{C}$ does respect the triangle inequalities.

Proposition 2 *Let* $\mathbf{C} \in \mathbb{R}^{(n+1)\times(m+1)}$ *be an instance of LSAPE that respects the triangle inequalities* (5). *Then the reduction principle specified in Theorem 1 can be carried out without knowledge of the removal costs* $\mathbf{c}_{\text{rem}}$. *If* $n = m$ *holds, too, then knowledge of the insertion costs* $\mathbf{c}_{\text{ins}}$ *is not necessary, either.*

Proof 1 *The proposition immediately follows from the facts that* $\delta^{\mathbf{C}}_{i,j,\epsilon} = 1$ *holds for all* $(i, j) \in I \times J$ *if* $\mathbf{C}$ *respects (5), and that* $\delta_{n<m} = 0$ *if* $n = m$.□.

Proposition 2 is useful, because in many application scenarios for LSAPE, $\mathbf{C}$ is not given but has to be computed. Furthermore, $\mathbf{C}$ is often known a priori to respect the triangle inequalities, for instance, because its entries contain the distances between elements in a metric space. In such settings, the overall performance of an algorithm that calls a LSAPE solver as a subroutine can improve significantly if only parts of the cost matrix $\mathbf{C}$ have to be computed. Table 2 summarizes which parts of $\mathbf{C}$ have to be known for our reduction from LSAPE to LSAP. The case $n > m$ can straightforwardly be obtained from the

case $n < m$ by transposing $\mathbf{C}$. Recall that both FBP and $\texttt{FBP}_0$ always require the entire cost matrix $\mathbf{C}$ to be known. SFBP requires the same parts of $\mathbf{C}$ as our approach, but reduces LSAPE to a larger instance of LSAP ($\max\{n,m\} \times \max\{n,m\}$ vs. $n \times m$).

Table 2: Required parts of $\mathbf{C}$ for our reduction from LSAPE to LSAP

	triangle ineqs. hold	**triangle ineqs. do not hold**
$n = m$	$\mathbf{C}_{\text{sub}}$	$\mathbf{C}_{\text{sub}}$, $\mathbf{c}_{\text{ins}}$, $\mathbf{c}_{\text{rem}}$
$n < m$	$\mathbf{C}_{\text{sub}}$, $\mathbf{c}_{\text{ins}}$	$\mathbf{C}_{\text{sub}}$, $\mathbf{c}_{\text{ins}}$, $\mathbf{c}_{\text{rem}}$
$n > m$	$\mathbf{C}_{\text{sub}}$, $\mathbf{c}_{\text{rem}}$	$\mathbf{C}_{\text{sub}}$, $\mathbf{c}_{\text{ins}}$, $\mathbf{c}_{\text{rem}}$

Algorithm 1 shows the algorithm FLWC, which turns our reduction from LSAPE to LSAP into a method for computing an optimal error-correcting matching. FLWC only uses those parts of $\mathbf{C}$ which are really required by the reduction (cf. Table 2).

Algorithm 1: FLWC

input: an instance $\mathbf{C} \in \mathbb{R}^{(n+1)\times(m+1)}$ of LSAPE
output: an optimal error-correcting matching $\mathbf{X}^\star \in \Pi_{n,m,\epsilon}$ for $\mathbf{C}$

1 **if** $n > m$ **then**
2 $\quad \mathbf{C} \leftarrow \mathbf{C}^T$; $(n,m) \leftarrow (m,n)$;
3 initialize $\overline{C} \in \mathbb{R}^{n\times m}$;
4 **for** $i \in \{1,\dots,n\}$ **do**
5 $\quad$ **for** $j \in \{1,\dots,m\}$ **do**
6 $\quad\quad$ **if** $\mathbf{C}$ *respects triangle inequalities* **then**
7 $\quad\quad\quad$ **if** $n = m$ **then**
8 $\quad\quad\quad\quad \overline{c}_{i,j} \leftarrow c_{i,j}$;
9 $\quad\quad\quad$ **else**
10 $\quad\quad\quad\quad \overline{c}_{i,j} \leftarrow c_{i,j} - c_{\epsilon,j}$;
11 $\quad\quad$ **else**
12 $\quad\quad\quad \overline{c}_{i,j} \leftarrow \delta^{\mathbf{C}}_{i,j,\epsilon} c_{i,j} + (1 - \delta^{\mathbf{C}}_{i,j,\epsilon})(c_{i,\epsilon} + c_{\epsilon,j}) - \delta_{n<m}\, c_{\epsilon,j}$;
13 call LSAP solver to compute opt. max. matching $\overline{\mathbf{X}}^\star \in \Pi_{n,m}$ for $\overline{\mathbf{C}}$;
14 $\mathbf{X}^\star \leftarrow f_{\mathbf{C}}(\overline{\mathbf{X}}^\star)$;
15 **if** $\mathbf{C}$ *was transposed in lines 1–2* **then**
16 $\quad \mathbf{X}^\star \leftarrow \mathbf{X}^{\star T}$;
17 **return** $\mathbf{X}^\star$;

If the adaption of the Hungarian Algorithm to unbalanced instances of LSAP is used in line 13, FLWC runs in $O(\min\{n,m\}^2 \max\{n,m\})$ time and $O(nm)$ space. Table 3 compares FLWC's time and space complexities to the complexities of existing competitors. Note that our reduction principle can also be used for the fast computation of a suboptimal solution for LSAPE. To this end, it suffices

to replace the optimal LSAP solver in line 13 of Algorithm 1 by a suboptimal one such as one of the the greedy heuristics suggested in [24].

Table 3: Time and space complexities of existing algorithms for LSAPE under the assumptions that reductions to LSAP use the Hungarian Algorithm for solving LSAP

method	**time**	**space**
cost-constrained methods		
FBP [27]	$O(\min\{n,m\}^2 \max\{n,m\})$	$O(nm)$
FBP_0 [27]	$O(\max\{n,m\}^3)$	$O(\max\{n,m\}^2)$
SFBP [29]	$O(\max\{n,m\}^3)$	$O(\max\{n,m\}^2)$
general methods		
EBP [21]	$O((n+m)^3)$	$O((n+m)^2)$
HNG_ϵ [4]	$O(\min\{n,m\}^2 \max\{n,m\})$	$O(nm)$
FLWC [our approach]	$O(\min\{n,m\}^2 \max\{n,m\})$	$O(nm)$

4.2 Proving the Correctness of the Reduction Principle

The first step towards the proof is the following Proposition 3, which constitutes a relation between error-correcting matchings and maximum matchings.

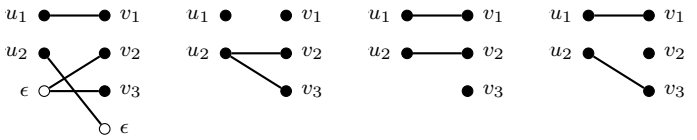
Proposition 3 *Let $\mathbf{X} \in \Pi_{n,m,\epsilon}$ be an error-correcting matching. Furthermore, let $Z_{\mathbf{X}} = (\mathcal{U}, \mathcal{V}, \{(u_i, v_j) \in \mathcal{U} \times \mathcal{V} : x_{i,\epsilon} x_{\epsilon,j} = 1\})$ be the bipartite graph between $\mathcal{U}$ and $\mathcal{V}$ whose edges encode all combinations of node removals and insertions, let $\mathcal{Z}^\star$ be the set of of maximum matchings for $Z_{\mathbf{X}}$, and let the set $\mathcal{Y}_{\mathbf{X}}$ be defined as follows:*

$$\mathcal{Y}_{\mathbf{X}} = \{\mathbf{X}_{\text{sub}} + \mathbf{Z}^\star : \mathbf{Z}^\star \in \mathcal{Z}^\star\} \tag{10}$$

Then $\mathbf{Y} \in \Pi_{n,m}$ holds for each $\mathbf{Y} \in \mathcal{Y}_{\mathbf{X}}$.

Proof 2 *Let $I_{\text{sub}} = \{i \in I : \sum_{j=1}^{m} x_{i,j} = 1\}$ and $J_{\text{sub}} = \{j \in J : \sum_{i=1}^{n} x_{i,j} = 1\}$ be the set of indices of those nodes of $\mathcal{U}$ and $\mathcal{V}$ that are substituted by $\mathbf{X}$. Furthermore, let $s = |I_{\text{sub}}|$ be the number of substitutions encoded by $\mathbf{X}$, and let $\mathbf{Z}^\star$ be a maximum matching for $Z_{\mathbf{X}}$. We observe that $Z_{\mathbf{X}}$ can be viewed as the complete bipartite graph between the nodes $\mathcal{U}_{rem} = \{u_i : i \in I \setminus I_{\text{sub}}\}$ and $\mathcal{V}_{ins} = \{v_j : j \in J \setminus J_{\text{sub}}\}$ that are removed and inserted by $\mathbf{X}$, and that we have $|\mathcal{U}_{rem}| = n - s \leq m - s = |\mathcal{V}_{ins}|$. These observations imply that we have $\sum_{j \in J} z^\star_{i,j} = 0$ for each $i \in I_{\text{sub}}$ and $\sum_{j \in J} z^\star_{i,j} = 1$ for each $i \in I \setminus I_{\text{sub}}$. Similarly, we have $\sum_{i \in I} z^\star_{i,j} = 0$ for each $j \in J_{\text{sub}}$ and $\sum_{i \in I} z^\star_{i,j} \leq 1$ for each $j \in J \setminus J_{\text{sub}}$. This gives us $\sum_{j \in J} x_{i,j} + z^\star_{i,j} = 1$ for each $i \in I$ and $\sum_{i \in I} x_{i,j} + z^\star_{i,j} \leq 1$ for each $j \in J$, which implies $\mathbf{X}_{sub} + \mathbf{Z}^\star \in \Pi_{n,m}$.*□

Example 2 *Consider the error-correcting matching $\mathbf{X}$ shown in Fig 2(a). Since $\mathbf{X}$ removes u_2 and inserts v_2 and v_3, $Z_{\mathbf{X}}$ contains the edges (u_2, v_2) and (u_2, v_3) (Fig 2(b)). There are exactly two maximum matchings for $Z_{\mathbf{X}}$, namely $\mathbf{Z}^\star_1 =$*



(a) $\mathbf{X}$ (b) $Z_{\mathbf{X}}$ (c) $\mathbf{Y}_1$ (d) $\mathbf{Y}_2$

Figure 2: Illustration of Proposition 3

$\{(u_2, v_2)\}$ *and* $\mathbf{Z}_2^\star = \{(u_2, v_3)\}$. *This implies that* $\mathcal{Y}_{\mathbf{X}} = \{\mathbf{Y}_1, \mathbf{Y_2}\}$ *with* $\mathbf{Y}_1 = \{(u_1, v_1), (u_2, v_2)\}$ *(Fig. 2(c)) and* $\mathbf{Y}_2 = \{(u_1, v_1), (u_2, v_3)\}$ *(Fig. 2(d)).*

We now introduce the notion of a minimally-sized error-correcting matching. To this purpose, we call two error-correcting matchings $\mathbf{X}, \mathbf{X}' \in \Pi_{n,m,\epsilon}$ *equivalent* w. r. t. an instance $\mathbf{C}$ of LSAPE (in symbols: $\mathbf{X} \sim_{\mathbf{C}} \mathbf{X}'$) if and only if, for all $(i, j) \in I \times J$, $x_{i,j} = x'_{i,j}$ or $c_{i,j} = c_{i,\epsilon} + c_{\epsilon,j}$ and $x_{i,j} = x'_{i,\epsilon} x'_{\epsilon,j}$ or $x'_{i,j} = x_{i,\epsilon} x_{\epsilon,j}$. By definition of $\sim_{\mathbf{C}}$, the cost L is invariant on the equivalence classes induced by $\sim_{\mathbf{C}}$.

Definition 1 (Minimally-Sized Error-Correcting Matching) *Let* $\mathbf{C} \in \mathbb{R}^{(n+1)\times(m+1)}$ *be an instance of LSAPE and* $\mathbf{X} \in \Pi_{n,m,\epsilon}$ *be an error-correcting matching. Then* $\mathbf{X}$ *is called* minimally-sized *if and only if* $|\operatorname{supp}(\mathbf{X})| < |\operatorname{supp}(\mathbf{X}')|$ *holds for all* $\mathbf{X}' \in [\mathbf{X}]_{\sim_{\mathbf{C}}}$*, where* $\operatorname{supp}(\mathbf{X})$ *is the support of* $\mathbf{X}$.

In other words, $\mathbf{X}$ is minimally-sized just in case it always favours substitution over removal plus insertion, if the costs are the same. By construction, each equivalence class $[\mathbf{X}]_{\sim_{\mathbf{C}}}$ contains exactly one minimally-sized error-correcting matching. In particular, there is always a minimally-sized optimal error-correcting matching.

The next step is to characterize a subset $\Pi_{n,m,\epsilon}(\mathbf{C}) \subseteq \Pi_{n,m,\epsilon}$ which contains all optimal minimally-sized error-correcting matchings for a given instance $\mathbf{C}$ of LSAPE.

Proposition 4 *Let* $\mathbf{C} \in \mathbb{R}^{(n+1)\times(m+1)}$ *be an instance of LSAPE, and let the set* $\Pi_{n,m,\epsilon}(\mathbf{C}) \subseteq \Pi_{n,m,\epsilon}$ *of cost-dependent error-correcting matchings be defined as follows:*

$$\Pi_{n,m,\epsilon}(\mathbf{C}) = \{\, \mathbf{X} \in \Pi_{n,m,\epsilon} \;:\; \begin{array}{l} \forall (i,j) \in I \times J,\ (1 - \delta^{\mathbf{C}}_{i,j,\epsilon}) x_{i,j} = 0, \\ \forall (i,j) \in I \times J,\ \delta^{\mathbf{C}}_{i,j,\epsilon} x_{i,\epsilon} x_{\epsilon,j} = 0 \} \end{array}$$

Then $\Pi_{n,m,\epsilon}(\mathbf{C})$ *contains all minimally-sized optimal error-correcting matchings for the instance* $\mathbf{C}$ *of LSAPE.*

Proof 3 *Assume that there is a minimally-sized optimal error-correcting matching* $\mathbf{X}^\star \in \Pi_{n,m,\epsilon} \setminus \Pi_{n,m,\epsilon}(\mathbf{C})$. *Then there is a pair* $(i, j) \in I \times J$ *such that* $(1 - \delta^{\mathbf{C}}_{i,j,\epsilon}) x^\star_{i,j} = 1$ *or* $\delta^{\mathbf{C}}_{i,j,\epsilon} x^\star_{i,\epsilon} x^\star_{\epsilon,j} = 1$. *Assume that we are in the first case, i. e.,*

that $\delta^{\mathbf{C}}_{i,j,\epsilon} = 0$ *and* $x^\star_{i,j} = 1$*. This implies* $c_{i,j} > c_{i,\epsilon} + c_{\epsilon,j}$*. Now consider the error-correcting matching* $\mathbf{X}'$*, which, instead of substituting* u_i *by* v_j*, removes* u_i *and inserts* v_j *(*$x'_{i,\epsilon}x'_{\epsilon,j} = 1$*). Since* $\delta^{\mathbf{C}}_{i,j,\epsilon} = 0$*,* $\mathbf{X}'$ *is cheaper than* $\mathbf{X}^\star$*. This contradicts* $\mathbf{X}^\star$*'s optimality. If we are in the second case, we have* $x^\star_{i,\epsilon}x^\star_{\epsilon,j} = 1$ *and* $c_{i,j} \leq c_{i,\epsilon} + c_{\epsilon,j}$*. Since* $\mathbf{X}^\star$ *is minimally-sized, we can strengthen the last inequality to* $c_{i,j} < c_{i,\epsilon} + c_{\epsilon,j}$*. Consider the error-correcting matching* $\mathbf{X}'$*, which, instead of removing* u_i *and inserting* v_j*, substitutes* u_i *by* v_j *(*$x'_{i,j} = 1$*). Again,* $\mathbf{X}'$ *is cheaper than* $\mathbf{X}^\star$*, which is a contradiction to* $\mathbf{X}^\star$*'s optimality.*□

The following Proposition 5 shows that the transformation function $f_{\mathbf{C}}$ defined in Theorem 1 indeed maps maximum matchings to error-correcting matchings and that it is surjective on $\Pi_{n,m,\epsilon}$.

Proposition 5 *Let* $f_{\mathbf{C}} : \Pi_{n,m} \to \Pi_{n,m,\epsilon}$ *be defined as in Theorem 1. Then it holds that* $\mathrm{img}(f_{\mathbf{C}}) \subseteq \Pi_{n,m,\epsilon}$ *and that* $\mathrm{img}(f_{\mathbf{C}}) \supseteq \Pi_{n,m,\epsilon}(\mathbf{C})$*.*

Proof 4 *Consider a maximum matching* $\overline{\mathbf{X}} \in \Pi_{n,m}$ *and let* $\mathbf{X} = f_{\mathbf{C}}(\overline{\mathbf{X}})$*. From (7) and (8), we have* $x_{i,\epsilon} + \sum_{j=1}^{m} x_{i,j} = 1$ *for each* $i \in I$*. From (7) and (9), we have* $x_{\epsilon,j} + \sum_{i=1}^{n} x_{i,j} = 1$ *for each* $j \in J$*. This implies* $\mathbf{X} \in \Pi_{n,m,\epsilon}$ *and thus* $\mathrm{img}(f_{\mathbf{C}}) \subseteq \Pi_{n,m,\epsilon}$*.*

For showing $\mathrm{img}(f_{\mathbf{C}}) \supseteq \Pi_{n,m,\epsilon}(\mathbf{C})$*, we fix an error-correcting matching* $\mathbf{X} \in \Pi_{n,m,\epsilon}(\mathbf{C})$*. From Proposition 3, there is a set* $\mathcal{Y}_{\mathbf{X}}$ *of maximum matchings representing* $\mathbf{X}$*. Consider a matching* $\overline{\mathbf{X}} \in \mathcal{Y}_{\mathbf{X}}$*, i. e.* $\overline{\mathbf{X}} = \mathbf{X}_{sub} + \mathbf{Z}^\star$ *with* $\mathbf{Z}^\star \in \mathcal{Z}^\star_{\mathbf{X}}$ *We will show that* $\mathbf{X} = f_{\mathbf{C}}(\overline{\mathbf{X}})$*, which proves the proposition. To this end, we first show the following equalities:*

$$\forall (i,j) \in I \times J, \quad \delta^{\mathbf{C}}_{i,j,\epsilon} z^\star_{i,j} = 0 \tag{11}$$

Consider a pair $(i,j) \in I \times J$ *with* $z^\star_{i,j} = 1$*. From* $Z_{\mathbf{X}} \in \mathcal{Z}^\star_{\mathbf{X}}$*, we know that* $x_{i,\epsilon}x_{\epsilon,j} = 1$*. As* $\mathbf{X} \in \Pi_{n,m,\epsilon}(\mathbf{C})$*, this implies* $\delta^{\mathbf{C}}_{i,j,\epsilon} = 0$ *and hence proves (11). Now let* $(i,j) \in I \times J$*. It holds that* $f_{\mathbf{C}}(\overline{\mathbf{X}})_{i,j} = \delta^{\mathbf{C}}_{i,j,\epsilon} x_{i,j} + \delta^{\mathbf{C}}_{i,j,\epsilon} z^\star_{i,j} = \delta^{\mathbf{C}}_{i,j,\epsilon} x_{i,j} = x_{i,j}$*, where the first equality follows from the definitions of* $f_{\mathbf{C}}$ *and* $\overline{\mathbf{X}}$*, the second equality follows from (11), and the third equality follows from* $\mathbf{X} \in \Pi_{n,m,\epsilon}(\mathbf{C})$*. We have hence shown that* $\mathbf{X}_{sub} = f_{\mathbf{C}}(\overline{\mathbf{X}})_{sub}$*. Next, we show that* $f_{\mathbf{C}}(\overline{\mathbf{X}})_{ins} = \mathbf{x}_{ins}$*. Let* $j \in J$*. We have* $f_{\mathbf{C}}(\overline{\mathbf{X}})_{\epsilon,j} = 1 - \sum_{i=1}^{n} \delta^{\mathbf{C}}_{i,j,\epsilon} x_{i,j} - \sum_{i=1}^{n} \delta^{\mathbf{C}}_{i,j,\epsilon} z^\star_{i,j} = 1 - \sum_{i=1}^{n} \delta^{\mathbf{C}}_{i,j,\epsilon} x_{i,j} = 1 - \sum_{i=1}^{n} x_{i,j} = x_{\epsilon,j}$*, as required. Again, the first equality follows from the definitions of* $f_{\mathbf{C}}$ *and* $\overline{\mathbf{X}}$*, the second equality follows from (11), and the third equality follows from* $\mathbf{X} \in \Pi_{n,m,\epsilon}(\mathbf{C})$*. The last equality follows from the fact that* $\mathbf{X}$ *is an error-correcting matching. The argument for showing that* $f_{\mathbf{C}}(\overline{\mathbf{X}})_{rem} = \mathbf{x}_{rem}$ *is analogous.*□

Now we prove the correctness of our reduction principle.

Proof of Theorem 1 1 *Let* $\overline{\mathbf{X}} \in \Pi_{n,m}$ *be a maximum matching. Its cost*

w. r. t. $\overline{\mathbf{C}}$ *is given by:*

$$
\begin{aligned}
L(\overline{\mathbf{X}}, \overline{\mathbf{C}}) &= \sum_{i=1}^{n}\sum_{j=1}^{m}(\delta^{\mathbf{C}}_{i,j,\epsilon}c_{i,j} + (1-\delta^{\mathbf{C}}_{i,j,\epsilon})(c_{i,\epsilon}+c_{\epsilon,j}) - \delta_{n<m}c_{\epsilon,j})\overline{x}_{i,j} \\
&= \sum_{i=1}^{n}\sum_{j=1}^{m} c_{i,j}\delta^{\mathbf{C}}_{i,j,\epsilon}\overline{x}_{i,j} + (c_{i,\epsilon}+c_{\epsilon,j})(1-\delta^{\mathbf{C}}_{i,j,\epsilon})\overline{x}_{i,j} \\
&\quad -\delta_{n<m}\left(\sum_{j=1}^{m} c_{\epsilon,j}\sum_{i=1}^{n}\overline{x}_{i,j} + \sum_{j=1}^{m} c_{\epsilon,j} - \sum_{j=1}^{m} c_{\epsilon,j}\right) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{m} c_{i,j}\underbrace{(\delta^{\mathbf{C}}_{i,j,\epsilon}\overline{x}_{i,j})}_{=f_{\mathbf{C}}(\overline{\mathbf{X}})_{i,j}} + \sum_{i=1}^{n} c_{i,\epsilon}\underbrace{\left(\sum_{j=1}^{m}(1-\delta^{\mathbf{C}}_{i,j,\epsilon})\overline{x}_{i,j}\right)}_{=A_i} \\
&\quad + \sum_{j=1}^{m} c_{\epsilon,j}\underbrace{\left(\delta_{n<m} + \sum_{i=1}^{n}(1-\delta^{\mathbf{C}}_{i,j,\epsilon}-\delta_{n<m})\overline{x}_{i,j}\right)}_{=B_j} - \delta_{n<m}\sum_{j=1}^{m} c_{\epsilon,j}
\end{aligned}
$$

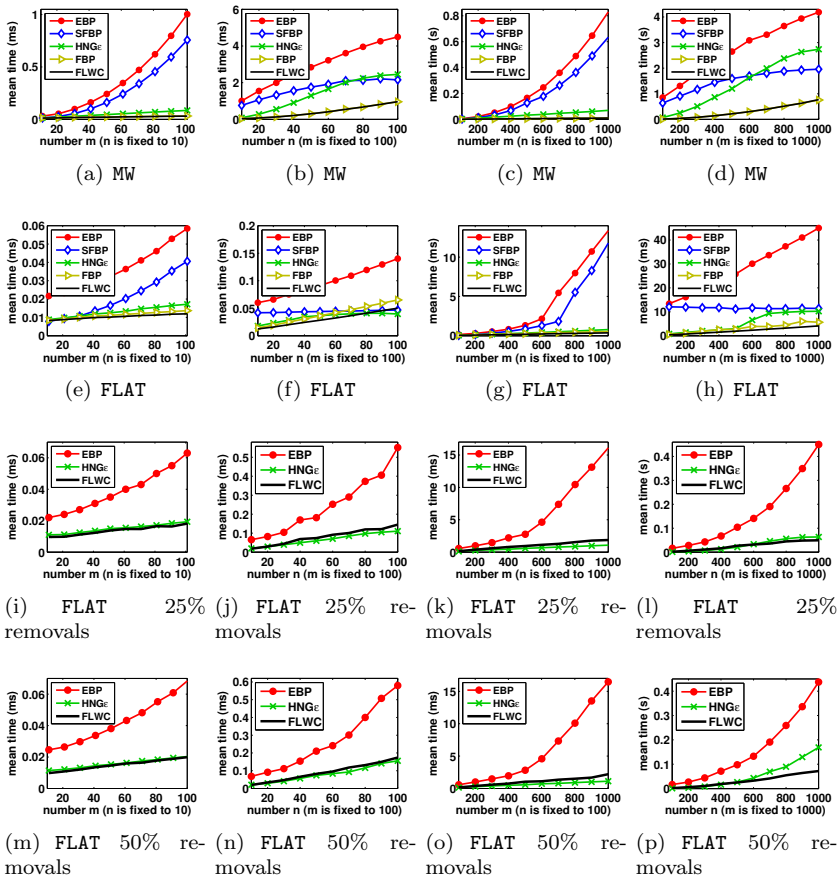
Since $\overline{\mathbf{X}}$ *is a maximum matching for* $\Pi_{n,m}$*, we know that* $A_i = 1-\sum_{j=1}^{m}\delta^{\mathbf{C}}_{i,j,\epsilon}\overline{x}_{i,j} = f_{\mathbf{C}}(\overline{\mathbf{X}})_{i,\epsilon}$ *for each* $i \in I$*. We now distinguish the cases* $\delta_{n<m}=1$ *and* $\delta_{n<m}=0$*. In the first case, we immediately have* $B_j = 1-\sum_{i=1}^{n}\delta^{\mathbf{C}}_{i,j,\epsilon}\overline{x}_{i,j} = f_{\mathbf{C}}(\overline{\mathbf{X}})_{\epsilon,j}$ *for each* $j \in J$*. In the second case, we have* $n=m$ *and* $B_j = f_{\mathbf{C}}(\overline{\mathbf{X}})_{\epsilon,j}$ *holds, too, since, for balanced instances, a maximum matching contains an edge* (u_i, v_j) *for each* $j \in J$*. We have thus shown the following equality:*

$$
\forall \overline{\mathbf{X}} \in \Pi_{n,m}, \quad L(\overline{\mathbf{X}}, \overline{\mathbf{C}}) = L(f_{\mathbf{C}}(\overline{\mathbf{X}}), \mathbf{C}) - \delta_{n<m}\sum_{j=1}^{m} c_{\epsilon,j} \tag{12}
$$

Now let $\mathbf{X}^\star$ *be a minimally-sized optimal error-correcting matching for* $\mathbf{C}$*,* $\overline{\mathbf{X}}^\star$ *be an optimal maximum matching for* $\overline{\mathbf{C}}$*, and* $\mathbf{X}' = f_{\mathbf{C}}(\overline{\mathbf{X}}^\star)$*. From (12), we know that* $L(\mathbf{X}', \mathbf{C}) = L(\overline{\mathbf{X}}^\star, \overline{C}) + \delta_{n<m}\sum_{j=1}^{m} c_{\epsilon,j}$*. Therefore, the theorem follows if we can show that* $L(\mathbf{X}', \mathbf{C}) = L(\mathbf{X}^\star, \mathbf{C})$*. The* $\geq$*-direction of the desired equality follows from* $\mathbf{X}^\star$*'s optimality and the fact that, from Proposition 5, we know that* $\mathbf{X}' \in \Pi_{n,m,\epsilon}$*. For showing the* $\leq$*-direction, we pick an* $\overline{\mathbf{X}}' \in \Pi_{n,m}$ *with* $f_{\mathbf{C}}(\overline{\mathbf{X}}') = \mathbf{X}^\star$*. Such an* $\overline{\mathbf{X}}'$ *exists because, from Proposition 4, we know that* $\overline{\mathbf{X}}^\star \in \Pi_{n,m,\epsilon}(\mathbf{C})$*, and, from Proposition 5, we have* $\mathrm{img}(f_{\mathbf{C}}) \supseteq \Pi_{n,m,\epsilon}(\mathbf{C})$*. Assume that* $L(\mathbf{X}', \mathbf{C}) > L(\mathbf{X}^\star, \mathbf{C})$*. Then (12) implies that* $L(\overline{\mathbf{X}}^\star, \overline{C}) > L(\overline{\mathbf{X}}', \overline{C})$*, which contradicts* $\overline{\mathbf{X}}^\star$*'s optimality.*□

5 Experimental Validation

In our experiments, we compared `FLWC` to all existing competitors mentioned in Table 3. We evaluated the runtime of the different methods (Section 5.1) and their performances when used within approximate approaches for the computation of GED (Section 5.2). All procedures use the same C++ implementation



(a) MW (b) MW (c) MW (d) MW

(e) FLAT (f) FLAT (g) FLAT (h) FLAT

(i) FLAT 25% removals (j) FLAT 25% removals (k) FLAT 25% removals (l) FLAT 25% removals

(m) FLAT 50% removals (n) FLAT 50% removals (o) FLAT 50% removals (p) FLAT 50% removals

Figure 3: Time comparison for Machol-Wien instances (first row) and flat instances (all other rows).

of the Hungarian Algorithm as LSAP solver, which is based on the version presented in [16, 8]. $\texttt{HNG}_\epsilon$ is also based on this version, so that all procedures are comparable.[1]

5.1 Time Comparison

To illustrate the differences between the methods, we recorded their execution time on three types of instances, some of which do and some of which do not respect the triangle inequalities: Machol-Wien instances, flat instances, and random instances. Experiments on further types yielded similar results. Fig .3 shows the results for Machol-Wien instances and flat instances, Fig. 4 displays

[1]Procedures are written both in C++ and MATLAB (or GNU Octave). The source code is available at `bougleux.users.greyc.fr/lsape/`.

the results for random instances. For each instance type, the tested methods' execution time is reported for several values of n and m. In the first and third columns of the figures, n is fixed and m is varied, while in the second and fourth columns, m is fixed and n is varied. In any case, we have $n \leq m$. The general methods FLWC, EBP, and $\texttt{HNG}_\epsilon$ were tested on all instances, the cost-constrained algorithms FBP, $\texttt{FBP}_0$, and SFBP only on those that respect the triangle inequalities. Since $\texttt{FBP}_0$ was slower than FBP across all instances, our plots do not contain curves for $\texttt{FBP}_0$.

Machol-Wien Instances [18] Machol-Wien instances are defined by $c_{i,j} = i * j \ \forall (i,j)$ and thus satisfy the triangle inequalities. They are known to be hard to optimize for classical LSAP solvers such as the Hungarian Algorithm. This is also the case for the LSAPE solver $\texttt{HNG}_\epsilon$, but not for the other methods. This is explained by the fact that all methods except for $\texttt{HNG}_\epsilon$ transform an instance $\mathbf{C}$ of LSAPE into an instance $\overline{\mathbf{C}}$ of LSAP, which reduces the difficulty of the problem. The first row of Fig. 3 shows the results of our experiments on Machol-Wien instances. We observe that FLWC and FBP perform very similarly, provide the best results in all cases, and are more stable than the other methods. EBP and SFBP are more time consuming, in particular when n and m increases.

Flat Instances In a first series of experiments, we considered flat instances of the form $\mathbf{C} = \alpha \mathbf{1}_{n \times m}$, with $\alpha = 10$. These instances satisfy the triangle inequalities. Moreover, they are easy to solve, which implies that a large part of the execution time is spent on the initialization step of the Hungarian Algorithm and the cost transformations. The results for instances of this kind are displayed in the second row of Fig. 3. As before, FLWC and FBP are more stable and efficient than the other methods. In a second series of experiments, we varied the construction of $\mathbf{C}$ in order to enforce that $k\%$ of the nodes contained in the smaller set be removed. This can be achieved by randomly selecting $m - n$ nodes in the larger set and setting their insertion cost to 0. Subsequently, $k\%$ of the remaining elements are selected and their insertion cost is set to $(\alpha/2) - 1$. Similarly, the removal costs of $k\%$ of the nodes contained in the smaller set are set to $(\alpha/2) - 1$. The resulting instances do not satisfy the triangle inequalities, and thus FBP, $\texttt{FBP}_0$, and SFBP are not tested (they do not compute an optimal solution). The results are shown in the third and the fourth row of Fig. 3. Both FLWC and $\texttt{HNG}_\epsilon$ clearly outperform EBP, and FLWC is slightly more stable than $\texttt{HNG}_\epsilon$.

Random instances We also carried out experiments on random instances similar to the ones presented in [28, 29]. These instances are very similar to the ones that occur in the context of approximation of GED. The cost $c_{i,j}$ encodes the cost of substituting a node v_i and its set of incident edges in a graph G by a node v_j and its set of incident edges in a graph H. Graphs are constructed locally by assigning node degrees randomly from 1 to $d_{\max} = (3/10)|V_G|$, where $|V_G|$ is the number of nodes in the graph G. Nodes are labeled randomly with

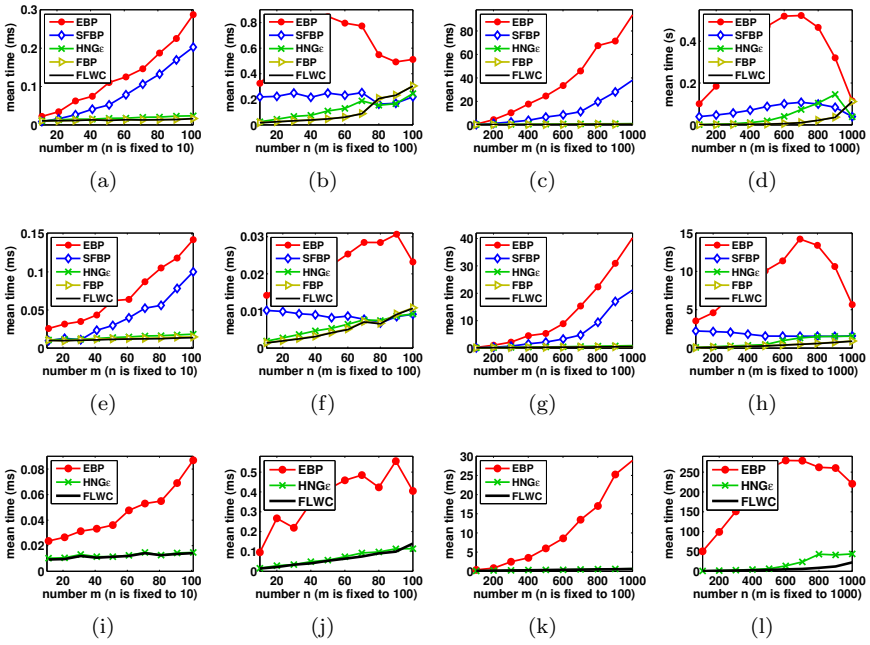


Figure 4: Time comparison for random instances of LSAPE (see text).

an integer value in $\{1, \ldots, l_{\max}\}$, where $l_{\max} = \sqrt{n * m}/10$. Similarly, edges are labeled with a binary value. Then $c_{i,j}$ is defined as the cost of substituting the labels (0 if they are the same, or the constant c_s else) plus the cost of an optimal error-correcting matching between the sets of incident edges. The cost of substituting two edges is defined as before, and the cost of removing or inserting an edge is defined by the constant c_{eri}. Similarly, removal and insertion costs are defined as the cost of removing the node plus the cost of removing its incident edges, which is given by $d_i\, c_{eri} + c_{nri}$, where d_i is the node degree and c_{nri} is its removal or insertion cost. The three following set of parameters (c_s, c_{nri}, c_{eri}) are considered: $(40, 20, 20)$, $(20, 40, 0)$, and $(40, 2, 2)$. The first parameter setting (first row of Fig. 4) satisfies the triangle inequality. The second (second row of Fig. 4) and the third parameter setting (third row of Fig. 4) do not satisfy the triangle inequalities. However, for the second setting, none of the computed optimal error-correcting contains removals and insertions, and so all methods were tested. On the contrary, optimal error-correcting matchings for the third setting indeed contain up to 20% of removals, which is why we did not carry out tests for the cost-constrained methods `FBP`, $\texttt{FBP}_0$, and `SFBP`. We again observe that our method `FLWC` is globally the most stable algorithm and obtains the best results on average.

5.2 Effect on Approximation of GED

As mentioned in the introduction, many methods approximate GED by lossily transforming the problem of its computation into an instance $\mathbf{C}$ of LSAPE [20, 12, 3, 21, 31, 32, 9]. Lossy transformations from GED to LSAPE can be used for computing both upper and lower bounds for GED. Given an error-correcting matching for $\mathbf{C}$ computed by any LSAPE algorithm, an upper bound for GED is derived by computing the distance associated to the edit path induced by the considered matching. Since this edit path may be suboptimal, the computed distance might be an overestimation of the exact GED, and thus constitutes an upper bound. With this paradigm, each error-correcting matching for $\mathbf{C}$ yields a valid upper bound, although the ones induced by optimal matchings are usually tighter. Note that, if the LSAPE instance $\mathbf{C}$ constructed by a lossy transformation has only one optimal solution, each optimal LSAPE solver yields the same upper bound for GED. If $\mathbf{C}$ has more than one optimal solution, there might be differences in accuracy, since different LSAPE solvers might pick different optimal solutions depending on the organization of the input data. However, these differences in accuracy are arbitrary and hence disappear once the upper bounds are averaged across enough pairs of input graphs. In other words, when it comes to the approximation of GED, the only relevant property of an exact LSAPE solver is its runtime behaviour.

This observation is empirically confirmed by the experiments reported in Table 4, which shows how different methods for solving LSAPE affect the performance of the algorithm `BP` suggested in [21], which computes an upper bound for GED. Given two graphs G_1 and G_2 on n_1 and n_2 nodes, respectively, `BP` constructs a LSAPE instance $\mathbf{C} \in \mathbb{R}^{(n_1+1)\times(n_2+1)}$. For computing the cell $c_{i,j}$ for $(i,j) \in \{1,\dots,n_1\} \times \{1,\dots,n_2\}$, `BP` has to solve another LSAPE instance of size $(\deg_{G_1}(u_i)+1) \times (\deg_{G_2}(v_j)+1)$, where $\deg_{G_1}(u_i)$ is the degree of node u_i in G_1 and $\deg_{G_2}(v_j)$ is the degree of node v_j in G_2. So altogether, `BP` has to solve $1 + n_1 n_2$ instances of LSAPE. The tests were carried out on the datasets Acyclic and MAO from the ICPR GED contest [1]. In order to ensure that also the cost-constrained LSAPE solvers compute optimal solutions, we defined metric edit costs by setting the cost of substituting nodes and edges to 1 and the cost of deleting and inserting nodes and edges to 3. We see that, as expected, there are no significant differences between the different solvers w. r. t. the tightness of the produced upper bounds. In terms of runtime, our solver `FLWC` performs best, followed by `FBP` and $\texttt{HNG}_\epsilon$.

In contrast to the situation for upper bounds, lossy transformations from GED to LSAPE which aim at the computation of lower bounds [3, 2] as well as methods based on conditional gradient descent [5, 4] where LSAPE occurs as a subproblem crucially depend on the optimality of the computed error-correcting matching. Therefore, these methods cannot use the existing fast LSAPE solvers `FBP`, $\texttt{FBP}_0$, and `SFBP`, unless the triangle inequalities are known to be satisfied. For instance, lower bounds for GED are obtained from the cost $L(\mathbf{X}^\star, \mathbf{C})$ of an optimal error-correcting matching for the LSAPE instance $\mathbf{C}$. If $\mathbf{X}^\star$ is not optimal, $L(\mathbf{X}^\star, \mathbf{C})$ is not in general a valid lower bound.

Table 4: Effect of LSAPE methods on algorithm BP suggested in [21] that computes an upper bound for GED on datasets with metric costs.

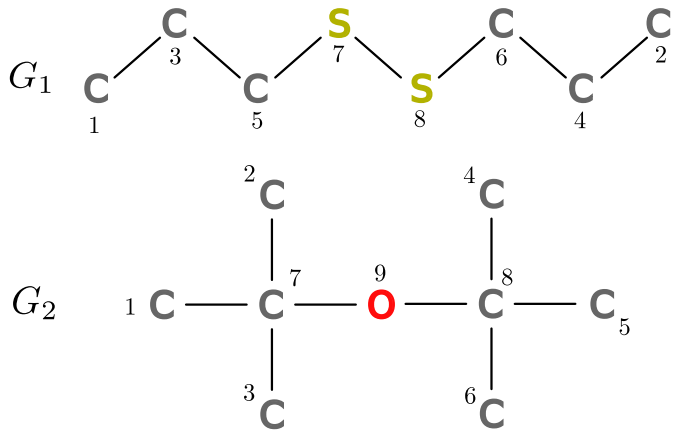
method	time in μs	average UB	time in μs	average UB
	Acyclic		MAO	
cost-constrained methods				
FBP	1.61	38.86	6.62	107.97
$\texttt{FBP}_0$	2.04	38.89	8.10	107.85
SFBP	2.14	39.28	11.10	107.48
general methods				
EBP	4.84	39.14	23.30	107.48
$\texttt{HNG}_\epsilon$	1.87	38.97	7.46	107.59
FLWC	1.53	38.86	6.05	108.04

Table 5 shows how different methods for solving LSAPE affect the performance of the algorithm BRANCH suggested in [3, 2], which computes a lower bound for GED. Like BP, given to graphs G_1 and G_2 on n and n_2 nodes, respectively, BRANCH has to solve $1 + n_1 n_2$ instances of LSAPE. One of them is of size $(n_1 + 1) \times (n_2 + 1)$, the other ones are of size $(\deg_{G_1}(u_i) + 1) \times (\deg_{G_2}(v_j) + 1)$. The tests were again carried out on the datasets Acyclic and MAO, but this time, edit costs that do not satisfy the triangle inequality were used (cf. [1] for more details on the edit costs, especially equation (1) and setting 3 in Table 2). First of all, we can see that the classic approach EBP requires more computational time than other optimized approaches, and that FLWC, FBP, and $\texttt{HNG}_\epsilon$ are the fastest methods. Second, we observe that the cost-constrained methods FBP, $\texttt{FBP}_0$ and SFBP are not able to deal with costs which do not satisfy the triangle inequality: Very often, they compute invalid lower bounds that exceed the exact GED, which we computed using a binary linear programming approach [17]. This is explained by the fact that, if the triangle inequality is not satisfied, optimal error-correcting matchings might well include both insertions and removals. However, those error-correcting matchings are not considered by cost-constrained methods for LSAPE.

Figure 5 shows a case where the cost-constrained methods FBP, $\texttt{FBP}_0$ and SFBP compute an invalid lower bound. Considering the two graphs G_1 and G_2 extracted from Acyclic dataset, cost-constrained methods provide a matching $\varphi_{\texttt{FBP}}$ that favours node substitution over removal plus insertion even if removal plus insertion is cheaper. For instance, $\varphi_{\texttt{FBP}}$ matches the nodes 7 and 8 of G_1 to the nodes 9 and 5 of G_2, although it is cheaper to first remove 7 and 8 and then insert 9 and 5, as done by the matching $\varphi_{\texttt{FLWC}}$ computed by FLWC. In conclusion, cost-constrained methods do not guarantee valid lower bounds for general edit costs, while our algorithm FLWC does.

Table 5: Effect of LSAPE methods on algorithm BRANCH suggested in [3, 2] that computes a lower bound for GED on datasets with non metric costs.

method	time in μs	# invalid LB	time in μs	# invalid LB
	Acyclic		MAO	
cost-constrained methods				
FBP	1.28	4	8.30	416
FBP_0	1.64	4	10.80	416
SFBP	1.91	4	12.81	416
general methods				
EBP	5.34	0	21.10	0
HNG_ϵ	1.54	0	7.59	0
FLWC	1.23	0	8.50	0



$$\varphi_{\texttt{FLWC}} = [1, 2, 7, 8, 5, 6, \varepsilon, \varepsilon] \quad \varphi_{\texttt{FBP}} = [1, 2, 7, 8, 3, 4, 9, 5]$$

Figure 5: Example of lower bound computation where FBP based methods do not allow to compute a valid lower bound.

6 Conclusions

In this paper, we presented FLWC, a new efficient method for solving the linear sum assignment problem with error correction. The technical backbone of our method is a cost-dependent factorization of substitutions into removals and insertions. FLWC runs in $O(\min\{n, m\}^2 \max\{n, m\})$ time and $O(nm)$ space. Its complexities are hence the same as the ones of the most efficient state of the art methods FBP and HNG_ϵ.

The advantage of our method FLWC over FBP is that, unlike FBP, FLWC remains valid for any configuration of the cost matrix and does not require the triangle inequality to holds. Therefore, our method allows to efficiently retrieve a lower

bound for the graph edit distance irrespectively of whether or not the edit costs respect the triangle inequality. This is especially important in settings where the edit costs are deduced from calculus and where one does not have an a priori guarantee that the triangle inequality will be satisfied. Situations of this kind occur, for instance, in some quadratic approximations of the graph edit distance problem [5, 4].

One advantage of `FLWC` over $\texttt{HNG}_\epsilon$ is that, although both algorithms have the same time complexity, `FLWC` is slightly faster in practice. In particular, `FLWC` is more stable than $\texttt{HNG}_\epsilon$, in the sense that it also allows to quickly solve difficult LSAPE instances with whom $\texttt{HNG}_\epsilon$ struggles. A second advantage is that `FLWC` is much easier to implement: While implementing $\texttt{HNG}_\epsilon$'s adaptation of the Hungarian Algorithm to LSAPE requires a thorough knowledge of matching theory, `FLWC` can be implemented by slightly transforming the cost matrix and then calling a solver for LSAP. Since LSAP is a very famous combinatorial optimization problem, libraries are available for all major programming languages.

For future work, we are planning to integrate the factorization of the assignment matrix, which is the core of our paper, in a modeling of the graph edit distance problem as a quadratic assignment problem.

References

[1] Z. Abu-Aisheh, B. Gaüzère, S. Bougleux, Jean-Yves Ramel, L. Brun, R. Raveaux, P. Héroux, and S. Adam. Graph edit distance contest 2016: Results and future challenges. *Pattern Recogn. Lett.*, 100:96–103, 2017.

[2] D. B. Blumenthal and J. Gamper. Improved lower bounds for graph edit distance. *IEEE Trans. Knowl. Data Eng.*, 30(3):503–516, 2018.

[3] David Benjamin Blumenthal and Johann Gamper. Correcting and speeding-up bounds for non-uniform graph edit distance. In *ICDE*, pages 131–134, 2017.

[4] S. Bougleux, B. Gaüzère, and L. Brun. A hungarian algorithm for error-correcting graph matching. In *GbRPR*, volume 10310 of *LNCS*, pages 118–127, 2017.

[5] Sébastien Bougleux, Luc Brun, Vincenzo Carletti, Pasquale Foggia, Benoit Gaüzère, and Mario Vento. Graph edit distance as a quadratic assignment problem. *Pattern Recogn. Lett.*, 87:38–46, 2017.

[6] F. Bourgeois and J.C. Lassalle. An extension of the Munkres algorithm for the assignment problem to rectangular matrices. *Commun. ACM*, 14:802–804, 1971.

[7] Horst Bunke and Gudrun Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recogn. Lett.*, 1(4):245–253, 1983.

[8] R. Burkard, M. Dell'Amico, and S. Martello. *Assignment Problems*. SIAM, 2009.

[9] Vincenzo Carletti, Benoit Gaüzère, Luc Brun, and Mario Vento. Approximate graph edit distance computation combining bipartite matching and exact neighborhood substructure distance. In *GbRPR*, volume 9069 of *LNCS*, pages 188–197, 2015.

[10] É. Daller, S. Bougleux, B. Gaüzère, and L. Brun. Approximate graph edit distance by several local searches in parallel. In *ICPRAM*, pages 149–158, 2018.

[11] Miquel Ferrer, Francesc Serratosa, and Kaspar Riesen. A first step towards exact graph edit distance using bipartite graph matching. In *GbRPR*, volume 9069 of *LNCS*, pages 77–86, 2015.

[12] B. Gaüzère, S. Bougleux, K. Riesen, and L. Brun. Approximate graph edit distance guided by bipartite matching of bags of walks. In *S+SSPR*, volume 8621 of *LNCS*, pages 73–82, 2014.

[13] W. Jones, A. Chawdhary, and A. King. Revisiting Volgenant-Jonker for approximating graph edit distance. In *GbRPR*, volume 9069 of *LNCS*, pages 98–107, 2015.

[14] W. Jones, A. Chawdhary, and A. King. Optimising the Volgenant-Jonker algorithm for approximating graph edit distance. *Pattern Recogn. Lett.*, 87:47–54, 2017.

[15] Harold W Kuhn. Variants of the hungarian method for the assignment problem. *Nav. Res. Logist. Q.*, 3:253–258, 1956.

[16] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, New York, 1976.

[17] Julien Lerouge, Zeina Abu-Aisheh, Romain Raveaux, Pierre Héroux, and Sébastien Adam. Exact graph edit distance computation using a binary linear program. In *S+SSPR*, volume 10029 of *LNCS*, pages 485–495, 2016.

[18] R. Machol and M. Wien. A 'hard' assignment problem. *Oper. Res.*, 24:190–192, 1976.

[19] James Munkres. Algorithms for the assignment and transportation problems. *SIAM J. Appl. Math.*, 5(1):32–38, 1957.

[20] K. Riesen. *Structural Pattern Recognition with Graph Edit Distance*. Advances in Computer Vision and Pattern Recognition. Springer, 2015.

[21] K. Riesen and H. Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image Vis. Comput.*, 27(7):950–959, 2009.

[22] K. Riesen and H. Bunke. Improving bipartite graph edit distance approximation using various search strategies. *Pattern Recogn.*, 28(4):1349–1363, 2015.

[23] K. Riesen, M. Neuhaus, and H. Bunke. Bipartite graph matching for computing the edit distance of graphs. In *GbRPR*, volume 4538 of *LNCS*, pages 1–12, 2007.

[24] Kaspar Riesen, Miquel Ferrer, Andreas Fischer, and Horst Bunke. Approximation of graph edit distance in quadratic time. In *GbRPR*, volume 9069 of *LNCS*, pages 3–12, 2015.

[25] Kaspar Riesen, Andreas Fischer, and Horst Bunke. Improved graph edit distance approximation with simulated annealing. In *GbRPR*, volume 10310 of *LNCS*, pages 222–231, 2017.

[26] A. Sanfeliu and K.-S. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Trans. Syst. Man Cybern.*, 13(3):353–362, 1983.

[27] F. Serratosa. Fast computation of bipartite graph matching. *Pattern Recogn. Lett.*, 45:244–250, 2014.

[28] F. Serratosa. Computation of graph edit distance: Reasoning about optimality and speed-up. *Image Vis. Comput.*, 40:38–48, 2015.

[29] F. Serratosa. Speeding up fast bipartite graph matching through a new cost matrix. *Int. J. Pattern Recogn.*, 29(2), 2015.

[30] Sousuke Takami and Akihiro Inokuchi. Accurate and fast computation of approximate graph edit distance based on graph relabeling. In *ICPRAM*, pages 17–26, 2018.

[31] Zhiping Zeng, Anthony K. H. Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: On approximating graph edit distance. *PVLDB*, 2(1):25–36, 2009.

[32] Weiguo Zheng, Lei Zou, Xiang Lian, Dong Wang, and Dongyan Zhao. Efficient graph similarity search over large graph databases. *IEEE Trans. Knowl. Data Eng*, 27(4):964–978, 2015.