Towards Efficient Graph Edit Distance Computation

Lijun Chang
The University of Sydney
lijun.chang@sydney.edu.au

Xing Feng
University of Technology Sydney
xing.feng@uts.edu.au

Xuemin Lin
University of New South Wales
lxue@cse.unsw.edu.au

Lu Qin
University of Technology Sydney
lu.qin@uts.edu.au

Wenjie Zhang
University of New South Wales
zhangw@cse.unsw.edu.au

ABSTRACT

Computing graph edit distance (GED), an important similarity measure between graphs, is a primitive operator in similarity-based graph data analysis. Partially due to the NP-hardness, the existing algorithms for GED computation are extremely slow, *e.g.*, they take more than one hour for graphs with 30 vertices. In order to speed up the GED computation, in this paper we systematically analyze the two key components — lower bound estimation and search strategy — of the branch-and-bound paradigm that is followed by the existing algorithms. Firstly, we propose a general anchor-aware technique to improve the tightness of the existing lower bound estimations, as well as algorithmic techniques to improve the efficiency of lower bound computation. Secondly, we illustrate that the best-first search strategy usually results in a smaller search space and a lower time complexity, and thus is better, than the depth-first search strategy when using the same lower bound estimation technique. Our extensive empirical studies (1) reveal that the tightness as well as the efficiency of lower bound estimation, which are largely ignored by the existing studies, have a significant impact on the efficiency of GED computation, and (2) validate that the best-first search strategy is better than the depth-first search strategy for GED computation. In particular, our AStar$^+$-BMao approach outperforms the state-of-the-art algorithms by more than four orders of magnitude. Finally, we also show that the above results carry over to the problem of GED verification.

PVLDB Reference Format:
Lijun Chang, Xing Feng, Xuemin Lin, Lu Qin, Wenjie Zhang. Towards Efficient Graph Edit Distance Computation. *PVLDB*, 12(xxx): xxxx-yyyy, 2019.
DOI: https://doi.org/TBD

1. INTRODUCTION

Graph model is ubiquitous and has been used to model the connectedness/relationships among entities in a wide spectrum of applications. Recent decades have witnessed significant research efforts towards many fundamental problems in managing and analyzing graph data. Among them, computing the similarity between graphs is a fundamental and essential operation in many applications. For example, it serves as a key building block in graph classification [20], graph clustering [25], and similarity search/join of graph databases [29, 30]. In addition, it also assists to identify functionally related enzyme clusters in biochemistry [21], to compare electroencephalogram in medicine [3], and to retrieve similar objects in videos [7].

Besides various similarity measures such as the maximum common subgraph [6, 10] and the number of miss-matching edges [32], the *graph edit distance* (GED) [11, 26, 29] has also been shown an important similarity measure since it gives the minimum amount of distortion needed to transform one graph into the other and it is a metric. As illustrated in [13], the GED between graphs q and g, denoted $\delta(q, g)$, equals the minimum value among the *editorial costs* of all vertex mappings from q to g, where the editorial cost of a mapping is the number of *edit operations* involved to transform q to g by obeying the mapping. Here, the edit operations are edge insertion/deletion/relabeling and vertex insertion/deletion/relabeling; note that, a vertex can be deleted only when having no adjacent edges, and inserting (deleting) an edge (or a vertex) together with its label is regarded as a unit cost. Consider q in Figure 1(a) and g in Figure 1(b). The mapping $\{v_1 \mapsto u_1, v_2 \mapsto u_2, v_3 \mapsto u_3, v_4 \mapsto u_4\}$ has an editorial cost of 3, *i.e.*, relabeling edge (v_1, v_2) to have label a, deleting edge (v_1, v_3), and inserting edge (v_2, v_4) to have the same label as edge (u_2, u_4); note that, the editorial cost is 3 regardless of the label of edge (u_2, u_4). Furthermore, it can be verified that this mapping has the minimum editorial cost among all mappings; thus, the GED between q and g is $\delta(q, g) = 3$.

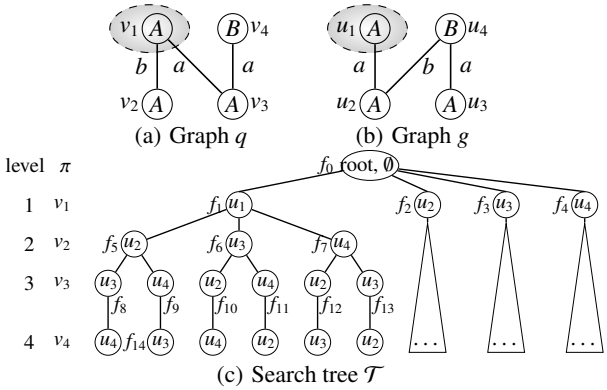


Figure 1: GED computation

As computing GED is NP-hard [28], the existing approaches follow the *branch-and-bound paradigm* [1, 5, 11, 23, 24, 29, 30], by conducting pruned searches on *a search tree* that represents all vertex mappings from q to g in a prefix-shared manner according

Proceedings of the VLDB Endowment, Vol. 12, No. xxx

DOI: https://doi.org/TBD

	$\delta^{\mathtt{LS}}(\cdot)$	$\delta^{\mathtt{LSa}}(\cdot)$	$\delta^{\mathtt{BM}}(\cdot)$	$\delta^{\mathtt{BMa}}(\cdot)$	$\delta^{\mathtt{BMo}}(\cdot)$	$\delta^{\mathtt{BMao}}(\cdot)$
AStar$^+$	AStar$^+$-LS (improves A*GED [24])	AStar$^+$-LSa	AStar$^+$-BM	AStar$^+$-BMa	AStar$^+$-BMo	AStar$^+$-BMao
DFS$^+$	DFS$^+$-LS (improves DF_GED [5])	DFS$^+$-LSa	DFS$^+$-BM	DFS$^+$-BMa	DFS$^+$-BMo	DFS$^+$-BMao

Table 1: Our GED computation algorithms

to a matching order of vertices of q. Given the matching order $\pi = (v_1, v_2, v_3, v_4)$, the search tree $\mathcal{T}$ for computing $\delta(q, g)$ in Figures 1(a) and 1(b) is shown in Figure 1(c). In the search tree $\mathcal{T}$, each node at level i represents a (partial) mapping from $(v_1, \ldots, v_i)$ to vertices of g, and all full mappings are represented by the leaf nodes. To avoid an exhaustive search of all mappings in $\mathcal{T}$, *a lower bound cost* is computed for each partial mapping f, denoted $\delta(f)$, which is a lower bound of the editorial cost of all full mappings that extend f (*i.e.*, are descendants of f).

Motivation. The branch-and-bound paradigm, as followed by the existing GED computation algorithms, has two key components: *search strategy* and *lower bound estimation*. The existing studies mainly focus on investigating the search paradigms while keeping the lower bound estimation as a silent feature. In particular, the first algorithm for GED computation, denoted A*GED [23, 24, 29, 30], uses the *best-first search* strategy, while recent studies such as DF_GED [1, 5] and CSI_GED [11] suggest conducting a *depth-first search*. It is reported in [11] that CSI_GED outperforms A*GED by over two orders of magnitude. However, (1) it is unclear whether this is due to the different search strategies or due to the different lower bound estimations; note that, A*GED and DF_GED use the *label set-based lower bound*, denoted $\delta^{\mathtt{LS}}(\cdot)$, while CSI_GED silently uses the *degree-based lower bound*. Moreover, (2) there lacks a systematical study of the impact of different lower bound estimations on the efficiency of GED computation.

Contributions. In this paper, we systematically investigate lower bound estimations and search strategies for speeding up GED computation. Our main contributions are summarized as follows.

① *Anchor-aware Lower Bound Estimation.* We develop a general anchor-aware technique to improve the tightness of the existing lower bound $\delta^{\mathtt{LS}}(\cdot)$ which is used in A*GED [23, 24, 29, 30] and DF_GED [1, 5]. Denote our improved lower bound by $\delta^{\mathtt{LSa}}(\cdot)$. We illustrate that $\delta^{\mathtt{LSa}}(f) \geq \delta^{\mathtt{LS}}(f)$ holds for every partial mapping f, and our experimental results show that $\delta^{\mathtt{LSa}}(\cdot)$ reduces the search space of $\delta^{\mathtt{LS}}(\cdot)$ by more than one order of magnitude. Furthermore, we also propose efficient techniques to compute the lower bound costs of *all children* of a partial mapping regarding $\delta^{\mathtt{LS}}(\cdot)$ and $\delta^{\mathtt{LSa}}(\cdot)$ in $O(|E(q)| + |E(g)|)$ total time; note that, the existing techniques (*i.e.*, A*GED and DF_GED) take $O\big(|V(q)| \times (|E(q)| + |E(g)|)\big)$ time. As a result, our best-first search algorithm AStar$^+$-LSa that uses the lower bound $\delta^{\mathtt{LSa}}(\cdot)$ outperforms the state-of-the-art algorithm CSI_GED by more than four orders of magnitude.

② *Tighter Lower Bound Estimation to Reduce Search Space.* To further reduce the search space and thus space consumption of best-first search algorithms, we propose tighter lower bound estimations that have higher time complexities to compute. Firstly, we adopt the *branch match-based lower bound* [31], denoted $\delta^{\mathtt{BM}}(\cdot)$, and improve it by our anchor-aware technique, denoted $\delta^{\mathtt{BMa}}(\cdot)$; we prove that $\delta^{\mathtt{BMa}}(f) \geq \delta^{\mathtt{LSa}}(f)$ holds for every partial mapping f. Then, to strike a balance between the tightness and the efficiency of lower bound estimation, we slightly loose the lower bounds $\delta^{\mathtt{BM}}(\cdot)$ and $\delta^{\mathtt{BMa}}(\cdot)$ into $\delta^{\mathtt{BMo}}(\cdot)$ and $\delta^{\mathtt{BMao}}(\cdot)$, respectively. As a result, our best-first search algorithm AStar$^+$-BMao that uses the lower bound $\delta^{\mathtt{BMao}}(\cdot)$ runs faster than the one using the lower bound $\delta^{\mathtt{BMa}}(\cdot)$. Moreover, AStar$^+$-BMao scales better due to having a smaller search space, than AStar$^+$-LSa.

③ *Contrast Best-first Search with Depth-first Search.* We develop a unified framework that can be instantiated into either a best-first search approach AStar$^+$ or a depth-first search approach DFS$^+$. Based on this unified framework, we then quantitatively contrast AStar$^+$ with DFS$^+$, and show that AStar$^+$ has a smaller search space and a lower time complexity than DFS$^+$ when using the same lower bound estimation technique; moreover, the tighter the lower bound estimation, the smaller the search space of AStar$^+$.

④ *Extensive Performance Studies.* Each combination of the search strategies with our lower bound estimations constitutes a GED computation algorithm, as summarized in Table 1. We conduct extensive performance studies of these algorithms. The results validate that AStar$^+$ *performs better than* DFS$^+$ *when using the same lower bound estimation technique*; this debunks the recent claims in [5, 11] that depth-first search is more suitable than best-first search for GED computation. The results also confirm that *our anchor-aware technique significantly improves the tightness of lower bound estimation* and thus reduces the search space, and our anchor-aware branch match-based lower bound $\delta^{\mathtt{BMa}}(\cdot)$ results in the smallest search space. Nevertheless, there needs a balance between the tightness and the efficiency of lower bound estimation, to achieve the fastest overall running time for GED computation. As a result, AStar$^+$-LSa runs the fastest, while AStar$^+$-BMao runs slightly slower but scales better due to having a significantly smaller search space. In conclusion, we recommend to use AStar$^+$-BMao for GED computation.

In addition, our experimental results show that AStar$^+$-BMao outperforms the state-of-the-art techniques CSI_GED and DF_GED by more than four orders of magnitude; note that, DFS$^+$-LS is our improved version of DF_GED by proposing a more efficient lower bound computation algorithm. It is also interesting to observe that our AStar$^+$-LS approach, which improves A*GED by merely speeding up lower bound computation and reducing memory consumption, runs faster than CSI_GED.

⑤ *Extension to* GED *Verification.* We also extend our algorithms to the problem of GED verification (*i.e.*, verify whether the GED between two graphs is no larger than a user-given threshold). Our experimental results show that for GED verification, our AStar$^+$-BMao approach outperforms the existing approaches CSI_GED and A*GED by more than one and more than two orders of magnitude, respectively. Note that, the existing works on graph similarity search in [17, 27, 29, 30, 31] claimed to use A*GED for GED verification.

Organization. The rest of the paper is organized as follows. A brief overview of related works is given below. Preliminaries are presented in Section 2. We illustrate a unified framework in Section 3. We propose anchor-aware lower bounds in Section 4, and develop efficient lower bound computation techniques in Section 5. We instantiate our unified framework into AStar$^+$ and DFS$^+$, and contrast them in Section 6. We report our experimental results in Section 7, and extend our algorithms to the problem of GED verification in Section 8. Finally, Section 9 concludes the paper. Proofs of all lemmas and theorems can be found in Section A.1 in Appendix.

Related Works. Related works are categorized in the following.
(1) GED *Computation.* The notion of graph edit distance (GED) was proposed in [26] to quantify the distance between two graphs. Zeng *et al.* [28] proved that computing the exact GED is NP-hard. Nevertheless, algorithms have been designed for computing the ex-

act GED in practice. A best-first search algorithm A*GED is developed in [24], and depth-first search algorithms DF_GED [1, 5] and CSI_GED [11] are recently proposed and shown to outperform A*GED. Nevertheless, all the existing algorithms cannot process graphs with more than 30 vertices. In this paper, we systematically study lower bound estimations and search strategies for speeding up GED computation. Our AStar+-BMao algorithm that uses best-first search and lower bound $\delta^{\text{BMao}}(\cdot)$ outperforms the state-of-the-art algorithm CSI_GED by more than four orders of magnitude.

(2) Graph Similarity Search. GED-based graph similarity search is studied in [17, 27, 29, 30, 31]; that is, given a graph database D and a query graph g, find all graphs g' in D such that the GED between g and g' is no larger than a user-given threshold (*i.e.*, g' is similar to g). All these works mainly focus on designing effective index structures — *e.g.*, q-gram-based index [30], star structure-based index [27], and subgraph-based index [17, 29] — to filter out from D as many dissimilar graphs to g as possible, while all remaining candidates are verified by A*GED. In this paper, we propose a new algorithm AStar+-BMao which outperforms A*GED by more than two orders of magnitude for GED verification. One consequence of our significantly improved GED verification algorithm is that, the overall efficiency of the existing indexing techniques for graph similarity search may need to be reevaluated; for example, light-weight indexing techniques may become more preferable.

(3) Compute Maximum Common Subgraph. Measuring the similarity between two graphs based on their maximum common subgraph, which is NP-hard to compute [18], is also studied in the literature. Mcgregor [18] proposed a depth-first search method, while more advanced pruning techniques are later proposed in [2, 15]. Another strategy is first constructing a product graph of the two input graphs, and then computing the maximum clique of the product graph [14, 22]. These techniques cannot be applied to GED computation due to the inherently different problem definition.

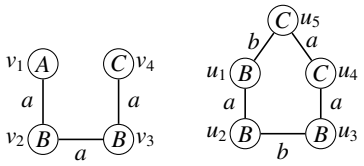
2. PRELIMINARIES

In this paper, we focus our discussions on labeled and undirected graphs $g = (V(g), E(g), l)$, while all our techniques can straightforwardly handle directed graphs. Here, $V(g)$ is the set of vertices, $E(g)$ is the set of edges, and $l : V(g) \cup E(g) \rightarrow \Sigma$ is a labelling function that assigns each vertex and/or edge a label from Σ; that is, $l(u)$ and $l(u, u')$ are the labels of vertex u and edge (u, u'), respectively. We denote the number of vertices and the number of edges of g by $|V(g)|$ and $|E(g)|$, respectively, and denote the degree of a vertex u by $d(u)$. Given a vertex subset $V_s \subseteq V(g)$, the subgraph of g induced by V_s is $g[V_s] = (V_s, \{(u, u') \in E(g) \mid u, u' \in V_s\}, l)$. In the following, for presentation simplicity we refer to a labeled and undirected graph simply as a graph.

A *graph edit operation* on a graph is an operation that transforms the graph. Specifically, it includes the following six edit operations: inserting/deleting an isolated vertex into/from the graph (*vertex insertion* and *vertex deletion*), adding/deleting an edge between two vertices (*edge insertion* and *edge deletion*), and changing the label of a vertex/edge (*vertex relabeling* and *edge relabeling*).

Definition 2.1: The **graph edit distance** (GED) between two graphs q and g, denoted $\delta(q, g)$, is the minimum number of edit operations that can transform q into g.

Note that, $\delta(q, g) = \delta(g, q)$ and GED is a metric [26]. Consider the two graphs q and g in Figure 2, where vertex labels are illustrated inside circles (*i.e.*, A, B, C) and edge labels are illustrated beside edges (*i.e.*, a, b). One possible sequence of edit operations for transforming q into g is as follows: (1) change the label of vertex v_1 from A to B, (2) change the label of edge (v_2, v_3) from a to b, (3) insert an isolated vertex v_5 with label C, (4) add an edge with label b between v_1 and v_5, and (5) add an edge with label a between v_4 and v_5. Thus, the GED between q and g is at most 5. Nevertheless, computing the exact GED is NP-hard [28].



(a) Graph q (b) Graph g

Figure 2: Sample graphs

Problem Statement. Given two graphs q and g, we study the problem of GED ***computation*** that computes the GED between q and g.

In the following, we use v and its variants, $v', v_1, v_2, \ldots$, to denote vertices in q, and use u and its variants, $u', u_1, u_2, \ldots$, to denote vertices in g. Frequently used notations are summarized in Table 2.

Notation	Description
q, g	Two graphs
$\delta(q, g)$	The GED between graphs q and g
$\text{UB}(q, g)$	Upper bound of $\delta(q, g)$
f	(Partial) mapping from $V(q)$ to $V(g)$
$f(v)$	The vertex of $V(g)$ to which $v \in V(q)$ maps
$f^-(u)$	The vertex of $V(q)$ that maps to $u \in V(g)$
$\delta_f(q, g)$	The editorial cost of the full mapping f from $V(q)$ to $V(g)$
$\delta(f)$	Lower bound of editorial costs of full mappings that extend f
$\mathcal{T}$	The search tree of all mappings from $V(q)$ to $V(g)$
$\mathcal{T}_{\leq\delta(q,g)}$	The set of partial mappings f in $\mathcal{T}$ s.t. $\delta(f) \leq \delta(q, g)$
$q[f]$	The subgraph of q induced by vertices of q that are in f
$q \backslash f$	The remaining subgraph of q by removing $q[f]$
$L_V(q \backslash f)$	Multi-set of vertex labels of $q \backslash f$
$L_E(q \backslash f)$	Multi-set of edge labels of $q \backslash f$
$L_{E_I}(q \backslash f)$	Multi-set of labels of inner edges of $q \backslash f$
$L_E(v)$	Multi-set of labels of v's adjacent edges
$L_{E_I}(v)$	Multi-set of labels of v's inner adjacent edges
$L_{E_C}(v)$	Multi-set of labels of v's cross adjacent edges
$\Upsilon(S_1, S_2)$	Edit distance between multi-sets, $\max\{\lvert S_1\rvert, \lvert S_2\rvert\} - \lvert S_1 \cap S_2\rvert$

Table 2: Frequently used notations

2.1 GED Computation via Vertex Mapping

We start with some simplifications, and then illustrate the idea of computing GED via enumerating vertex mappings.

Simplifications. As GED is a metric, any of the two graphs can be regarded as q. In our algorithms, we choose q to be the graph with fewer vertices (*i.e.*, $|V(q)| \leq |V(g)|$); if a tie occurs, an arbitrary one is chosen. We prove in the following lemma that, there is no vertex deletion in the optimal sequence of edit operations that transform q into g, given that $|V(q)| \leq |V(g)|$; the proof can be found Section A.1 in Appendix.

Lemma 2.1: *Given graphs q and g with $|V(q)| \leq |V(g)|$, there is no vertex deletion in the optimal sequence (i.e., with the minimum number) of edit operations that transform q into g.*

It is easy to verify that if $|V(q)| < |V(g)|$, then $\delta(q, g) = \delta(q', g)$ where q' is obtained from q by adding $|V(g)| - |V(q)|$ isolated vertices with the unique label $\perp \notin \Sigma$. Moreover, similar to Lemma 2.1, we can prove that if $|V(q)| = |V(g)|$, then there is no vertex deletion nor vertex insertion in the optimal sequence of edit operations that transform q into g. Thus, in the following, for presentation simplicity ***we assume q and g have the same number of vertices***,

Algorithm 1: EditorialCost

Input: Graphs q and g, and a mapping f from $V(q)$ to $V(g)$
Output: Editorial cost $\delta_f(q, g)$

```
1 δ_f(q,g) ← 0;
  /* Vertex relabeling                                   */
2 for each vertex v in q do
3     if l(v) ≠ l(f(v)) then δ_f(q,g) ← δ_f(q,g) + 1;
  /* Edge deletion or relabeling                         */
4 for each edge (v,v') in q do
5     if edge (f(v), f(v')) ∉ g or l(v,v') ≠ l(f(v), f(v')) then
6         δ_f(q,g) ← δ_f(q,g) + 1;
  /* Edge insertion                                      */
7 for each edge (u,u') in g do
8     if edge (f⁻(u), f⁻(u')) ∉ q then δ_f(q,g) ← δ_f(q,g) + 1;
9 return δ_f(q,g);
```

and thus we only need to consider four edit operations (*i.e.*, edge insertion/deletion, and vertex/edge relabeling); nevertheless, we have $|V(q)| \neq |V(g)|$ in our experiments.

GED **Computation via Vertex Mapping.** As vertex insertion and vertex deletion are not allowed in the edit operations, there is a natural *one-to-one mapping* from $V(q)$ to $V(g)$ that is preserved in the final isomorphism between the transformed graph q', obtained from q by applying the edit operations, and g. For presentation simplicity, we refer to one-to-one mapping as mapping in the following.

It has been shown in [13] that the GED between two graphs can be computed via enumerating vertex mappings, as follows.

Definition 2.2: Given a mapping f from $V(q)$ to $V(g)$, the **editorial cost** of f, denoted $\delta_f(q, g)$, is the minimum number of edit operations that is required to transform q into g by obeying the mapping f (*i.e.*, $v \in V(q)$ maps to $f(v) \in V(g)$).

Lemma 2.2: *[13] The* GED *between q and g equals the minimum editorial cost among all mappings from $V(q)$ to $V(g)$; that is, $\delta(q, g) = \min_{f \in \mathcal{F}(q,g)} \delta_f(q, g)$, where $\mathcal{F}(q, g)$ denotes the set of all mappings from $V(q)$ to $V(g)$.*

Thus, the GED between q and g can be computed by enumerating all mappings from $V(q)$ to $V(g)$ and computing their editorial costs, and then the minimum editorial cost is the result. The editorial cost of a mapping f from $V(q)$ to $V(g)$ can be computed in $O(|E(q)| + |E(g)|)$ time,[1] where such an algorithm is shown in Algorithm 1. Note that, this does not contradict with the NP-hardness of GED computation, since there is an exponential number of mappings from $V(q)$ to $V(g)$.

3. A UNIFIED FRAMEWORK

In this section, we develop a unified framework to conduct an efficient *pruned search* on the search tree, which compactly represents the exponential number of mappings from $V(q)$ to $V(g)$, for finding the mapping with the minimum editorial cost.

Search Tree. Given graphs q and g, and a matching order $\pi = (v_1, \ldots, v_{|V(q)|})$ of $V(q)$, we compactly represent all mappings from $V(q)$ to $V(g)$, according to the matching order π, in a prefix-shared *search tree* $\mathcal{T}$. A node at level i of $\mathcal{T}$ represents a partial mapping from $(v_1, \ldots, v_i)$ to $V(g)$, which *extends* that of its parent at level $i-1$ by additionally mapping v_i to a vertex of $V(g)$. To distinguish nodes in the search tree $\mathcal{T}$ from vertices in graphs q and g, we refer to the former as nodes. We use the term "mapping" when the context applies to both partial mapping and full mapping; otherwise, we explicitly specify "partial mapping" or "full mapping". Based on the parent-child relationships of nodes in $\mathcal{T}$, parent-child (as well as ancestor-descendant) relationships are also defined for mappings.

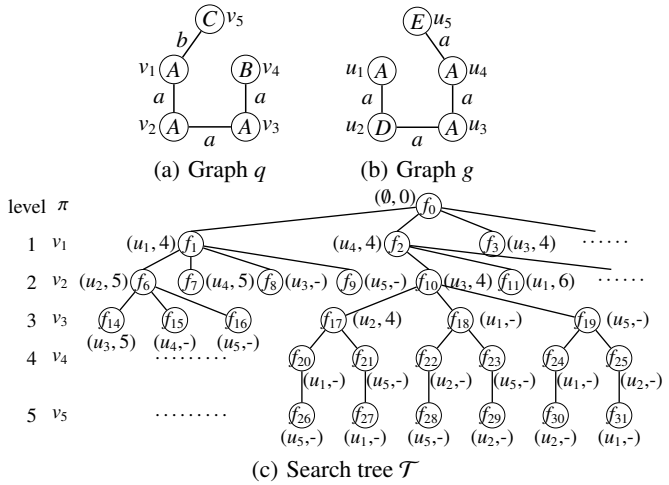


Figure 3: The search tree $\mathcal{T}$ for computing $\delta(q, g)$: f_i is a partial mapping, and beside f at level j is a pair $(u, \delta(f))$ where $u \in V(g)$ is the vertex to which v_j maps and $\delta(f)$ is the lower bound of f

A snippet of the search tree $\mathcal{T}$ of all vertex mappings from the graph q in Figure 3(a) to the graph g in Figure 3(b) is shown in Figure 3(c). The root node of $\mathcal{T}$ is at level 0, and represents an empty mapping. Beside each node f at level i in $\mathcal{T}$, we show two values: the vertex of $V(g)$ to which v_i maps in the mapping f, and the lower bound cost of f which will be introduced shortly; note that, we use f to denote both a mapping and its corresponding node in $\mathcal{T}$. For example, the partial mapping f_6 is $\{v_1 \mapsto u_1, v_2 \mapsto u_2\}$ and has a lower bound cost 5. The full mappings from $V(q)$ to $V(g)$ are at level $|V(q)|$ of $\mathcal{T}$.

The Framework. To avoid enumerating all mappings for computing $\delta(q, g)$, a lower bound cost is computed for each partial mapping for the purpose of pruning.

Definition 3.1: The **lower bound cost of a mapping** f, denoted $\delta(f)$, is a value that is no larger than the minimum editorial cost among all full mappings that extend f.

As a result, we can safely prune all mappings that extend a partial mapping f if $\delta(f)$ is already no smaller than the minimum editorial cost among all currently enumerated full mappings. To do so, we maintain the minimum editorial cost among all enumerated full mappings as the *upper bound* $\mathtt{UB}(q, g)$ of $\delta(q, g)$. Based on this idea, the pseudocode of computing $\delta(q, g)$ by conducting a pruned search on $\mathcal{T}$ is shown in Algorithm 2.

We first compute a matching order of vertices of q and let it be $\pi = (v_1, \ldots, v_{|V(q)|})$ (Line 1), initialize the upper bound $\mathtt{UB}(q, g)$ (Line 2), and initialize a priority queue Q which stores all mappings to be extended (Line 3). Each entry of the priority queue stores a mapping f, its level i in $\mathcal{T}$, its lower bound cost $\delta(f)$, and also the set $C(v_i)$ of candidates to which v_i can map in the ungenerated siblings of f; here, the siblings of f are the mappings in $\mathcal{T}$ that share the same parent as f. In this way, *the ungenerated siblings of f, each of which has a lower bound cost at least $\delta(f)$, are compactly represented by $C(v_i)$*. Then, we iteratively pop an entry $(f, i, \delta(f), C(v_i))$ from Q (Line 5). If the lower bound cost $\delta(f)$ is smaller than $\mathtt{UB}(q, g)$, we generate the best (*i.e.*, with the smallest

[1] In this paper, without loss of generality, we assume that $|V(q)| \leq |E(q)|$ and $|V(g)| \leq |E(g)|$ for time complexity analysis.

Algorithm 2: A Unified Framework

Input: Graphs q and g
Output: GED between q and g: $\delta(q, g)$

1 Compute a matching order $\pi = (v_1, \ldots, v_{|V(q)|})$ of vertices of q;
2 $\mathtt{UB}(q, g) \leftarrow +\infty$; /* Set the upper bound */;
3 Initialize a priority queue Q with an empty mapping $(\emptyset, 0, 0, \emptyset)$;
4 **while** $Q \neq \emptyset$ **do**
5 Pop $(f, i, \delta(f), C(v_i))$ from Q;
6 **if** $\delta(f) < \mathtt{UB}(q, g)$ **then**
/* Generate f's next best sibling */
7 **if** $i > 0$ **and** $C(v_i) \neq \emptyset$ **then** GenNext$(f, i, C(v_i))$;
/* Generate f's best child */
8 **if** $i < |V(q)|$ **then** GenNext$(f, i + 1, V(g\backslash f))$;
9 **return** $\mathtt{UB}(q, g)$;

Procedure GenNext(partial mapping f, level i, candidates $C(v_i)$)
10 $u^* \leftarrow \arg\min_{u \in C(v_i)} \delta(f \cup \{v_i \mapsto u\})$;
11 **if** $\delta(f \cup \{v_i \mapsto u^*\}) < \mathtt{UB}(q, g)$ **then**
12 Push $(f \cup \{v_i \mapsto u^*\}, i, \delta(f \cup \{v_i \mapsto u^*\}), C(v_i) \backslash u^*)$ into Q;
13 Generate a full mapping f' by extending $f \cup \{v_i \mapsto u^*\}$, and update $\mathtt{UB}(q, g)$ based on $\delta_{f'}(q, g)$; /* Optional */;

lower bound cost) ungenerated sibling (Line 7) and the best child (Line 8) of f in $\mathcal{T}$, and push them into Q (Lines 11–12); *in this case, we call f as being extended*. Here, $V(g\backslash f)$ denotes the set of vertices of g that are not used in f. At Line 7, in computing the best ungenerated sibling of f, we first remove the current mapping of v_i from f; thus, the best sibling and the best child of f are computed by the same procedure GenNext, which finds *the best extension of f by additionally mapping v_i*. For a generated partial mapping f, we may optionally construct a full mapping f' by extending f, and update the upper bound $\mathtt{UB}(q, g)$ by $\delta_{f'}(q, g)$ (Line 13).

Generality of the Framework. The framework in Algorithm 2 is general and can be instantiated into different algorithms as follows. (1) At Line 5, priority queues with different strategies for the popping operation can be adopted, based on which Algorithm 2 can be instantiated into either a best-first search approach (see Section 6.1), or a depth-first search approach (see Section 6.2). (2) At Lines 10–11, different lower bound estimation techniques can be adopted (see Sections 4 and 5). Moreover, all instantiations of our general framework in Algorithm 2 correctly compute the GED between two graphs, as proved below.

Theorem 3.1: *Algorithm 2 correctly computes the* GED *between two graphs.*

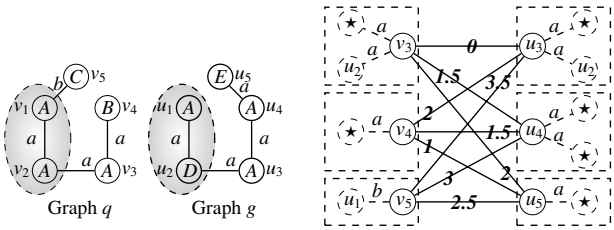
4. ANCHOR-AWARE LOWER BOUNDS

In this section, we propose a general anchor-aware technique to improve the tightness of lower bound estimations.

4.1 Lower Bound Estimation Framework

For a given partial mapping f, we denote the subgraph of q (resp. g) induced by vertices in f as $q[f]$ (resp. $g[f]$), and denote the remaining subgraph of q (resp. g) as $q\backslash f$ (resp. $g\backslash f$). Note that, $q\backslash f$ contains none of the vertices of $q[f]$ but includes edges that have exactly one end-point in $q[f]$. Thus, $q\backslash f$ contains both *inner edges* whose both end-points are in $q\backslash f$, and *cross edges* between vertices of $q\backslash f$ and vertices of $q[f]$. This also holds for $g\backslash f$.

Example 4.1: Consider the partial mapping $f = \{v_1 \mapsto u_1, v_2 \mapsto u_2\}$ for the graphs q and g in Figure 4(a). $q[f]$ and $g[f]$ are the parts in the shadowed area, while $q\backslash f$ and $g\backslash f$ are the remaining



(a) Graphs q and g (b) Branch match

Figure 4: Graphs and branch match

parts; specifically, $q\backslash f$ consists of three vertices $\{v_3, v_4, v_5\}$, one inner edge $\{(v_3, v_4)\}$ and two cross edges $\{(v_5, v_1), (v_3, v_2)\}$. □

Based on $q[f], g[f], q\backslash f$ and $g\backslash f$, we decompose the lower bound cost $\delta(f)$ of a partial mapping f into two parts, (1) the cost to transform $q[f]$ into $g[f]$ by obeying the mapping f, and (2) a lower bound cost for editing vertices and edges of $q\backslash f$ to be the same as $g\backslash f$. It is easy to see that the first part is exactly $\delta_f(q[f], g[f])$ which can be computed in linear time (see Section 2.1). We denote the second part by $\mathtt{LB}(q\backslash f, g\backslash f)$, whose computation is the focus of the remaining subsections of this section. Thus,

$$\delta(f) := \delta_f(q[f], g[f]) + \mathtt{LB}(q\backslash f, g\backslash f). \quad (1)$$

4.2 Existing Lower Bounds

The existing techniques for computing a *global lower bound* of $\delta(q, g)$ are mainly label set-based [5, 23, 24], branch match-based [31], and star match-based [28]. While the first one has been used for estimating $\mathtt{LB}(q\backslash f, g\backslash f)$ in A*GED and DF_GED, *the latter two have never been utilized in the context of* GED *computation*. In the following, due to limit of space we illustrate the first two lower bounds for estimating $\mathtt{LB}(q\backslash f, g\backslash f)$, while omitting the star match-based lower bound which can be found in Section A.3 in Appendix; note that, the star match-based lower bound is inferior to the branch match-based lower bound in practice.

Label Set-based Lower Bound $\mathtt{LS}(\cdot, \cdot)$**.** Let $L_V(q\backslash f)$ and $L_V(g\backslash f)$ denote the multi-sets of vertex labels of $q\backslash f$ and $g\backslash f$, respectively, and $L_E(q\backslash f)$ and $L_E(g\backslash f)$ denote the multi-sets of edge labels. The label set-based lower bound is the difference between $L_V(q\backslash f)$ and $L_V(g\backslash f)$ plus the difference between $L_E(q\backslash f)$ and $L_E(g\backslash f)$ [5], *i.e.*,

$$\mathtt{LS}(q\backslash f, g\backslash f) := \Upsilon\Big(L_V(q\backslash f), L_V(g\backslash f)\Big) + \Upsilon\Big(L_E(q\backslash f), L_E(g\backslash f)\Big) \quad (2)$$

where $\Upsilon(\cdot, \cdot)$ denotes the edit distance between two multi-sets and $\Upsilon(S_1, S_2) = \max\{|S_1|, |S_2|\} - |S_1 \cap S_2|$ for multi-sets S_1 and S_2 (please refer to Section A.2 for more details).[2] For the partial mapping f in Example 4.1, $L_V(q\backslash f) = \{A, B, C\}$, $L_V(g\backslash f) = \{A, A, E\}$, $L_E(q\backslash f) = \{a, a, b\}$, and $L_E(g\backslash f) = \{a, a, a\}$. Thus, $\mathtt{LS}(q\backslash f, g\backslash f) = 2 + 1 = 3$.

Branch Match-based Lower Bound $\mathtt{BM}(\cdot, \cdot)$**.** Lower bound based on the branch structure is proposed in [31] for computing a *global* lower bound of the GED between two graphs. We *extend* it to estimate $\mathtt{LB}(q\backslash f, g\backslash f)$ in the following.

Definition 4.1: The **branch** of a vertex v is $B(v) = (l(v), L_E(v))$, where $L_E(v)$ denotes the multi-set of labels of v's adjacent edges.

Definition 4.2: Based on the branch structures $B(v)$ and $B(u)$, the **mapping cost** of mapping $v \in q\backslash f$ to $u \in g\backslash f$ is defined as,

$$\lambda^{\mathtt{BM}}(v, u) := \mathbb{1}_{l(v) \neq l(u)} + \tfrac{1}{2} \times \Upsilon\Big(L_E(v), L_E(u)\Big),$$

[2]Note that, all unions ($\cup$) and intersections ($\cap$) in this paper follow the multi-set-based semantic.

where $\mathbb{1}_\phi$ is an indicator function that equals 1 if the expression ϕ evaluates `true` and 0 otherwise.

The branch match-based lower bound is the minimum mapping cost for mapping the set of branch structures of $q\backslash f$ into the set of branch structures of $g\backslash f$, *i.e.*,

$$\texttt{BM}(q\backslash f, g\backslash f) := \min_{\sigma\in\mathcal{F}(q\backslash f, g\backslash f)} \sum_{v\in q\backslash f} \lambda^{\texttt{BM}}(v, \sigma(v)) \quad (3)$$

where $\mathcal{F}(q\backslash f, g\backslash f)$ denotes the set of all mappings from vertices of $q\backslash f$ to vertices of $g\backslash f$. For the partial mapping f in Example 4.1, $B(v_3) = (A, \{a, a\})$ and $B(u_4) = (A, \{a, a\})$; thus, $\lambda^{\texttt{BM}}(v_3, u_4) = 0$. It can be verified that $\texttt{BM}(q\backslash f, g\backslash f) = 3$.

4.3 Our Anchor-aware Lower Bounds

The above lower bounds $\texttt{LS}(\cdot,\cdot)$ and $\texttt{BM}(\cdot,\cdot)$ are loose, since they do not exploit the mapping information of the partial mapping f. In the following, we propose anchor-aware techniques to improve $\texttt{LS}(\cdot,\cdot)$ and $\texttt{BM}(\cdot,\cdot)$. Before that, we first define anchored vertex.

Definition 4.3: We call vertices of q that are in f as **anchored vertices** since their mappings are fixed, and other vertices of q as **free vertices** since they can freely map to any vertex of $g\backslash f$.

Anchor-aware Label Set-based Lower Bound $\texttt{LSa}(\cdot,\cdot)$. As each anchored vertex v in $q[f]$ must map to $f(v)$, the set of cross adjacent edges of v must be edited to map to that of $f(v)$ in every full mapping that extends f. Thus, we separate the set of edges of $q\backslash f$ (and also $g\backslash f$) into cross edges and inner edges, and treat them differently. Let $L_{E_C}(v)$ denote the multi-set of labels of v's cross adjacent edges, and let $L_{E_I}(q\backslash f)$ and $L_{E_I}(g\backslash f)$ denote the multi-sets of labels of inner edges of $q\backslash f$ and $g\backslash f$, respectively. Then, we define the anchor-aware label set-based lower bound as,

$$\begin{aligned}\texttt{LSa}(q\backslash f, g\backslash f) :=& \Upsilon\Big(L_V(q\backslash f), L_V(g\backslash f)\Big) + \Upsilon\Big(L_{E_I}(q\backslash f), L_{E_I}(g\backslash f)\Big) \\ &+ \textstyle\sum_{v\in q[f]} \Upsilon\Big(L_{E_C}(v), L_{E_C}(f(v))\Big) \quad (4)\end{aligned}$$

whose correctness can be easily verified. For the partial mapping f in Example 4.1, we have $L_{E_C}(v_1) = \{b\}$, $L_{E_C}(v_2) = \{a\}$, $L_{E_C}(u_1) = \emptyset$ and $L_{E_C}(u_2) = \{a\}$. Thus, $\texttt{LSa}(q\backslash f, g\backslash f) = 4$.

It is easy to see that $L_{E_I}(q\backslash f)$ and $L_{E_C}(v)$ for anchored vertices v refine $L_E(q\backslash f)$; that is,

$$L_E(q\backslash f) = L_{E_I}(q\backslash f) \cup (\textstyle\bigcup_{v\in q[f]} L_{E_C}(v)).$$

Thus,

$$\texttt{LSa}(q\backslash f, g\backslash f) \geq \texttt{LS}(q\backslash f, g\backslash f);$$

that is, $\texttt{LSa}(\cdot,\cdot)$ computes a tighter lower bound than $\texttt{LS}(\cdot,\cdot)$. Note that, for lower bounds, the larger the better.

Anchor-aware Branch Match-based Lower Bound $\texttt{BMa}(\cdot,\cdot)$. To improve the branch match-based lower bound $\texttt{BM}(\cdot,\cdot)$ by the anchored vertices, we revise the branch structure of a vertex $v \in q\backslash f$ as $B'(v) = (l(v), L_{E_I}(v), \bigcup_{v'\in q[f]} \{(f(v'), l(v, v'))\})$, where $l(v, v') = \perp$ if the edge does not exist; that is, we explicitly consider each anchored vertex v' and its connection $l(v, v')$ to v. Similarly, we revise the branch structure for vertices of $g\backslash f$. Then, the mapping cost of mapping $v \in q\backslash f$ to $u \in g\backslash f$ is revised to be the sum of the edit distances between the three corresponding components of $B'(v)$ and $B'(u)$, *i.e.*,

$$\lambda^{\texttt{BMa}}(v, u) := \mathbb{1}_{l(v)\neq l(u)} + \tfrac{1}{2} \times \Upsilon\Big(L_{E_I}(v), L_{E_I}(u)\Big) + \textstyle\sum_{v'\in q[f]} \mathbb{1}_{l(v,v')\neq l(u,f(v'))}$$

where the label of a non-existence edge is defined as $\perp$. Intuitively, $\lambda^{\texttt{BMa}}(v, u)$ equals the minimum cost to edit v and its adjacent edges (*i.e.*, the branch structure of v) to be the same as u and u's adjacent edges, subject to the constraint that *an adjacent edge of v connecting to an anchored vertex v' must map to the adjacent edge of u connecting to $f(v')$*. For example, in Figure 4(b), the structure in each dotted rectangle is a branch, where non-existence edges (*i.e.*, with label $\perp$) are omitted; $\star$ denotes a free vertex and can map to any free vertex. The mapping cost between two vertices are illustrated, in the middle part, on the solid edge connecting the vertices. In particular, we have $B'(v_3) = (A, \{a\}, \{(u_1, \perp), (u_2, a)\})$ and $B'(u_4) = (A, \{a, a\}, \{(u_1, \perp), (u_2, \perp)\})$; thus, $\lambda^{\texttt{BMa}}(v_3, u_4) = 1.5$.

Then, we define the anchor-aware branch match-based lower bound as,

$$\texttt{BMa}(q\backslash f, g\backslash f) := \min_{\sigma\in\mathcal{F}(q\backslash f, g\backslash f)} \sum_{v\in q\backslash f} \lambda^{\texttt{BMa}}(v, \sigma(v)) \quad (5)$$

whose correctness can be proved in a similar way to the proof of $\texttt{BM}(q\backslash f, g\backslash f)$ in [31]. For the partial mapping f in Example 4.1, we have $\texttt{BMa}(q\backslash f, g\backslash f) = 4$, where the mapping costs are shown in Figure 4(b).

It can be easily verified that $\lambda^{\texttt{BMa}}(v, u) \geq \lambda^{\texttt{BM}}(v, u)$. Consequently,

$$\texttt{BMa}(q\backslash f, g\backslash f) \geq \texttt{BM}(q\backslash f, g\backslash f).$$

Moreover, we prove in below that $\texttt{BMa}(\cdot,\cdot)$ is tighter than $\texttt{LSa}(\cdot,\cdot)$.

Lemma 4.1: *For any partial mapping f, we have*

$$\texttt{BMa}(q\backslash f, g\backslash f) \geq \texttt{LSa}(q\backslash f, g\backslash f).$$

4.4 Putting It Together

In Equation (1), by replacing $\texttt{LB}(q\backslash f, g\backslash f)$ with $\texttt{LS}(q\backslash f, g\backslash f)$, $\texttt{BM}(q\backslash f, g\backslash f)$, and $\texttt{LSa}(q\backslash f, g\backslash f)$ and $\texttt{BMa}(q\backslash f, g\backslash f)$ in Equations (2)–(5), we can respectively get the lower bounds of a partial mapping f, $\delta^{\texttt{LS}}(f)$, $\delta^{\texttt{BM}}(f)$, $\delta^{\texttt{LSa}}(f)$, and $\delta^{\texttt{BMa}}(f)$. Among them, $\delta^{\texttt{LS}}(\cdot)$ is the lower bound definition used in the existing algorithms A*GED [23, 24, 30] and DF_GED [1, 5], while all other lower bounds are new and have not been studied for exact GED computation.

Note that, CSI_GED [11] uses a different lower bound than the above ones; we denote it by $\texttt{DE}(\cdot,\cdot)$, which is defined similarly to Equation (4) but by revising $\Upsilon(S_1, S_2)$ to be $\max\{|S_1|, |S_2|\} - \min\{|S_1|, |S_2|\}$. Specifically, (1) regarding each anchored vertex $v \in q[f]$, $\texttt{DE}(\cdot,\cdot)$ considers the difference between the cardinalities of $L_{E_C}(v)$ and $L_{E_C}(f(v))$, and (2) regarding inner edges, $\texttt{DE}(\cdot,\cdot)$ considers the differences between the cardinalities of $L_{E_I}(q\backslash f)$ and $L_{E_I}(g\backslash f)$. Due to entirely ignoring the edge labels, $\texttt{DE}(q\backslash f, g\backslash f)$ is always smaller than $\texttt{LSa}(q\backslash f, g\backslash f)$. Moreover, our experimental results show that our depth-first search approach incorporated with our $\texttt{LSa}(\cdot,\cdot)$ lower bound significantly outperforms CSI_GED (see Section 7). Thus, we omit $\texttt{DE}(\cdot,\cdot)$ in this paper.

5. BEST EXTENSION COMPUTATION

In Algorithm 2, given a partial mapping f and a lower bound definition $\delta(\cdot)$, we need to compute the best extension of f; that is, compute $\arg\min_{u\in C(v_i)} \delta(f \cup \{v_i \mapsto u\})$, where $v_i \notin f$ is the next vertex of q to map according to the matching order π and $C(v_i)$ is the subset vertices of g that v_i can map to. In this section, we propose techniques to efficiently conduct such computations.

5.1 Best Extension regarding $\delta^{\texttt{LS}}(\cdot)$ and $\delta^{\texttt{LSa}}(\cdot)$

We mainly focus our discussions on $\delta^{\texttt{LS}}(\cdot)$, while computing best extension regarding $\delta^{\texttt{LSa}}(\cdot)$ uses similar ideas but is more involved.

A Straightforward Quadratic-time Approach. To compute the best extension of a partial mapping f regarding $\delta^{\texttt{LS}}(\cdot)$, a straightforward approach is computing $\delta^{\texttt{LS}}(f \cup \{v_i \mapsto u\})$ *independently*

for each vertex $u \in C(v_i)$, and then choosing the one that results in the minimum lower bound cost. This approach has been used in A*GED [23, 24, 30] and DF_GED [1, 5]. As each lower bound computation takes $O(|E(q)| + |E(g)|)$ time, the time complexity of this straightforward approach is $O\Big(|V(q)| \times (|E(q)| + |E(g)|)\Big)$.

An Efficient Linear-time Approach. As the efficiency of best extension computation is critical to the efficiency of GED computation, we propose an efficient algorithm to compute the best extension as well as the lower bound costs of all children of f regarding $\delta^{\mathtt{LS}}(\cdot)$ in $O(|E(q)| + |E(g)|)$ total time. The general idea is to obtain $\delta^{\mathtt{LS}}(f \cup \{v_i \mapsto u\})$ from $\delta^{\mathtt{LS}}(f)$ in $O(d(u))$ time by maintaining some auxiliary information, where $d(u)$ is the degree of u in g. Let f' denote $f \cup \{v_i \mapsto u\}$. Then, $\delta^{\mathtt{LS}}(f')$ consists of three parts and is

$$\delta_{f'}(q[f'], g[f']) + \Upsilon(L_E(q\backslash f'), L_E(g\backslash f')) + \Upsilon(L_V(q\backslash f'), L_V(g\backslash f'))$$

Firstly, $\delta_{f'}(q[f'], g[f'])$ can be obtained from $\delta_f(q[f], g[f])$ by adding the minimum cost of editing edges of $E(v_i, q[f])$ to map to edges of $E(u, g[f])$ according to the mapping f', where $E(v_i, q[f])$ denotes the set of edges between v_i and $q[f]$ and $E(u, g[f])$ is similarly defined. Specifically, we have

$$\delta_{f'}(q[f'], g[f']) = \delta_f(q[f], g[f]) + (d_1 + d_2 - 2 \times c_2 - c_1 + \mathbb{1}_{l(v_i) \neq l(u)})$$

where $d_1 = |E(v_i, q[f])|$, $d_2 = |E(u, g[f])|$, c_2 is the number of matched edges with the same labels between $E(v_i, q[f])$ and $E(u, g[f])$ (*i.e.*, no edit operation is required), and c_1 is the number of matched edges with different labels between $E(v_i, q[f])$ and $E(u, g[f])$ (*i.e.*, edge relabeling is required). Note that, for the above quantities, $\delta_f(q[f], g[f])$ and d_1 are irrelevant to u and thus can be pre-computed, while d_2, c_1, and c_2 can be computed online in $O(d(u))$ time.

Secondly, according to the definition of $\Upsilon(\cdot, \cdot)$, we have

$$\Upsilon(L_E(q\backslash f'), L_E(g\backslash f')) = \max\{n_1, n_2\} - c_E$$

where $n_1 = |L_E(q\backslash f')|$, $n_2 = |L_E(g\backslash f')|$ and $c_E = |L_E(q\backslash f') \cap L_E(g\backslash f')|$; n_2 also equals $|L_E(g\backslash f)| - |E(u, g[f])| = |L_E(g\backslash f)| - d_2$. Note that, n_1 and $|L_E(g\backslash f)|$ are irrelevant to u and can be pre-computed; thus, n_2 can be obtained online in $O(d(u))$ time. In order to efficiently obtain c_E online, we compute c_E from $|L_E(q\backslash f') \cap L_E(g\backslash f)|$ which is irrelevant to u and can be precomputed; note that, $L_E(g\backslash f) = L_E(g\backslash f') \cup L_{E_C}(u)$ where $L_{E_C}(u)$ is the set of edge labels of $E(u, g[f])$. Specifically, let $n_E(a)$ (resp. $n'_E(a)$) be the number of edges of $g\backslash f$ (resp. $g\backslash f'$) with label a minus the number of edges of $q\backslash f'$ with label a. Then, it can be verified that c_E equals

$$|L_E(q\backslash f') \cap L_E(g\backslash f)| - \sum_{a \in \Sigma: n_E(a) > n'_E(a) < 0} (\min\{0, n_E(a)\} - n'_E(a))$$

where the second part can be computed online in $O(d(u))$ time.

Thirdly, $\Upsilon(L_V(q\backslash f'), L_V(g\backslash f'))$ can be computed similarly.

Based on the above ideas, the pseudocode of our algorithm for computing the best extension of f regarding $\delta^{\mathtt{LS}}(\cdot)$ is shown in Algorithm 3, which is self-explanatory. The following is immediate.

Lemma 5.1: *Algorithm 3 correctly computes the best extension of f as well as the lower bound costs of all children of f regarding $\delta^{\mathtt{LS}}(\cdot)$ in $O(|E(q)| + |E(g)|)$ total time.*

Similar to Algorithm 3, we can compute the best extension of f as well as the lower bound costs of all children of f regarding $\delta^{\mathtt{LSa}}(\cdot)$ in $O(|E(q)| + |E(g)|)$ total time. We omit the details.

5.2 Best Extension regarding $\delta^{\mathtt{BM}}(\cdot)$ and $\delta^{\mathtt{BMa}}(\cdot)$

The computations for lower bounds $\delta^{\mathtt{BM}}(\cdot)$ and $\delta^{\mathtt{BMa}}(\cdot)$ are similar to each other by only differing in the mapping cost $\lambda(v, u)$, which can be computed in $O(d(v) + d(u))$ time for both. Thus, in the following we mainly focus our discussions on $\delta^{\mathtt{BMa}}(\cdot)$. Similar to

Algorithm 3: Compute best extension of f regarding $\delta^{\mathtt{LS}}(\cdot)$

Input: Graphs q and g, a partial mapping f, and candidates $C(v_i)$ of the next mapping vertex $v_i \in q\backslash f$
Output: Best extension $u^* \leftarrow \arg\min_{u \in C(v_i)} \delta^{\mathtt{LS}}(f \cup \{v_i \mapsto u\})$

Let d_1 be the number of edges between v_i and vertices of $q[f]$;
Let n_1 and n_2 be the numbers of edges in $q\backslash(f \cup \{v_i\})$ and $g\backslash f$, respectively;
Let $n_E(a)$ be the number of edges in $g\backslash f$ with label a minus the number of edges in $q\backslash(f \cup \{v_i\})$ with label a for each edge label a appeared in $g\backslash f$;
Let c_E be the cardinality of the multi-set intersection between the edge labels of $q\backslash(f \cup \{v_i\})$ and the edge labels of $g\backslash f$;
Let $n_V(A)$ be the number of vertices in $g\backslash f$ with label A minus the number of vertices in $q\backslash(f \cup \{v_i\})$ with label A for each vertex label A appeared in $g\backslash f$;
Let c_V be the cardinality of the multi-set intersection between the vertex labels of $q\backslash(f \cup \{v_i\})$ and the vertex labels of $g\backslash f$;
$max_V \leftarrow \max\{|V(g\backslash f)| - 1, |V(g\backslash f)| - 1\}$;
$u^* \leftarrow$ null;
for each vertex u in $C(v_i)$ **do**
  $d_2 \leftarrow 0, c_1 \leftarrow 0, c_2 \leftarrow 0$;
  for each edge (u, u') between u and $g[f]$ **do**
    $d_2 \leftarrow d_2 + 1$;
    if edge $(v_i, f^-(u'))$ exists in q **then**
      if $l(u, u') = l(v_i, f^-(u'))$ **then** $c_2 \leftarrow c_2 + 1$;
      else $c_1 \leftarrow c_1 + 1$;
    $n_2 \leftarrow n_2 - 1, a \leftarrow l(u, u'), n_E(a) \leftarrow n_E(a) - 1$;
    if $n_E(a) < 0$ **then** $c_E \leftarrow c_E - 1$;
  $\delta^{\mathtt{LS}}(f \cup \{v_i \mapsto u\}) \leftarrow \delta_f(q[f], g[f]) + (d_1 + d_2 - 2 \times c_2 - c_1 + \mathbb{1}_{l(v_i) \neq l(u)}) + ((\max\{n_1, n_2\} - c_E) + (max_V - c_V + \mathbb{1}_{n_V(l(u)) \leq 0}))$;
  if $u^* =$ null **or** $\delta^{\mathtt{LS}}(f \cup \{v_i \mapsto u\}) < \delta^{\mathtt{LS}}(f \cup \{v_i \mapsto u^*\})$ **then**
    $u^* \leftarrow u$;
  /* Restore $n_2, n_E(a), c_E$ */
  for each edge (u, u') between u and $g[f]$ **do**
    $n_2 \leftarrow n_2 + 1, a \leftarrow l(u, u'), n_E(a) \leftarrow n_E(a) + 1$;
    if $n_E(a) \leq 0$ **then** $c_E \leftarrow c_E + 1$;
return u^*;

the straightforward approach in Section 5.1, we can compute the best extension of f regarding $\delta^{\mathtt{BMa}}(\cdot)$ by computing the lower bound cost $\delta^{\mathtt{BMa}}(f \cup \{v_i \mapsto u\})$ for each vertex $u \in C(v_i)$ (*i.e.*, each child of f), and then choosing the one with the minimum lower bound cost.

Given a partial mapping f', the lower bound cost $\delta^{\mathtt{BMa}}(f')$ of f' is $\delta_{f'}(q[f'], g[f']) + \mathtt{BMa}(q\backslash f', g\backslash f')$, equivalently, $\delta_{f'}(q[f'], g[f']) + \min_{\sigma \in \mathcal{F}(q\backslash f', g\backslash f')} \sum_{v \in q\backslash f'} \lambda^{\mathtt{BMa}}(v, \sigma(v))$. The first part can be computed in linear time by Algorithm 1, and the second part can be computed in $O\Big((|V(q)| + |V(g)|)^3\Big)$ time by the classic Hungarian algorithm that computes the minimum cost perfect matching in a bipartite graph [9, 16, 19]. Here, the bipartite graph consists of vertices of $q\backslash f'$ on one side and vertices of $g\backslash f'$ on the other side, where each edge (v, u) has a cost $\lambda^{\mathtt{BMa}}(v, u)$. For example, consider the partial mapping $f' = \{v_1 \mapsto u_1, v_2 \mapsto u_2\}$ for the graphs in Figure 4(a), the constructed bipartite graph is illustrated in Figure 4(b), where edges of the bipartite graph are depicted as solid lines with their costs shown on the edges. By running the Hungarian algorithm, we obtain the minimum cost perfect matching as $\{v_3 \mapsto u_3, v_4 \mapsto u_4, v_5 \mapsto u_5\}$ whose cost is 4; thus, $\mathtt{BMa}(q\backslash f', g\backslash f') = 4$.

As a result, the time complexity for computing the best extension of f as well the time complexity for computing the lower bound costs of all children of f regarding $\delta^{\mathtt{BMa}}(\cdot)$ are $O\Big((|V(q)| + |V(g)|)^4\Big)$. Note that, this time complexity cannot be improved via sharing computation among the different lower bound computations, because the costs of edges in the bipartite graph constructed for com-

puting $\delta^{\texttt{BMa}}(f \cup \{v_i \mapsto u\})$ may be completely different from that for computing $\delta^{\texttt{BMa}}(f)$ and $\delta^{\texttt{BMa}}(f \cup \{v_i \mapsto u'\})$, in the worst case.

5.3 Optimizing $\delta^{\texttt{BMa}}(\cdot)$ into $\delta^{\texttt{BMao}}(\cdot)$

From the above, we know that $\delta^{\texttt{BMa}}(\cdot)$ and $\delta^{\texttt{BM}}(\cdot)$ provide tighter lower bounds than $\delta^{\texttt{LSa}}(\cdot)$ and $\delta^{\texttt{LS}}(\cdot)$, but at the cost of a significantly higher time complexity for computing the best extension of a partial mapping. As will be demonstrated by our experiments, there needs a balance between the tightness and the efficiency of lower bound estimation. Thus, we slightly loose $\delta^{\texttt{BMa}}(\cdot)$ into the lower bound

$$\delta^{\texttt{BMao}}(f \cup \{v_i \mapsto u\}) := \delta_f(q[f], g[f]) + \texttt{BMa}_{v_i \mapsto u}(q \backslash f, g \backslash f) \quad (6)$$

where $\texttt{BMa}_{v_i \mapsto u}(q \backslash f, g \backslash f)$ is similar to $\texttt{BMa}(q \backslash f, g \backslash f)$ except that v_i is restricted to map to u. Moreover, *v_i is treated as a free rather than anchored vertex* in computing $\texttt{BMa}_{v_i \mapsto u}(q \backslash f, g \backslash f)$ and thus $\delta^{\texttt{BMao}}(f \cup \{v_i \mapsto u\})$. As a result, we intuitively have

$$\delta^{\texttt{BMao}}(f \cup \{v_i \mapsto u\}) \leq \delta^{\texttt{BMa}}(f \cup \{v_i \mapsto u\}).$$

Nevertheless, the gap between these two lower bounds are not large as will be illustrated in Section 6.1.

Algorithm 4: Compute the best extension regarding $\delta^{\texttt{BMao}}(\cdot)$

Input: Graphs q and g, a partial mapping f, and candidates $C(v_i)$ of the next mapping vertex $v_i \in q \backslash f$
Output: Best extension $u^* \leftarrow \arg\min_{u \in C(v_i)} \delta^{\texttt{BMao}}(f \cup \{v_i \mapsto u\})$

```
1 for each vertex v in q\f do
2     for each vertex u in g\f do
3         if v = v_i and u ∉ C(v_i) then λ^BMa(v, u) ← +∞;
4         else Compute the mapping cost λ^BMa(v, u) of mapping v to u;
5 Compute the minimum cost perfect matching M between vertices of
    q\f and vertices of g\f by invoking the Hungarian method in [16, 19];
6 Let u* be the vertex to which v_i maps in M;
7 return u*;
```

The advantage of the lower bound $\delta^{\texttt{BMao}}(\cdot)$ is that the best extension of a partial mapping f regarding it can be computed in $O\Big(\big(|V(q)| + |V(g)|\big)^3\Big)$ time, without computing the lower bound cost of all children of f. Such an algorithm is shown in Algorithm 4, which conducts only one computation of the minimum cost perfect matching based on the property that $\min_{u \in C(v_i)} \texttt{BMa}_{v_i \mapsto u}(q \backslash f, g \backslash f) = \texttt{BMa}_{v_i \mapsto C(v_i)}(q \backslash f, g \backslash f)$. Algorithm 4 is self-explanatory.

Theorem 5.1: *Algorithm 4 correctly computes the best extension of a partial mapping f regarding $\delta^{\texttt{BMao}}(\cdot)$.*

Note that, Algorithm 4 can also be extended to compute the lower bound costs of all children of f in $O\Big(\big(|V(q)| + |V(g)|\big)^3\Big)$ total time. That is, each time after obtaining a minimum cost perfect matching M, we change the cost $\lambda(v_i, u')$ for the mapping $v_i \mapsto u'$ in M into $+\infty$, and then extend $M \backslash \{v_i \mapsto u'\}$ into a minimum cost perfect matching in $O\Big(\big(|V(q)| + |V(g)|\big)^2\Big)$ time.

Similarly, we optimize $\delta^{\texttt{BM}}(\cdot)$ into $\delta^{\texttt{BMo}}(\cdot)$, and we can compute the best extension as well as the lower bound costs of all children of a partial mapping regarding $\delta^{\texttt{BMo}}(\cdot)$ in $O\Big(\big(|V(q)| + |V(g)|\big)^3\Big)$ time.

5.4 Maintaining an Upper Bound

In Algorithm 2, we maintain an upper bound $\texttt{UB}(q, g)$, which equals the minimum editorial cost among all enumerated full mappings, to reduce the memory consumption in practice. Thus, after generating a new partial mapping at Line 12 of Algorithm 2, we can heuristically extend it to a full mapping for updating $\texttt{UB}(q, g)$. It is easy to see that the minimum cost perfect matching M computed by Algorithm 4 actually extends f to a full mapping. Thus, we use the editorial cost of the full mapping $f \cup M$ to update $\texttt{UB}(q, g)$, when any of the lower bounds $\delta^{\texttt{BM}}(\cdot)$, $\delta^{\texttt{BMa}}(\cdot)$, $\delta^{\texttt{BMo}}(\cdot)$, $\delta^{\texttt{BMao}}(\cdot)$ is used. However, we do not heuristically extend f to a full mapping if lower bound $\delta^{\texttt{LS}}(\cdot)$ or $\delta^{\texttt{LSa}}(\cdot)$ is used.

6. OUR APPROACHES

In this section, we instantiate our framework (Algorithm 2) into a best-first search approach AStar$^+$ in Section 6.1, and into a depth-first search approach DFS$^+$ in Section 6.2. Then, we contrast these two approaches in Section 6.3.

6.1 Our AStar$^+$ Approach

Our framework in Algorithm 2 works in a best-first search fashion if at Line 5, we pop from the priority queue Q the partial mapping with the minimum lower bound cost; if there is a tie on the minimum lower bound cost, then we break the tie by preferring large level numbers. Note that, if a tie still occurs, then we break the tie arbitrarily. We denote our approach with this strategy as AStar$^+$. A running example of AStar$^+$ is given in Section A.5 in Appendix.

Analysis of AStar$^+$. Let $\mathcal{T}_{\leq \delta(q,g)}$ be the set of non-leaf nodes/partial mappings in $\mathcal{T}$ whose lower bound costs are no larger than $\delta(q, g)$, and $|\mathcal{T}_{\leq \delta(q,g)}|$ be its cardinality. Here, we assume that *the sequence of lower bound costs for any root to leaf path in $\mathcal{T}$ is non-decreasing*. Note that, in Algorithm 2, it is possible that when extending a partial mapping f to get its best ungenerated sibling f' (Line 7) and its best child f'' (Line 8), the computed lower bound cost of f' (or f'') is smaller than that of f; nevertheless, if this is the case, then we can safely change the lower bound cost of f' (or f'') to be that of f. We have the following lemmas.

Lemma 6.1: *The set of partial mappings extended by AStar$^+$ (i.e., reach Lines 7–8 of Algorithm 2) is a subset of $\mathcal{T}_{\leq \delta(q,g)}$.*

Lemma 6.2: *Any algorithm that correctly computes $\delta(q, g)$ by traversing the search tree $\mathcal{T}$ needs to extend all partial mappings of $\mathcal{T}_{< \delta(q,g)}$,* i.e.*, with lower bound cost smaller than $\delta(q, g)$.*

Thus, $\mathcal{T}_{\leq \delta(q,g)}$ roughly indicates the search space of AStar$^+$. Intuitively, the tighter the lower bound, the smaller the search space. In the following, we use the superscript to denote the lower bound used. Following from Section 4, we have $|\mathcal{T}^{\texttt{BMa}}_{\leq \delta(q,g)}| \leq |\mathcal{T}^{\texttt{LSa}}_{\leq \delta(q,g)}| \leq |\mathcal{T}^{\texttt{LS}}_{\leq \delta(q,g)}|$, $|\mathcal{T}^{\texttt{BMa}}_{\leq \delta(q,g)}| \leq |\mathcal{T}^{\texttt{BM}}_{\leq \delta(q,g)}|$, and $|\mathcal{T}^{\texttt{BMa}}_{\leq \delta(q,g)}| \leq |\mathcal{T}^{\texttt{BMao}}_{\leq \delta(q,g)}| \leq |\mathcal{T}^{\texttt{BMo}}_{\leq \delta(q,g)}|$. Moreover, we can prove the following lemma.

Lemma 6.3: $|\mathcal{T}^{\texttt{BMao}}_{\leq \delta(q,g)}| \leq |V(g)| \times |\mathcal{T}^{\texttt{BMa}}_{\leq \delta(q,g)}|$.

Space Complexity of AStar$^+$. As the space consumption of AStar$^+$ is mainly dominated by the priority queue Q, we first bound the number of partial mappings that were pushed into Q, as follows.

Lemma 6.4: *The total number of partial mappings that were pushed into the priority queue Q when running AStar$^+$ is $O(|\mathcal{T}_{\leq \delta(q,g)}|)$.*

Although Q may also include partial mappings whose lower bound costs are larger than $\delta(q, g)$, the number of partial mappings that were ever pushed into Q is still bounded by $O(|\mathcal{T}_{\leq \delta(q,g)}|)$. Now, we prove the space complexity of AStar$^+$ by the following theorem.

Theorem 6.1: *The space complexity of AStar$^+$ is $O(|\mathcal{T}_{\leq \delta(q,g)}|)$.*

In practice, we further reduce the space consumption of AStar$^+$ based on the maintained upper bound $\texttt{UB}(q, g)$. Specifically, 1) we remove from Q all partial mappings whose lower bound costs are no smaller than $\texttt{UB}(q, g)$, and 2) we also remove from main memory the partial mappings that have no descendants in Q.

Time Complexity of AStar$^+$. Let $\mathtt{T_{BE}}$ be the time of computing the best extension of a partial mapping at Line 10 of Algorithm 2, which depends on the actual lower bound estimation technique and is discussed in Section 5. The time complexity of AStar$^+$ is proved in the following theorem.

Theorem 6.2: *The time complexity of* AStar$^+$ *is* $O(|\mathcal{T}_{\leq\delta(q,g)}| \times (\log|\mathcal{T}_{\leq\delta(q,g)}| + \mathtt{T_{BE}}))$.

As $\log|\mathcal{T}_{\leq\delta(q,g)}|$ is bounded by $|V(q)| \times \log|V(g)|$ and is usually smaller than $\mathtt{T_{BE}}$, we can simply regard the time complexity of AStar$^+$ as $O(|\mathcal{T}_{\leq\delta(q,g)}| \times \mathtt{T_{BE}})$.

Expand All Strategy. In the above discussions, to save memory space, each time when computing the best child of a partial mapping, we only materialize the best child and its lower bound cost. Then, to obtain the best ungenerated sibling of a partial mapping, we need to run the expensive best extension computation algorithm again (Line 7 of Algorithm 2). As a result, the same lower bound cost $\delta(f \cup \{v_i \mapsto u\})$ may be computed up-to $O(|V(g)|)$ times, once in computing the best sibling for each f' that is a sibling of $f \cup \{v_i \mapsto u\}$; thus, there are redundant computations.

We can trade a little memory for time efficiency by the expand all strategy. That is, when computing the best child of a partial mapping, we generate all its children and compute their lower bound costs; note that, this still can be conducted in $\mathtt{T_{BE}}$ time (see Section 5). Nevertheless, rather than pushing all these generated children into the priority queue Q, we only push the best child into Q and attach all the remaining children to this best child. Subsequently, to obtain the best ungenerated sibling of a partial mapping, we directly select the best one that is attached to it.

Comparison to the Existing Best-First Search Approach. Although the best-first search strategy has been adopted in existing works (*e.g.*, A*GED [23, 24, 29, 30]), one unique feature of AStar$^+$ is reducing memory consumption by storing each partial mapping in a constant memory space and utilizing the upper bound $\mathtt{UB}(q, g)$ to prune the priority queue. Consequently, AStar$^+$ largely resolves the issue of running out-of-memory of the existing best-first search approach A*GED. Moreover, we propose anchor-aware lower bounds and also efficient algorithms to compute the best extension of a partial mapping. In summary, A*GED has both a larger space complexity (*i.e.*, $O\big(|\mathcal{T}_{\leq\delta(q,g)}| \times |V(g)|\big)$) and a larger time complexity (*i.e.*, $O\big(|\mathcal{T}_{\leq\delta(q,g)}| \times |V(g)| \times \mathtt{T_{LB}}\big)$), where $\mathtt{T_{LB}}$ is the time to compute the lower bound cost of a partial mapping; note that, $\mathtt{T_{LB}} = \mathtt{T_{BE}}$ for lower bounds $\delta^{\mathtt{LS}}(\cdot)$, $\delta^{\mathtt{LSa}}(\cdot)$, $\delta^{\mathtt{BMo}}(\cdot)$ and $\delta^{\mathtt{BMao}}(\cdot)$ (see Section 5).

6.2 Our DFS$^+$ Approach

Algorithm 2 works in a depth-first search fashion if at Line 5, we pop from the priority queue Q the partial mapping that has the largest level number according to the search tree $\mathcal{T}$. Then, at any time, there is at most one partial mapping kept in Q for each distinct level number. Thus, we can simulate the priority queue by an array of size $|V(q)|$. We denote our algorithm with this strategy as DFS$^+$. A running example of DFS$^+$ is given in Section A.6 in Appendix.

Space Complexity of DFS$^+$**.** The space complexity of DFS$^+$ is $O(|V(q)| \times |V(g)|)$. Firstly, the number of partial mappings in the priority queue Q of DFS$^+$ is $O(|V(q)|)$. Secondly, for each partial mapping, we need to store either $C(v_i)$ or its remaining siblings depending on whether or not the expand all strategy is used.

Time Complexity of DFS$^+$**.** Let $\mathcal{T}_{\mathsf{DFS}}$ be the set of all nodes (*i.e.*, partial mappings) in $\mathcal{T}$ that are extended by DFS$^+$ (*i.e.*, reach Lines 7–8 of Algorithm 2). Then, the search space of DFS$^+$ is $O(|\mathcal{T}_{\mathsf{DFS}}|)$. It is easy to verify that the time complexity of DFS$^+$ is $O(|\mathcal{T}_{\mathsf{DFS}}| \times \mathtt{T_{BE}})$, since it takes $O(\mathtt{T_{BE}})$ time to find the best sibling/child of a partial mapping. Note that, in DFS$^+$ each push/pop operation of the priority queue takes constant time, since it is simulated by an array.

Comparison to the Existing Depth-First Search Approaches. The idea of using depth-first search for GED computation has also been exploited in the existing works; for example, DF_GED [1, 5] and CSI_GED [11]. DF_GED is similar to DFS$^+$ incorporated with the lower bound $\delta^{\mathtt{LS}}(\cdot)$, but we propose a more efficient algorithm to compute the lower bound costs of all children of a partial mapping. On the other hand, CSI_GED enumerates edge mappings, while DFS$^+$ enumerates vertex mappings. Moreover, it is worth mentioning that the lower bounds discussed in this paper, except $\delta^{\mathtt{LS}}(\cdot)$, have not been used in the existing approaches.

6.3 AStar$^+$ v.s. DFS$^+$

In the following, we contrast AStar$^+$ with DFS$^+$ for GED computation. On one hand, AStar$^+$ usually has a smaller search space (and thus lower time complexity), by extending fewer partial mappings, than DFS$^+$ based on the following intuitions. (1) As proved in Lemma 6.2, each partial mapping that has a lower bound cost strictly smaller than $\delta(q, g)$ must be extended by DFS$^+$; that is, $\mathcal{T}_{<\delta(q,g)} \subseteq \mathcal{T}_{\mathsf{DFS}}$. Intuitively, even if DFS$^+$ generates the minimum-cost full mapping f^* at a very early stage of the searching, it still needs to exhaust all mappings of $\mathcal{T}_{<\delta(q,g)}$ to certify that there is no full mapping with a smaller editorial cost than f^*. (2) More often, DFS$^+$ also extends many partial mappings with lower bound costs larger than $\delta(q, g)$ due to a large $\mathtt{UB}(q, g)$ obtained at early stages of the searching. Recall that, the time complexities of AStar$^+$ and DFS$^+$ are $O(|\mathcal{T}_{\leq\delta(q,g)}| \times \mathtt{T_{BE}})$ and $O(|\mathcal{T}_{\mathsf{DFS}}| \times \mathtt{T_{BE}})$, respectively.

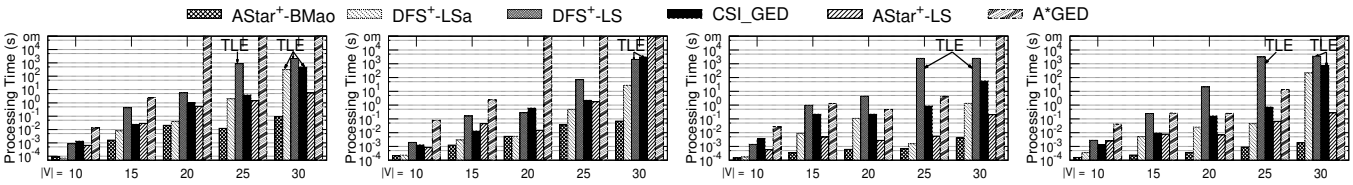
On the other hand, AStar$^+$ has a larger space complexity than DFS$^+$. One may think that it will run out-of-memory, as observed for the existing best-first search approach A*GED [1, 11]. In fact, AStar$^+$ largely resolves this issue by (1) representing each partial mapping by a constant memory space, (2) incorporating upper bound to prune the priority queue, and (3) significantly reducing the search space $\mathcal{T}_{\leq\delta(q,g)}$ based on our anchor-aware branch match-based lower bound $\delta^{\mathtt{BMao}}(\cdot)$. Moreover, when using the lower bound $\delta^{\mathtt{BMao}}(\cdot)$, if AStar$^+$ runs out-of-memory, then DFS$^+$ is likely to take an excessive long time by noting the cubic time of lower bound computation based on $\delta^{\mathtt{BMao}}(\cdot)$. Consequently, AStar$^+$ *is better than* DFS$^+$ *as demonstrated by our experiments in Section 7.*

7. EXPERIMENTS

We conduct extensive empirical studies to evaluate the effectiveness and efficiency of our techniques. To do so, we compare the following algorithms.

- **Our Algorithms.** We implemented several variants of our AStar$^+$ and DFS$^+$ algorithms by incorporating different lower bounds: AStar$^+$-LS, AStar$^+$-LSa, AStar$^+$-BMo, AStar$^+$-BMao, AStar$^+$-BMa, AStar$^+$-SMa, DFS$^+$-LS, DFS$^+$-LSa, DFS$^+$-BMao;[3] the lower bound $\delta^{\mathtt{SMa}}(\cdot)$ is discussed in Section A.3 in Appendix. For all the algorithms, we use the expand all strategy, and we use the same matching order selection technique as presented in Section A.4 in Appendix. The algorithms are implemented in C++ and compiled with GNU GCC with the -O3 flag.
- **Existing Algorithms.** We obtained the binary executable codes of the existing best-first search algorithm A*GED and

[3] A single binary executable code of all these algorithms can be downloaded from https://github.com/LijunChang/Graph_Edit_Distance



(a) AIDS (b) PubChem (c) Protein (d) G_R

Figure 5: Against existing algorithms regarding processing time (vary $|V|$)

Graphs	Avg #vertices	Avg #edges	#vertex labels	#edge labels
AIDS	25.6	27.6	62	3
PubChem	24	25.8	81	3
Protein	32.6	62	3	5

Table 3: Statistics of real graphs

the existing depth-first search algorithm CSI_GED from the authors of [29] and [11], respectively.

Note that, our DFS$^+$-LS algorithm is an improved version of the existing depth-first search algorithm DF_GED [1, 5] by merely speeding up the lower bound computation.

Real Graphs. We evaluate the algorithms on three sets of widely-used real graphs [11, 17, 29, 30]: AIDS, PubChem, and Protein. AIDS is an antivirus screen chemical compound dataset published by the Developmental Therapeutics Program at NCI/NIH[4], which contains 42,687 chemical compounds. PubChem is a chemical compound dataset[5] that contains one million graphs. Protein is a protein database from the Protein Data Bank[6], consisting of 600 protein structures. The average numbers of vertices and edges, and the numbers of distinct vertex labels and edge labels of these datasets are illustrated in Table 3.

Synthetic Graphs. We also generate synthetic random graphs G_R by the graph generator GraphGen[7], to evaluate the algorithms. Specifically, we generate 10 groups of random graphs G_R, with the number of vertices chosen from {10, 15, 20, 25, 30, 64, 128, 256, 512, 1024}, where large graphs (*i.e.*, with $|V| \geq 64$) are used for scalability testing and small graphs are used to compare the different algorithms. Each group of G_R contains 51 graphs with the same number of vertices, and is generated as follows. We first generate a graph with i vertices by invoking GraphGen, and then randomly apply x edit operations on the graph 10 times to get 10 graphs, where x is chosen from $\{1, 2, 3, 4, 5\}$ for small graphs and is chosen from $\{2, 5, 10, 20, 40\}$ for large graphs. Each graph generated by GraphGen has an edge density of 20%, 5 distinct vertex labels, and 2 distinct edge labels, similar to that used in [11].

Generate Query Graph Pairs. For each graph dataset and a specific number i of vertices, we first select the graphs whose sizes are within the range of $[i-2, i+2]$, and then partition the set of all graph pairs among the selected graphs into different groups with respect to their GED values. Finally, 10 graph pairs are randomly sampled from each group, for which we run the algorithms. Note that, for different graph pairs of similar sizes, in general the larger the GED value, the hard the computation. We choose GED = 9 by default.

Evaluation Metrics. For each testing, we report the processing time and the search space, where each reported result is an average of processing 10 graph pairs in a group. The search space of an algorithm is defined as the number of best extension computations, and indicates the space consumption of AStar$^+$ algorithms. We set a timeout of 1 hour (*i.e.*, 3.6×10^3 seconds). If an algorithm takes more than 1 hour to process one graph pair, then we record this time as 1 hour and label the algorithm with "TLE" in the plot. We report the processing time and search space of an algorithm as "om" if it runs out-of-memory. All experiments are conducted on a machine with an Intel Xeon(R) 3.40GHz CPU and 16GB main memory.

[4] http://dtp.nci.nih.gov/docs/aids/aids_data.html

[5] http://pubchem.ncbi.nlm.nih.gov

[6] http://www.iam.unibe.ch/fki/databases/iam-graph-database/download-the-iam-graph-database

[7] http://www.cse.cuhk.edu.hk/~jcheng/graphgen1.0.zip

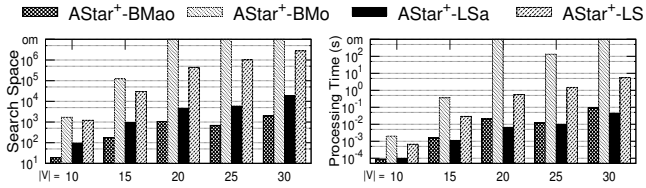
7.1 Experimental Results

Eval-I: Against Existing Algorithms. We first evaluate our algorithms AStar$^+$-BMao, DFS$^+$-LSa and AStar$^+$-LS against the existing algorithms DF_GED, CSI_GED and A*GED. The processing time of these algorithms on the four graphs by varying $|V|$ is illustrated in Figure 5, where the query graph pairs have GED 9. The time of DF_GED is demonstrated by that of DFS$^+$-LS; recall that DFS$^+$-LS uses the same search strategy and the same lower bound (*i.e.*, $\delta^{\text{LS}}(\cdot)$) as DF_GED, but uses a faster lower bound computation algorithm than DF_GED. We set the largest $|V|$ as 30 because the existing algorithms fail to process graphs with 30 or more vertices. *Our* AStar$^+$-BMao *algorithm significantly outperforms all existing algorithms, and the improvement of* AStar$^+$-BMao *over* CSI_GED*,* DFS$^+$-LS*, and* A*GED *can be more than* 4 *orders of magnitude.* For example, the average processing time of AStar$^+$-BMao on the PubChem graph with 30 vertices is less than 0.1 seconds, while CSI_GED takes more than 3×10^3 seconds.

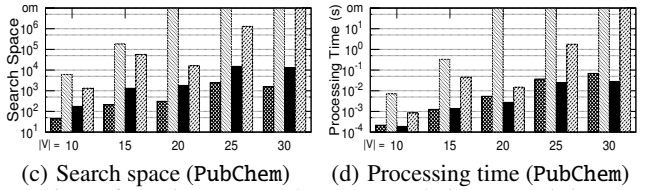
From Figure 5, we can also see that CSI_GED outperforms A*GED, but our AStar$^+$-LS algorithm that uses the same search strategy and the same lower bound as A*GED outperforms CSI_GED. This is because AStar$^+$-LS significantly improves A*GED by (1) our memory consumption reducing techniques (see Section 6.1) that make it not run out-of-memory, and (2) our linear-time best extension computation algorithm (see Section 5.1) that makes it run faster. *This demonstrates that the efficiency of best extension computation is critical to the efficiency of* GED *computation algorithms.*

In addition, it is interesting to observe that our depth-first algorithm DFS$^+$-LSa also outperforms the state-of-the-art depth-first algorithm CSI_GED. This is because our anchor-aware lower bound $\delta^{\text{LSa}}(\cdot)$ used in DFS$^+$-LSa is tighter than the degree-based lower bound $\delta^{\text{DE}}(\cdot)$ used in CSI_GED. *This demonstrates that the tightness of a lower bound also has a great impact on the performance of* GED *computation algorithms.* Thus in the following, we do not consider the lower bound $\delta^{\text{DE}}(\cdot)$, and only evaluate our algorithms.

Eval-II: Evaluate the Effectiveness of Anchor Aware. In this testing, we evaluate the effectiveness of our anchor-aware techniques for improving lower bound estimations. The search space and processing time of AStar$^+$-BMao, AStar$^+$-BMo, AStar$^+$-LSa, and AStar$^+$-LS are shown in Figure 6. We can see that *our anchor-aware technique significantly reduces the search space and thus the processing time of the algorithms*, *e.g.*, AStar$^+$-BMao v.s. AStar$^+$-BMo, and AStar$^+$-LSa v.s. AStar$^+$-LS. Note that, applying the anchor-aware techniques does not change the time complexity of best extension computation (see Section 5). Thus, in the following, we only consider anchor-aware lower bounds.

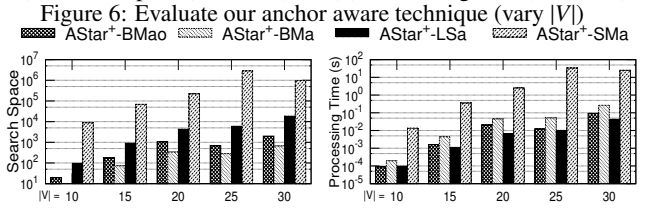


(a) Search space (AIDS) (b) Processing time (AIDS)

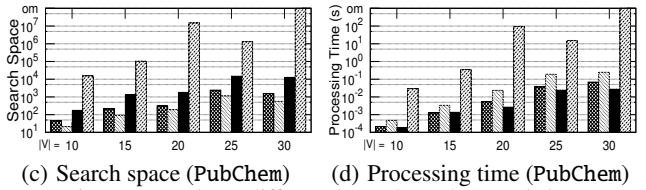


(c) Search space (PubChem) (d) Processing time (PubChem)

Figure 6: Evaluate our anchor aware technique (vary $|V|$)



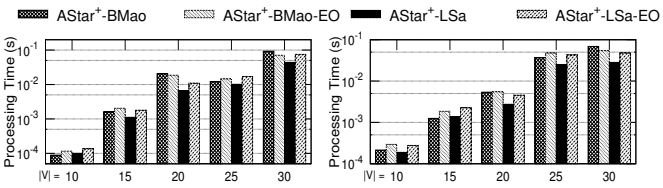
(a) Search space (AIDS) (b) Processing time (AIDS)



(c) Search space (PubChem) (d) Processing time (PubChem)

Figure 7: Evaluate different lower bounds (vary $|V|$)

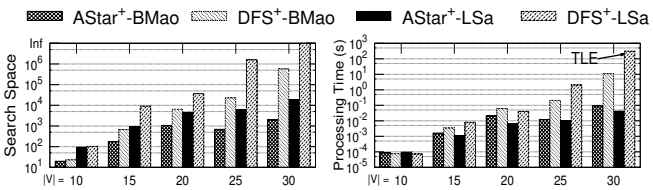
Eval-III: Evaluate Different Lower Bounds. In this set of experiments, we evaluate the effect of different anchor-aware lower bounds within our AStar$^+$ algorithm. Specifically, we evaluate $\delta^{\texttt{BMao}}(\cdot)$, $\delta^{\texttt{BMa}}(\cdot)$, $\delta^{\texttt{LSa}}(\cdot)$, and $\delta^{\texttt{SMa}}(\cdot)$. The results of search space and processing time are shown in Figure 7. The algorithms in increasing order w.r.t. the search space are: AStar$^+$-BMa, AStar$^+$-BMao, AStar$^+$-LSa, AStar$^+$-SMa. This indicates that, in practice $\delta^{\texttt{BMa}}(f) \geq \delta^{\texttt{BMao}}(f) \geq \delta^{\texttt{LSa}}(f) \geq \delta^{\texttt{SMa}}(f)$, since intuitively the tighter the lower bound the smaller the search space. However, regarding the processing time, the order of the algorithms AStar$^+$-BMa, AStar$^+$-BMao, and AStar$^+$-LSa is reversed. Recall that, the time complexities of best extension computation regarding $\delta^{\texttt{BMa}}(\cdot)$, $\delta^{\texttt{BMao}}(\cdot)$, and $\delta^{\texttt{LSa}}(\cdot)$ are to the power of four, three, and one, respectively (see Section 5). Thus, *there needs a trade-off between the tightness of a lower bound estimation and its computational efficiency, and a good lower bound estimation should be both tight and easy to compute.* This is also our motivation of designing the lower bound $\delta^{\texttt{BMao}}(\cdot)$. As the lower bounds $\delta^{\texttt{BMa}}(\cdot)$ and $\delta^{\texttt{SMa}}(\cdot)$ result in longer processing time for GED computation than $\delta^{\texttt{BMao}}(\cdot)$ and $\delta^{\texttt{LSa}}(\cdot)$, we exclude the former two lower bounds in the following.



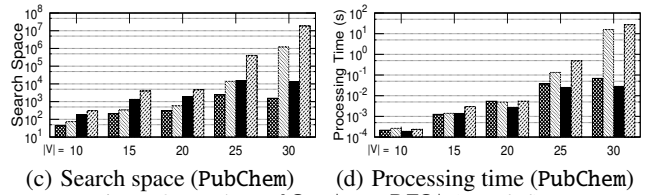
(a) AIDS (b) PubChem

Figure 8: Evaluate expand all strategy (processing time, vary $|V|$)

Eval-IV: Evaluate Expand All Strategy. To evaluate the effectiveness of the expand all strategy, we compare AStar$^+$-BMao and AStar$^+$-LSa with their counterparts AStar$^+$-BMao-EO, AStar$^+$-LSa-EO that only compute and materialize the best child when extending a partial mapping. The results are shown in Figure 8. Recall from Section 5 that for $\delta^{\texttt{LSa}}(\cdot)$, we need to compute the lower bound costs of all children of a partial mapping f in order to obtain the best extension of f, while for $\delta^{\texttt{BMao}}(\cdot)$, we can directly compute the best extension of f without computing the lower bound costs of all children of f. Thus, the expand all strategy consistently improves AStar$^+$-LSa while having little effect on AStar$^+$-BMao. In the following, we adopt the expand all strategy.



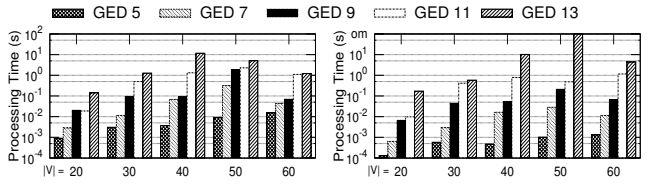
(a) Search space (AIDS) (b) Processing time (AIDS)



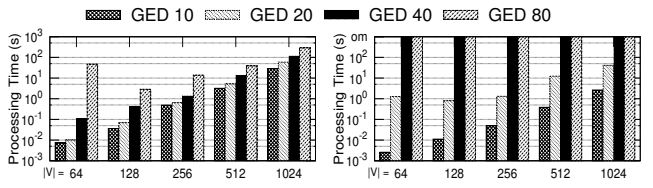
(c) Search space (PubChem) (d) Processing time (PubChem)

Figure 9: Evaluate AStar$^+$ v.s. DFS$^+$ (vary $|V|$)

Eval-V: Evaluate AStar$^+$ Against DFS$^+$. The results of evaluating the paradigms of AStar$^+$ and DFS$^+$ for GED computation are shown in Figure 9. We can see that *when using the same lower bound estimation technique,* AStar$^+$ *consistently has a smaller search space and thus runs faster than* DFS$^+$, and the improvement of AStar$^+$ over DFS$^+$ is more evident when the graph size increases. This confirms with our analytical results in Section 6.3. Thus, we recommend to use the best-first search strategy for GED computation.



(a) AStar$^+$-BMao on AIDS (b) AStar$^+$-LSa on AIDS



(c) AStar$^+$-BMao on G_R (d) AStar$^+$-LSa on G_R

Figure 10: Scalability testing of AStar$^+$-BMao and AStar$^+$-LSa

Eval-VI: Scalability Testing of AStar$^+$-BMao and AStar$^+$-LSa. We evaluate the scalability of AStar$^+$-BMao and AStar$^+$-LSa on AIDS and G_R for different GED values by varying $|V|$. The largest AIDS graph has around 60 vertices, and we generate G_R graphs with up-to 1024 vertices. The results in Figure 10 show that AStar$^+$-BMao scales well to large graphs and large GED values, while AStar$^+$-LSa may run out-of-memory due to using a looser lower bound (and thus having a larger search space); note that, $\delta^{\texttt{BMao}}(f) \geq \delta^{\texttt{LS}}(f)$ holds for every partial mapping f. Thus, AStar$^+$-BMao *is more scalable than* AStar$^+$-LSa *for* GED *computation.*

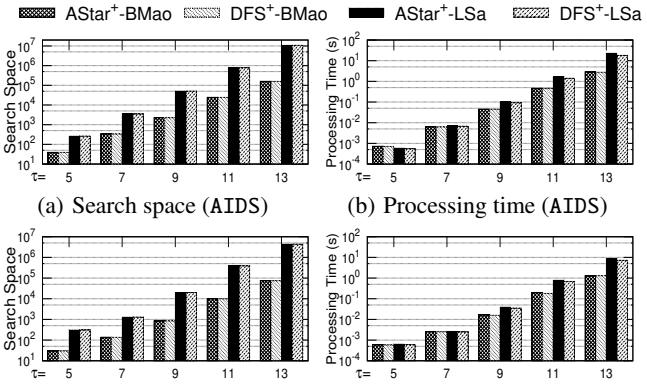
8. EXTENSION TO GED VERIFICATION

In this section, we investigate the problem of GED verification that verifies whether $\delta(q, g) \leq \tau$ for a user-given threshold τ. Our framework in Algorithm 2 as well as all our algorithms for GED computation can be easily extended for GED verification, as follows. The upper bound $\mathtt{UB}(q, g)$ is initialized to be $\tau + \epsilon$ at Line 2 for a small real value ϵ (*e.g.*, 0.001), and the framework returns `true` once the full mapping generated at Line 13 has an editorial cost at most τ. It is easy to see that both AStar$^+$ and DFS$^+$ will run faster for GED verification than for GED computation.

8.1 AStar$^+$ v.s. DFS$^+$ for GED Verification

AStar$^+$ is also better than DFS$^+$ for GED verification due to the following reasons. Firstly, if q and g are dissimilar (*i.e.*, $\delta(q, g) > \tau$), then the sets of partial mappings in $\mathcal{T}$ that are extended by AStar$^+$ and DFS$^+$ are the same (*i.e.*, $\mathcal{T}_{\leq\tau}$). Thus, AStar$^+$ and DFS$^+$ perform similarly for dissimilar pairs. Secondly, if q and g are similar (*i.e.*, $\delta(q, g) \leq \tau$), then both approaches can terminate early after finding a full mapping with editorial cost no larger than τ, which although may not be the one with the minimum editorial cost. Nevertheless, AStar$^+$ has a higher chance of finding a full mapping with editorial cost no larger than τ, due to the strategy of always extending the partial mapping with the minimum lower bound cost.

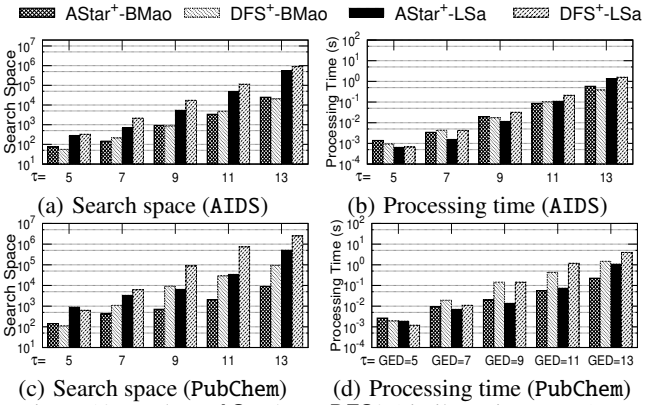
8.2 Experimental Results for GED Verification



(a) Search space (AIDS) (b) Processing time (AIDS)

(c) Search space (PubChem) (d) Processing time (PubChem)

Figure 11: Evaluate AStar$^+$ v.s. DFS$^+$ (dissimilar pairs, vary τ)

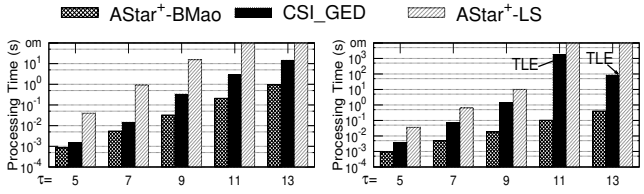


(a) Search space (AIDS) (b) Processing time (AIDS)

(c) Search space (PubChem) (d) Processing time (PubChem)

Figure 12: Evaluate AStar$^+$ v.s. DFS$^+$ (similar pairs, vary τ)

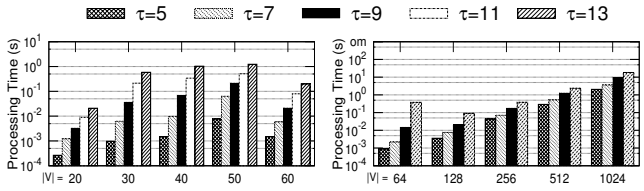
Eval-VII: Evaluate AStar$^+$ Against DFS$^+$. In this testing, we evaluate the paradigms of AStar$^+$ and DFS$^+$ for GED verification, where the set of query graph pairs is obtained as the union of the groups with 30 vertices and GED values in $\{5, 7, 9, 11, 13, \geq 14\}$. The results for dissimilar graph pairs are shown in Figure 11. We can see that AStar$^+$ and DFS$^+$ perform similarly by having the same search space, which conforms with the theoretical analysis in Section 8.1. The results for similar graph pairs are shown in Figure 12. We can see that, AStar$^+$ *runs slightly faster than* DFS$^+$ *by having a smaller search space.* Nevertheless, the difference is not as significant as for GED computation in Figure 9; this is because, the initial setting of $\mathtt{UB}(q, g)$ to be $\tau + \epsilon$ significantly reduces the search space of DFS$^+$. From Figures 11 and 12, we can also see that the lower bound $\delta^{\mathtt{BMao}}(\cdot)$ performs better than $\delta^{\mathtt{LSa}}(\cdot)$ for GED verification. Thus, in the following, we only consider AStar$^+$-BMao.



(a) AIDS (b) PubChem

Figure 13: Against CSI_GED for GED verification (vary τ)

Eval-VIII: Against Existing Algorithms. The results of evaluating AStar$^+$-BMao against CSI_GED and AStar$^+$-LS for GED verification are shown in Figure 13, where AStar$^+$-LS is an improved version of A*GED. AStar$^+$-BMao significantly outperforms CSI_GED and AStar$^+$-LS. This is mainly due to the tighter lower bound $\delta^{\mathtt{BMao}}(\cdot)$ used in AStar$^+$-BMao; that is, $\delta^{\mathtt{BMao}}(f) \geq \delta^{\mathtt{LSa}}(f) \geq \delta^{\mathtt{DE}}(f)$ (see Eval-III and Eval-I). It is also worth mentioning that, A*GED has been adopted for GED verification in the existing works on graph similarity search, but its improved version AStar$^+$-LS is significantly outperformed by AStar$^+$-BMao.



(a) AStar$^+$-BMao on AIDS (b) AStar$^+$-BMao on G_R

Figure 14: Scalability testing for GED verification (vary $|V|$)

Eval-IX: Scalability Testing of AStar$^+$-BMao. The results of evaluating the scalability of AStar$^+$-BMao for GED verification are shown in Figure 14. We can see that, AStar$^+$-BMao scales well to large graphs, and it scales better for GED verification than for GED computation in Figure 10 due to two reasons. Firstly, the initial upper bound $\mathtt{UB}(q, g)$ is smaller for GED verification. Secondly, AStar$^+$-BMao may terminate earlier if the heuristically generated full mapping at Line 13 of Algorithm 2 is no larger than τ.

9. CONCLUSION

In this paper, we proposed a general anchor-aware technique for improving the tightness of lower bound estimations, as well as efficient algorithms for computing the best extension of a partial mapping. Extensive performance studies confirmed our theoretical analysis of the superiority of AStar$^+$ over DFS$^+$, and demonstrated the effectiveness and efficiency of our techniques. In particular, our AStar$^+$-BMao algorithm outperforms the state-of-the-art algorithms by several orders of magnitude for both GED computation and GED verification. One possible direction of future work is to design better lower bound estimation techniques than $\delta^{\mathtt{BMao}}(\cdot)$. Another possible direction of future work is to extend our techniques to the problem of graph edit distance with non-uniform costs. It

will also be interesting to conduct an experimental study of the existing indexing techniques for graph similarity search by adopting our much better algorithm AStar$^+$-BMao for GED verification.

10. REFERENCES

[1] Zeina Abu-Aisheh, Romain Raveaux, Jean-Yves Ramel, and Patrick Martineau. An exact graph edit distance algorithm for solving pattern recognition problems. In *Proc. of ICPRAM'15*, pages 271–278, 2015.

[2] Faisal N. Abu-Khzam, Nagiza F. Samatova, Mohamad A. Rizk, and Michael A. Langston. The maximum common subgraph problem: Faster solutions via vertex cover. In *Proc. of AICCSA'07*, 2007.

[3] Montaine Bernard, Noël Richard, and Joël Paquereau. Functional brain imaging by eeg graph-matching. In *Proc. of EMB'06*, 2006.

[4] Fei Bi, Lijun Chang, Xuemin Lin, Lu Qin, and Wenjie Zhang. Efficient subgraph matching by postponing cartesian products. In *Proc. of SIGMOD'16*, 2016.

[5] David B. Blumenthal and Johann Gamper. Exact computation of graph edit distance for uniform and non-uniform metric edit costs. In *Proc. of GbRPR'17*, pages 211–221, 2017.

[6] Horst Bunke and Kim Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3-4):255–259, 1998.

[7] Fanny Chevalier, Jean-Philippe Domenger, Jenny Benois-Pineau, and Maylis Delest. Retrieval of objects in video by similarity based on graph matching. *Pattern Recognition Letters*, 2007.

[8] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2004.

[9] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

[10] Mirtha-Lina Fernández and Gabriel Valiente. A graph distance metric combining maximum common subgraph and minimum common supergraph. *Pattern Recognition Letters*, 22(6/7):753–758, 2001.

[11] Karam Gouda and Mosab Hassaan. CSI_GED: An efficient approach for graph edit similarity computation. In *Proc. of ICDE'16*, 2016.

[12] Wook-Shin Han, Jinsoo Lee, and Jeong-Hoon Lee. Turbo$_{\text{iso}}$: towards ultrafast and robust subgraph isomorphism search in large graph databases. In *Proc. of SIGMOD'13*, 2013.

[13] Derek Justice and Alfred Hero. A binary linear programming formulation of the graph edit distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(8):1200–1214, 2006.

[14] Ina Koch. Enumerating all connected maximal common subgraphs in two graphs. *Theor. Comput. Sci.*, 2001.

[15] Evgeny B. Krissinel and Kim Henrick. Common subgraph isomorphism detection by backtracking search. *Softw., Pract. Exper.*, 2004.

[16] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 1955.

[17] Yongjiang Liang and Peixiang Zhao. Similarity search in graph databases: A multi-layered indexing approach. In *Proc. of ICDE'17*, pages 783–794, 2017.

[18] James J. McGregor. Backtrack search algorithms and the maximal common subgraph problem. *Softw., Pract. Exper.*, 1982.

[19] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 1957.

[20] Michel Neuhaus and Horst Bunke. Edit distance-based kernel functions for structural pattern classification. *Pattern Recognition*, 2006.

[21] Hiroyuki Ogata, Wataru Fujibuchi, Susumu Goto, and Minoru Kanehisa. A heuristic graph comparison algorithm and its application to detect functionally related enzyme clusters. *Nucleic acids research*, 2000.

[22] John W. Raymond, Eleanor J. Gardiner, and Peter Willett. RASCAL: calculation of graph similarity using maximum common edge subgraphs. *Comput. J.*, 2002.

[23] Kaspar Riesen, Sandro Emmenegger, and Horst Bunke. A novel software toolkit for graph edit distance computation. In *Proc. of GbRPR'13*, pages 142–151, 2013.

[24] Kaspar Riesen, Stefan Fankhauser, and Horst Bunke. Speeding up graph edit distance computation with a bipartite heuristic. In *Proc. of MLG'07*, 2007.

[25] Antonio Robles-Kelly and Edwin R. Hancock. Graph edit distance from spectral seriation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(3):365–378, 2005.

[26] Alberto Sanfeliu and King-Sun Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Trans. Systems, Man, and Cybernetics*, 13(3):353–362, 1983.

[27] Xiaoli Wang, Xiaofeng Ding, Anthony K. H. Tung, Shanshan Ying, and Hai Jin. An efficient graph indexing method. In *Proc. of ICDE'12*, pages 210–221, 2012.

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

[29] Xiang Zhao, Chuan Xiao, Xuemin Lin, Qing Liu, and Wenjie Zhang. A partition-based approach to structure similarity search. *PVLDB*, 7(3):169–180, 2013.

[30] Xiang Zhao, Chuan Xiao, Xuemin Lin, and Wei Wang. Efficient graph similarity joins with edit distance constraints. In *Proc. of ICDE'12*, pages 834–845, 2012.

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

[32] Gaoping Zhu, Xuemin Lin, Ke Zhu, Wenjie Zhang, and Jeffrey Xu Yu. Treespan: efficiently computing similarity all-matching. In *Proc. of SIGMOD'12*, pages 529–540, 2012.

A. APPENDIX

A.1 Proofs of Lemmas and Theorems

Proof of Lemma 2.1. We prove the lemma by contradiction. Assume there is such a sequence of edit operations, $P = (eo_1, \ldots, eo_i, \ldots, eo_n)$ with length $n = \delta(q, g)$, that contains a vertex deletion. Without loss of generality, let eo_i be the operation of deleting a vertex v from q. Then, there must also exist a vertex insertion operation since $|V(q)| \leq |V(g)|$; let eo_j be the operation of inserting a vertex v' with label a, and j can be either smaller or larger than i. Now consider another sequence P' of edit operations that differs from P by removing eo_i and changing eo_j from vertex insertion to vertex relabeling (*i.e.*, change the label of v to a); note that, we also need to replace v', occurred in P, with v. It is easy to verify that P'

also transforms q into g, and $|P'| = |P| - 1$, which contradicts that P is optimal. Thus, the lemma holds. □

Proof of Theorem 3.1. We prove the theorem by contradiction. Assume Algorithm 2 outputs a wrong value for $\delta(q, g)$. Then, when the algorithm terminates, we must have $\texttt{UB}(q, g) > \delta(q, g)$ and $Q = \emptyset$. From Lemma 2.2, we know that $\delta(q, g)$ equals the minimum editorial cost among all full mappings from $V(q)$ to $V(g)$; without loss of generality, we assume that there is only one full mapping f^* with editorial cost $\delta(q, g)$. Then, at some point, an ancestor of f^* in the search tree $\mathcal{T}$ must be pruned by the algorithm, since the empty mapping f_0 which is an ancestor of f^* is initially contained in Q. A partial mapping f is pruned (at Lines 6 and 11) only if $\delta(f) \geq \texttt{UB}(q, g)$. However, all ancestors of f^* have lower bound costs no larger than $\delta(q, g)$, and $\texttt{UB}(q, g) > \delta(q, g)$. We reach a contradiction. Thus, the theorem holds. □

Proof of Lemma 4.1. It suffices to prove that for any mapping $\sigma \in \mathcal{F}(q\backslash f, g\backslash f)$, the following holds: $\sum_{v\in q\backslash f} \lambda^{\texttt{BMa}}(v, \sigma(v)) \geq \texttt{LSa}(q\backslash f, g\backslash f) = \Upsilon\Big(L_V(q\backslash f), L_V(g\backslash f)\Big) + \Upsilon\Big(L_{E_I}(q\backslash f), L_{E_I}(g\backslash f)\Big) + \sum_{v\in q[f]} \Upsilon\Big(L_{E_C}(v), L_{E_C}(f(v))\Big)$.

Firstly, we have,

$$\begin{aligned}
&\sum_{v\in q\backslash f} \lambda^{\texttt{BMa}}(v, \sigma(v)) \\
&= \sum_{v\in q\backslash f} \Big(\mathbb{1}_{l(v)\neq l(\sigma(v))} + \tfrac{1}{2} \times \Upsilon(L_{E_I}(v), L_{E_I}(\sigma(v))) + \sum_{v'\in q[f]} \mathbb{1}_{l(v,v')\neq l(\sigma(v),f(v'))}\Big) \\
&= \sum_{v\in q\backslash f} \mathbb{1}_{l(v)\neq l(\sigma(v))} + \tfrac{1}{2} \times \sum_{v\in q\backslash f} \Upsilon(L_{E_I}(v), L_{E_I}(\sigma(v))) \\
&\quad + \sum_{v\in q\backslash f} \sum_{v'\in q[f]} \mathbb{1}_{l(v,v')\neq l(\sigma(v),f(v'))}.
\end{aligned}$$

Secondly, it is easy to verify the following three inequalities.

(1) $\sum_{v\in q\backslash f} \mathbb{1}_{l(v)\neq l(\sigma(v))} \geq \Upsilon(L_V(q\backslash f), L_V(g\backslash f))$

since $L_V(q\backslash f) = \bigcup_{v\in q\backslash f} \{l(v)\}$ and $L_V(g\backslash f) = \bigcup_{v\in q\backslash f} \{l(\sigma(v))\}$.

(2) $\frac{1}{2} \times \sum_{v\in q\backslash f} \Upsilon(L_{E_I}(v), L_{E_I}(\sigma(v))) \geq \Upsilon(L_{E_I}(q\backslash f), L_{E_I}(g\backslash f))$

since $L_{E_I}(q\backslash f) \cup L_{E_I}(q\backslash f) = \bigcup_{v\in q\backslash f} L_{E_I}(v)$ and similarly for $L_{E_I}(g\backslash f)$.

(3) $\sum_{v\in q\backslash f} \sum_{v'\in q[f]} \mathbb{1}_{l(v,v')\neq l(\sigma(v),f(v'))} \geq \sum_{v\in q[f]} \Upsilon(L_{E_C}(v), L_{E_C}(f(v)))$

since $\sum_{v'\in q\backslash f} \mathbb{1}_{l(v',v)\neq l(\sigma(v'),f(v))} \geq \Upsilon(L_{E_C}(v), L_{E_C}(f(v)))$ holds for a vertex v, and

$$\sum_{v\in q\backslash f} \sum_{v'\in q[f]} \mathbb{1}_{l(v,v')\neq l(\sigma(v),f(v'))} = \sum_{v\in q[f]} \sum_{v'\in q\backslash f} \mathbb{1}_{l(v',v)\neq l(\sigma(v'),f(v))}.$$

Thus, the lemma holds. □

Proof of Theorem 5.1. It is easy to see that, if $C(v_i)$ consists of only one vertex (*i.e.*, $C(v_i) = \{u^*\}$), then Algorithm 4 correctly maps v_i to u^* and moreover the cost of the matching M equals $\texttt{BMa}_{v_i\mapsto u^*}(q\backslash f, g\backslash f)$. Now, we prove the theorem by contradiction. Assume the best extension of f is $\{v_i \mapsto u'\}$, *i.e.*, $\texttt{BMa}_{v_i\mapsto u'}(q\backslash f, g\backslash f) < \texttt{BMa}_{v_i\mapsto u^*}(q\backslash f, g\backslash f)$. Then, the computation of $\texttt{BMa}_{v_i\mapsto u'}(q\backslash f, g\backslash f)$ also implies a bipartite matching, which exists in the bipartite graph constructed by Algorithm 4 and has a smaller cost than M. This contradicts that M is the minimum cost perfect matching. Thus, the theorem holds. □

Proof of Lemma 6.1. Note that, a partial mapping is extended only after being popped from the priority queue Q. Consider the moment of popping from the priority queue Q a partial mapping f whose lower bound cost is larger than $\delta(q, g)$. The lower bound costs of all remaining partial mappings in Q are no smaller than the lower bound cost $\delta(f)$ of f, based on our strategy of popping from Q the partial mapping with the minimum lower bound cost. Then, the upper bound $\texttt{UB}(q, g)$ must equal $\delta(q, g)$; otherwise, an ancestor f' of the minimum-cost full mapping f^* (*i.e.*, with editorial cost $\delta(q, g)$) must be in Q since the ancestor f_0 (*i.e.*, the empty mapping) is initially in Q, which would contradict that f is popped from Q since $\delta(f') \leq \delta(q, g) < \delta(f)$. Thus, f will not be extended, and the lemma holds. □

Proof of Lemma 6.2. Consider an algorithm that does not extend all partial mappings of $\mathcal{T}_{<\delta(q,g)}$, and let f be such a partial mapping (*i.e.*, $\delta(f) < \delta(q, g)$). It is possible that the unique full mapping f^* with the minimum editorial cost from q to g is a descendant of f. Then, this algorithm will not visit f^* and thus will not report the correct GED between q and g. Therefore, the lemma holds. □

Proof of Lemma 6.3. It is easy to see that $\delta^{\texttt{BMao}}(f_2) \geq \delta^{\texttt{BMa}}(f_1)$ holds for each child f_2 of f_1 in the search tree $\mathcal{T}$; note that, for lower bounds, the larger the better. As a result, for each partial mapping $f \notin \mathcal{T}^{\texttt{BMa}}_{\leq\delta(q,g)}$, all its children are not in $\mathcal{T}^{\texttt{BMao}}_{\leq\delta(q,g)}$. Thus, the lemma holds. □

Proof of Lemma 6.4. Firstly, we prove that the number of partial mappings in Q after i iterations is $\leq i$ by induction, where an iteration is running once Lines 4–8 of Algorithm 2. Initially, the claim holds for the first iteration, since the empty mapping has no sibling; thus, we pop the empty mapping from Q and push its best child into Q. Assume the claim holds for $i \geq 1$, we prove that it also holds for $i + 1$. At the $(i + 1)$-th iteration, we pop one partial mapping from Q and push at most two partial mappings into Q (one at Line 7 and another at Line 8); the number of partial mappings in Q increases by at most 1 and thus becomes $\leq i + 1$. Therefore, the claim holds for all iterations.

Secondly, following from Lemma 6.1, the number of iterations before popping a partial mapping f with $\delta(f) \geq \texttt{UB}(q, g)$ is at most $|\mathcal{T}_{\leq\delta(q,g)}|$. Moreover, after popping the first partial mapping with lower bound cost $\geq \texttt{UB}(q, g)$, no new partial mappings will be generated or pushed into Q. Thus, the lemma holds. □

Proof of Theorem 6.1. *The information of each partial mapping (*i.e.*, each entry) in Q can be stored in constant space* as follows. (1) For a partial mapping f at level i, we only store the vertex of $V(g)$ to which $v_i \in V(q)$ maps, while other parts of f can be retrieved from its ancestors in the search tree $\mathcal{T}$. (2) The level number and the lower bound cost of f also take constant spaces. (3) We store a pointer for f pointing to its immediate preceding sibling, from which f is generated; thus, $C(v_i)$ as needed at Line 7 can be retrieved online in $O(|V(g\backslash f)|)$ time. As a result, after a partial mapping being popped from Q, we do not remove it from the main memory. Nevertheless, the space complexity of AStar$^+$ is still bounded by $O(|\mathcal{T}_{\leq\delta(q,g)}|)$. □

Proof of Theorem 6.2. Following from Lemma 6.4, AStar$^+$ runs for at most $O(|\mathcal{T}_{\leq\delta(q,g)}|)$ iterations; that is, *the search space of* AStar$^+$ *is* $O(|\mathcal{T}_{\leq\delta(q,g)}|)$. Each iteration consists of one pop operation (Line 5) and at most two push operations (Line 12) of Q, each with time complexity $O(\log|\mathcal{T}_{\leq\delta(q,g)}|)$, and at most two computations of the best extension (Line 10), each with time complexity $\texttt{T}_{\texttt{BE}}$. Thus, the theorem holds. □

A.2 Edit Distance between Two Multi-sets

Given two multi-sets S_1 and S_2, the edit distance between S_1 and S_2, denoted by $\Upsilon(S_1, S_2)$, is the minimum number of edit operations that transform S_1 to S_2, where edit operations are (1) inserting an element, (2) deleting an element, and (3) replacing an

element. $\Upsilon(\cdot,\cdot)$ is a metric, and $\Upsilon(\emptyset,\emptyset) = 0$ and $\Upsilon(\emptyset, S) = |S|$. In general,

$$\Upsilon(S_1, S_2) = \max\{|S_1|, |S_2|\} - |S_1 \cap S_2|,$$

where $S_1 \cap S_2$ and $S_1 \cup S_2$ denote the multi-set intersection and multi-set union, respectively. For example, if $S_1 = \{a, a, b\}$ and $S_2 = \{a, a, a\}$, the edit distance between S_1 and S_2 is 1 (*i.e.*, replace b in S_1 with a); $S_1 \cap S_2 = \{a, a\}$, and $S_1 \cup S_2 = \{a, a, a, a, a, b\}$. It is easy to see that by using a hash structure, $\Upsilon(S_1, S_2)$ can be computed in $O(|S_1| + |S_2|)$ time.

The properties of $\Upsilon(\cdot,\cdot)$ are shown in the following two lemmas.

Lemma A.1: *Given four multi-sets S_1, S_2, S_1', S_2', we have $\Upsilon(S_1 \cup S_1', S_2 \cup S_2') \leq \Upsilon(S_1, S_2) + \Upsilon(S_1', S_2')$.*

Proof: We have

$$\begin{aligned}
&\Upsilon(S_1, S_2) + \Upsilon(S_1', S_2') - \Upsilon(S_1 \cup S_1', S_2 \cup S_2') \\
=&\max\{|S_1|, |S_2|\} - |S_1 \cap S_2| + \max\{|S_1'|, |S_2'|\} - |S_1' \cap S_2'| \\
&- \max\{|S_1 \cup S_1'|, |S_2 \cup S_2'|\} + |(S_1 \cup S_1') \cap (S_2 \cup S_2')| \\
=&(\max\{|S_1|, |S_2|\} + \max\{|S_1'|, |S_2'|\} - \max\{|S_1| + |S_1'|, |S_2| + |S_2'|\}) \\
&+ (|(S_1 \cup S_1') \cap (S_2 \cup S_2')| - |S_1 \cap S_2| - |S_1' \cap S_2'|) \\
\geq&0 + 0 = 0
\end{aligned}$$

Thus, the lemma holds. □

Lemma A.2: *Given two multi-sets S_1, S_2, we have $\Upsilon(S_1 \cup S_1, S_2 \cup S_2) = \Upsilon(S_1, S_2) + \Upsilon(S_1, S_2)$.*

Proof: This lemma can be proved in a similar way to the proof of Lemma A.1. □

The above two lemmas can also be naturally extended to the union of multiple multi-sets.

A.3 Star Match-based Lower Bounds

Star Match-based Lower Bound $\mathtt{SM}(\cdot,\cdot)$. Lower bound based on the star structure is proposed in [28], where edges have no labels. We extend it to handle edge labels as follows.

Definition A.1: The **star** of a vertex v in a graph q is $S(v) = (l(v), L_E(v), L_V(v))$, where $L_V(v)$ denotes the multi-set of labels of v's one-hop neighbors.

Based on the star structures $S(v)$ and $S(u)$, we define the mapping cost of mapping $v \in q\backslash f$ to $u \in g\backslash f$ as,

$$\lambda^{\mathtt{SM}}(v, u) := \mathbb{1}_{l(v)\neq l(u)} + \tfrac{1}{2} \times \Upsilon(L_E(v), L_E(u)) + \Upsilon(L_V(v), L_V(u))$$

where $L_V(v)$ and $L_V(u)$ only consider the neighbors of v and u in $q\backslash f$ and $g\backslash f$, respectively. Thus, $\lambda^{\mathtt{SM}}(v, u) = \lambda^{\mathtt{BM}}(v, u) + \Upsilon(L_V(v), L_V(u))$. The star match-based lower bound [28] is,

$$\mathtt{SM}(q\backslash f, g\backslash f) := \frac{\min_{\sigma\in\mathcal{F}(q\backslash f, g\backslash f)} \sum_{v\in q\backslash f} \lambda^{\mathtt{SM}}(v, \sigma(v))}{\max\{4, \Delta(q\backslash f) + 1, \Delta(g\backslash f) + 1\}}$$

where $\Delta(q\backslash f)$ and $\Delta(g\backslash f)$ denote the maximum vertex degree in $q\backslash f$ and $g\backslash f$, respectively.

Anchor-aware Star Match-based Lower Bound. Similarly, we revise the star structure to define the mapping cost of mapping $v \in q\backslash f$ to $u \in g\backslash f$ as,

$$\begin{aligned}
\lambda^{\mathtt{SMa}}(v, u) :=& \mathbb{1}_{l(v)\neq l(u)} + \tfrac{1}{2} \times \Upsilon(L_{E_I}(v), L_{E_I}(u)) \\
&+ \textstyle\sum_{v'\in q[f]} \mathbb{1}_{l(v,v')\neq l(u, f(v'))} + \Upsilon(L_V(v), L_V(u)).
\end{aligned}$$

That is, $\lambda^{\mathtt{SMa}}(v, u) = \lambda^{\mathtt{BMa}}(v, u) + \Upsilon(L_V(v), L_V(u))$. Then, we define the anchor-aware star match-based lower bound as,

$$\mathtt{SMa}(q\backslash f, g\backslash f) := \frac{\min_{\sigma\in\mathcal{F}(q\backslash f, g\backslash f)} \sum_{v\in q\backslash f} \lambda^{\mathtt{SMa}}(v, \sigma(v))}{\max\{4, \Delta(q\backslash f) + 1, \Delta(g\backslash f) + 1\}}.$$

It can be easily verified that $\lambda^{\mathtt{SMa}}(v, u) \geq \lambda^{\mathtt{SM}}(v, u)$, and thus we have $\mathtt{SMa}(q\backslash f, g\backslash f) \geq \mathtt{SM}(q\backslash f, g\backslash f)$.

We prove the following lemma regarding $\mathtt{BMa}(\cdot,\cdot)$ and $\mathtt{SMa}(\cdot,\cdot)$.

Lemma A.3: *For a partial mapping f, we have $\mathtt{BMa}(q\backslash f, g\backslash f) \geq \mathtt{SMa}(q\backslash f, g\backslash f)$ if $\mathtt{BMa}(q\backslash f, g\backslash f) \geq |V(q\backslash f)|$.*

Proof: Let d be $\max\{4, \Delta(q\backslash f) + 1, \Delta(g\backslash f) + 1\}$, and σ be the mapping obtained by

$$\arg\min_{\sigma'\in\mathcal{F}(q\backslash f, g\backslash f)} \textstyle\sum_{v\in q\backslash f} \lambda^{\mathtt{BMa}}(v, \sigma'(v)).$$

Then, we have

$$\begin{aligned}
&d \times (\mathtt{BMa}(q\backslash f, g\backslash f) - \mathtt{SMa}(q\backslash f, g\backslash f)) \\
=&(d \times \min_{\sigma'\in\mathcal{F}(q\backslash f, g\backslash f)} \textstyle\sum_{v\in q\backslash f} \lambda^{\mathtt{BMa}}(v, \sigma'(v))) \\
&- \min_{\sigma''\in\mathcal{F}(q\backslash f, g\backslash f)} \textstyle\sum_{v\in q\backslash f} \lambda^{\mathtt{SMa}}(v, \sigma''(v)) \\
\geq&(d \times \textstyle\sum_{v\in q\backslash f} \lambda^{\mathtt{BMa}}(v, \sigma(v))) - \sum_{v\in q\backslash f} \lambda^{\mathtt{SMa}}(v, \sigma(v)) \\
=&\textstyle\sum_{v\in q\backslash f} (d \times \lambda^{\mathtt{BMa}}(v, \sigma(v)) - \lambda^{\mathtt{SMa}}(v, \sigma(v)))
\end{aligned}$$

Consider each component in the last expression and let u denote $\sigma(v)$. Based on the property that $\lambda^{\mathtt{SMa}}(v, u) = \lambda^{\mathtt{BMa}}(v, u) + \Upsilon(L_V(v), L_V(u))$, we have

$$\begin{aligned}
&d \times \lambda^{\mathtt{BMa}}(v, u) - \lambda^{\mathtt{SMa}}(v, u) \\
=&d \times \lambda^{\mathtt{BMa}}(v, u) - \left(\lambda^{\mathtt{BMa}}(v, u) + \Upsilon(L_V(v), L_V(u))\right) \\
=&(d - 1) \times \lambda^{\mathtt{BMa}}(v, u) - \Upsilon(L_V(v), L_V(u)) \\
\geq&(d - 1) \times (\lambda^{\mathtt{BMa}}(v, u) - 1)
\end{aligned}$$

where the last inequality follows from the fact that $\Upsilon(L_V(v), L_V(u)) \leq \max\{|L_V(v)|, |L_V(u)|\} \leq d - 1$.

Thus, from the above, we have

$$\begin{aligned}
&d \times (\mathtt{BMa}(q\backslash f, g\backslash f) - \mathtt{SMa}(q\backslash f, g\backslash f)) \\
\geq&\textstyle\sum_{v\in q\backslash f} (d \times \lambda^{\mathtt{BMa}}(v, \sigma(v)) - \lambda^{\mathtt{SMa}}(v, \sigma(v))) \\
\geq&(d - 1) \times \textstyle\sum_{v\in q\backslash f} (\lambda^{\mathtt{BMa}}(v, \sigma(v)) - 1) \\
=&(d - 1) \times (\mathtt{BMa}(q\backslash f, g\backslash f) - |V(q\backslash f)|)
\end{aligned}$$

Therefore, if $\mathtt{BMa}(q\backslash f, g\backslash f) \geq |V(q\backslash f)|$, then we have $\mathtt{BMa}(q\backslash f, g\backslash f) \geq \mathtt{SMa}(q\backslash f, g\backslash f)$. □

Note that, the above lemma is conservative, while in practice, $\mathtt{SMa}(q\backslash f, g\backslash f)$ is even smaller than $\mathtt{LSa}(q\backslash f, g\backslash f)$ as verified by our experiments in Section 7. The main reason is that, as the label of a vertex v is considered multiple times in the star structures of v's neighbors, the mapping cost $\lambda^{\mathtt{SMa}}(v, u)$ has to be normalized by a large factor of $\max\{4, \Delta(q\backslash f) + 1, \Delta(g\backslash f) + 1\}$.

A.4 A Frequency-Aware Mapping Order

The mapping order π also has an impact on the performance of Algorithm 2, in a similar way to the impact of search order on the performance of subgraph matching algorithms (*e.g.*, see [4, 8, 12]). However, the existing techniques of computing a search order for subgraph matching cannot be applied to GED computation, since the vertex mapping in GED is unconstrained (*i.e.*, a vertex of q can map to any vertex of g regardless of their labels). Thus, we propose techniques to compute a mapping order for GED computation. We start with our two main intuitions.

Firstly, a connected mapping order is preferred; that is, each vertex v should be connected to one of the vertices preceding v in π. The intuition is that partial mappings obtained by a connected mapping order tend to have tighter lower bound costs, and the number of partial mappings enumerated by a GED computation algorithm

becomes smaller if the lower bound gets tighter. Secondly, infrequent part of a graph should be mapped first. The intuition is that an infrequent subgraph of q has a smaller number of similar mapping subgraphs in g. Consequently, most of the partial mapping generated by this infrequent subgraph have large lower bound costs, due to the large $\delta_f(q[f], g[f])$ which is a part of the lower bound cost $\delta(f)$, and thus will not be extended or generated by a GED computation algorithm. In contrast, a frequent subgraph of q has a lot of similar mapping subgraphs in g, and thus will generate a lot of partial mappings with small lower bound costs.

Based on the above intuitions, we propose our mapping order computation algorithm, denoted by MappingOrder. To quantify the infrequency of a subgraph, we compute an infrequency weight $w(\cdot)$ for each vertex and each edge of q, which is one minus the label's frequency in g for the corresponding vertex or edge. Thus, a subgraph is more infrequent if it has a larger total weight for vertices and edges. We use a greedy strategy to construct the mapping order π. The first vertex v_1 is chosen as the one with the largest total weight for the vertex and its adjacent edges. Then, we iteratively add into π the vertex that has the largest total weight for the vertex and its adjacent edges to vertices in π.

A.5 Running Example of AStar$^+$

Consider the graphs q and g in Figures 3(a) and 3(b), respectively. Assume the matching order in computing $\delta(q, g)$ is $\pi = (v_1, v_2, v_3, v_4, v_5)$, and the lower bound costs of partial mappings are as shown in Figure 3(c) which are actually computed by $\delta^{\text{BMa}}(\cdot)$. AStar$^+$ runs as follows. Initially, $Q = \{f_0\}$, $\text{UB}(q, g) = +\infty$. In the first iteration, we pop f_0 from Q, and push its best child f_1 into Q. Assume the full mapping obtained by extending f_1 is $(u_1, u_4, u_3, u_5, u_2)$ which has an editorial cost 7, we update $\text{UB}(q, g)$ as 7. Thus, $Q = \{f_1\}$ and $\text{UB}(q, g) = 7$. In the second iteration, we pop f_1 from Q, and push both its best ungenerated sibling f_2 and its best child f_6 into Q. The full mappings that extend f_2 and f_6 for updating the upper bound are $(u_4, u_1, u_3, u_5, u_2)$ and $(u_1, u_2, u_3, u_5, u_4)$, respectively; the upper bound $\text{UB}(q, g)$ remains 7. Thus, $Q = \{f_2, f_6\}$ and $\text{UB}(q, g) = 7$. In the third iteration, we pop f_2 from Q and push both its best ungenerated sibling f_3 and its best child f_{10} into Q. The full mappings that extend f_3 and f_{10} for updating the upper bound are $(u_3, u_1, u_4, u_5, u_2)$ and $(u_4, u_3, u_1, u_2, u_5)$, respectively; the upper bound $\text{UB}(q, g)$ is updated as 5. Thus, $Q = \{f_6, f_3, f_{10}\}$ and $\text{UB}(q, g) = 5$. In the fourth iteration, we pop f_{10} from Q and push its best child f_{17} into Q; note that, its best ungenerated sibling f_{11} is not pushed into Q since its lower bound cost is larger than $\text{UB}(q, g)$. The full mapping that extends f_{17} for updating the upper bound is $(u_4, u_3, u_2, u_1, u_5)$, which updates $\text{UB}(q, g)$ as 4. Thus, $Q = \{f_6, f_3, f_{17}\}$ and $\text{UB}(q, g) = 4$. Then, partial mappings are iteratively popped from Q without being extended since their lower bound costs are no smaller than $\text{UB}(q, g)$. As a result, the GED between q and g is reported as 4.

A.6 Running Example of DFS$^+$

Reconsider the above running example. Now, we run DFS$^+$ to compute the GED between q and g in Figures 3(a) and 3(b), respectively. The initialization and the first two iterations are the same as that of running AStar$^+$, and we have $Q = \{f_2, f_6\}$ and $\text{UB}(q, g) = 7$. In the third iteration, we pop f_6 from Q since its level number is the largest, and push both its best ungenerated sibling f_7 and its best child f_{14} into Q. The full mappings that extend f_7 and f_{14} for updating the upper bound are $(u_1, u_4, u_3, u_5, u_2)$ and $(u_1, u_2, u_3, u_4, u_5)$, respectively; the upper bound $\text{UB}(q, g)$ is updated as 5. Thus $Q = \{f_2, f_7, f_{14}\}$ and $\text{UB}(q, g) = 5$. Then, f_{14} and f_7 are popped from Q without being extended in the fourth and fifth iterations, respectively, since their lower bound costs are no less than $\text{UB}(q, g)$. Thus $Q = \{f_2\}$ and $\text{UB}(q, g) = 5$. In the sixth iteration, We pop f_2 from Q and update Q and $\text{UB}(q, g)$ in a similar fashion to the third iteration of the running example of AStar$^+$. Thus $Q = \{f_3, f_{10}\}$ and $\text{UB}(q, g) = 5$. In the seventh iteration, we pop f_{10} from Q and update Q and $\text{UB}(q, g)$ in a similar way to the fourth iteration of running AStar$^+$. Thus $Q = \{f_3, f_{17}\}$ and $\text{UB}(q, g) = 4$. Then, partial mappings f_{17} and f_3 are iteratively popped from Q without being extended since their lower bound costs are no smaller than $\text{UB}(q, g)$. As a result, the GED between q and g is reported as 4.