An Efficient Heuristic for Graph Edit Distance

Xiaoyang Chen
Department of Computer Science, Xidian University, Xi'an, China

Yujia Wang
Department of Computer Science, Xidian University, Xi'an, China

Hongwei Huo[1]
Department of Computer Science, Xidian University, Xi'an, China

Jeffrey Scott Vitter[1]
Department of Computer Science, Tulane University, New Orleans, LA, USA
The University of Mississippi, MS, USA

Abstract

The graph edit distance (GED) is a flexible distance measure widely used in many applications. Existing GED computation methods are usually based upon the tree-based search algorithm that explores all possible vertex (or edge) mappings between two compared graphs. During this process, various GED lower bounds are adopted as heuristic estimations to accelerate the tree-based search algorithm. For the first time, we analyze the relationship among three state-of-the-art GED lower bounds, label edit distance (LED), Hausdorff edit distance (HED), and branch edit distance (BED). Specifically, we demonstrate that $BED(G, Q) \geq HED(G, Q)$ and $BED(G, Q) \geq LED(G, Q)$ for any two undirected graphs G and Q. Furthermore, for BED we propose an efficient heuristic BED$^+$ for improving the tree-based search algorithm. Extensive experiments on real and synthetic datasets confirm that BED$^+$ achieves smaller deviation and larger solvable ratios than LED, HED and BED when they are employed as heuristic estimations. The source code is available online.

2012 ACM Subject Classification Information systems → Query optimization

Keywords and phrases Graph edit distance, Label edit distance, Hausdorff edit distance, Branch edit distance, Tree-based search, Heuristics

Digital Object Identifier 10.4230/OASIcs.Grossi.2025.1

Category Research

Supplementary Material
Software (Source Code): `https://github.com/Hongweihuo-Lab/Heur-GED` [11]

Funding This work was supported in part by the National Natural Science Foundation of China under Grant No. 62272358.

1 Introduction

Graphs are frequently used to represent a wide variety of various objects, such as networks, maps, handwriting, molecular compounds, and protein structures. The process of evaluating the similarity of two graphs is referred to as error-tolerant graph matching, aiming to find a correspondence between their vertices. In this paper, we focus upon the similarity measure *graph edit distance (GED)* because it can be applied to all types of graphs and can precisely capture structural differences between the compared graphs. The GED of two graphs is defined as the minimum cost of transforming one graph into another through a sequence

[1] corresponding author

of edit operations (inserting, deleting and substituting vertices or edges). An edit cost is assigned to each edit operation to measure its strength, which can be obtained by combining specific knowledge of the domain or learning from a set of sample graphs [14].

However, computing the GED is an NP-hard problem [30] and usually based upon the tree-based search algorithm. This search tree enumerates all possible mappings between vertices (or edges) of two compared graphs, where the inner nodes denote partial mappings and the leaf nodes denote complete mappings. Most existing GED computation methods employ different search paradigms to traverse this search tree to seek for the *optimal mapping* that induces the GED. Riesen et al. [25, 26] proposed the standard method, A*-GED, based upon the best-first search paradigm. It needs to store numerous inner nodes, resulting in high memory consumption. To overcome this bottleneck, Abu-Aisheh et al. [3] proposed a depth-first search based algorithm, DF-GED, whose memory requirement increases linearly with the number of vertices of graphs. On the other hand, Chen et al. [9] introduced a method for the GED computation based upon beam-stack search [28], achieving a flexible tradeoff between memory consumption and the time overhead of backtracking in the depth-first search. Chang et al. [8] developed a unified framework that can be instantiated into either a best-first search approach or a depth-first search approach. Gouda et al. [17] proposed a novel edge-mapping based approach, CSI_GED, and also employed the depth-first search paradigm. CSI_GED works only for the uniform cost model, and Blumenthal et al. [4, 6] generalized it to cover the non-uniform cost model. Kim [19] developed an efficient GED computation algorithm using isomorphic vertices [9]. Liu et al. [22] explored a learning-based method for the approximate GED computation. Piao et al. [23] propose a deep learning method for the GED computation. It is worth mentioning that many researchers have proposed various indexing techniques [31, 8, 10] to accelerate graph similarity searches under the GED metric. They use the above GED computation methods as the final phase to verify the candidate graphs that satisfy the GED constraint.

In the tree-based search algorithm, a heuristic estimation is usually adopted to prune the useless search space to accelerate the search process. In order to ensure that the optimal mapping is not erroneously pruned, this heuristic function must be *admissible*; namely, it estimates the cost of a tree node that is less than or equal to the real cost. In the previous works A*-GED and DF-GED, they adopted the label edit distance (LED) as the heuristic, which calculates the minimum substitution cost of vertices and edges of two compared graphs. After that, Fischer et al. [15, 16] proposed the Hausdorff edit distance (HED) as a heuristic estimation. HED, based upon Hausdorff matching [29], performs a bidirectional matching between two graphs and allows multiple assignments between their vertices. Recently, Blumenthal et al. [5] proposed another effective GED lower bound, branch edit distance (BED), which also can be adopted as a heuristic estimation.

As observed in other studies [7, 15, 27], the higher the heuristic estimates the cost, the better the tree-based search algorithm performs. The following question naturally arises: Which of these three state-of-the-art GED lower bounds (namely, LED, HED, or BED) is more effective? In this paper, we first analyze the relationship among these three lower bounds and then propose an effective heuristic estimation. Our contributions are summarized as follows:

(1) We analyze the relationship among LED, HED and BED for the first time, and we derive that $BED(G, Q) \geq HED(G, Q)$ and $BED(G, Q) \geq LED(G, Q)$ for any two undirected graphs G and Q.

(2) We propose an efficient heuristic estimation BED$^+$ based upon BED, and demonstrate that BED$^+$ is still admissible.

(3) We conduct extensive experiments to confirm BED$^+$'s effectiveness on the real and synthetic datasets. The source code is available online [11].

The rest of this paper is organized as follows. In Section 2, we give the definition of the graph edit distance and revisit three state-of-the-art GED lower bounds. In Section 3, we theoretically analyze the relationship between LED, HED and BED. In Section 4, we propose the heuristic function BED$^+$ for improving the GED computation. In Section 5, we report the experimental results. Finally, we summarize this paper in Section 6.

2 Graph edit distance

In this paper, we consider *undirected*, *labeled* graphs without multi-edges or self-loops. A labeled graph is a triplet $G = (V_G, E_G, L)$, where V_G is the set of vertices, E_G is the set of edges, $L : V_G \cup E_G \to \Sigma$ is a labeling function that assigns a label to a vertex or an edge, and Σ is a set of labels. Also, we use a special symbol ε to denote a *dummy vertex* or a *dummy edge*.

Given two graphs G and Q, six edit operations [18, 25, 21, 5] can be used to transform G to Q (or vice versa): inserting or deleting a vertex or an edge, and substituting the label of a vertex or an edge. We denote the *label substitution* (or simply *substitution*) of vertices $u \in V_G$ and $v \in V_Q$ by $(u \to v)$, the *deletion* of u by $(u \to \varepsilon)$, and the *insertion* of v by $(\varepsilon \to v)$. For the three edit operations on edges, we use similar notation.

An *edit path* $\mathcal{P} = \langle p_1, p_2, \ldots, p_k \rangle$ between G and Q is a sequence of edit operations that transforms G to Q, such as $G = G^0 \xrightarrow{p_1} \ldots G^i \xrightarrow{p_{i+1}} G^{i+1} \ldots \xrightarrow{p_k} G^k = Q$, where graph G^{i+1} is obtained by performing the edit operation p_{i+1} on graph G^i, for $0 \le i \le k-1$. During this transformation, each edit operation p_i is assigned a penalty cost $c(p_i)$ to reflect whether it can strongly change a graph. Note that the cost of editing two dummy vertices (or edges) is 0; that is, $c(\varepsilon \to \varepsilon) = 0$. Thus, $\mathcal{P}$'s edit cost is defined as $\sum_{i=1}^{k} c(p_i)$. We define the *graph edit distance* as follows:

▶ **Definition 1.** *Given two graphs G and Q, the graph edit distance between them, denoted by $ged(G, Q)$, is defined as the minimum cost of transforming G to Q, namely,*

$$ged(G, Q) = \min_{\mathcal{P} \in \Upsilon(G,Q)} \sum\nolimits_{p_i \in \mathcal{P}} c(p_i) \tag{1}$$

where $\Upsilon(G, Q)$ is the set of all edit paths between G and Q, and $c(p_i)$ is edit operation p_i's cost.

Hereafter, for ease of presentation, we denote $V_G^\varepsilon = V_G \cup \overbrace{\{\varepsilon, \ldots, \varepsilon\}}^{|V_Q|}$ and $V_Q^\varepsilon = V_Q \cup \overbrace{\{\varepsilon, \ldots, \varepsilon\}}^{|V_G|}$ as the expanded sets of V_G and E_G, respectively, so that V_G^ε and V_Q^ε have the same number of vertices. Similarly, $E_G^\varepsilon = E_G \cup \overbrace{\{\varepsilon, \ldots, \varepsilon\}}^{|E_Q|}$ and $E_Q^\varepsilon = E_Q \cup \overbrace{\{\varepsilon, \ldots, \varepsilon\}}^{|E_G|}$ denote the expanded sets of E_G and E_Q, respectively.

2.1 State-of-the-art GED lower bounds

Below we introduce three state-of-the-art GED lower bounds, which can be used as heuristic estimations in the tree-based search algorithm to compute GED. Each of the methods gives a lower bound on GED because the operations are done in sets that do not have to be consistent with one another. For example, in the first method LED described below, the edit operations on the vertex labels can be done independently of the edit operations on the edge labels, and thus they may not be globally consistent.

Label Edit Distance. Riesen et al. [27, 26] proposed the label edit distance (LED), which is the minimum cost of substituting vertices and edges of two graphs.

▶ **Definition 2** (Label edit distance). *Given two graphs G and Q, the label edit distance between them is defined as $LED(G,Q) = \lambda_V(G,Q) + \lambda_E(G,Q)$, where $\lambda_V(G,Q) = \min_{\phi:V_G^\varepsilon \to V_Q^\varepsilon} \sum_{u \in V_G^\varepsilon} c(u \to \phi(u))$ is the minimum cost of substituting vertices of G and Q, and ϕ is a bijection from V_G^ε to V_Q^ε; and $\lambda_E(G,Q) = \min_{\varphi:E_G^\varepsilon \to E_Q^\varepsilon} \sum_{e(u,u') \in E_G^\varepsilon} c(e(u,u') \to \varphi(e(u,u')))$ is the minimum cost of substituting edges of G and Q, and φ is a bijection from E_G^ε to E_Q^ε.*

Hausdorff Edit Distance. Inspired by the Hausdorff distance [29] between two finite sets, Fischer et al. [16] proposed the Hausdorff edit distance (HED) between two graphs G and Q. The key ideas of HED are to perform a bidirectional matching between G and Q and to allow multiple assignments between their vertices.

▶ **Definition 3** (Hausdorff edit distance). *Given two graphs G and Q, their Hausdorff edit distance is defined as $HED(G,Q) = \sum_{u \in V_G} \min_{v \in V_Q \cup \{\varepsilon\}} f_H(u,v) + \sum_{v \in V_Q} \min_{u \in V_G \cup \{\varepsilon\}} f_H(u,v)$, where $f_H(u,v)$ is the Hausdorff cost of matching vertex u to vertex v.*

The Hausdorff vertex matching cost $f_H(u,v)$ considers not only the two vertices $u \in V_G$ and $v \in V_Q$ but also their neighboring edges.

▶ **Definition 4** (Neighboring edges). *Given graph G and a vertex $u \in V_G$, the* neighboring edges N_u *of u are defined as $N_u = \{e(u,u') : u' \in V_G \wedge e(u,u') \in E_G\}$.*

We define $f_H(u,v)$ as

$$f_H(u,v) = \begin{cases} c(u \to \varepsilon) + \sum_{e_1 \in N_u} \frac{c(e_1 \to \varepsilon)}{2} & \text{if } v = \varepsilon; \\ c(\varepsilon \to v) + \sum_{e_2 \in N_v} \frac{c(\varepsilon \to e_2)}{2} & \text{if } u = \varepsilon; \\ \frac{c(u \to v) + \frac{HED(N_u, N_v)}{2}}{2} & \text{otherwise.} \end{cases} \tag{2}$$

Similarly to Definition 3, the Hausdorff edit distance $HED(N_u, N_v)$ between N_u and N_v is defined as

$$HED(N_u, N_v) = \sum_{e_1 \in N_u} \min_{e_2 \in N_v \cup \{\varepsilon\}} f_H(e_1, e_2) + \sum_{e_2 \in N_v} \min_{e_1 \in N_u \cup \{\varepsilon\}} f_H(e_1, e_2) \tag{3}$$

where $f_H(e_1, e_2)$ is the cost of matching two edges such that

$$f_H(e_1,e_2) = \begin{cases} c(e_1 \to \varepsilon) & \text{if } e_2 = \varepsilon; \\ c(\varepsilon \to e_2) & \text{if } e_1 = \varepsilon; \\ \frac{c(e_1 \to e_2)}{2} & \text{otherwise.} \end{cases} \tag{4}$$

Branch Edit Distance. Blumenthal et al. [5] recently proposed the branch edit distance (BED), which computes the minimum cost of editing branch structures of two graphs.

▶ **Definition 5** (Branch structure). *The branch structure of vertex u in graph G is defined as $B_u = (u, N_u)$, where N_u is the set of neighboring edges of u.*

Given two branch structures B_u and B_v, the minimum cost of editing B_u into B_v is defined as

$$f_B(u,v) = c(u \to v) + \frac{1}{2} \min_{\varrho: N_u^\varepsilon \to N_v^\varepsilon} \sum_{e(u,u') \in N_u^\varepsilon} c(e(u,u') \to \varrho(e(u,u'))), \tag{5}$$

where $N_u^\varepsilon = N_u \cup \overbrace{\{\varepsilon, \ldots, \varepsilon\}}^{|N_v|}$ and $N_v^\varepsilon = N_v \cup \overbrace{\{\varepsilon, \ldots, \varepsilon\}}^{|N_u|}$ are expanded sets of N_u and N_v, respectively, and ϱ is a bijection from N_u^ε to N_v^ε.

▶ **Definition 6** (Branch edit distance). *Given two graphs G and Q, the branch edit distance between them is defined as $BED(G,Q) = \min_{\rho: V_G^\varepsilon \to V_Q^\varepsilon} \sum_{u \in V_G^\varepsilon} f_B(u, \rho(u))$, where ρ is a bijection from V_G^ε to V_Q^ε, and $f_B(\cdot,\cdot)$ is defined in (5).*

3 Tightness analysis

In this section, we analyze the tightness of the three GED lower bounds: LED, HED and BED. Specifically, we will prove that BED is the strongest of all; that is, for any two undirected graphs G and Q, we have $BED(G,Q) \geq LED(G,Q)$ and $BED(G,Q) \geq HED(G,Q)$.

3.1 Relation of LED and BED

▶ **Theorem 7.** *Given two graphs G and Q, we have $BED(G,Q) \geq LED(G,Q)$.*

Proof. For ease of proof, we insert dummy vertices and edges into G to make it become a complete graph with $(|V_G| + |V_Q|)$ vertices. Similarly, we transform Q into a complete graph that also has $(|V_G| + |V_Q|)$ vertices. Then, we can simplify (5) as

$$\begin{aligned} f_B(u,v) &= c(u \to v) + \frac{1}{2} \min_{\varrho: N_u^\varepsilon \to N_v^\varepsilon} \sum_{e(u,u') \in N_u^\varepsilon} c(e(u,u') \to \varrho(e(u,u'))) \\ &= c(u \to v) + \frac{1}{2} \min_{\zeta: V_G \setminus \{u\} \to V_Q \setminus \{v\}} \sum_{u' \in V_G \setminus \{u\}} c(e(u,u') \to e(v, \zeta(u'))) \\ &= c(u \to v) + \frac{1}{2} \sum_{u' \in V_G \setminus \{u\}} c(e(u,u') \to e(v, \zeta_{\min}^{u,v}(u'))) \end{aligned}$$

where $\zeta_{\min}^{u,v}$ is the bijection from $V_G \setminus \{u\}$ to $V_Q \setminus \{v\}$ for which $f_B(u,v)$ achieves the minimum value. Thus, we have

$$\begin{aligned} BED(G,Q) &= \min_{\rho: V_G \to V_Q} \sum_{u \in V_G} f_B(u, \rho(u)) \\ &= \sum_{u \in V_G} \left\{ c(u \to \rho_{\min}(u)) + \frac{1}{2} \sum_{u' \in V_G \setminus \{u\}} c(e(u,u') \to e(\rho_{\min}(u), \zeta_{\min}^{u,u'}(u')) \right\} \\ &= \sum_{u \in V_G} c(u \to \rho_{\min}(u)) + \frac{1}{2} \sum_{u \in V_G} \sum_{u' \in V_G \setminus \{u\}} c(e(u,u') \to e(\rho_{\min}(u), \zeta_{\min}^{u,u'}(u'))) \\ &= \sum_{u \in V_G} c(u \to \rho_{\min}(u)) + \sum_{e(u,u') \in E_G} c(e(u,u') \to \xi(e(u,u'))) \\ &\geq \min_{\phi: V_G \to V_Q} \sum_{u \in V_G} c(u \to \phi(u)) + \min_{\varphi: E_G \to E_Q} \sum_{e(u,u') \in E_G} c(e(u,u') \to \varphi(e(u,u')) \\ &= \lambda_V(G,Q) + \lambda_E(E_G, E_Q) = LED(G,Q) \end{aligned}$$

where $\rho_{\min}$ is the bijection from V_G to V_Q for which $BED(G,Q)$ achieves the minimum value, and ξ is the bijection from E_G to E_Q satisfying $e(\rho_{\min}(u), \zeta_{\min}^{u,u'}(u'))) = \xi(e(u,u'))$ for $\forall u \in V_G, u' \in V_G \backslash \{u\}$. ◀

3.2 Relation of HED and BED

▶ **Lemma 8.** *Given two vertices $u \in V_G^\varepsilon$ and $v \in V_Q^\varepsilon$, then we have*

$$f_H(u,v) \leq \begin{cases} f_B(u,v) & \text{if } u = \varepsilon \text{ or } v = \varepsilon; \\ \frac{1}{2} f_B(u,v) & \text{otherwise.} \end{cases}$$

where $f_H(u,v)$ and $f_B(u,v)$ are defined in (2) *and* (5)*, respectively.*

The proof of Lemma 8 is in Appendix A.

▶ **Theorem 9.** *Given two graphs G and Q, we have $BED(G,Q) \geq HED(G,Q)$.*

Proof. By $f_H(u,v)$'s definition in (2), we know that when $u = \varepsilon$, $\min_{v \in V_Q \cup \{\varepsilon\}} f_H(\varepsilon, v) = f_H(\varepsilon,\varepsilon) = 0$; and similarly when $v = \varepsilon$, $\min_{u \in V_G \cup \{\varepsilon\}} f_H(u,\varepsilon) = f_H(\varepsilon,\varepsilon) = 0$. We can rewrite $HED(G,Q)$ as

$$\begin{aligned} HED(G,Q) &= \sum_{u \in V_G} \min_{v \in V_Q \cup \{\varepsilon\}} f_H(u,v) + \sum_{v \in V_Q} \min_{u \in V_G \cup \{\varepsilon\}} f_H(u,v) \\ &= \sum_{u \in V_G^\varepsilon} \min_{v \in V_Q^\varepsilon} f_H(u,v) + \sum_{v \in V_Q^\varepsilon} \min_{u \in V_G^\varepsilon} f_H(u,v) \\ &= \sum_{u \in V_G^\varepsilon} f_H(u, \pi_1(u)) + \sum_{v \in V_Q^\varepsilon} f_H(\pi_2(v), v)) \\ &= \sum_{u \in V_G^\varepsilon} \Big\{ f_H(u, \pi_1(u)) + f_H\big(\pi_2(\rho_{\min}(u)), \rho_{\min}(u)\big) \Big\}, \end{aligned} \tag{6}$$

where π_1 is a *mapping* from V_G^ε to V_Q^ε satisfying $\pi_1(u) = \arg\min_{v \in V_Q^\varepsilon} f_H(u,v)$,; π_2 is a *mapping* from V_Q^ε to V_G^ε satisfying $\pi_2(v) = \arg\min_{u \in V_G^\varepsilon} f_H(u,v)$; and $\rho_{\min}$ is the bijection from V_G^ε to V_Q^ε for which $BED(G,Q)$ achieves the minimum value. We know that

$$BED(G,Q) = \sum_{u \in V_G^\varepsilon} f_B(u, \rho_{\min}(u)). \tag{7}$$

By (6) and (7), we can complete the proof by showing that

$$f_H\big(u, \pi_1(u)\big) + f_H\big(\pi_2(\rho_{\min}(u)), \rho_{\min}(u)\big) \leq f_B(u, \rho_{\min}(u)).$$

We do so by considering the following four exhaustive cases:

Case I. When $u = \varepsilon$ and $\rho_{\min}(u) = \varepsilon$, then $f_H(u, \pi_1(u)) + f_H(\pi_2(\rho_{\min}(u)), \rho_{\min}(u)) \leq f_H(\varepsilon,\varepsilon) + f_H(\varepsilon,\varepsilon) = f_B(\varepsilon,\varepsilon) = 0$, by the definitions of π_1, π_2, and $\rho_{\min}$.

Case II. When $u \neq \varepsilon$ and $\rho_{\min}(u) = \varepsilon$, then $f_H(u, \pi_1(u)) + f_H(\pi_2(\rho_{\min}(u)), \rho_{\min}(u)) \leq f_H(u,\varepsilon) + f_H(\varepsilon,\varepsilon) = f_H(u,\varepsilon)$, by the definitions of π_1, π_2, and $\rho_{\min}$. By Lemma 8, we know that $f_H(u,\varepsilon) \leq f_B(u,\varepsilon) = f_B(u, \rho_{\min}(u))$.

Case III. When $u = \varepsilon$ and $\rho_{\min}(u) \neq \varepsilon$, the analysis is similar to that of Case II.

Case IV. When $u \neq \varepsilon$ and $\rho_{\min}(u) \neq \varepsilon$, then we have $f_H(u, \pi_1(u)) \leq f_H(u, \rho_{\min}(u))$ and $f_H(\pi_2(\rho_{\min}(u)), \rho_{\min}(u)) \leq f_H(u, \rho_{\min}(u))$, by the definitions of π_1, π_2, and $\rho_{\min}$. By Lemma 8, we know that $f_H(u, \rho_{\min}(u)) \leq \frac{1}{2} f_B(u, \rho_{\min}(u))$. Thus, we have $f_H(u, \pi_1(u)) + f_H\big(\pi_2(\rho_{\min}(u)), \rho_{\min}(u)\big) \leq 2 \times \frac{1}{2} f_B(u, \rho_{\min}(u)) = f_B(u, \rho_{\min}(u))$. ◀

4 Tree-based search algorithm

The previous section showed that BED achieves the tightest GED lower bound. In this section, based upon BED we propose an efficient heuristic estimation to improve the tree-based search algorithm [2, 3] for the GED computation.

4.1 Search tree

Computing the GED of graphs G and Q is typically based upon a tree-based search procedure that explores all possible graph mappings from G to Q. Starting from a dummy node, root $= \emptyset$, we logically create the search tree layer by layer by iteratively generating successors using BasicGenSuccr [9]. This search space can be organized as an ordered search tree, where the inner nodes denote partial graph mappings and the leaf nodes denote complete graph mappings. Such a search tree is created dynamically at runtime by iteratively generating successors linked by edges to the currently considered node. For more details, please refer to Section 2 in the reference [9].

4.2 Heuristic cost estimation

For a node r in the search tree, let $h(r)$ be the estimated cost from r to its descendant leaf node that is less than or equal to the real cost. Based upon BED, we introduce how to estimate $h(r)$ in the tree-based search algorithm.

4.2.1 Heuristic function

Consider an inner node $r = \{(u_1 \to v_{j_1}), \ldots, (u_l \to v_{j_\ell})\}$, where v_{j_k} is u_k's mapped vertex, for $1 \le k \le \ell$. We divide G into two subgraphs G_r^1 and G_r^2, where G_r^1 is the mapped part of G such that $V_{G_r^1} = \{u_1, \ldots, u_l\}$ and $E_{G_r^1} = \{e(u,v) : u, v \in V_{G_r^1} \wedge e(u,v) \in E_G\}$, and G_r^2 is the unmapped part such that $V_{G_r^2} = V_G \backslash V_{G_r^1}$ and $E_{G_r^2} = \{e(u,v) : u, v \in V_{G_r^2} \wedge e(u,v) \in E_G\}$. We obtain Q_r^1 and Q_r^2 similarly.

Clearly, the lower bound $BED(G_r^2, Q_r^2)$ can be used to estimate r's cost. However, $BED(G_r^2, Q_r^2)$ has not covered the potential edit cost on the edges between G_r^1 (resp., Q_r^1) and G_r^2 (resp., Q_r^2). Recently, [8, 9] proposed two different methods to cover this potential cost; nevertheless, these two methods only worked for the *uniform cost function* (i.e., for which the cost of each edit operation is 1). We expand the method in [8] to support for any cost function.

▶ **Definition 10.** *Given vertices $u \in G_r^2$ and $v \in Q_r^2$, we define the cost of matching u to v as $f_B^+(u,v) = f_B(u,v) + \sum_{u' \in V_{G_r^1}} c(e(u,u') \to e(v,v'))$, where v' is the mapped vertex of the already processed vertex u', and $f_B(u,v)$ is the minimum cost of transforming B_u to B_v, which we defined in* (5).

When there is no edge between u and u', we set $e(u,u') = \varepsilon$, and similarly for $e(v,v')$. Based upon $f_B^+(u,v)$, we define the improved lower bound BED$^+$ as

$$BED^+(G_r^2, Q_r^2) = \min_{\rho: V_{G_r^2}^\varepsilon \to V_{Q_r^2}^\varepsilon} \sum_{u \in V_{G_r^2}^\varepsilon} f_B^+(u, \rho(u)) \tag{8}$$

▶ **Theorem 11.** *Given a node r in the GED tree of graphs G and Q, then $BED^+(G_r^2, Q_r^2) \ge BED(G_r^2, Q_r^2)$, where G_r^2 and Q_r^2 are the unmapped subgraphs of G and Q, respectively, $BED^+(\cdot,\cdot)$ and $BED(\cdot,\cdot)$ are defined in* (8) *and Definition 6, respectively.*

Proof. We trivially obtain this theorem since $f_B^+(u,v) \geq f_B(u,v)$ for $\forall u \in V_{G_r^2}, v \in V_{Q_r^2}$. ◀

▶ **Theorem 12.** *Given a descendant leaf node s of r, the heuristic estimation $h(r) = BED^+(G_r^2, Q_r^2)$ is admissible; that is, $h(r) \leq g(s) - g(r)$, where $g(\cdot)$ gives the incurred cost from the root node to the currently considered node.*

Proof. For ease of proof, we insert dummy vertices and edges into G to transform it to a complete graph with $(|V_G| + |V_Q|)$ vertices. Similarly, we transform Q to a complete graph that also has $(|V_G| + |V_Q|)$ vertices.

Consider an internal node $r = \{(u_1 \to v_{j_1}), \ldots, (u_\ell \to v_{j_\ell})\}$ in the search tree, where v_{j_k} is the mapped vertex of u_k, for $1 \leq k \leq \ell$. For easy presentation, hereafter we use $r(u_k)$ to denote u_k's mapped vertex, i.e., $v_{j_k} = r(u_k)$. Given a descendent leaf node s (i.e., s is a complete vertex mapping from G to Q) of r, then the incurred cost of s is

$$\begin{aligned} g(s) &= \sum_{u \in V_G} c(u \to s(u)) \ + \ \frac{1}{2} \sum_{u \in V_G} \sum_{u' \in V_G \backslash \{u\}} c(e(u,u') \to e(s(u), s(u'))) \\ &= \sum_{u \in V_G} c(u \to s(u)) \ + \sum_{e(u,u') \in \binom{V_G}{2}} c(e(u,u') \to e(s(u), s(u'))) \end{aligned} \tag{9}$$

As we know, r induces an edit path transforming G_r^1 to Q_r^1, where G_r^1 and Q_r^1 are the already mapped subgraphs of G and Q, respectively, and $V_{G_r^1} = \{u_1, \ldots, u_\ell\}$ and $V_{Q_r^1} = \{r(u_1), \ldots, r(u_\ell)\}$. According to (9), we know that

$$g(r) = \sum_{u \in V_{G_r^1}} c(u \to r(u)) \ + \sum_{e(u,u') \in \binom{V_{G_r^1}}{2}} c(e(u,u') \to e(r(u), r(u')))$$

Let $\omega = s \backslash r$ be the partial mapping that contains the vertex mapping pairs belong to s but not r; namely, $\omega = \{(u \to s(u)) : u \in V_G \backslash V_{G_r^1}\}$. We can obtain that

$$\begin{aligned} g(s) - g(r) &= \sum_{u \in V_G} c(u \to s(u)) \ + \sum_{e(u,u') \in \binom{V_G}{2}} c(e(u,u') \to e(s(u), s(u'))) \\ &\qquad - \left\{ \sum_{u \in V_{G_r^1}} c(u \to r(u)) \ + \sum_{e(u,u') \in \binom{V_{G_r^1}}{2}} c(e(u,u') \to e(r(u), r(u'))) \right\} \\ &= \sum_{u \in V_{G_r^2}} c(u \to \omega(u)) \ + \sum_{e(u,u') \in \binom{V_{G_r^2}}{2}} c(e(u,u') \to e(\omega(u), \omega(u'))) \\ &\qquad + \sum_{u \in V_{G_r^2}} \sum_{u' \in v_{G_r^1}} c(e(u,u') \to e(\omega(u), r(u'))) \\ &= \sum_{u \in V_{G_r^2}} \left\{ c(u \to \omega(u)) + \frac{1}{2} \sum_{u' \in V_{G_r^2} \backslash \{u\}} c(e(u,u') \to e(\omega(u), \omega(u'))) \right. \\ &\qquad \left. + \sum_{u' \in V_{G_r^1}} c(e(u,u') \to e(\omega(u), r(u'))) \right\} \\ &= \sum_{u \in V_{G_r^2}} f_B^+(u, \omega(u)) \geq \min_{\rho : V_{G_r^2} \to V_{Q_r^2}} f_B^+(u, \rho(u)) = BED^+(G_r^2, Q_r^2) = h(r). \end{aligned}$$

where $V_{G_r^2} = V_G \backslash V_{G_r^1}$ and $V_{Q_r^2} = V_Q \backslash V_{Q_r^1}$. The second equality is due to $\binom{V_G}{2} = \binom{V_{G_r^1}}{2} \cup \binom{V_{G_r^2}}{2} \cup (V_{G_r^1} \times V_{G_r^2})$ when V_G is partitioned into two disjoint sets $V_{G_r^1}$ and $V_{G_r^2}$. ◀

We give an example in Appendix B of computing three GED lower bounds: LED, HED and BED. The same optimization that produces BED+ from BED can be applied to LED and HED to achieve enhanced heuristics LED+ and HED+, but we do not include them in this paper.

4.3 Algorithm

In this section, we show how to incorporate the heuristic estimation BED^+ into the anytime-based GED computation algorithm [2]. The reason we consider the anytime-based algorithm is that it is flexible and can control the algorithm to output tighter and tighter GED upper bounds until the exact GED value by setting more and more running time.

Algorithm 1 gives the anytime-based algorithm for computing the GED, where $t_{\max}$ is the user-defined maximum running time. We perform a depth-first search over the GED search tree of G and Q to find better and better GED upper bounds until the running time $\#t$ reaches $t_{\max}$. To accomplish this, we first employ the BP [25] algorithm to fast compute an initial GED upper bound ub; then, we adopt a stack $\mathcal{S}$ to finish the depth-first search. Each time we pop a node q from $\mathcal{S}$. If q is a leaf node, then we find a better solution. Otherwise, we call *procedure* BasicGenSuccr [9] (see Appendix C) to generate q's successors and then insert them into $\mathcal{S}$. During this process, we adopt the *branch-and-bound* strategy to prune the useless space: for each successor r, if $g(r) + h(r) \geq ub$, we can safely prune it, where $h(r) = BED^+(G_r^2, Q_r^2)$ is defined in (8).

Algorithm 1 Anytime-based GED computation.

Input : G and Q, and $t_{\max}$
Output : best GED upper bound for $t_{\max}$

```
1  initialize ub with the BP algorithm
2  root ← {}, S ← {}
3  S.push(root)
4  while S ≠ ∅ and #t ≤ t_max do
5  |  q ← S.pop()
6  |  if q is a complete mapping then
7  |  |  ub ← g(q), export ub and q
8  |  |  continue
9  |  succ ← BasicGenSuccr(q)
10 |  foreach r ∈ succ do
11 |  |  if g(r) + h(r) < ub then
12 |  |  |  S.push(r)
13 export ub and q
```

5 Experiments

5.1 Datasets and settings

Datasets. We chose four real (GREC, MUTA, PRO, and CMU) and one synthetic (SYN) datasets in the experiments. The datasets GREC, MUTA, and PRO were taken from the *IAM Graph Database Repository* [24]; the CMU dataset could be found at the *CMU website* [13]; and the SYN dataset was generated by the synthetic graph generator *GraphGen* [12]. Following the same procedure in [2, 21], we selected some subsets of GREC, MUTA, and PRO as the tested datasets, respectively, where each subset consists of graphs that have the same

Table 1 Summary of characteristics of datasets and cost functions used.

Dataset	#Graphs	$\|V\|$	$\|E\|$	vertex labels	edge labels	c_v	c_e	α	c_{vs}	c_{es}
GREC	40	12.5	17.5	(x, y) coord.	Line type	90	15	0.5	Ext. ED	Dirac
MUTA	70	40	41.5	Chem. symbol	Valence	11	1.1	0.25	Dirac	Dirac
PRO	30	30	58.6	Type/AA-seq.	Type/length	11	1	0.75	Ext. SED	Dirac
CMU	111	30	79.1	None	Distance	∞	–	0.5	0	L1 norm
SYN	100	14.5	20	Symbol	Symbol	0.3	0.5	0.75	Dirac	Dirac

number of vertices. Specifically, the subsets of CREC contain 5, 10, 15, and 20 vertices, respectively; the subsets of MUTA contain 10, 20, ..., 70 vertices, respectively; the subsets of PRO contain 20, 30, and 40 vertices, respectively; and each subset consists of 10 graphs.

Table 1 summarizes the characteristic and applied cost function of each dataset. ED and SED are short for *Euclidean distance* and *string edit distance* functions, respectively. c_v is the cost of inserting/deleting a vertex; c_e is the cost of inserting/deleting an edge; c_{vs} and c_{es} are the costs of substituting a vertex and an edge, respectively. In addition, we introduce a parameter α to control whether edit operations on vertices or edges are more important.

Settings. We conducted all the experiments on a HP Z800 PC running the Ubuntu 12.04 LTS operating system and equipped with a 2.67GHz CPU and 24 GB of memory. We implemented the algorithm in C++, using -O3 to compile and run it.

5.2 Evaluation metrics

We discuss two metrics to evaluate algorithm performance: *deviation* (*dev*) [1] and *solvable ratio* (*sr*) [9]. The metric *dev* measures the deviation generated by an algorithm. Formally, given two graphs G and Q, the deviation of the two graphs can be computed as $deviation(G,Q) = |dis(G,Q) - R(G,Q)|/R(G,Q)$, where $dis(G,Q)$ is the (approximate) GED value produced by the algorithm, and $R(G,Q)$ is the best GED value produced in all the experiments done on the graph database repository in [1]. Based upon the pairwise comparison model, the deviation on the dataset $\mathcal{G}$ can be computed as

$$dev = \frac{1}{|\mathcal{G}| \times |\mathcal{G}|} \sum\nolimits_{G \in \mathcal{G}} \sum\nolimits_{Q \in \mathcal{G}} deviation(G,Q) \tag{10}$$

The metric *sr* measures how often the exact GED value is obtained when reaching the maximum running time threshold $t_{\max}$. Formally, let $slove(G,Q)$ indicate whether an algorithm outputs the exact GED of G and Q within $t_{\max}$ time; in other words, if the algorithm requires less than $t_{\max}$ time to output the GED, $slove(G,Q) = 1$; otherwise, $slove(G,Q) = 0$. The solvable ratio (*sr*) on the dataset $\mathcal{G}$ can be computed as

$$sr = \frac{1}{|\mathcal{G}| \times |\mathcal{G}|} \sum\nolimits_{G \in \mathcal{G}} \sum\nolimits_{Q \in \mathcal{G}} slove(G,Q) \tag{11}$$

Obviously, a smaller *dev* and a larger *sr* reflects a better performance of an algorithm.

5.3 Experimental results

As described earlier in this paper, we first analyzed the relation of three GED lower bounds (i.e., LED, HED and BED). Then based upon BED we proposed an efficient heuristic estimation BED$^+$. Thus, it is necessary to evaluate the contribution of these lower bounds to the GED computation.

5.3.1 Tightness of LED, HED and BED

In this section, we evaluate the tightness of three GED lower bounds LED, HED and BED as well as their running time. Table 2 shows the obtained results, where the abbreviation "ms" represents milliseconds.

As shown in Table 2, BED achieves the smallest *dev*, which means that BED is closest to the exact GED value. This result is consistent with the analysis in Section 3, i.e., $BED(G, Q) \geq HED(G, Q)$ and $BED(G, Q) \geq LED(G, Q)$ for any two graphs G and Q. We also find in most cases tthat LED performs better than HED; the reason is that HED allows multiple assignments between vertices of G and Q and greedily selects matched vertices with the lowest cost.

We also list the running time of each method in Table 2. It can be seen from this table that HED runs faster than LED and LED runs faster than BED. The reason is that HED runs in quadratic time, while both LED and BED run in cubic time. LED independently considers the cost of substituting vertices and edges and ignores the structures, thus it has a better running time than BED.

Table 2 Deviation (%) and running time (ms) of LED, HED, and BED.

Datasets	LED		HED		BED	
	dev	*time*	*dev*	*time*	*dev*	*time*
GREC	4.41	0.38	17.45	**0.29**	**3.54**	0.52
MUTA	12.54	1.07	30.13	**0.47**	**11.49**	2.72
PRO	4.61	3.75	21.31	**2.84**	**3.25**	6.07
CMU	61.56	9.53	57.6	**3.77**	**25.1**	13.2
SYN	67.41	0.28	91.92	**0.14**	**46.61**	0.45

5.3.2 Effect of heuristic

Observing that BED produces the tighter lower bound than LED and HED, we propose BED$^+$ as a heuristic estimation to improve the GED computation. To achieve the comparison, we adopted LED, HED, BED, and BED$^+$ as the heuristic estimations, respectively, and fixed the running time $t_{\max} = 10^4$ ms. Table 3 lists the obtained deviation *dev* and solvable ration *sr*.

Table 3 Deviation (%) and solvable ratio (%) of of LED, HED, BED, and BED$^+$.

Datasets	LED		HED		BED		BED$^+$	
	dev	*sr*	*dev*	*sr*	*dev*	*sr*	*dev*	*sr*
GREC	0.36	69.87	0.56	54.38	0.22	67.88	**0.01**	**90**
MUTA	5.56	3.47	4.85	3.27	4.49	3.51	**2.6**	**22.33**
PRO	1.27	4	0.71	3.56	0.68	4.22	**0.06**	**4.22**
CMU	109.89	19.06	49.18	19.06	52.83	31.16	**2.97**	**75.79**
SYN	8.79	8.4	9.48	4.18	7.96	7.48	**0.17**	**94.34**

From Table 3, we know that using BED$^+$ as a heuristic can produce the smallest *dev*. This is due to the fact that BED$^+$ produces a higher estimated bound. For the solvable ratio *sr*, we also find that BED$^+$ achieves the best performance.

We also varied the running time $t_{\max}$ from 10^1 ms to 10^5 ms in order to evaluate the above heuristic estimations under different running times. From Figure 1, as the running time $t_{\max}$ increases, using the above four heuristic estimations we obtain lower and lower

deviation *dev* as well as higher and higher solvable ratio *sr*. Also, we find in most cases that BED$^+$ achieves the best *dev* and *sr* under both small (e.g., 10^2 ms) and large (e.g., 10^5 ms) running times. Compared with the widely used heuristic estimation LED, using BED$^+$ can decrease the *dev* by 72.2%, 34.1%, 48.1%, 39.4%, and 52.1% on average on the GREC, MUTA, PRO, CMU, and SYN datasets, respectively. Using BED$^+$ can increase the *sr* by 54.3%, 293.2%, 3.4%, 113.1%, and 702.8% on average on the five above datasets, respectively. Thus, we conclude that using BED$^+$ as a heuristic estimation can greatly improve the GED computation.

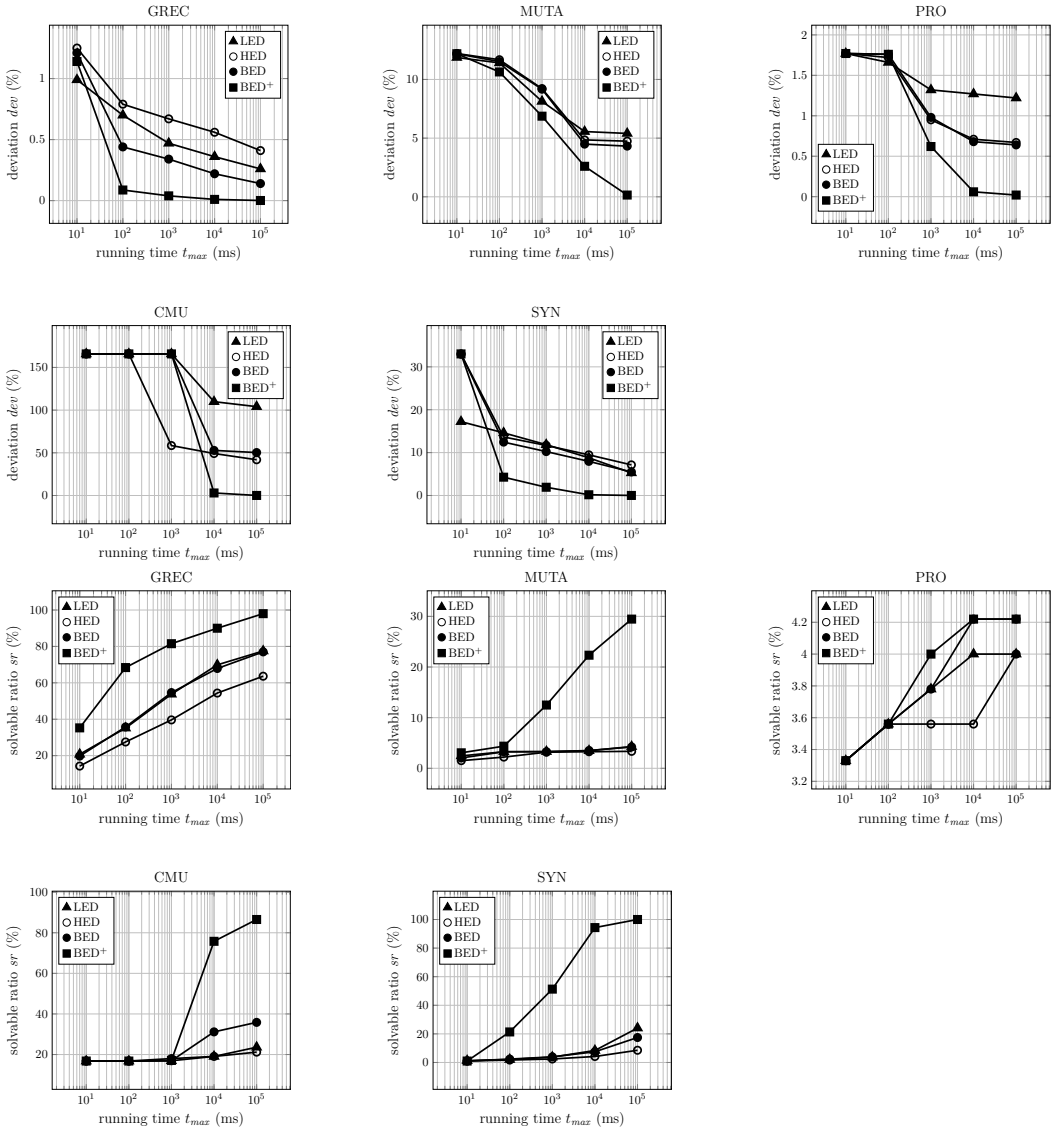

Figure 1 The *dev* (top two rows) and *sr* (bottom two rows) under different running time.

6 Conclusion and future works

In this paper, we analyze the relationship among three state-of-the-art GED lower bounds that are widely used as heuristic estimations in the tree-based search algorithm for the GED computation. Specifically, we demonstrate that $BED(G, Q) \geq LED(G, Q)$ and $BED(G, Q) \geq$

$HED(G, Q)$ for any two undirected graphs G and Q. Furthermore, based upon BED we propose an efficient heuristic estimation BED^+ and demonstrate that BED^+ still estimates a cost that is not greater than the real cost. Experimental results on four real and one synthetic datasets confirm that BED^+ can achieve the best performance under both small and large running time.

When calculating the heuristic estimation $BED^+(G_r^2, Q_r^2)$, we first compute the transformation cost (i.e., $f_B(\cdot, \cdot)$) of two compared branch structures. In fact, the transformation cost of these two branch structures may have been calculated many times in the previous traversal of the search tree. Future work will consider how to build a suitable index structure to maintain the transformation cost of these traversed branch structures in order to accelerate the computation of $BED^+(G_r^2, Q_r^2)$.

References

1 Z. Abu-Aisheh, R. Raveaux, and J. Y. Ramel. A graph database repository and performance evaluation metrics for graph edit distance. In *GbRPR*, pages 138–147, 2015.

2 Z. Abu-Aisheh, R. Raveaux, and J. Y. Ramel. Anytime graph matching. *Pattern Recogn Lett.*, 84:215–224, 2016. doi:10.1016/J.PATREC.2016.10.004.

3 Z. Abu-Aisheh, R. Raveaux, J. Y. Ramel, and P. Martineau. An exact graph edit distance algorithm for solving pattern recognition problems. In *ICPRAM*, pages 271–278, 2015.

4 D. B. Blumenthal and J. Gamper. Exact computation of graph edit distance for uniform and non-uniform metric edit costs. In *GbRPR*, pages 211–221, 2017.

5 D. B. Blumenthal and J. Gamper. Improved lower bounds for graph edit distance. *IEEE Trans. Knowl Data Eng.*, 30(3):503–516, 2018. doi:10.1109/TKDE.2017.2772243.

6 D. B. Blumenthal and J. Gamper. On the exact computation of the graph edit distance. *Pattern Recogn Lett.*, 134:46–57, 2020. doi:10.1016/J.PATREC.2018.05.002.

7 B. Bonet and H. Geffner. Planning as heuristic search. *Artif. Intell.*, 129(1-2):5–33, 2001. doi:10.1016/S0004-3702(01)00108-4.

8 L. Chang, X. Feng, X. Lin, L. Qin, and W. Zhang. Efficient graph edit distance computation and verification via anchor-aware lower bound estimation. *CoRR*, 2017. arXiv:1709.06810.

9 X. Chen, H. Huo, J. Huan, and J. S. Vitter. An efficient algorithm for graph edit distance computation. *Knowl.-Based Syst.*, 163:762–775, 2019. doi:10.1016/J.KNOSYS.2018.10.002.

10 X. Chen, H. Huo, J. Huan, J. S. Vitter, W. Zheng, and L. Zou. MSQ-Index: A succinct index for fast graph similarity search. *IEEE Trans. Knowl Data Eng.*, 33(6):2654–2668, 2021. doi:10.1109/TKDE.2019.2954527.

11 X. Chen, Y. Wang, H. Huo, and J. S. Vitter. An efficient heuristic for graph edit distance [source code], June 2019. URL: https://github.com/Hongweihuo-Lab/Heur-GED.

12 James Cheng, Yiping Ke, and Wilfred Ng. GraphGen — a synthetic graph data generator. URL: https://cse.hkust.edu.hk/graphgen/.

13 CMU house and hotel datasets. URL: https://github.com/dbblumenthal/gedlib/blob/master/data/datasets/CMU-GED.

14 X. Cortés and F. Serratosa. Learning graph-matching edit-costs based on the optimality of the oracle's node correspondences. *Pattern Recogn Lett.*, 56:22–29, 2015. doi:10.1016/J.PATREC.2015.01.009.

15 A. Fischer, R. Plamondon, Y. Savaria, K. Riesen, and H. Bunke. A Hausdorff heuristic for efficient computation of graph edit distance. *Structural, Syntactic, and Statistical Pattern Recognition*, LNCS 8621:83–92, 2014.

16 A. Fischer, C. Y. Suen, V. Frinken, K. Riesen, and H. Bunke. Approximation of graph edit distance based on Hausdorff matching. *Pattern Recogn.*, 48(2):331–343, 2015. doi:10.1016/J.PATCOG.2014.07.015.

17 K. Gouda and M. Hassaan. CSI_GED: An efficient approach for graph edit similarity computation. In *ICDE*, pages 256–275, 2016.

18 D. Justice and A. Hero. A binary linear programming formulation of the graph edit distance. *IEEE Trans. Pattern Anal Mach Intell.*, 28(8):1200–1214, 2006. doi:10.1109/TPAMI.2006.152.

19 J. Kim. Efficient graph edit distance computation using isomorphic vertices. *Pattern Recogn Lett.*, 168(2023):71–78, 2023. doi:10.1016/J.PATREC.2023.03.002.

20 H.W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955. doi:10.1002/nav.3800020109.

21 J. Lerouge, Z. Abu-Aisheh, R. Raveaux, P. Héroux, and S. Adam. New binary linear programming formulation to compute the graph edit distance. *Pattern Recogn.*, 72:254–265, 2017. doi:10.1016/J.PATCOG.2017.07.029.

22 J. Liu, M. Zhou, S. Ma, and L. Pan. MATA*: Combining learnable node matching with A* algorithm for approximate graph edit distance computation. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM '23)*, pages 1503–1512, 2023.

23 Chengzhi Piao, Tingyang Xu, Xiangguo Sun, Yu Rong, Kangfei Zhao, and Hong Cheng. Computing graph edit distance via neural graph matching. *Proceedings of the VLDB Endowment*, 16(8):1817–1829, 2023. doi:10.14778/3594512.3594514.

24 K. Riesen and H. Bunke. IAM graph database repository for graph based pattern recognition and machine learning. *Structural, Syntactic, and Statistical Pattern Recognition*, pages 287–297, 2008.

25 K. Riesen and H. Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image Vision Comput.*, 27(7):950–959, 2009. doi:10.1016/J.IMAVIS.2008.04.004.

26 K. Riesen, S. Emmenegger, and H. Bunke. A novel software toolkit for graph edit distance computation. In *GbRPR*, pages 142–151, 2013.

27 K. Riesen, S. Fankhauser, and H. Bunke. Speeding up graph edit distance computation with a bipartite heuristic. In *MLG*, pages 21–24, 2007.

28 S. Russell and P. Norvig. *Artificial Intelligence: a Modern Approach (2nd ed.)*. Prentice-Hall, New Jersey, USA, 2002.

29 O Schütze, X. Esquivel, A. Lara, and C. A. C. Carlos. Using the averaged Hausdorff distance as a performance measure in evolutionary multiobjective optimization. *IEEE Trans. Evol. Comput.*, 16(4):504–522, 2012. doi:10.1109/TEVC.2011.2161872.

30 Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou. Comparing stars: On approximating graph edit distance. *PVLDB*, 2(1):25–36, 2009. doi:10.14778/1687627.1687631.

31 W. Zheng, L. Zou, X. Lian, D. Wang, and D. Zhao. Efficient graph similarity search over large graph databases. *IEEE Trans. Knowl Data Eng.*, 27(4):964–978, 2015. doi:10.1109/TKDE.2014.2349924.

A Proof of Lemma 8

Proof. We prove this lemma by considering the following two cases:

Case I. when $u = \varepsilon$ or $v = \varepsilon$. We first discuss the case $u = \varepsilon$. It is trivial to know that $f_H(u, \varepsilon) = f_B(u, \varepsilon) = c(u \to \varepsilon) + \frac{1}{2}\sum_{e \in N_u} c(e \to \varepsilon)$. Similarly, when $v = \varepsilon$, we also have $f_H(u, v) = f_B(u, v)$. Thus, when $u = \varepsilon$ or $v = \varepsilon$, the lemma follows.

Case II. when $u \neq \varepsilon$ and $v \neq \varepsilon$. Then, we know that

$$f_B(u, v) = c(u \to v) + \frac{1}{2} MLS(N_u, N_v), \tag{12}$$

$$f_H(u, v) = \frac{1}{2}\left\{ c(u \to v) + \frac{1}{2} HED(N_u, N_v) \right\} \tag{13}$$

where $MLS(N_u, N_v) = \min_{\varrho: N_u^\varepsilon \to N_v^\varepsilon} \sum_{e \in N_u^\varepsilon} c(e \to \varrho(e))$, and ϱ is a bijection from N_u^ε to N_v^ε.

In order to prove $f_H(u,v) \leq \frac{1}{2} f_B(u,v)$, it suffices from (12) and (13) to prove

$$HED(N_u, N_v) \leq MLS(N_u, N_v),$$

which we do as follows:

(i) Rewriting $MLS(N_u, N_v)$ and $HED(N_u, N_v)$:

$$MLS(N_u, N_v) = \sum_{e \in N_u^\varepsilon} c(e \to \varrho_{\min}(e)) = \sum_{e \in N_u^\varepsilon} c(e \to y),$$

where $\varrho_{\min}$ is the *bijection* from N_u^ε to N_v^ε that $MLS(N_u, N_v)$ achieves the minimum value; $y = \varrho_{\min}(e) \in N_v^\varepsilon$ is e's mapped edge under the bijection $\varrho_{\min}$, for $\forall e \in N_u^\varepsilon$;

$$HED(N_u, N_v) = \sum_{e \in N_u^\varepsilon} \Big\{ f_H(e, \chi_1(e)) + f_H(\chi_2(y), y) \Big\},$$

where χ_1 is the *mapping* from N_u^ε to N_v^ε satisfying $\chi_1(e) = \arg\min_{e' \in N_v^\varepsilon} f_H(e, e')$, for $\forall e \in N_u^\varepsilon$; and χ_2 is the *mapping* from N_v^ε to N_u^ε satisfying $\chi_2(y) = \arg\min_{e \in N_u^\varepsilon} f_H(e, y)$.

(ii) Proving $f_H(e, \chi_1(e)) + f_H(\chi_2(y), y) \leq c(e \to y)$: According to the definition of χ_1 and χ_2, $f_H(e, \chi_1(e)) \leq \min\{f_H(e, y), f_H(e, \varepsilon)\}$ and $f_H(\chi_2(y), y) \leq \min\{f_H(e, y), f_H(\varepsilon, y)\}$. We discuss the following cases (a)–(d):

- **(a)** When $e = \varepsilon$ and $y = \varepsilon$, then $f_H(e, \chi_1(e)) + f_H(\chi_2(y), y) \leq f_H(\varepsilon, \varepsilon) + f_H(\varepsilon, \varepsilon) = c(\varepsilon \to \varepsilon) = 0$;
- **(b)** When $e \neq \varepsilon$ and $y = \varepsilon$, then $f_H(e, \chi_1(e)) + f_H(\chi_2(y), y) \leq f_H(e, \varepsilon) + f_H(\varepsilon, \varepsilon) = c(e \to \varepsilon)$;
- **(c)** When $e = \varepsilon$ and $y \neq \varepsilon$, the analysis is similar to that of (b);
- **(d)** When $e \neq \varepsilon$ and $y \neq \varepsilon$, then $f_H(e, \chi_1(e)) + f_H(\chi_2(y), y) \leq f_H(e, y) + f_H(e, y) = 2 f_H(e, y) = 2 \times \frac{1}{2} c(e \to y) = c(e \to y)$.

(iii) Combining both (i) and (ii), we have

$$HED(N_u, N_v) = \sum_{e \in N_u^\varepsilon} \Big\{ f_H(e, \chi_1(e)) + f_H(\chi_2(y), y) \Big\} \leq \sum_{e \in N_u^\varepsilon} c(e \to y) = MLS(N_u, N_v)$$

Therefore, $f_H(u,v) \leq \frac{1}{2} f_B(u,v)$ when $u \neq \varepsilon$ and $v \neq \varepsilon$. This completes the proof. ◀

B Examples of computing LED, HED and BED

In this section, we give an example of calculating three GED lower bounds, LED, HED and BED.

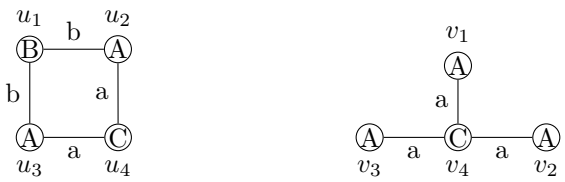

Figure 2 Graphs G (left) and Q (right).

Figure 2 shows two graphs G and Q, where "A", "B" and "C" denote vertex labels, and "a" and "b" denote edge labels. Consider the cost function c satisfying: (i) the cost of each vertex edit operation is 2, that is, $c(u \to v) = 2$ when two vertices $u \in V_G^\varepsilon$ and $v \in V_Q^\varepsilon$ have different labels, and $c(u \to v) = 0$ otherwise; (ii) the cost of each edge edit operation is 1, that is, $c(e_1 \to e_2) = 1$ when two edges $e_1 \in E_G^\varepsilon$ and $e_2 \in E_Q^\varepsilon$ have different labels, and $c(e_1 \to e_2) = 0$ otherwise. Based upon this cost function c, we discuss how to compute $LED(G, Q)$, $HED(G, Q)$ and $BED(G, Q)$ below using the examples shown in Figure 2.

(1) Computing $LED(G, Q)$

In $LED(G, Q)$ (see Definition 2 in main text), we need to compute the minimum substitution cost of vertices and edges of G and Q, i.e., $\lambda_V(G, Q)$ and $\lambda_E(G, Q)$. For $\lambda_V(G, Q) = \min_{\phi: V_G^\varepsilon \to V_Q^\varepsilon} \sum_{u \in V_G^\varepsilon} c(u \to \phi(u))$, we seek for a bijection ϕ from V_G^ε to V_Q^ε to minimize the linear sum $\lambda_V(G, Q)$; this is a well-investigated *linear sum assignment problem* (LSAP) and can be solved by the Hungarian algorithm [20] through the following two steps:

(1) Construct the vertex substitution cost matrix W^V, such that $W^V_{u,v} = c(u \to v)$ is the cost of substituting vertices $u \in V_G$ and $v \in V_Q$; $W^V_{u,\varepsilon} = c(u \to \varepsilon)$ is the cost of deleting u; and $W^V_{\varepsilon,v} = c(\varepsilon \to v)$ is the cost of inserting v. In this example, we compute W^V as

$$W^V = \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ \varepsilon \\ \varepsilon \\ \varepsilon \\ \varepsilon \end{array} \begin{array}{c} \begin{array}{cccccccc} v_1 & v_2 & v_3 & v_4 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{array} \\ \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & \infty & \infty & \infty \\ 0 & 0 & 0 & 2 & \infty & 2 & \infty & \infty \\ 0 & 0 & 0 & 2 & \infty & \infty & 2 & \infty \\ 2 & 2 & 2 & 0 & \infty & \infty & \infty & 2 \\ 2 & \infty & \infty & \infty & 0 & 0 & 0 & 0 \\ \infty & 2 & \infty & \infty & 0 & 0 & 0 & 0 \\ \infty & \infty & 2 & \infty & 0 & 0 & 0 & 0 \\ \infty & \infty & \infty & 2 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

(2) Find the *optimal assignment* $\phi_{\min}$ that minimizes the linear sum on W^V. In this example, we find that $\phi_{\min} = \{(u_1 \to v_1), (u_2 \to v_2), (u_3 \to v_3), (u_4 \to v_4), (\varepsilon \to \varepsilon)\}$ is the optimal assignment, and then obtain $\lambda_V(G, Q) = W^V_{u_1,v_1} + W^V_{u_2,v_2} + W^V_{u_3,v_3} + W^V_{u_4,v_4} + W^V_{\varepsilon,\varepsilon} = 2$.

Similar to the above process, we can compute the edge substitution cost matrix W^E as follows:

$$W^E = \begin{array}{c} \\ e(u_1, u_2) \\ e(u_1, u_3) \\ e(u_2, u_4) \\ e(u_3, u_4) \\ \varepsilon \\ \varepsilon \\ \varepsilon \end{array} \begin{array}{c} \begin{array}{cccccccc} e(v_1, v_4) & e(v_2, v_4) & e(v_3, v_4) & \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 & \infty & \infty & \infty \\ 1 & 1 & 1 & \infty & 1 & \infty & \infty \\ 0 & 0 & 0 & \infty & \infty & 1 & \infty \\ 0 & 0 & 0 & \infty & \infty & \infty & 1 \\ 1 & \infty & \infty & 0 & 0 & 0 & 0 \\ \infty & 1 & \infty & 0 & 0 & 0 & 0 \\ \infty & \infty & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

With the Hungarian algorithm, we know that the optimal assignment on W^E is $\varphi_{\min} = \{(e(u_1, u_2) \to \varepsilon), (e(u_1, u_3) \to e(v_1, v_4)), (e(u_2, u_4) \to e(v_2, v_4)), (e(u_3, u_4) \to e(v_3, v_4)), (\varepsilon \to \varepsilon)\}$. Then, $\lambda_E(G, Q) = W^E_{e(u_1,u_2),\varepsilon} + W^E_{e(u_1,u_3),e(v_1,v_4)} + W^E_{e(u_2,u_4),e(v_2,v_4)} + W^E_{e(u_3,u_4),e(v_3,v_4)} + W^E_{\varepsilon,\varepsilon} = 2$. Combing $\lambda_V(G, Q)$ and $\lambda_E(G, Q)$, we have $LED(G, Q) = \lambda_V(G, Q) + \lambda_E(G, Q) = 2 + 2 = 4$.

(2) Computing $HED(G, Q)$

According to the definition of $HED(G, Q)$ (see Definition 3 in main text), we need to calculate the hausdorff matching cost $f_H(u, v)$ between two vertices $u \in V_G^\varepsilon$ and $v \in V_Q^\varepsilon$, and then perform a bidirectional matching between G and Q. When performing a matching from G to Q, we greedily seek for the minimum matching cost $\min_{v \in V_Q^\varepsilon} f_H(u, v)$ of each vertex u; then, the sum of these minimum costs is the matching cost from G to Q, i.e., $\sum_{u \in V_G} \min_{v \in V_Q^\varepsilon} f_H(u, v)$. Similarly, the matching cost from Q to G is $\sum_{v \in V_Q} \min_{u \in V_G^\varepsilon} f_H(u, v)$. Finally, the sum of the above two matching costs is $HED(G, Q)$. We can summarize the computation process of $HED(G, Q)$ as two steps:

(1) Construct the hausdorff matching cost matrix W^H, such that $W^H_{u,v} = f_H(u, v)$ is the hausdorff cost of matching vertex $u \in V_G$ to vertex $v \in V_Q$; $W^H_{u,\varepsilon} = f_H(u, \varepsilon)$ is the hausdorff cost of deleting u; and $W^H_{\varepsilon,v} = f_H(\varepsilon, v)$ is the hausdorff cost of inserting v, where $f_H(\cdot, \cdot)$ is defined in (2) in main text. In this example, we can compute W^H as

$$W^H = \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ \varepsilon \end{array} \begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & \varepsilon \end{array} \\ \begin{pmatrix} 1.375 & 1.375 & 1.375 & 1.625 & 3 \\ 0.125 & 0.125 & 0.125 & 1.125 & 3 \\ 0.125 & 0.125 & 0.125 & 1.125 & 3 \\ 1 & 1 & 1 & 0 & 3 \\ 2.5 & 2.5 & 2.5 & 3.5 & 0 \end{pmatrix} \end{array}$$

(2) Based upon W^H, compute $\sum_{u \in V_G} \min_{v \in V_Q^\varepsilon} W^H_{u,v}$ and $\sum_{v \in V_Q} \min_{u \in V_G^\varepsilon} W^H_{u,v}$. In this example, we trivially obtain $HED(G, Q) = \sum_{u \in V_G} \min_{v \in V_Q^\varepsilon} W^H_{u,v} + \sum_{v \in V_Q} \min_{u \in V_G^\varepsilon} W^H_{u,v} = (W^H_{u_1,v_1} + W^H_{u_2,v_1} + W^H_{u_3,v_1} + W^H_{u_4,v_4}) + (W^H_{u_2,v_1} + W^H_{u_2,v_2} + W^H_{u_3,v_2} + W^H_{u_4,v_4}) = (1.375 + 0.125 + 0.125 + 0) + (0.125 + 0.125 + 0.125 + 0) = 2$.

Note that when calculating $W^H_{u,v}$ (i.e., $f_H(u, v)$), we need to calculate $HED(N_u, N_v)$ (see Equation (3) in main context), where N_u and N_v are the sets of edges adjacent to u and v, respectively. The computation of $HED(N_u, N_v)$ is similar to the above process of computing $HED(G, Q)$; and thus, we omit the detailed calculation here.

(3) Computing $BED(G, Q)$

The process of calculating $BED(G, Q)$ is similar to that of calculating $\lambda_V(G, Q)$, which is also looking for a bijection ρ from V_G^ε to V_Q^ε to minimize the linear sum $\sum_{u \in V_G^\varepsilon} f_B(u, \rho(u))$. The computation contains two steps:

(1) Construct the branch matching cost matrix W^B, such that $W^B_{u,v} = f_B(u, v)$ is the branch cost of matching vertex $u \in V_G$ to vertex $v \in V_Q$; $W^B_{u,\varepsilon} = f_B(u, \varepsilon)$ is the branch cost of deleting u; and $W^B_{\varepsilon,v} = f_B(\varepsilon, v)$ is the branch cost of inserting v, where $f_B(\cdot, \cdot)$ is defined in (5) in main text. In this example, we can compute W^B as

$$W^B = \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ \varepsilon \\ \varepsilon \\ \varepsilon \\ \varepsilon \end{array} \begin{array}{c} \begin{array}{cccccccc} v_1 & v_2 & v_3 & v_4 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{array} \\ \begin{pmatrix} 3 & 3 & 3 & 3.5 & 3 & \infty & \infty & \infty \\ 0.5 & 0.5 & 0.5 & 3 & \infty & 3 & \infty & \infty \\ 0.5 & 0.5 & 0.5 & 3 & \infty & \infty & 3 & \infty \\ 2.5 & 2.5 & 2.5 & 0.5 & \infty & \infty & \infty & 3 \\ 2.5 & \infty & \infty & \infty & 0 & 0 & 0 & 0 \\ \infty & 2.5 & \infty & \infty & 0 & 0 & 0 & 0 \\ \infty & \infty & 2.5 & \infty & 0 & 0 & 0 & 0 \\ \infty & \infty & \infty & 3.5 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

(2) Find the *optimal assignment* $\rho_{\min}$ that minimizes the linear sum on W^B. In this example, we find that $\rho_{\min} = \{(u_1 \to v_1), (u_2 \to v_2), (u_3 \to v_3), (u_4 \to v_4), (\varepsilon \to \varepsilon)\}$ is the optimal assignment, and then obtain $BED(G,Q) = W^B_{u_1,v_1} + W^B_{u_2,v_2} + W^B_{u_3,v_3} + W^B_{u_4,v_4} + W^B_{\varepsilon,\varepsilon} = 4.5$.

Note that when calculating $W^B_{u,v}$ (i.e., $f_B(u,v)$), we need to calculate the minimum edge substitution cost between N_u and N_v, which is similar to the process of calculating $\lambda_E(,)$; and thus, we omit the detailed computation here.

For graphs G and Q in Figure 2, we finally obtain that $LED(G,Q) = 4$, $HED(G,Q) = 2$ and $BED(G,Q) = 4.5$. Clearly, $BED(G,Q) \geq LED(G,Q)$ and $BED(G,Q) \geq HED(G,Q)$.

C Successor generation

We discuss how to generate successors of each node in the GED search tree with Algorithm 2.

Consider an inner node $r = \{(u_1 \to v_{j_1}), \ldots, (u_\ell \to v_{j_\ell})\}$, where v_{j_k} is the mapped vertex of u_k in the GED search tree, for $1 \leq k \leq \ell$. BasicGenSuccr generates all the possible successors of r. First, we compute the sets of unmapped vertices in G and Q, respectively, i.e., $V^r_G = V_G \backslash \{u_1, \ldots, u_l\}$ and $V^r_Q = V_Q \backslash \{v_{j_1}, \ldots, v_{j_l}\}$. If $|V^r_G| > 0$, then we select a vertex z from $V^r_Q \cup \{\varepsilon\}$ as the mapped vertex of $u_{\ell+1}$, and consequently, obtain a successor *child* of r such that $child = r \cup \{(u_{\ell+1} \to z)\}$. Otherwise, all the vertices of G are processed; trivially, we obtain a leaf node $s = r \cup \bigcup_{z \in V^r_Q} \{(\varepsilon \to z)\}$.

Algorithm 2 BasicGenSuccr(r).

```
1  succ ← {}
2  V_G^r ← V_G \ {u_1, ..., u_ℓ}
3  V_Q^r ← V_Q \ {v_{j_1}, ..., v_{j_ℓ}}
4  if |V_G^r| > 0 then
5  |   foreach z ∈ V_Q^r ∪ {ε} do
6  |   |   child ← r ∪ {(u_{ℓ+1} → z)}
7  |   |   succ ← succ ∪ {child}
8  else
9  |   s ← r ∪ ⋃_{z ∈ V_Q^r} {(ε → z)}
10 |   succ ← succ ∪ {s}
11 return succ
```