Enhancing Graph Edit Distance Computation: Stronger and Orientation-based ILP Formulations

Andrea D'Ascenzo
Luiss University
Rome, Italy
Gran Sasso Science Institute
L'Aquila, Italy
adascenzo@luiss.it

Julian Meffert
University of Bonn
Bonn, Germany
s6jumeff@uni-bonn.de

Petra Mutzel
University of Bonn
Bonn, Germany
pmutzel@uni-bonn.de

Fabrizio Rossi
University of L'Aquila
L'Aquila, Italy
fabrizio.rossi@univaq.it

ABSTRACT

The graph edit distance (GED) is among the most widely used graph similarity measures in practice. It asks for a minimum cost edit path between two given labeled graphs G and H, where the edit path is defined as a sequence of operations (e.g., node and edge insertions, deletions or substitutions) that successively transform the graph G into H.

In this work, we suggest a new ILP formulation (FORI) based on orienting the corresponding edge variables. Moreover, we suggest enhancing two state-of-the-art ILP formulations by incorporating additional inequalities. We theoretically compare the strength of the formulations with respect to their Linear Programming relaxations. The result is a hierarchy with (FORI) at the top.

Our extensive evaluation on widely used benchmark sets shows that our improved formulations run significantly faster than the previous ones. These allow to solve to proven optimality all the reference instances from common databases, such as the IAM Graph Database, many of which were prohibitive with state-of-the-art methods. Moreover, we are able to compute the GED of a small pattern and a large graph such as CORA and PUBMED, having up to 19,717 nodes and 44,327 edges.

PVLDB Reference Format:
Andrea D'Ascenzo, Julian Meffert, Petra Mutzel, and Fabrizio Rossi. Enhancing Graph Edit Distance Computation:
Stronger and Orientation-based ILP Formulations. PVLDB, 18(11): 4737 - 4749, 2025.
doi:10.14778/3749646.3749726

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/meffertj/FORI-GED.

Proceedings of the VLDB Endowment, Vol. 18, No. 11 ISSN 2150-8097.
doi:10.14778/3749646.3749726

1 INTRODUCTION

Graph data has become ubiquitous for many disciplines ranging from semantic data modeling, over cheminformatics, to computer vision. Rather than strictly determining if two graphs are structurally identical, graph similarity measures are able to provide a quantitative assessment of the (dis)similarity between two graphs and can thus be viewed as an error-tolerant extension of graph isomorphism. Graph similarity measures are the basis for major machine learning tasks such as classification, clustering, associated pattern mining, and outlier detection for data with network structure. For example, they are essential for graph database applications, enabling the user to retrieve, e.g., a small set of graphs that are most similar to a queried graph, placing them at the center of numerous disciplines such as data-driven drug discovery, social network analysis, pattern recognition, and fraud detection. In computer vision, graph similarity often occurs in image matching [35], (multi-) object tracking [23] and pedestrian re-identification [38]. In rational drug design usage of graph similarity is tied to the assumption that structurally similar molecules frequently have similar properties [11]. In fraud detection, graph similarity measures are employed to identify subgraphs similar to a graph pattern of fraudulent behavior or connections among entities [26]. Searching small query graphs in large database graphs, or performing subgraph isomorphism queries is important, e.g., for graph-based data management [12], analogy reasoning in knowledge graphs [13], and for identifying motifs in biological and chemical networks [28, 29, 33].

A widely used graph similarity measure in practice is the graph edit distance (GED), since it is conceptually simple and highly flexible. The GED of two graphs G and H with node and/or edge labels is defined as the minimum cost of a sequence of edit operations that transforms G into a graph that is isomorphic to H. Edit operations consist of insertion, deletion or substitution of nodes and edges, each associated with a non-negative (often dataset-specific) cost. Computing the GED is NP-hard, even for planar graphs or when using unit edit costs (every operation has a cost of 1) [22, 39].

The exact computation of the GED has been the subject of numerous contributions in the literature from various research communities. Complete surveys on exact and approximate GED algorithms are given in [4, 14, 30]. Riesen et al. [32] apply the well-known A^*

search algorithm to compute the GED exactly. The algorithm explores a search tree, where each node of the tree represents a partial edit path, i.e., a sequence of operations applied so far. Complete edit paths that transform one graph into the other correspond to leaves of this tree. The exploration of the tree is guided by traversing the most promising path on the basis of a heuristic estimation of the GED. Unfortunately, in practice it performs poorly due to the large search space it needs to explore, thus yielding too large time and space requirements [5, 21]. Recent developments in A^*-search-based algorithms have been obtained, e.g., by Chen et al., who applied the beam-stack-search paradigm to A^*-search [9], or by exploiting the isomorphism between vertices in order to avoid repeated computations, as done in [20]. However, experimental evaluation of these methods is limited to graphs with up to 30 nodes. In 2023, Chang et al. [7, 8] proposed two improvements of the A^*-search paradigm, namely A^{*+}-LSa and A^*-BMao. By leveraging on improved lower bound estimations and specialized data structure that stores graph node labels compactly, the authors managed to run their algorithms on larger GED instances with up to 64 nodes per graph, establishing novel state-of-the-art algorithms for computing the exact GED. A^*-based search algorithm are also the basis for many heuristic-aided search algorithms [2, 15].

Integer Linear Programming techniques are considered some of the best methods for exactly computing the GED (see, e.g., [1, 5, 21]). Justice and Hero [19] developed an ILP formulation which is fast in practice. However, its applicability is limited because it cannot account for edge labels. Lerouge et al. [21] introduced two different ILP formulations (called (F1) and (F2)) that also take into account the edge labels. In practice, their formulation (F2) outperforms all other methods for exactly computing the GED on general labeled graphs [5], and is thus used as subroutine in many pattern recognition applications (see e.g. the recent work of Xu et al. [37]). Recently, Blumenthal and Gamper [5] have suggested a Mixed Integer Linear Programming formulation for the GED problem. In their practical experiments they confirmed the dominance of the (F2) formulation given in [21]. However, up to now, exact approaches have shown to regularly solve instances of sizes up to ~ 20 nodes on the used benchmark sets, only [5].

Therefore, in practice often heuristics, like local-search and beam-search [4] are used, although they may lead to an arbitrarily poor solution. Recent advancements have focused on neural GED estimators that approximate the GED by leveraging data distribution characteristics [3, 18, 27, 28, 40]. A recent study [17] has pointed out that these methods are constrained by the commonly used datasets for training, since the GED values are known for very small or for trivial instances, only. Finally, often heuristics and learning-based approaches are not designed to also provide the sequence of operations that transform one of the input graphs in the other.

In this work, we make a major step forward on exactly computing the GED based on ILPs. Indeed, ILP based methods (i) are able to compute the exact GED; (ii) are designed to provide both the GED distance and the sequence of edit operations; (iii) do not require preprocessing or training/test phases; (iv) known formulations work on both labeled and unlabeled graphs and on general cost functions; (v) can also be used to train neural networks to provide better performance [37].

Our contribution:

- We introduce a novel ILP formulation (FORI), based on orienting the corresponding edge variables, which theoretically and practically outperforms all state-of-the-art approaches. Unlike previous ILP formulations, which typically establish correspondences between subsets of edges in the two input graphs, our approach introduces variables that encode directed edges. This representation more accurately captures both edge directionality and structural dependencies between the corresponding node pairs. As a result, it enables a decomposition of certain constraints present in existing ILP models, leading to a stronger formulation.
- Furthermore, we improve known state-of-the-art ILP formulations, namely (F1) and (F2) from Lerouge et al. [21], by adding additional inequalities leading to formulations (F1+) and (F2+), respectively.
- We theoretically analyze the strength of state-of-the-art and our new ILP models by studying and comparing their LP relaxations. Our analysis shows that our new formulations (F1+) and (F2+) are strictly stronger than the existing ILP formulations (F1) and (F2), in the sense that their LP relaxations yield tighter lower bounds. Among all formulations considered, (FORI) emerges as the strongest, with its LP relaxation producing the tightest lower bounds overall, including improvements over (F1+) and (F2+).
- Our theoretical findings are confirmed by practical experiments. An extensive experimental study shows that our formulations run significantly faster than the previous ones, allowing to solve to proven optimality many instances whose GED computation was prohibitive with state-of-the-art methods. In particular, our new model (FORI) is able to solve all instances from the *IAM Graph Database* considered in [5]. Indeed, the model can optimally solve GED instances of significantly larger dimensions. Notably, our method makes it possible — for the first time — to routinely solve instances involving graphs with up to 100 nodes, a scale previously considered intractable for exact approaches. We also document the computation of the GED of a small pattern and a large graph such as IMDB, CORA and PUBMED, having up to 19,717 nodes and 44,327 edges.

2 PRELIMINARIES

We follow the notation used in [5]. Let $G = (V_G, E_G, L_V, L_E)$ be an undirected, labeled, simple graph with node set V_G, edge set E_G, and labeling functions $L_V : V_G \to \Sigma_V$ and $L_E : E_G \to \Sigma_E$ assigning labels to nodes and edges.[1]

We say that a node $u \in V_G$ is a neighbor of (or adjacent to) a node $v \in V_G$ if there is an edge $\{u, v\} \in E_G$. Sometimes, the notation uv will be used instead of $\{u, v\}$. The neighborhood $\delta_G(u)$ of a node $u \in V_G$ is the set of all neighbors of u, i.e., $\delta_G(u) = \{v \in V_G \mid \{u, v\} \in E_G\}$. For deletion and insertion we consider the node set as extended by a dummy node ε_V, the edge set extended by a dummy edge ε_E, with $V_{G+\varepsilon} = V \cup \{\varepsilon_V\}$ and $E_{G+\varepsilon} = E_G \cup \{\varepsilon_E\}$. Node and

[1]Throughout this paper, we use the term label to denote node or edge annotations, which are also commonly referred to as attributes in other communities.

edge labels as well as the cost function with respect to these labels are specific to each dataset and application.

Note that the definitions and notations can be straightforwardly extended to directed graphs. In particular, we will denote an edge directed from node u to node v by (u, v), the set of outgoing neighbors of a node $u \in V_G$ by $\delta_G^+(u)$, and the set of ingoing neighbors of a node $u \in V_G$ by $\delta_G^-(u)$.

Two labeled graphs G and H, on common label alphabets L_V and L_E, are called isomorphic if there exists a bijection $\pi : V_G \to V_H$, such that for each node $v \in V_G$ we have $L_V(v) = L_V(\pi(v))$, and for each pair of nodes $u, v \in V_G$ we have that $\pi(u)$ and $\pi(v)$ are adjacent in H if and only if u and v are adjacent in G. Moreover, for all adjacent pairs of nodes $\{u, v\} \in E_G$ we have $L_E(\{u, v\}) = L_E(\{\pi(u), \pi(v)\})$.

The GED is characterized by the application of *edit operations* to the graph G until one obtains a graph that is isomorphic to H. Edit operations o_i consist of:

- insertion or deletion of an isolated, labeled node;
- insertion or deletion of a labeled edge;
- substitution of a node (label) by another one;
- substitution of an edge (label) by another one.

with a non-negative *edit cost* associated with each edit operation: $c_V : \Sigma_V \times \Sigma_V \to \mathbb{R}_{\geq 0}$ for node operations, $c_E : \Sigma_E \times \Sigma_E \to \mathbb{R}_{\geq 0}$ for edge operations.

An *edit path* $P = (G, o_1, G^1, \ldots, o_k, G^k)$, alternatively denoted as $P = (o_1, \ldots, o_k)$, encodes a sequence of graphs G^i, $(1 \leq i \leq k)$, where each G^i is obtained from G^{i-1} by applying the edit operation o_i. This sequence transforms the graph G into a graph G^k that is isomorphic to H. The cost of an edit path is equal to the sum of the costs of its edit operations. The *graph edit distance* $GED(G, H)$ is defined as the minimum cost $c(P)$ of an edit path P between G and H. Edit paths are in general not unique under this definition. Given two labeled graphs G and H, the *GED problem* is to find a minimum cost edit path.

In all the published algorithmic approaches, the GED problem is restricted to edit paths that are induced by complete node maps [5]. Such a *node map* is defined as a relation $\pi \subset V_{G+\varepsilon} \times V_{H+\varepsilon}$ that is an injective function when restricted to V_G and V_H, and every node $i \in V_G$ is mapped either to a node $k \in V_H$ or to the dummy node ε_{V_H} (deleted), and every node $k \in V_H$ is either the mapping of a node $i \in V_G$ or the dummy node ε_{V_G} (inserted). This restriction, which we also accept, is justified, since it has been shown that for metric cost functions, the node map induced GED is equivalent to the original GED defined above (see [25] and [6]). Moreover, in the case that the given cost function is not metric, it can be transformed into a metric cost function [5].

Given a GED problem instance $D = (G, H, c)$ with substitution, insertion, and deletion costs c for node and edge pairs, a complete node map $\pi \subset V_{G+\varepsilon} \times V_{H+\varepsilon}$, defines a unique mapping π' between the edges from $E_{G+\varepsilon}$ to $E_{H+\varepsilon}$. Then the (node map induced) GED can be derived as follows:

- If $\pi(i) = k$ for two nodes $i \in V_G$ and $k \in V_H$, then the costs are given by the node label substitution costs $c_{i,k}$.
- If node i is deleted (i.e, assigned to node ε_{V_H}), the costs are $c_{i,\varepsilon}$.
- If node k is inserted (i.e, assigned from node ε_{V_G}), the costs are $c_{\varepsilon,k}$.
- If $\pi'(\{i, j\}) = \{k, l\}$ for two edges $\{i, j\} \in E_G$ and $\{k, l\} \in E_H$, the costs are given by the edge label substitution costs $c_{ij,kl}$.
- If edge $\{i, j\} \in E_G$ is deleted, (i.e. $\pi'(\{i, j\}) = \varepsilon_{E_H}$), then the costs are given by the edge deletion costs $c_{ij,\varepsilon}$.
- If edge $\{k, l\} \in E_H$ is inserted, (i.e. $\pi'(\varepsilon_{E_G}) = \{k, l\}$), then the costs are given by the edge insertion costs $c_{\varepsilon,kl}$.

Hence, the considered GED problem in this work is to find a complete node map $\pi \subset V_{G+\varepsilon} \times V_{H+\varepsilon}$ that minimizes the sum of the above node and edge mapping costs.

2.1 Integer programming and valid inequalities

An Integer Linear Program (ILP) [10, 36]

$$\min\{c^T x \mid x \in \mathbb{Z}^n,\ Ax \geq b\} \tag{1}$$

consists of a system of linear inequalities and a linear objective function with a constraint matrix $A \in \mathbb{Q}^{m \times n}$, a cost vector $c \in \mathbb{Q}^n$, and a right-hand side vector $b \in \mathbb{Q}^m$. A variable vector $\bar{x} \in \mathbb{Z}^n$ satisfying all the constraints $a_i^T \bar{x} \geq b_i$ for all $i = 1, \ldots, m$ is called a *feasible solution*. The set of feasible solutions to (1) is denoted by $\mathcal{X}$. The inequality $\pi x \geq \pi_0$ is said to be *valid* for $\mathcal{X}$ if $\pi x \geq \pi_0$ for all $x \in \mathcal{X}$.

We obtain the natural *LP relaxation* of an ILP by substituting the requirement $x \in \mathbb{Z}^n$ by $x \in \mathbb{R}^n$. The subset $\mathcal{P} = \{x \in \mathbb{R}^n \mid Ax \geq b\}$ of $\mathbb{R}^n$ defines a *polyhedron*. In the case of minimization problems, such as the GED, the optimal value of an LP relaxation provides a lower bound to the optimal value of the corresponding integer formulation. In this paper, we restrict the variables x to be in $\{0, 1\}^n$.

Strength of LP relaxations. In this paper, we theoretically compare the strength of ILP models. Our comparisons are based on the natural LP relaxations which are the starting points of our practical computations. Let $v(\mathcal{P}_M(I))$ be the optimal value of the (natural) LP relaxation for a formulation M on instance I. We call an ILP formulation (model) M *weakly stronger* than a formulation M', denoted by $M \succeq M'$, if the optimal value of their natural LP relaxations fulfills $v(\mathcal{P}_M(I)) \geq v(\mathcal{P}_{M'}(I))$ for all instances I of the problem. We say that M is *strictly stronger* than M', denoted by $M \succ M'$, if M is weakly stronger than M', and there exists a problem instance I for which $v(\mathcal{P}_M(I)) > v(\mathcal{P}_{M'}(I))$, i.e., the optimal value of the LP relaxation of M leads to stronger (larger) lower bounds than that of M'. If M is weakly stronger than M', and M' is weakly stronger than M, we call them *equivalent*, and we denote this fact by $M \equiv M'$.

3 STATE-OF-THE-ART ILP FORMULATIONS FOR THE GED PROBLEM

Lerouge et al. [21] suggest two ILP-formulations for the GED problem and evaluate them in comparison with existing ILP formulations.

Their first formulation (F1) (see Figure 1) is a straightforward model using assignment variables to encode the mapping for the nodes and for the edges. It contains two types of decision variables: $x_{i,k} = 1$ indicates that node $i \in V_G$ is mapped to node $k \in V_H$, while $x_{i,\varepsilon} = 1$ and $x_{\varepsilon,k} = 1$ encode the deletion of node i and

$$\text{(F1)} \quad \min \sum_{i\in V_G}\sum_{k\in V_H} c_{i,k}x_{i,k} + \sum_{i\in V_G} c_{i,\varepsilon}x_{i,\varepsilon} + \sum_{k\in V_H} c_{\varepsilon,k}x_{\varepsilon,k} + \sum_{ij\in E_G}\sum_{kl\in E_H} c_{ij,kl}y_{ij,kl} + \sum_{ij\in E_G} c_{ij,\varepsilon}y_{ij,\varepsilon} + \sum_{kl\in E_H} c_{\varepsilon,kl}y_{\varepsilon,kl}$$

$$\text{s.t.} \quad x_{i,\varepsilon} + \sum_{k\in V_H} x_{i,k} = 1 \qquad \forall\, i \in V_G \tag{2a}$$

$$x_{\varepsilon,k} + \sum_{i\in V_G} x_{i,k} = 1 \qquad \forall\, k \in V_H \tag{2b}$$

$$y_{ij,\varepsilon} + \sum_{kl\in E_H} y_{ij,kl} = 1 \qquad \forall\, ij \in E_G \tag{2c}$$

$$y_{\varepsilon,kl} + \sum_{ij\in E_G} y_{ij,kl} = 1 \qquad \forall\, kl \in E_H \tag{2d}$$

$$y_{ij,kl} \le x_{i,k} + x_{j,k} \qquad \forall\, ij \in E_G,\ kl \in E_H \tag{2e}$$

$$y_{ij,kl} \le x_{i,l} + x_{j,l} \qquad \forall\, ij \in E_G,\ kl \in E_H \tag{2f}$$

$$x \in \{0,1\}^{xcard_{F1}} \tag{2g}$$

$$y \in \{0,1\}^{ycard_{F1}} \tag{2h}$$

Figure 1: Formulation (F1) proposed in [21].

the insertion of node k, respectively. Similarly, $y_{ij,kl} = 1, y_{ij,\varepsilon} = 1$ and $y_{\varepsilon,kl} = 1$ correspond to the mapping, deletion, and insertion of edges $\{i,j\} \in E_G$ and $\{k,l\} \in E_H$. The formulation involves $xcard_{F1} = |V_G| \cdot |V_H| + |V_G| + |V_H|$ node assignment variables and $ycard_{F1} = |E_G| \cdot |E_H| + |E_G| + |E_H|$ edge assignment variables. The objective function naturally reflects the total cost of the resulting node-map-based assignment.

The constraints (2a) ensure that every node in graph G is either mapped to a node in graph H or deleted. Similarly, constraints (2b) guarantee that every node in H is either the image of a node in G or inserted. Similarly, constraints (2c) and (2d) enforce the same for edges: each edge must either be matched with a corresponding edge in the other graph, deleted or inserted. The so-called topological constraints (2e) and (2f) link node and edge assignment variables. These constraints ensure that an edge $\{i,j\} \in E_G$ can only be mapped to an edge $\{k,l\} \in E_H$ if one of i or j is mapped to k, and the other to l. If this condition is not satisfied, the corresponding edge mapping variable $y_{ij,kl}$ must be zero.

Lerouge et al. [21] also suggest formulation (F2) which they derive from (F1). In a first step, they argue that the variables for deleting and inserting nodes and edges are not needed, since they are implicitly given by the assignment variables. Hence they substitute the equations (2a) – (2d) by inequalities, reducing the number of variables and constraints.

Variables $x_{i,k} = 1$ correspond to mapping node $i \in V_G$ to $k \in V_H$, $x_{i,k} = 0$ otherwise. Similarly, variables $y_{ij,kl} = 1$ correspond to mapping edge $\{i,j\} \in E_G$ to $\{k,l\} \in E_H$.

The cost function implicitly takes the deletion and insertion of nodes and edges into consideration. The constant term

$$K = \sum_{i\in V_G} c_{i,\varepsilon} + \sum_{k\in V_H} c_{\varepsilon,k} + \sum_{ij\in E_G} c_{ij,\varepsilon} + \sum_{kl\in E_H} c_{\varepsilon,kl}$$

$$\text{(F2)} \quad \min \sum_{i\in V_G}\sum_{k\in V_H} \bar{c}_{i,k} \cdot x_{i,k} + \sum_{ij\in E_G}\sum_{kl\in E_H} \bar{c}_{ij,kl} \cdot y_{ij,kl} + K$$

$$\text{s.t.} \quad \sum_{k\in V_H} x_{i,k} \le 1 \qquad \forall\, i \in V_G \tag{3a}$$

$$\sum_{i\in V_G} x_{i,k} \le 1 \qquad \forall\, k \in V_H \tag{3b}$$

$$\sum_{l\in\delta_H(k)} y_{ij,kl} \le x_{i,k} + x_{j,k} \qquad \forall\, k \in V_H,\ ij \in E_G \tag{3c}$$

$$x \in \{0,1\}^{|V_G|\cdot|V_H|} \tag{3d}$$

$$y \in \{0,1\}^{|E_G|\cdot|E_H|} \tag{3e}$$

Figure 2: Formulation (F2) proposed in [21].

adds the cost for deleting and subsequently inserting every node and edge to the objective function value. Substitution costs are defined as $\bar{c}_{i,k} = (c_{i,k} - c_{i,\varepsilon} - c_{\varepsilon,k})$ for all pairs of nodes $(i,k) \in V_G \times V_H$ and $\bar{c}_{ij,kl} = (c_{ij,kl} - c_{ij,\varepsilon} - c_{\varepsilon,kl})$ for all pairs of edges $(ij,kl) \in E_G \times E_H$. Thus, for any variable with $x_{i,k} = 1$ the objective function coefficient adds the cost of the corresponding mapping and cancels out the cost of the deletion of $i \in V_G$ and the insertion of $k \in V_H$, analogously for $y_{ij,kl} = 1$. We will call this formulation (F2-).

In order to reduce the number of constraints, Lerouge et al. [21] suggest to substitute the constraints (2e) and (2f) in (F2-) by the new topological constraints (3c). Moreover, they show that due to this substitution, constraints (2c) and (2d) are no longer necessary to be a valid formulation for the GED problem. This reduces the number of topological constraints from $2|E_G| \cdot |E_H|$ to $|V_H| \cdot |E_G|$. The model (F2) suggested by Lerouge et al. [21] is shown in Figure 2.

Lerouge et al. [21] show that their new type of constraints (3c) are valid for formulation (F1), which is equivalent to (F2-) in terms of the set of feasible solutions. In their computational experiments, which showed that (F2) dominates (F1) on the tested benchmark sets, they only incorporated them into formulation (F2). We will incorporate them also into (F1) leading to formulation (F1').

Lemma 3.1. *[Proposition 2 in [21]] Replacing inequalities (2e) and (2f) in (F1) with (3c) leaves the set of feasible solutions untouched. I.e. constraints (2c),(2d) plus (2e), (2f) are implied by inequality (3c).*

We will show in Section 6 that using the topological constraints (3c) strengthens the formulations in the sense that their LP relaxations lead to better bounds than that of (F1) and (F2-), respectively. Additionally, we will show that both formulations (F1) and (F2-) are equivalent with respect to their LP relaxation bounds.

4 STRENGTHENING STATE-OF-THE-ART MODELS

In the same spirit of the inequalities (3c) in formulation (F2) that link the y variables with the x variables, we introduce inequalities over a different combination of node and edge indices. In detail, inequalities (3c) are defined starting from the pair $(k \in V_H, ij \in E_G)$. Then, they involve variables $y_{ij,kl}$, where $l \in \delta_H(k)$ is a node in the neighborhood of k in H. Swapping the role of the graphs, one

can consider the pair $(i \in V_G, kl \in E_H)$ and variables $y_{ij,kl}$ where j is a node in the neighborhood of i in G, i.e., $j \in \delta_G(i)$. The new topological constraints have the form:

$$\sum_{j \in \delta_G(i)} y_{ij,kl} \leq x_{i,k} + x_{i,l} \quad \forall\, i \in V_G,\ kl \in E_H \tag{4}$$

Since the y variables in (F2) and (F1) induce a valid edge mapping, the left hand side of (4) is either 0 or 1. If it is equal to 1, then there exists an edge incident to $i \in V_G$, which is mapped to $\{k, l\} \in E_H$. In this case, node i needs to be assigned to either $k \in V_H$ or $l \in V_H$ showing validity of (4).

Lemma 4.1. *Inequalities (4) are valid for the set of integer feasible solutions of the ILP formulation (F2) as well as for (F1).*

Proof. We show that every integral feasible solution of (F2) and (F1), resp., satisfies the constraints (4). If the constraints (4) would be violated for (F1), then there would exist an $i' \in V_G$ and $\{k', l'\} \in E_H$ for which

$$\sum_{j \in \delta_G(i')} y_{i'j,k'l'} > x_{i',k'} + x_{i',l'}.$$

Since from constraints (2a) and (3a), resp., it follows that $x_{i',k'} + x_{i',l'} \leq 1$, we know that then there must exist at least two nodes $j', j'' \in V_G$ for which $y_{i',j',k'l'} = y_{i',j'',k'l'} = 1$. But this is in contradiction with (2d) in (F1). For (F2), Lerouge et al. [21] have shown that constraints (2d) are implied by (3b) and (3c). □

4.1 Formulations (F2+) and (F1+)

Our formulations (F2+) and (F1+), resp., are defined as the model (F2) and (F1'), resp., extended by the new class of constraints (4). Lemma 4.1 together with the fact that the new models extend (F2) and (F1), resp., which correctly encode the set of feasible solutions of the GED problem, leads to the correctness of both new models.

Proposition 4.2. *The formulations (F2+) and (F1+), resp., correctly model the GED problem. In particular, the models (F2+) and (F1+), resp., applied to a GED instance (G, H, c), provide the correct GED and the corresponding node and edge mappings.*

Both formulations (F2+) and (F1+) have the same number of variables as their basic models (F2) and (F1'), resp., and $|V_G| \cdot |E_H|$ additional constraints. Despite an increase in constraints, in Section 7 we will demonstrate that our approach significantly enhances the computational performance. Moreover, we will show in Section 6 that their LP relaxations lead to strictly stronger lower bounds.

5 A NEW ILP FORMULATION BASED ON EDGE ORIENTATIONS

Our new formulation is based on respecting newly introduced edge orientations of the given graphs. The x variables are identical to those of (F2) and induce a node map. Variable $x_{i,k} = 1$ if node $i \in V_G$ is mapped to node $k \in V_H$, for all $i \in V_G$ and $k \in V_H$. In G, we orient every undirected edge $\{i, j\}$ so that $i < j$, and denote the resulting directed graph by $\overrightarrow{G}$. Moreover, in H, we introduce two arcs (k, l) and (l, k) for each edge $\{k, l\} \in E_H$ with $k \neq l$, leading to the graph $\overleftrightarrow{H}$. We introduce z variables $z_{ij,kl}$ for the set of all arcs $(i, j) \in E_{\overrightarrow{G}}$ $(i < j)$ and all arcs $(k, l) \in E_{\overleftrightarrow{H}}$. A variable $z_{ij,kl}$ is set to 1 if arc $(i, j) \in E_{\overrightarrow{G}}$ gets mapped to arc $(k, l) \in E_{\overleftrightarrow{H}}$ and zero otherwise.

(FORI) $$\min \sum_{i \in V_G} \sum_{k \in V_H} \bar{c}_{i,k} \cdot x_{i,k} + \sum_{(i,j) \in E_{\overrightarrow{G}}} \sum_{(k,l) \in E_{\overleftrightarrow{H}}} \bar{c}_{ij,kl} \cdot z_{ij,kl} + K$$

$$\text{s.t.} \sum_{k \in V_H} x_{i,k} \leq 1 \quad \forall\, i \in V_G \tag{5a}$$

$$\sum_{i \in V_G} x_{i,k} \leq 1 \quad \forall\, k \in V_H \tag{5b}$$

$$\sum_{l \in \delta^+_{\overleftrightarrow{H}}(k)} z_{ij,kl} \leq x_{i,k} \quad \forall\, k \in V_H,\ (i,j) \in E_{\overrightarrow{G}} \tag{5c}$$

$$\sum_{l \in \delta^-_{\overleftrightarrow{H}}(k)} z_{ij,lk} \leq x_{j,k} \quad \forall\, k \in V_H,\ (i,j) \in E_{\overrightarrow{G}} \tag{5d}$$

$$\sum_{j \in \delta^+_{\overrightarrow{G}}(i)} z_{ij,kl} + \sum_{j \in \delta^-_{\overrightarrow{G}}(i)} z_{ji,lk} \leq x_{i,k} \quad \forall\, i \in V_G,\ (k,l) \in E_{\overleftrightarrow{H}} \tag{5e}$$

$$x \in \{0,1\}^{|V_G| \cdot |V_H|} \tag{5f}$$

$$z \in \{0,1\}^{|E_G| \cdot 2 |E_H|} \tag{5g}$$

Figure 3: Formulation (FORI).

Figure 4: GED(G,H) instance used in Lemmas 6.3–6.5.

Figure 5: Example of $\overrightarrow{G}$ (where a < b < c) and $\overleftrightarrow{H}$.

For the objective function, our new model (FORI) copies the costs from (F2), where the z variables $z_{ij,kl}$ and $z_{ij,lk}$ take over the costs of the y variables $y_{ij,kl}$ from (F2). The formulation (FORI) is shown in Figure 3.

Introducing variables that explicitly model directed edges enables us to enforce the relationships between corresponding nodes more precisely. Consider the topological constraints (3c) in (F2), which ensure that an edge $\{i, j\} \in E_G$ can be mapped to an edge $\{k, l\} \in E_H$ only if one of the nodes i or j is mapped to k, and the other to l. Since we do not know if i is mapped to k or to l, the right-hand side of (3c) contains the sum of both assignment variables, $x_{i,k} + x_{j,k}$. In contrast, our new formulation (FORI) provides directional information: we know whether the directed edge (i, j) with $i < j$ in

$\overrightarrow{G}$ is mapped to (k,l) or to (l,k) in $\overleftrightarrow{H}$, thus providing information which of the nodes $i \in V_G$ and $j \in V_G$ is mapped to node $k \in V_H$ (see, e.g., Fig. 5 with $i = a$, $j = b$, $k = 1$, and $l = 2$). This allows us to decompose the original constraint (3c) into two distinct types, each involving only a single assignment variable on the right-hand side (see constraints (5c) and (5d)). For the example shown in Fig. 4, we can consider a possible feasible solution which will map the edge (a,b) to the edge $(1,2)$. In formulation (F2+) this breaks down to $y_{ab,12} = 1$ and due to constraints (3c) we get $x_{a,1} + x_{b,1} \geq 1$. However, in (FORI) we get either $z_{ab,12} = 1$, which will lead to $x_{a,1} \geq 1$ or we get $z_{ab,21} = 1$, which will lead to $x_{b,1} \geq 1$ (see Figure 5). The following Lemma shows that at most one of both related z-variables will be 1.

Lemma 5.1. *In the model (FORI) the following holds: Given an edge $\{i,j\} \in E_G$ and an edge $\{k,l\} \in E_H$, either $\{z_{ij,kl}, z_{ij,lk}\}$ or $\{z_{ji,kl}, z_{ji,lk}\}$ are defined. Moreover, at most one of the variables in both sets can be 1.*

Proof. The first fact follows directly from the definition of our formulation. Consider now, w.l.o.g., the variables in $\{z_{ij,kl}, z_{ij,lk}\}$. Assume that $z_{ij,kl} = z_{ij,lk} = 1$. Then, constraint (5c) forces $x_{i,k}$ to 1, while constraint (5d) pushes $x_{j,k}$ up to 1, violating constraint (5b). □

Theorem 5.2. *The formulation (FORI) correctly models the GED problem. In particular, applied to a GED instance (G,H,c), it provides the correct GED and the corresponding node and edge mappings.*

Proof. We prove this by showing that for any feasible (FORI) solution we can construct a feasible (F2) solution with the same costs, and vice versa. We start with a given (F2) solution $(\hat{x},\hat{y})$ from which we will construct a (FORI) solution $(\hat{x},\hat{z})$. Obviously, the x variables satisfy constraints (5a) and (5b). For the construction of the z variables, consider a pair of edges $\{i,j\} \in E_G$ (we can assume that $i < j$) and $\{k,l\} \in E_H$. If $\hat{y}_{ij,kl} = 0$, then either we have $\hat{x}_{i,k} = \hat{x}_{j,k} = 0$, $\hat{x}_{i,l} = \hat{x}_{j,l} = 0$, or the costs $c_{ij,kl} - c_{ij,\varepsilon} - c_{\varepsilon,kl} \geq 0$. In these cases, we can safely set $\hat{z}_{ij,kl} = 0$ without violating a constraint of (FORI). Otherwise, $\hat{y}_{ij,kl} = 1$ and due to constraints (3c) we have that either $\hat{x}_{i,k} = \hat{x}_{j,l} = 1$ or $\hat{x}_{i,l} = \hat{x}_{j,k} = 1$. In the case $\hat{x}_{i,k} = \hat{x}_{j,l} = 1$ we set $\hat{z}_{ij,kl} = 1$ and $\hat{z}_{ij,lk} = 0$. Otherwise ($\hat{x}_{i,l} = \hat{x}_{j,k} = 1$), we set $\hat{z}_{ij,lk} = 1$ and $\hat{z}_{ij,kl} = 0$. In all cases, constraints (5c), (5d), and (5e) are satisfied, since we set at most one z-variable on the left hand side of the constraint to 1, and we do this only if the right hand side is also 1. Since the cost function of (F2) and (FORI) is the same, both solutions $(\hat{x},\hat{z})$ and $(\hat{x},\hat{y})$ have the same cost.

Next we start with a feasible solution $(\hat{x},\hat{z})$ of (FORI) and construct one for (F2). Let $(\hat{x},\hat{y})$ such that $\hat{y}_{ij,kl} = \hat{z}_{ij,kl} + \hat{z}_{ij,lk}$ for $\{i,j\} \in E_G$, $i < j$, and $\{k,l\} \in E_H$. Obviously, the assignment constraints (3a) and (3b) of the x variables in (F2) are satisfied. Due to Lemma 5.1, we have that $y_{ij,kl} \in \{0,1\}$. Consider the topological constraints for a fixed $k' \in V_H$ and $(i',j') \in E_{\overrightarrow{G}}$ with $i' < j'$. Summing one inequality from each of the constraints (5c) and (5d) we get:

$$\sum_{l \in \delta_H(k')} \hat{y}_{i'j',k'l} =$$
$$= \sum_{l \in \delta^+_{\overleftrightarrow{H}}(k')} \hat{z}_{i'j',k'l} + \sum_{l \in \delta^-_{\overleftrightarrow{H}}(k')} \hat{z}_{i'j',lk'} \leq \hat{x}_{i',k'} + \hat{x}_{j',k'}.$$

This shows that also the topological constraints (3c) are satisfied. Also in this case, the cost functions of both solutions $(\hat{x},\hat{y})$ and $(\hat{x},\hat{z})$ have the same cost. □

The new model (FORI) uses $|V_G| \cdot |V_H| + 2|E_G| \cdot |E_H|$ variables and has $|V_G| + |V_H| + 2|V_H| \cdot |E_G| + 2|V_G| \cdot |E_H|$ linear constraints. Although the orientation-based formulation enlarges the variable space w.r.t. that of (F1+) and (F2+), resp., it turns out that (FORI) outperforms the state-of-the-art formulations theoretically (see Lemma 6.5) and practically (see Section 7).

6 THEORETICAL COMPARISON OF ILP MODELS

In this section, we will theoretically analyze the strength of the state-of-the-art and new ILP models suggested in this work, namely (F1), (F2-), (F2), (F2+), (F1+), and (FORI). We will show that

$$\text{(FORI)} \succ \text{(F1+)} \equiv \text{(F2+)} \succ \text{(F2)} \succ \text{(F2-)} \equiv \text{(F1)}.$$

The strength of a formulation is important for practical problem solving, since stronger LP relaxations lead to a smaller relaxation gap, i.e., the gap between the optimal value $v(M(I))$ and the optimal value of the linear relaxation $v(\mathcal{P}_M(I))$ for a formulation M applied to a GED instance $I = (G,H,c)$. Our theoretical findings have been supported by our computational experiments (see Section 7).

Lemma 6.1. *The formulation (F1) is equivalent to formulation (F2-).*

Proof. Formulation (F1) has one equation of type (2a) for each $i \in V_G$. In each of those equations we have the variable $x_{i,\varepsilon}$ involved. Hence, we can project out the variables $x_{i,\varepsilon}$ for all $i \in V_G$, by removing them from the formulation, and obtain inequalities (3a). Similarly, we can project out the variables $x_{\varepsilon,k}$ for all $k \in V_H$. The equations of type (2c) for each $\{i,j\} \in E_G$ contain the variables $y_{ij,\varepsilon}$. Hence, we can project out the variables $y_{ij,\varepsilon}$ and get the corresponding inequalities. Similarly, we can project out the variables $y_{\varepsilon,kl}$ for all $\{k,l\} \in E_H$. It follows that the value of the LP relaxations of both formulations is the same. □

Lemma 6.2. *The formulation (F2) is strictly stronger than (F1).*

Proof. Obviously, every solution of $\mathcal{P}_{\text{F2}}$ is a feasible solution of $\mathcal{P}_{\text{F1}}$ by adding the corresponding deletion and insertion variables. We show that the opposite is not true. We consider a fractional solution of (F1) with $x_{i,k} = x_{j,l'} = 0.3$, $x_{i,l} = x_{j,l} = 0.2$ and $y_{ij,kl} = y_{ij,kl'} = 0.3$ for the edges $\{i,j\} \in E_G$, $\{k,l\} \in E_H$, and $\{k,l'\} \in E_H$. All remaining variables $x_{i,k}$ and $y_{ij,kl}$ are set to 0. It is possible to augment this solution to a feasible LP solution (F1) by setting the remaining deletion, insertion and all the edge variables to 1 and 0, respectively. This solution satisfies constraints (2a)-(2f) of (F1).

However, this solution violates constraints (3c) of (F2), since we have

$$0.6 = \sum_{kl \in E_H} y_{ij,kl} \geq y_{ij,kl} + y_{ij,kl'} > x_{i,k} + x_{j,k} = 0.3.$$

If the variables $y_{ij,kl}$ and $y_{ij,kl'}$ have cost $\bar{c}_{ij,kl} = \bar{c}_{ij,kl'} = -1$, and all other variables have cost 0, the LP relaxation bound of (F2) is larger than that of (F1). □

Lemma 6.3. *The formulation (F2+) is strictly stronger than (F2).*

Proof. We show that the related polytopes of the LP relaxations satisfy $\mathcal{P}_{\text{F2+}} \subseteq \mathcal{P}_{\text{F2}}$ for all problem instances, i.e.,

$$\min\{cz \mid z \in \mathcal{P}_{\text{F2}}\} \leq \min\{cz \mid z \in \mathcal{P}_{\text{F2+}}\},$$

and we provide an instance with a cost function for which optimizing over $\mathcal{P}_{\text{F2+}}$ leads to stronger lower bounds as over $\mathcal{P}_{\text{F2}}$.

Obviously, every feasible solution of $\mathcal{P}_{\text{F2+}}$ is also feasible for $\mathcal{P}_{\text{F2}}$. Consider the graphs shown in Figure 4 (represented without labels for the sake of simplicity) for which we want to compute the GED. Let G be the graph on the left, and H be the graph on the right. Assume that the x variables have cost 0, while the y variables have cost $\bar{c} = -1$. The optimal solution (x^*, y^*) having value $-3.0 + K$ to the linear relaxation of (F2) is $y^*_{ab,23} = 1$, $x^*_{a2} = x^*_{a3} = x^*_{b2} = x^*_{b3} = x^*_{c1} = x^*_{c4} = y^*_{ac,12} = y^*_{ac,34} = y^*_{bc,12} = y^*_{bc,34} = 0.5$ (omitted variables are set to 0). The class of valid inequalities (4) introduced for (F2+) is able to cut off the point (x^*, y^*), e.g., by the inequality $y_{ac,12} + y_{bc,12} - x_{c1} - x_{c2} \leq 0$ which becomes $0.5 > 0$ when evaluated on (x^*, y^*). Indeed, the optimal solution to the linear relaxation of (F2+) is equal to $-2.5 + K$, which is strictly larger than that of (F2). □

Lemma 6.4. *The formulation (F1+) is equivalent to (F2+).*

Proof. In Lemma 6.1 we have shown that formulations (F1+) and (F2+) are equivalent. Adding the same number and type of constraints to both formulations, leads to the same LP relaxations. □

We show that the optimum value of the LP relaxation of our new formulation (FORI) is strictly greater than that of (F2+). The reason lies in the fact that the relationship between the node assignment variables and the edge assignment variables can be represented more accurately in (FORI).

Lemma 6.5. *The formulation (FORI) is* strictly stronger *than (F2+).*

Proof. Let $\mathcal{P}_{\text{FORI}}$ (and $\mathcal{P}_{\text{F2+}}$, resp.) denote the polytope of the LP relaxation of (FORI) (and (F2+), resp.) and let $\nu(\mathcal{P}_{\text{FORI}}(I))$ (resp. $\nu(\mathcal{P}_{\text{F2+}}(I))$ be the optimal LP value for instance I. We show that by using the (natural) projection

$$proj_y : y_{ij,kl} = z_{ij,kl} + z_{ij,lk},$$

we have

$$proj_y(\mathcal{P}_{\text{FORI}}) \subseteq \mathcal{P}_{\text{F2+}}.$$

Moreover, there is an instance (G, H, c) and a cost function $\bar{c}$ for which optimizing over $\mathcal{P}_{\text{FORI}}$ leads to stronger lower bounds as over $\mathcal{P}_{\text{F2+}}$, i.e.,

$$\min\{\bar{c}z \mid z \in \mathcal{P}_{\text{FORI}}\} > \min\{cy \mid y \in \mathcal{P}_{\text{F2+}}\}.$$

Consider the topological constraints for a fixed $k' \in V_H$ and $(i', j') \in E_{\overrightarrow{G}}$. In Theorem 5.2 we have shown that the constraints (5c) and (5d), and using the equation $y_{ij,kl} = z_{ij,kl} + z_{ij,lk}$ of the projection $proj_y$, we get a constraint of type (3c). Next we consider a fixed $i' \in V_G$ and an edge $\{k', l'\} \in E_H$. By adding the constraints (5e) for $i' \in V_G$ and $(k', l') \in E_{\overleftrightarrow{H}}$ to those for $i' \in V_G$ and $(l', k') \in E_{\overleftrightarrow{H}}$, we get:

$$\sum_{j \in \delta^+_{\overrightarrow{G}}(i')} z_{i'j,k'l'} + \sum_{j \in \delta^-_{\overrightarrow{G}}(i')} z_{ji',l'k'} + \sum_{j \in \delta^+_{\overrightarrow{G}}(i')} z_{i'j,l'k'} + \sum_{j \in \delta^-_{\overrightarrow{G}}(i')} z_{ji',k'l'} \leq x_{i',k'} + x_{i',l'}$$

Using the projection $y_{ij,kl} = z_{ij,kl} + z_{ij,lk}$ results in constraints (4). Hence, all feasible points of the LP relaxation $\mathcal{P}_{\text{FORI}}$ are also feasible for $\mathcal{P}_{\text{F2+}}$ leading to $\nu(\mathcal{P}_{\text{FORI}}(I)) \geq \nu(\mathcal{P}_{\text{F2+}}(I))$.

Moreover, we again consider the instance in Figure 4, and assume that the x variables have cost 0, and the y and z variables have cost -1. Then the optimal solution obtained by optimizing over $\mathcal{P}_{\text{F2+}}$ is equal to $-2.50 + K$. On the other hand, optimizing the same instance over $\mathcal{P}_{\text{FORI}}$ leads to an optimal solution of value $-2.25 + K$. □

7 EXPERIMENTAL EVALUATION

A comparison between the models (F2) and (F1) has been provided by Lerouge et al. in [21]. The main finding is that formulation (F2) performs best for graphs with more than 10 nodes. Therefore, we decided to concentrate our practical evaluation on the most promising models (F2), (F2+), (F1+), and (FORI). In both the evaluation by Lerouge et al. [21] and the experimental study conducted by Blumenthal et al. [5], the formulation (F2) has been compared also with A^*-search based approaches such as the one by Riesen et al. [32], CSI-GED [15], and DF-GED [2]. Both studies show that these algorithms become impractical when the graph size reaches 20 nodes. To the best of our knowledge, among the most recent developments on exact algorithms for GED computation–including the beam-stack search algorithm by Chen et al. [9] and Kim's approach based on isomorphic vertices [20]–only two algorithms, namely A^{*+}-LSa and A^*-BMao proposed by Chang et al. [8], are capable of handling graphs with more than 50 nodes in their experimental evaluation. For this reason, we include both algorithms in our comparison with the FORI formulation.

We evaluate the presented approaches with respect to the following research questions:

- **Q1**: Will the theoretical results about the strengths of the ILP models be reflected in practice? In particular:
  - Will the new models (F2+) and (F1+) be able to solve more instances within a fixed time limit than (F2)?
  - Will (FORI) be able to solve significantly more instances than models (F2), (F1+), and (F2+), respectively?
- **Q2**: How does our new model (FORI) compare to A^{*+}-LSa and A^*-BMao?
- **Q3**: How efficient is our model (FORI) compared to the previous models in computing the GED for a small graph pattern and a very large graph? What are the sizes that can be solved routinely?

7.1 Experimental Setup

Problem instances. In order to answer question (Q1), we conduct tests on the datasets PROTEIN, AIDS, and MUTAGENICITY, taken from the widely used IAM graph database repository [31]. In particular, we use the instances in the test folder *VLDBJ2020* from the *GEDlib* library of Blumenthal et al. [4] in which each dataset is subdivided into different bins, each one containing 10 graphs. Bins are named DATASET-P-Q, where DATASET is the name of the dataset and P, Q are two integers that denote the range of nodes in the graphs within the bin. For example, PROT-21-30 contains 10 graphs from the PROTEIN dataset having between 21 and 30 nodes. Table 1 provides a summary of the graph characteristics for each dataset.

Table 1: Datasets for Question Q1 (labeled graphs).

	Dataset	min $\|V_G\|$	max $\|V_G\|$	min $\|E_G\|$	max $\|E_G\|$
PROTEIN	PROT-21-30	22	30	40	60
	PROT-31-40	32	39	53	91
	PROT-41-50	41	50	74	98
	PROT-51-60	51	60	95	116
AIDS	AIDS-21-30	22	30	23	30
	AIDS-31-40	32	39	33	42
	AIDS-41-50	43	50	47	55
	AIDS-51-60	52	59	55	65
	AIDS-61-70	62	69	68	75
	AIDS-71-80	73	80	79	96
MUTAGENICITY	MUTA-21-30	22	30	24	32
	MUTA-31-40	31	40	34	44
	MUTA-41-50	42	49	44	51
	MUTA-51-60	51	60	52	63
	MUTA-61-70	62	70	61	74
	MUTA-71-80	72	79	74	81
	MUTA-81-90	82	89	86	91
	MUTA-91-100	91	97	94	105

Graphs contained in AIDS and MUTAGENICITY represent molecular compounds. The nodes of the graphs contained in AIDS and MUTAGENICITY are labeled with a chemical symbol chosen out of a set of 13 different symbols, and their edges are labeled with a valence (either 1, 2, or 3). Node edit costs are in the range $[0, 5.5]$, while edge edit costs are in the range $[0, 1.65]$. Graphs contained in PROTEIN represent proteins which are annotated with their EC classes [34]. Nodes are labeled with tuples (t, s), where t is the node's type (helix, sheet, or loop) and s is its amino acid sequence. The dataset under consideration comprises 905 unique sequences. Nodes are connected via structural or sequential edges or both, i. e., edges $\{u_i, u_j\}$ are labeled with tuples (t_1, t_2), where t_1 is the type of the first edge connecting u_i and u_j and t_2 is the type of the second edge connecting u_i and u_j (possibly `null`). We count a total of 5 distinct edge types. Node and edge edit costs are instance dependent. For a complete description of the datasets and how edit costs are computed, the reader is referred to the detailed explanation in [4]. Note, however, that the cost function satisfies the triangle inequality for each instance. For each dataset bin of Table 1, GED instances are obtained by taking as inputs unordered pairs of graphs, for a total of 45 instances from each bin.

To address (Q2), we compare (FORI) with the A^{*+}-LSa and A^*-BMao algorithms proposed by Chang et al. [8]. Since both A^{*+}-LSa and A^*-BMao are designed to operate under unit edit costs only, we restrict our evaluation to this setting. As a consequence, we use the AIDS and MUTAGENICITY datasets for the comparison, as their original cost function can be naturally translated into the unit cost model.

In order to answer (Q3), we conduct tests on the famous IMDB, CORA, and PUBMED datasets, widely used as benchmark datasets in machine learning [3, 27, 28, 40]. Each graph in IMDB is an unlabeled ego-network, where each node denotes a film actor/actress and each edge denotes a co-star relation. We take IMDB graphs from the work of Piao et al. [27]. Specifically, a GED instance (G, H, c) for IMDB tests is obtained by setting G as the largest graph in the dataset, and H as a so-called *pattern* or *query* graph selected from four different bins. Each bin consists of six graphs, and graphs in the same bin have the same number of nodes. Constructed bins are IMDB-10, IMDB-20, IMDB-30, IMDB-43, containing graphs of 10, 20, 30, and 43 nodes, respectively. CORA and PUBMED are citation networks. For both datasets, a GED instance (G, H, c) is obtained by selecting G as the entire network, while H consists of query graphs sampled as ego-networks centered on a randomly chosen node and constructed using BFS up to a 3-hop distance. Table 2 summarizes graph statistics from these datasets. All such instances consist of unlabeled graphs. Node and edge edit operations have uniform cost equal to 1.

Table 2: Datasets for Question Q3 (unlabeled graphs). Subscript Q denotes a query graph.

	Dataset	$\|V_G\|$	$\|E_G\|$	avg / max $\|V_Q\|$	avg / max $\|E_Q\|$
IMDB	IMDB-10	89	1467	10 / 10	33 / 45
	IMDB-20	89	1467	20 / 20	113 / 190
	IMDB-30	89	1467	30 / 30	279 / 435
	IMDB-43	89	1467	43 / 43	435 / 719
	CORA	2,708	5,278	15 / 28	16 / 38
	PUBMED	19,717	44,327	17 / 29	16 / 29

Computational setting. The experiments were run on a workstation with AMD EPYC 7282 32-core 3.2GHz CPU, 504GB RAM, Ubuntu 20.04.6. All algorithms are implemented in C++ and compiled using GCC 10.5.0. We use Gurobi 12.0.0 [16] to solve the ILPs and use 5 different random seeds to mitigate the impact of performance variability [24]. Then, we take as instance runtime the shifted geometric mean[2] ($s = 1$) of the time spent by Gurobi over each of the 5 different seeds. The Gurobi solution time limit has been set to 600 seconds for all instances. A GED instance is declared solved by a formulation ($F*$) if at least 1 out of the 5 different runs is solved to optimality within the time limit.

For both A^{*+}-LSa and A^*-BMao, we used the source code provided by the authors, written in C++. As parameter configuration, we selected *pair* as running-mode, *astar* as search paradigm, and *LSa* and *BMao*, respectively, as lower bound methods. A GED instance is considered solved by these algorithms if a provably optimal solution is found within the time limit of 600 seconds.

Used datasets, implementation, and optimal GED solutions are provided in the URL on the first page of the paper.

[2]The shifted geometric mean, for $t_1, \ldots, t_n$, is $[\prod_{i=1}^{n}(t_i + s)]^{\frac{1}{n}} - s$, where s is the shift term.

Table 3: Question Q1: results on PROTEIN dataset.

MODEL	21-30	31-40	41-50	51-60
		avg. runtime in sec.		
(F2)	35.74	198.97	322.12	263.55
(F1+)	1.76	27.47	102.86	120.96
(F2+)	1.15	23.49	88.29	116.70
(FORI)	**0.31**	**8.29**	**8.62**	**59.81**
		timeouts		
(F2)	0	14 (31.1%)	18 (40.0%)	19 (42.2%)
(F1+)	0	**0**	3 (6.6%)	8 (17.7%)
(F2+)	0	1 (2.2%)	4 (8.8%)	8 (17.7%)
(FORI)	0	**0**	**0**	**1 (2.2%)**
		avg. / max gap in %		
(F2)	0 / 0	0.3 / 1.8	0.3 / 1.4	0.4 / 1.9
(F1+)	0 / 0	**0 / 0**	<0.1 / 0.3	0.1 / 0.5
(F2+)	0 / 0	< 0.1/0.2	0.01 / 0.3	<0.1 / 0.4
(FORI)	0 / 0	**0 / 0**	**0 / 0**	**< 0.01 / 0.1**

Table 4: Question Q1: results on AIDS dataset.

MODEL	21-30	31-40	41-50	51-60	61-70	71-80
			avg. runtime in sec.			
(F2)	24.40	128.09	493.27	448.06	444.37	442.77
(F1+)	0.91	2.31	37.01	124.27	52.40	277.47
(F2+)	0.58	1.28	25.23	95.11	31.96	230.41
(FORI)	**0.20**	**0.42**	**5.42**	**14.73**	**7.72**	**99.28**
			timeouts			
(F2)	0	2 (4.4%)	34 (75.5%)	31 (68.8%)	29 (64.4%)	33 (73.3%)
(F1+)	0	**0**	**0**	3 (6.6%)	**0**	10 (22.2%)
(F2+)	0	**0**	**0**	2 (4.4%)	**0**	8 (17.7%)
(FORI)	0	**0**	**0**	**0**	**0**	**3 (6.6%)**
			avg. / max gap in %			
(F2)	0 / 0	0.2 / 5.6	10.4 / 41.7	8.4 / 24.5	9.0 / 21.8	10.5 / 23.4
(F1+)	0 / 0	**0 / 0**	**0 / 0**	0.1 / 3.4	**0 / 0**	0.5 / 5.2
(F2+)	0 / 0	**0 / 0**	**0 / 0**	0.1 / 2.2	**0 / 0**	0.4 / 4.3
(FORI)	0 / 0	**0 / 0**	**0 / 0**	**0 / 0**	**0 / 0**	**0.1 / 2.8**

7.2 Evaluation Results

Answering (Q1). We experimentally evaluate the strengths of our new ILP formulations (F1+), (F2+), and (FORI) against the state-of-the-art (F2) formulation on the AIDS, MUTAGENICITY, and PROTEIN datasets. In these experiments, Gurobi was run using 8 threads with the default settings. An upshot of the results is provided in Tables 3–5 and in Figures 7–8. Tables 3–5 report, for each tested formulation and for each dataset bin, (i) the average runtime in seconds; (ii) the number of timeouts, i.e., the number of instances not solved by Gurobi in the given time limit; (iii) the average and the maximum gap reached by Gurobi at the end of the computation[3]. Bold font highlights the best values over all formulations.

Surprisingly, the (F2) formulation addresses some large instances that were previously considered unsolvable. This enhancement can be attributed to the advancements in the Gurobi solver over time. Nonetheless, performance statistics for the (F2) model indicate that it is still impractical and unsatisfactory for several cases in the benchmark set, as detailed below for each dataset.

PROTEIN dataset: Table 3 shows that (F2) struggles with larger graphs, requiring over 3 minutes on average and failing to solve 31.1% and 42.2% of instances in the PROT-31-40 and PROT-51-60 bins, respectively. In contrast, (F1+) solves all instances up to 40 nodes and maintains sub-2-minute runtimes even for the largest cases, leaving significantly smaller gaps in unsolved bins. (F2+) further reduces times and gaps across all bins, though it solves one fewer instance than (F1+) in the PROT-31-40 and PROT-41-50 bins, still maintaining high solution quality. The results for the (FORI) formulation are impressive: it solves all instances in the PROT-41-50 bin with an average runtime of just 8.62 seconds. In the PROT-51-60 bin, (FORI) solves all but one instance, achieving the lowest gap values among all formulations within 600 seconds of computation.

AIDS dataset: Table 4 demonstrates the limitations of (F2) on the AIDS dataset, failing over 60% of instances in bins starting from AIDS-41-50 with persistently high gaps. (F1+) and (F2+) outperform it significantly—solving all instances up to AIDS-61-70 with drastically reduced runtimes (e.g., 10× faster for AIDS-21-30 and AIDS-41-50). Notably, (F2+) dominates (F1+), solving more instances and lowering both time and gap values. On the other hand, the practical effectiveness of the (FORI) formulation also stands out on this dataset. All but three instances (from the AIDS-71-80 bin) are solved to proven optimality, with a maximum average runtime of 99 seconds on the last bin. As expected, (FORI) proves to be more effective than (F1+) and (F2+) across all considered measures, achieving a significantly lower average runtime, smaller gaps statistics, and closing all but one unsolved instance.

MUTAGENICITY dataset: The (FORI) formulation outperforms all other models also on the MUTAGENICITY dataset, as shown by the statistics reported in Table 5. In particular, (FORI) is able to close all the instances within ~25 seconds, even on the largest bin MUTA-91-100. Compared to our strengthened formulation (F2+), the achieved speed-ups are at least an order of magnitude, and reach up to 73x against (F2) on MUTA-21-30. On the other hand, the inadequacy of (F2) is further highlighted in this dataset: it fails to close all the instances starting from MUTA-31-40, and leaves most of them unsolved in the MUTA-61-70 to MUTA-91-100 bins. Moreover, gaps after 600 seconds of computation remain very high, from a maximum of 14.7% in MUTA-31-40 bin up to 44.7% maximum gap on MUTA-91-100 bin. Concerning our strengthened formulations (F1+) and (F2+), the latter confirms to be better than the former over all measured statistics. In particular, from MUTA-61-70 to MUTA-81-90, (F2+) leaves unsolved only half as many instances as (F1+). This is also reflected in the average runtime and gap statistics, as those of (F2+) are consistently smaller than those of (F1+).

(Q1) final remarks: The experimental analysis conducted to answer (Q1) suggests that the (F2) formulation is not suitable when the input graphs exceed 30 or 40 nodes. The strengthened formulations (F1+) and (F2+) successfully solve most of the instances that are not solved by (F2) in 600 seconds. However, (F1+) and (F2+) leave several instances unsolved when the size of the graphs reaches 60 nodes, particularly in the PROTEIN and MUTAGENICITY datasets.

Ultimately, (FORI) demonstrates a significant performance improvement over (F1+) and (F2+): (FORI) is able to solve all but four instances within 600 seconds, while maintaining an average runtime of under two minutes across all datasets. We also mention that (FORI) effectively solves *all* the instances examined in [4] by allowing Gurobi to run for up to ~ 9 hours, thereby certifying the optimality for the remaining four unsolved instances. This is one of the most important aspects of our contribution. Indeed, Blumenthal et al. [4] have evaluated various GED approximation techniques on the same datasets considered in our study. The results indicate that the best of all the tested approximation methods had gaps between the largest lower bound and the smallest upper bound of 3.6% on

[3]The percentage gap is defined as (best objective - best bound)/best objective · 100, where best objective is the value of the best feasible solution found at time limit and best bound is the smallest bound among unexplored subproblems in the branch-and-bound enumeration tree.

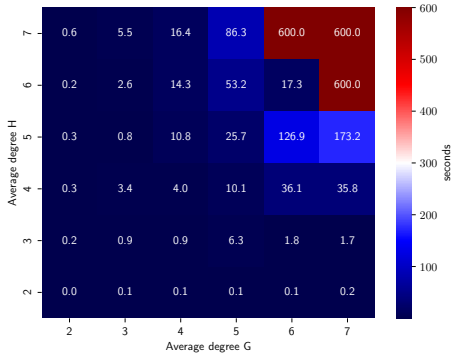


Figure 6: Heatmap of average degree of graphs G and H with runtime in seconds as color.

Table 5: Question Q1: results on MUTAGENICITY dataset.

MODEL	21-30	31-40	41-50	51-60	61-70	71-80	81-90	91-100
				avg. runtime in sec.				
(F2)	14.62	103.05	254.77	492.80	551.61	599.96	587.13	579.81
(F1+)	1.89	3.89	23.56	103.61	198.69	329.06	291.93	349.28
(F2+)	1.25	2.50	18.82	76.92	163.01	259.99	254.63	262.17
(FORI)	**0.20**	**0.41**	**1.43**	**2.06**	**6.06**	**8.04**	**8.45**	**25.64**
				timeouts				
(F2)	0	1 (2.2%)	9 (20.0%)	29 (64.4%)	39 (86.6%)	44 (97.7%)	43 (95.5%)	43 (95.5%)
(F1+)	0	**0**	**0**	3 (6.6%)	8 (17.7%)	16 (35.5%)	16 (35.5%)	13 (28.8%)
(F2+)	0	**0**	**0**	3 (6.6%)	5 (11.1%)	8 (17.7%)	9 (20.0%)	10 (22.2%)
(FORI)	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
				avg. / max gap in %				
(F2)	**0 / 0**	14.7 / 14.7	3.0 / 22.7	6.2 / 18.9	10.7 / 35.2	13.9 / 36.5	14.4 / 40.8	21.3 / 44.7
(F1+)	**0 / 0**	**0 / 0**	**0 / 0**	0.1 / 2.9	0.3 / 3.0	0.6 / 3.1	0.6 / 4.2	0.7 / 6.1
(F2+)	**0 / 0**	**0 / 0**	**0 / 0**	0.1 / 2.0	0.2 / 2.0	0.3 / 2.5	0.4 / 4.2	0.6 / 6.1
(FORI)	**0 / 0**	**0 / 0**	**0 / 0**	**0 / 0**	**0 / 0**	**0 / 0**	**0 / 0**	**0 / 0**

the AIDS dataset, 4.2% on the Mutagenicity dataset, and 1.6% on the Protein dataset. The best lower and upper bounds were determined within a time span of a few seconds to tenths of a second, depending on the considered dataset. While more recent heuristics may have reduced gaps and / or running time, and occasionally achieved optimal solutions, a fundamental limitation of heuristic methods is their inability to guarantee optimality, which is of interest for practical applications that rely on optimal GED values.

Figure 7 presents scatter plots comparing the runtime of (F2) (x-axis) and log-scaled (FORI) (y-axis) across datasets. Yellow dots indicate instances solved by both formulations, while blue dots mark those solved only by (FORI). Nearly all yellow and blue dots lie below the 10x speed-up curve, showing (FORI) consistently outperforms (F2), often solving previously unsolved instances in under 60 seconds.

Figure 8 compares (F2+) and (FORI) runtimes (x- and y-axes, respectively), with colors and the 5x speed-up line consistent with earlier plots. In Figure 8a, almost all yellow points fall below the 5x line, confirming (FORI)'s consistent outperformance on the PROTEIN dataset. Blue points, representing cases solved only by (FORI), show speed-ups from 1x to 12x. Figure 8b shows a similar blue-point distribution, highlighting (FORI)'s ability to handle harder instances on the AIDS dataset. Yellow points are split around the 5x line, reflecting trends seen in Table 4. In Figure 8c for the MUTAGENICITY graphs, most yellow dots cluster below the 5x line, near the x-axis, while all blue points lie well below it— i.e. (FORI) allows to obtain a speed-up of at least 6x over (F2+) on unsolved instances.

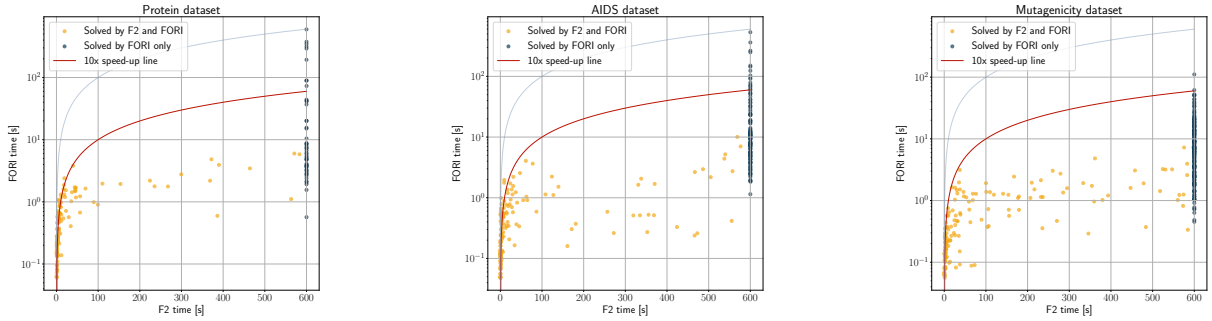
Finally, we remark that the runtime of (FORI) increases with the size of the input graphs. This is expected since the size of the formulation increases as a function of the nodes and the edges of the graphs. A closely related parameter that condition the computational time of our new formulation appears to be the graph density. Indeed, PROTEIN and AIDS datasets, in which (FORI) reaches the time limit in some instances, are denser than the MUTAGENICITY ones, where (FORI) perform extremely well (cf. the values in Table 1 to have a glimpse on graph densities). To further highlight this behaviour, we have run additional experiments with Erdős–Rényi graphs. We have created a series of graphs of sizes 30 nodes (G) and 10 nodes (query graphs H) each with increasing average degree from 2 to 7. A heatmap of the runtime is displayed in Figure 6. Again, the runtime increases with the average degree of the graphs.

Answering (Q2). The comparison between A^{*+}-LSa, A^*-BMao, and (FORI) shows a clear picture of performance differences, in line with known comparison between integer programming based methods and A^* search algorithms [5, 21]. Tables 6–7 report the measured statistics on the AIDS and MUTAGENICITY datasets, respectively. As in Q1, we report the average runtime in seconds and the number of timeouts, i.e., the number of instances not solved to optimality within the time limit of 10 minutes. Finally, we report the average memory required by each method instead of the gap statistics, which could not be computed on A^*-based approaches.

Overall, A^{*+}-LSa exhibits significant performance degradation as instance size grows, with runtime exceeding 586 seconds on AIDS-41-50 and consistent 10-minute timeouts from MUTA-41-50 onward. Timeout rates rise from 30% on small AIDS instances to 97.7% on AIDS-41-50, and from 60% to 100% on MUTAGENICITY starting at MUTA-21-30. In contrast, A^*-BMao shows improved performance on the first two bins of each dataset. However, as the graph size reaches $\sim$ 50 nodes, A^*-BMao behavior matches that of A^{*+}-LSa, leaving most of the tested instances unsolved (e.g., from MUTA-41-50 to MUTA-91-100, no instance is solved within the time limit).

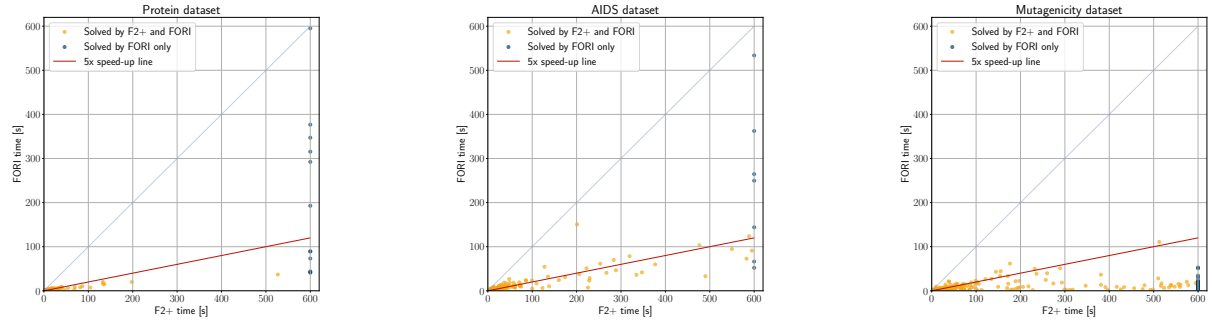
On the other hand, (FORI) scales efficiently with instance size, showing low runtime and timeout rates. On the AIDS dataset, it keeps average runtime below 12 seconds—except for AIDS-71-80, where it rises to 138.91 due to four unsolved cases. Even so, it solves 41 out of 45 instances, outperforming A^{*+}-LSa and A^*-BMao, which solved only 9 on that bin. Across other AIDS bins, (FORI) solves all instances. On MUTAGENICITY, it closes every instance in under 31.77 seconds, including on the largest bin MUTA-91-100, where A^*-based methods fail.

Memory usage is another area where (FORI) stands out. First of all, while A^{*+}-LSa consumes tens of gigabytes—up to 50GB on MUTA-51-60 —A^*-BMao is significantly more efficient, having a memory footprint that is orders of magnitude lower than that of A^{*+}-LSa across all tested bins. (FORI) is even leaner, peaking at just 369MB on AIDS and 253MB on MUTAGENICITY, making it the most



(a) (b) (c)

Figure 7: Scatter plots comparing runtimes of (F2) vs (FORI) over each dataset considered in Question Q1.



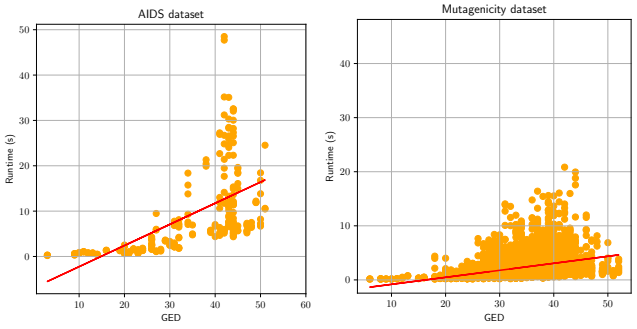
(a) (b) (c)

Figure 8: Scatter plots comparing runtimes of (F2+) vs (FORI) over each dataset considered in Question Q1.

memory-efficient approach in nearly all cases, except AIDS-71-80, where A^*-BMAO uses slightly less.

Note that this experiment also shows that (FORI) is robust under changes in the edit costs. Indeed, measured statistics on unit edit costs (those in Tables 6–7) are comparable with the statistics collected on non-uniform edit costs (those in Tables 4–5, resp.).

Finally, unit edit costs provide insight into the trend linking optimal GED values with execution time. In Figure 9 we compare the optimal GED value with the (FORI) runtime on instances generated over all AIDS (Figure 9a) and MUTAGENICITY (Figure 9b) graphs in the IAM graph repository having between 50 and 52 vertices. The plots reveal an almost linear relationship between the two quantities, highlighting a reasonable scalability of our model, even in relation to increasing GED values.



(a) (b)

Figure 9: Scatter plots comparing optimal GED values under unit edit costs vs runtime of (FORI) in seconds.

Answering (Q3). The graph sizes of the IMDB, CORA, and PUBMED instances result in large ILPs for all models. This necessitates some adjustments to the Gurobi parameters. The solver was configured to use 32 threads, set `Method` to 2, `NodeMethod` to 2, and `Crossover` to 0 to ensure it relied solely on the barrier method. Additionally, we disabled cuts and the RINS heuristics by setting `Cuts` to 0 and `RINS` to 0. Finally, the solution time benefits from `Disconnected=2`, aggressively exploiting the LP structure. The effectiveness of (FORI) in computing the GED for a small graph pattern and a very large graph has been compared to that of the previous state-of-the-art model, (F2), and our best strengthening of this formulation, namely (F2+). The results of the experiments are summarized in Table 8.

Table 6: Question Q2: comparison on AIDS dataset.

METHOD	21-30	31-40	41-50	51-60	61-70	71-80
avg. runtime in sec.						
A^{*+}-LSa	257.83	213.73	586.66	551.77	546.48	482.04
A^{*}-BMao	64.16	155.17	586.66	567.26	568.60	490.46
(FORI)	**0.22**	**0.45**	**6.13**	**11.79**	**11.64**	**138.91**
timeouts						
A^{*+}-LSa	13 (28.9%)	16 (35.5%)	44 (97.7%)	41 (91.1%)	40 (88.9%)	36 (80.0%)
A^{*}-BMao	1 (2.2%)	11 (24.4%)	44 (97.7%)	42 (93.3 %)	42 (93.3%)	36 (80.0%)
(FORI)	**0 (0%)**	**0 (0%)**	**0 (0%)**	**0 (0%)**	**0 (0%)**	**4 (8.8 %)**
avg. memory consumption in MB						
A^{*+}-LSa	23120	17000	42415	40121	37484	32215
A^{*}-BMao	275	502	939	749	578	**339**
(FORI)	**33**	**42**	**110**	**135**	**136**	369

Table 7: Question Q2: comparison on MUTAGENICITY dataset.

METHOD	21-30	31-40	41-50	51-60	61-70	71-80	81-90	91-100
avg. runtime in sec.								
A^{*+}-LSa	373.12	563.36	600.00	600.00	600.00	600.00	600.00	600.00
A^{*}-BMao	233.75	383.39	600.00	600.00	600.00	600.00	600.00	600.00
(FORI)	**0.41**	**0.67**	**1.85**	**2.62**	**6.06**	**14.49**	**18.95**	**31.77**
timeouts								
A^{*+}-LSa	27 (60.0%)	41 (91.1%)	45 (100%)	45 (100%)	45 (100%)	45 (100%)	45 (100%)	45 (100%)
A^{*}-BMao	14 (31.1%)	25 (55.5%)	45 (100%)	45 (100%)	45 (100%)	45 (100%)	45 (100%)	45 (100%)
(FORI)	**0 (0%)**	**0 (0%)**	**0 (0%)**	**0 (0%)**	**0 (0%)**	**0 (0%)**	**0 (0%)**	**0 (0%)**
avg. memory consumption in MB								
A^{*+}-LSa	34210	46609	48372	48801	47661	45445	45426	46108
A^{*}-BMao	1131	1231	1197	877	617	435	374	396
(FORI)	**32**	**48**	**71**	**76**	**112**	**159**	**194**	**253**

Table 8: Question Q3: results on IMDB, CORA, and PUBMED.

MODEL	IMDB-10	IMDB-20	IMDB-30	IMDB-43	CORA	PUBMED
avg. runtime in sec.						
(F2)	600.22	600.70	601.40	602.05	600.34	628.78
(F2+)	160.50	576.51	539.42	602.36	446.70	342.91
(FORI)	**1.77**	**8.45**	**22.95**	604.12	**235.60**	**293.84**
timeouts						
(F2)	6 (100%)	6 (100%)	6 (100%)	6 (100%)	10 (100%)	10 (100%)
(F2+)	**0 (0%)**	5 (83.3%)	5 (83.3%)	6 (100%)	7 (70%)	3 (30%)
(FORI)	**0 (0%)**	**0 (0%)**	**0 (0%)**	6 (100%)	**3 (30%)**	**2 (20%)**
avg. / max gap in %						
(F2)	29.2 / 29.7	56.6 / 61.6	68.7 / 78.6	84.7 / 103.6	3.8 / 10.6	0.7 / 1.6
(F2+)	**0 / 0**	1.5 / 2.6	4.5 / 8.1	21.8 / 52.3	0.1 / 0.7	**<0.1 / <0.1**
(FORI)	**0 / 0**	**0 / 0**	**0 / 0**	**10.2 / 29.4**	**<0.1 / 0.2**	<0.1 / 0.1[4]

The ineffectiveness of (F2) stands out in this setting, as it fails to solve any instance from each dataset within the time limit. Specifically, on the IMDB dataset, even in the smallest bin IMDB-10, gaps remain as high as 29% after 600 seconds of computation, reaching a maximum of 103.6% in the IMDB-43 bin. However, the documented gaps on the CORA and PUBMED datasets are smaller, on average equal to 3.82 and 0.73, respectively.

The strengthened (F2+) formulation solves all IMDB-10 instances, a few from IMDB-20, IMDB-30, CORA, and up to seven PUBMED instances. It performs efficiently on small IMDB graphs (~10 nodes), averaging under three minutes. For larger query graphs (~20+ nodes), it rarely converges within 600 seconds but still achieves tighter gaps than (F2). On CORA and PUBMED, it solves 30% and 70% of instances, respectively, averaging under eight minutes with consistently small gaps (~0.1, max 0.71 for CORA).

Finally, (FORI) effectively solves all instances in IMDB-10 –IMDB-30 with average runtime of ~2–23 seconds. However, it fails to solve any instance in IMDB-43, though yielding smaller gaps than (F2+) and (F2) on IMDB-43. On CORA, it solves 7/10 instances in ~4 minutes with a <0.1% average gap. For PUBMED, it closes one more instance than (F2+) in 5 minutes. Yet, a discrepancy on the average gap values is observed in this dataset, where the (F2+) formulation achieves a smaller gap than (FORI).

To summarize, (FORI) outperforms both (F2) and (F2+) by optimally solving 33 out of 45 instances containing very large graphs, whereas (F2) and (F2+) solve only 0 and 18 instances, respectively.

8 CONCLUSION AND FUTURE WORK

We explored a hierarchy of Integer Linear Programming models that provide theoretical and computational improvements over the current state-of-the-art methods for GED computation. The new ILP model (FORI) solves all benchmark instances from common databases to provable optimality. (FORI) is also able to solve the GED for a small graph pattern and a very large graph, widely extending the applicability of ILP-based methods for computing the GED. Future research directions include investigating the polyhedral structure of (FORI) to identify possible strengthening, exploiting the model structure to accelerate the solution of the LP relaxation, and developing tailored LP-based heuristics for very large instances.

We remark that the efficiency of (FORI) in computing the exact GED may have significant practical implications, which we intend to explore in future work. One key area is neural network-based heuristic GED approaches, which are typically trained using GED values [37]. However, due to the NP-hardness of the GED problem, the training data is often based on sub-optimal approximations of the true distance. The ability to generate exact ground-truth values would improve the quality and accuracy of such learning-based methods. Another impactful application lies in GED verification, where the goal is to identify, within a (potentially large) graph database, all graphs whose GED with respect to a given query graph lies below a specified threshold. This process typically involves a filtering phase, in which graphs are discarded based on lower bounds that exceed the threshold, followed by exact computation on the remaining candidates. We believe that our new formulations can enhance both stages of this pipeline. To improve filtering efficiency, we can use the dual bound from the LP relaxation as a quick and effective criterion. In the verification step, our experimental results show that our novel methods outperform the state-of-the-art techniques A^{*}-BMao and A^{*+}-LSa by several orders of magnitude, highlighting their strong potential for accelerating this process. Furthermore, using ILP solvers, we can efficiently prune the exact computation of GED between the query graph and each graph in the dataset by monitoring the optimization bounds: If the current best upper bound falls below the threshold, the corresponding graph is accepted. Conversely, if the best lower bound exceeds the threshold, the candidate graph can be safely discarded.

ACKNOWLEDGMENTS

This work was supported by the DFG under grant FOR-5361 – 459420781, by the BMBF (Germany) and state of NRW as part of the Lamarr-Institute, LAMARR22B, by MUR, the Italian Ministry of University and Research, under PRIN Project n. 2022TS4Y3N - EXPAND, and partially funded by the European Union–NextGenerationEU under the Italian Ministry of University and Research (MUR) National Innovation Ecosystem grant ECS00000041–VITALITY–CUP E13C22001060006.

[4] On instance 8603_29_29, barrier algorithm converges with tolerance 10^{-7}.

REFERENCES

[1] Zeina Abu-Aisheh, Benoit Gauzere, Sébastien Bougleux, Jean-Yves Ramel, Luc Brun, Romain Raveaux, Pierre Heroux, and Sébastien Adam. 2017. Graph edit distance contest: Results and future challenges. *Pattern Recognition Letters* 100 (2017), 96–103. https://doi.org/10.1016/j.patrec.2017.10.007

[2] Zeina Abu-Aisheh, Romain Raveaux, Jean-Yves Ramel, and Patrick Martineau. 2015. An Exact Graph Edit Distance Algorithm for Solving Pattern Recognition Problems. In *ICPRAM 2015 - Proceedings of the International Conference on Pattern Recognition Applications and Methods, Volume 1, Lisbon, Portugal, 10-12 January, 2015*, Maria De Marsico, Mário A. T. Figueiredo, and Ana L. N. Fred (Eds.). SciTePress, 271–278.

[3] Jiyang Bai and Peixiang Zhao. 2021. TaGSim: Type-aware Graph Similarity Learning and Computation. *Proc. VLDB Endow.* 15, 2 (2021), 335–347. https://doi.org/10.14778/3489496.3489513

[4] David B. Blumenthal, Nicolas Boria, Johann Gamper, Sébastien Bougleux, and Luc Brun. 2020. Comparing heuristics for graph edit distance computation. *VLDB J.* 29, 1 (2020), 419–458. https://doi.org/10.1007/S00778-019-00544-1

[5] David B. Blumenthal and Johann Gamper. 2020. On the exact computation of the graph edit distance. *Pattern Recognit. Lett.* 134 (2020), 46–57. https://doi.org/10.1016/J.PATREC.2018.05.002

[6] Sébastien Bougleux, Luc Brun, Vincenzo Carletti, Pasquale Foggia, Benoit Gaüzère, and Mario Vento. 2017. Graph edit distance as a quadratic assignment problem. *Pattern Recognit. Lett.* 87 (2017), 38–46. https://doi.org/10.1016/J.PATREC.2016.10.001

[7] Lijun Chang, Xing Feng, Xuemin Lin, Lu Qin, Wenjie Zhang, and Dian Ouyang. 2020. Speeding Up GED Verification for Graph Similarity Search. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*. IEEE, 793–804. https://doi.org/10.1109/ICDE48307.2020.00074

[8] Lijun Chang, Xing Feng, Kai Yao, Lu Qin, and Wenjie Zhang. 2023. Accelerating Graph Similarity Search via Efficient GED Computation. *IEEE Trans. Knowl. Data Eng.* 35, 5 (2023), 4485–4498. https://doi.org/10.1109/TKDE.2022.3153523

[9] Xiaoyang Chen, Hongwei Huo, Jun Huan, and Jeffrey Scott Vitter. 2019. An efficient algorithm for graph edit distance computation. *Knowl. Based Syst.* 163 (2019), 762–775. https://doi.org/10.1016/J.KNOSYS.2018.10.002

[10] M. Conforti, G. Cornuéjols, and G. Zambelli. 2014. *Integer Programming*. Springer International Publishing.

[11] Andre Droschinsky, Lina Humbeck, Oliver Koch, Nils M. Kriege, Petra Mutzel, and Till Schäfer. 2022. Graph-Based Methods for Rational Drug Design. In *Algorithms for Big Data - DFG Priority Program 1736*, Hannah Bast, Claudius Korzen, Ulrich Meyer, and Manuel Penschuck (Eds.). Lecture Notes in Computer Science, Vol. 13201. Springer, 76–96. https://doi.org/10.1007/978-3-031-21534-6_5

[12] Chi Thang Duong, Trung Dung Hoang, Hongzhi Yin, Matthias Weidlich, Quoc Viet Hung Nguyen, and Karl Aberer. 2021. Efficient streaming subgraph isomorphism with graph neural networks. *Proceedings of the VLDB Endowment* 14, 5 (2021), 730–742.

[13] Christopher L. Ebsch, Joseph A. Cottam, Natalie C. Heller, Rahul D. Deshmukh, and George Chin. 2020. Using Graph Edit Distance for Noisy Subgraph Matching of Semantic Property Graphs. In *2020 IEEE International Conference on Big Data (Big Data)*. 2520–2525. https://doi.org/10.1109/BigData50022.2020.9378349

[14] Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. 2010. A survey of graph edit distance. *Pattern Anal. Appl.* 13, 1 (2010), 113–129. https://doi.org/10.1007/S10044-008-0141-Y

[15] Karam Gouda and Mosab Hassaan. 2016. CSI_GED: An efficient approach for graph edit similarity computation. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*. IEEE Computer Society, 265–276. https://doi.org/10.1109/ICDE.2016.7498246

[16] Gurobi Optimization, LLC. 2023. Gurobi Optimizer Reference Manual. https://www.gurobi.com

[17] Eeshaan Jain, Indradyumna Roy, Saswat Meher, Soumen Chakrabarti, and Abir De. 2024. Graph Edit Distance Evaluation Datasets: Pitfalls and Mitigation. In *The Third Learning on Graphs Conference*. https://openreview.net/forum?id=guapIeLs02 Virtual Event), November 26.29, 2024.

[18] Eeshaan Jain, Indradyumna Roy, Saswat Meher, Soumen Chakrabarti, and Abir De. 2024. Graph Edit Distance with General Costs Using Neural Set Divergence. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

[19] Derek Justice and Alfred O. Hero III. 2006. A Binary Linear Programming Formulation of the Graph Edit Distance. *IEEE Trans. Pattern Anal. Mach. Intell.* 28, 8 (2006), 1200–1214. https://doi.org/10.1109/TPAMI.2006.152

[20] Jongik Kim. 2023. Efficient graph edit distance computation using isomorphic vertices. *Pattern Recognit. Lett.* 168 (2023), 71–78. https://doi.org/10.1016/J.PATREC.2023.03.002

[21] Julien Lerouge, Zeina Abu-Aisheh, Romain Raveaux, Pierre Héroux, and Sébastien Adam. 2017. New binary linear programming formulation to compute the graph edit distance. *Pattern Recognit.* 72 (2017), 254–265. https://doi.org/10.1016/J.PATCOG.2017.07.029

[22] Chih-Long Lin. 1994. Hardness of Approximating Graph Transformation Problem. In *Algorithms and Computation, 5th International Symposium, ISAAC '94, Beijing, P. R. China, August 25-27, 1994, Proceedings (Lecture Notes in Computer Science)*, Ding-Zhu Du and Xiang-Sun Zhang (Eds.), Vol. 834. Springer, 74–82. https://doi.org/10.1007/3-540-58325-4_168

[23] Qiankun Liu, Qi Chu, Bin Liu, and Nenghai Yu. 2020. GSM: Graph Similarity Model for Multi-Object Tracking. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, Christian Bessiere (Ed.). ijcai.org, 530–536. https://doi.org/10.24963/IJCAI.2020/74

[24] Andrea Lodi and Andrea Tramontani. 2013. Performance variability in mixed-integer programming. In *Theory driven by influential applications*. INFORMS, 1–12.

[25] Michel Neuhaus and Horst Bunke. 2007. *Bridging the Gap between Graph Edit Distance and Kernel Machines*. Series in Machine Perception and Artificial Intelligence, Vol. 68. WorldScientific. https://doi.org/10.1142/6523

[26] Thanh Tam Nguyen, Thanh Cong Phan, Hien Thu Pham, Thanh Thi Nguyen, Jun Jo, and Quoc Viet Hung Nguyen. 2023. Example-based explanations for streaming fraud detection on graphs. *Inf. Sci.* 621 (2023), 319–340. https://doi.org/10.1016/J.INS.2022.11.119

[27] Chengzhi Piao, Tingyang Xu, Xiangguo Sun, Yu Rong, Kangfei Zhao, and Hong Cheng. 2023. Computing Graph Edit Distance via Neural Graph Matching. *Proc. VLDB Endow.* 16, 8 (2023), 1817–1829. https://doi.org/10.14778/3594512.3594514

[28] Rishabh Ranjan, Siddharth Grover, Sourav Medya, Venkatesan Chakaravarthy, Yogish Sabharwal, and Sayan Ranu. 2022. Greed: A neural framework for learning graph distance functions. *Advances in Neural Information Processing Systems* 35 (2022), 22518–22530.

[29] Sayan Ranu and Ambuj K. Singh. 2009. Mining Statistically Significant Molecular Substructures for Efficient Molecular Classification. *Journal of Chemical Information and Modeling* 49, 11 (2009), 2537–2550. https://doi.org/10.1021/ci900035z arXiv:https://doi.org/10.1021/ci900035z

[30] Kaspar Riesen. 2015. *Structural Pattern Recognition with Graph Edit Distance - Approximation Algorithms and Applications*. Springer. https://doi.org/10.1007/978-3-319-27252-8

[31] Kaspar Riesen and Horst Bunke. 2008. IAM Graph Database Repository for Graph Based Pattern Recognition and Machine Learning. In *Structural, Syntactic, and Statistical Pattern Recognition, Joint IAPR International Workshop, SSPR & SPR 2008, Orlando, USA, December 4-6, 2008. Proceedings (Lecture Notes in Computer Science)*, Niels da Vitoria Lobo, Takis Kasparis, Fabio Roli, James Tin-Yau Kwok, Michael Georgiopoulos, Georgios C. Anagnostopoulos, and Marco Loog (Eds.), Vol. 5342. Springer, 287–297. https://doi.org/10.1007/978-3-540-89689-0_33

[32] Kaspar Riesen, Stefan Fankhauser, and Horst Bunke. 2007. Speeding Up Graph Edit Distance Computation with a Bipartite Heuristic. In *Mining and Learning with Graphs, MLG 2007, Firence, Italy, August 1-3, 2007, Proceedings*, Paolo Frasconi, Kristian Kersting, and Koji Tsuda (Eds.). http://mlg07.dsi.unifi.it/pdf/02_Riesen.pdf

[33] Aravind Sankar, Sayan Ranu, and Karthik Raman. 2017. Predicting novel metabolic pathways through subgraph mining. *Bioinformatics* 33, 24 (2017), 3955–3963. https://doi.org/10.1093/bioinformatics/btx481

[34] Ida Schomburg, Antje Chang, Christian Ebeling, Marion Gremse, Christian Heldt, Gregor Huhn, and Dietmar Schomburg. 2004. BRENDA, the enzyme database: updates and major new developments. *Nucleic Acids Res.* 32, Database-Issue (2004), 431–433. https://doi.org/10.1093/NAR/GKH081

[35] Runzhong Wang, Tianqi Zhang, Tianshu Yu, Junchi Yan, and Xiaokang Yang. 2021. Combinatorial Learning of Graph Edit Distance via Dynamic Embedding. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 5241–5250. https://doi.org/10.1109/CVPR46437.2021.00520

[36] Laurence A Wolsey. 2020. *Integer programming*. John Wiley & Sons.

[37] Zhoubo Xu, Puqing Chen, Romain Raveaux, Xin Yang, and Huadong Liu. 2024. Deep graph matching meets mixed-integer linear programming: Relax or not ? *Pattern Recognition* 155 (2024), 110697. https://doi.org/10.1016/j.patcog.2024.110697

[38] Mang Ye, Jianbing Shen, Gaojie Lin, Tao Xiang, Ling Shao, and Steven C. H. Hoi. 2022. Deep Learning for Person Re-Identification: A Survey and Outlook. *IEEE Trans. Pattern Anal. Mach. Intell.* 44, 6 (2022), 2872–2893. https://doi.org/10.1109/TPAMI.2021.3054775

[39] Zhiping Zeng, Anthony K. H. Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. 2009. Comparing Stars: On Approximating Graph Edit Distance. *Proc. VLDB Endow.* 2, 1 (2009), 25–36. https://doi.org/10.14778/1687627.1687631

[40] Zhen Zhang, Jiajun Bu, Martin Ester, Zhao Li, Chengwei Yao, Zhi Yu, and Can Wang. 2021. H2MN: Graph Similarity Learning with Hierarchical Hypergraph Matching Networks. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, Feida Zhu, Beng Chin Ooi, and Chunyan Miao (Eds.). ACM, 2274–2284. https://doi.org/10.1145/3447548.3467328