Accelerating Graph Similarity Search via Efficient GED Computation

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

Xing Feng
University of Technology Sydney
xingfeng.unsw@gmail.com

Kai Yao
The University of Sydney
kyao8420@uni.sydney.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 the graph edit distance (GED) between graphs is the core operation in graph similarity search. Recent studies suggest that the existing index structures are ineffective in reducing the overall processing time of graph similarity search, and that directly verifying the GED between the query graph and every data graph in the database is still the best option. The state-of-the-art algorithm for GED verification is the recently proposed AStar-LSa. However, AStar-LSa may consume an extremely large amount of main memory or even run out-of-memory, when the graphs become larger and/or the GED threshold becomes larger. In this paper, we aim to improve the efficiency of GED verification and simultaneously lower the main memory consumption. To achieve that, we propose a new estimation for the lower bounds of partial mappings between graphs. We formally prove that our new lower bound is tighter than the one used in AStar-LSa. Moreover, we also propose efficient algorithms to compute the lower bounds, as well as optimization techniques to improve the efficiency. Empirical studies on real datasets demonstrate that our newly proposed algorithm AStar-BMao runs faster, and at the same time consumes much less main memory, than AStar-LSa.**

I. Introduction

Retrieving all occurrences of a query graph in a graph database is a fundamental problem in graph database research. Here, the graph database contains a large quantity (*e.g.*, thousands or millions) of small to medium sized graphs. Example graph databases include a database of chemical compounds, a database of proteins, a database of program call graphs, and etc.[1] In many applications, searching for the exact occurrences of a query graph may return no result or very few results that are not sufficient for the applications. This could be the result of erroneous data entry, data noise, or even the nature of the applications. An immediate remedy is to search for inexact occurrences, *i.e.*, retrieving all graphs in the database that are *similar* to the query graph, which has been extensively studied recently [7], [9], [11], [15], [26], [28], [29], [30].

Given a graph database $\mathcal{D}$ that contains a collection of small to medium sized data graphs, the problem of *graph similarity search* takes a query graph q and a threshold τ as input, and outputs all data graphs in $\mathcal{D}$ that are similar to q, *e.g.*, see Figure 1. That is, $\mathsf{results}(q,\tau) = \{g \in \mathcal{D} \mid \mathtt{sim}(q,g) \geq \tau\}$

[1] http://www.fki.inf.unibe.ch/databases/iam-graph-database

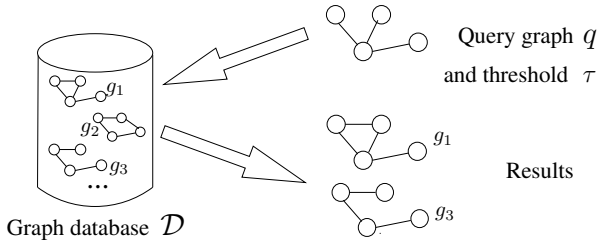


Fig. 1. Graph similarity search

where $\mathtt{sim}(q,g)$ is the similarity between q and g. Among various (dis-)similarity measures, *graph edit distance* (GED) has been widely adopted by the existing works (*e.g.*, see [7], [9], [11], [15], [26], [28], [29], [30]). This is because GED has several nice properties, *e.g.*, it is a metric, it is applicable to all types of graphs, and it captures the structural difference between graphs. Specifically, the GED between graphs q and g, denoted $\mathsf{ged}(q,g)$, is the minimum number of edit operations that are needed to transform q into g, where the edit operations are *edge insertion/deletion/relabeling* and *vertex insertion/deletion/relabeling*; note that, a vertex can be deleted only when it has no adjacent edges. GED gives the minimum amount of distortion needed to transform one graph into the other, and $\mathsf{ged}(q,g) = \mathsf{ged}(g,q)$. The result of (GED-based) graph similarity search then is $\{g \in \mathcal{D} \mid \mathsf{ged}(q,g) \leq \tau\}$.

As GED computation is NP-hard [27], most of the existing works for graph similarity search adopt the filtering-and-verification paradigm aiming to reduce the number of GED verifications (*i.e.*, verifying whether $\mathsf{ged}(q,g) \leq \tau$) by various offline constructed indexes, *e.g.*, q-gram-based index [28], star structure-based index [26], and subgraph-based index [15], [29]. That is, for a given query, the index is firstly probed online to filter out unpromising data graphs. The main focus of the existing studies is on designing different index structures, based on the premise that GED verification is extremely slow. Recently, an algorithm AStar-LSa is proposed in [7], which significantly improves the efficiency of GED verification compared to the algorithms used in previous works. Moreover, it is observed in [7] that, when AStar-LSa is adopted for GED verification, the existing index structures have very limited

effectiveness, *e.g.*, the improvement of using Pars [29] for filtering will be at most 52% than directly verifying all graphs of $\mathcal{D}$ by AStar-LSa.

In this paper, along the same line as [7], [9], [11], we aim to further improve the efficiency of GED verification and computation. Note that, besides graph similarity search, GED verification/computation also has many other applications, *e.g.*, in graph classification [17], graph clustering [23], biochemistry [18] and medicine [4]. The state-of-the-art algorithm for GED verification/computation is AStar-LSa [7]. It computes $\mathsf{ged}(q, g)$ by enumerating (vertex) mappings from $V(q)$ to $V(g)$, where each mapping $f : V(q) \rightarrow V(g)$ induces an editorial cost. The minimum editorial cost among all full mappings from $V(q)$ to $V(g)$ equals $\mathsf{ged}(q, g)$. As there is an exponential number of vertex mappings, AStar-LSa prunes all full mappings that share the same prefix (*i.e.*, partial mapping) f if the lower bound cost of f, denoted lb_f, is larger than τ (which implies that all full mappings that take f as a prefix have editorial costs larger than τ). The efficiency of AStar-LSa mainly comes from two aspects. Firstly, AStar-LSa designed the anchor-aware label set-based lower bound $\mathsf{lb}^{\mathtt{LSa}}$ which is much tighter than the label set-based lower bound $\mathsf{lb}^{\mathtt{LS}}$ used in previous algorithms. Secondly, AStar-LSa proposed a novel algorithm for computing the lower bound costs of all children of a partial mapping in totally linear time.

We observe that the main memory consumption of AStar-LSa increases very fast when either the graph size or the threshold τ becomes larger. This limits AStar-LSa from scaling to larger graphs or larger threshold values. As the main memory consumption of AStar-LSa, and more generally of the search paradigm of AStar-LSa, is inversely related to the tightness of the lower bound estimation, in this paper we first propose an anchor-aware branch match-based lower bound $\mathsf{lb}^{\mathtt{BMa}}$. We formally prove the correctness of $\mathsf{lb}^{\mathtt{BMa}}$ and that $\mathsf{lb}^{\mathtt{BMa}}$ is tighter than $\mathsf{lb}^{\mathtt{LSa}}$, *i.e.*, $\mathsf{lb}^{\mathtt{BMa}}_f \geq \mathsf{lb}^{\mathtt{LSa}}_f$ holds for any mapping f. We show that $\mathsf{lb}^{\mathtt{BMa}}_h$ for all children h of f can be computed in $O\left((|V(q)| + |V(g)|)^4\right)$ total time. However, due to the high time complexity of computing the lower bound costs regarding $\mathsf{lb}^{\mathtt{BMa}}$, the resulting algorithm AStar-BMa, despite of having a much smaller search space, may run slower than AStar-LSa. To strike a balance between the tightness and the efficiency of lower bound estimation, we then slightly loose the lower bound $\mathsf{lb}^{\mathtt{BMa}}$ into $\mathsf{lb}^{\mathtt{BMao}}$ such that the lower bound costs of all children of f regarding $\mathsf{lb}^{\mathtt{BMao}}$ can be computed in $O\left((|V(q)| + |V(g)|)^3\right)$ total time. As a result, AStar-BMao runs faster and scales better, due to having a much smaller search space, than the state-of-the-art algorithm AStar-LSa.

Contributions. Our main contributions are as follows.

- We design an anchor-aware branch match-based lower bound $\mathsf{lb}^{\mathtt{BMa}}$, and an algorithm to compute the lower bound cost for all children of a partial mapping in $O\left((|V(q)| + |V(g)|)^4\right)$ total time.
- We formally prove the correctness of $\mathsf{lb}^{\mathtt{BMa}}$, and prove that $\mathsf{lb}^{\mathtt{BMa}}$ is tighter than $\mathsf{lb}^{\mathtt{LSa}}$.
- We slightly loose the lower bound $\mathsf{lb}^{\mathtt{BMa}}$ into $\mathsf{lb}^{\mathtt{BMao}}$ such that we are able to propose an algorithm to compute the lower bound cost for all children of a partial mapping in $O\left((|V(q)| + |V(g)|)^3\right)$ total time.

We conduct extensive performance studies on both real datasets and synthetic datasets. The results confirm that our algorithm AStar-BMa that uses the lower bound $\mathsf{lb}^{\mathtt{BMa}}$ has the smallest search space and thus smallest main memory consumption. Nevertheless, our algorithm AStar-BMao that uses the lower bound $\mathsf{lb}^{\mathtt{BMao}}$ runs faster, as it computes the lower bounds much faster while only increasing the search space slightly. When compared to the state-of-the-art algorithm AStar-LSa, our algorithm AStar-BMao has a much smaller main memory consumption and also runs faster for both graph similarity search and GED verification/computation.

Organization. The rest of the paper is organized as follows. A brief overview of related works is given below. Preliminaries are presented in Section II. We present the state-of-the-art approach AStar-LSa in Section III. We design a tighter lower bound $\mathsf{lb}^{\mathtt{BMa}}$ and compare it with the existing lower bound $\mathsf{lb}^{\mathtt{LSa}}$ in Section IV, which results in our algorithm AStar-BMa. We propose to trade tightness for efficiency by slightly loosing the tightness of $\mathsf{lb}^{\mathtt{BMa}}$ in Section V-C, which results in our algorithm AStar-BMao. We report the results of our experimental studies in Section VI. Finally, Section VII concludes the paper.

Related Works. Related works are categorized as follows.
(1) Graph Similarity Search. GED-based graph similarity search has been extensively studied, *e.g.*, in [7], [9], [11], [15], [26], [28], [29], [30]. Most of the existing works focus on designing effective index structures — such as q-gram-based index [28], star structure-based index [26], and subgraph-based index [15], [29] — to filter out as many unpromising data graphs (*i.e.*, dissimilar to the query graph) as possible. Some recent works suggest index-free approaches for graph similarity search without pre-constructing an index offline, *e.g.*, [7], [9], [11]. Among them, Inves [11] conducts online graph partitioning-based filtering for GED verification, while AStar-LSa [7] and CSI_GED [9] directly verify every data graph in the database with the query graph. It is shown in [7] that the existing index structures are ineffective in reducing the overall processing time of graph similarity search, and that AStar-LSa outperforms both Inves and CSI_GED. In this paper, we follow the index-free approach, and propose a better algorithm for GED verification/computation.
(2) GED Computation. The notion of GED was proposed in [24] to quantify the distance between two graphs. Zeng *et al.* [27] proved that computing the exact GED is NP-hard. Nevertheless, algorithms have been designed for computing the exact GED in practice. A best-first search algorithm A*GED is developed in [21], [22], and depth-first search algorithms DF_GED [2], [5] and CSI_GED [9] are shown to outperform A*GED. The recently proposed AStar-LSa [7] is the state-of-the-art algorithm. In this paper, we propose a new algorithm AStar-BMao which improves AStar-LSa regarding both processing time and main memory usage. Note that, all these algorithms can compute as well as verify GED.

(3) Maximum Common Subgraph. Measuring the similarity between two graphs based on their maximum common subgraph, which is NP-hard to compute [16], is also studied in the literature. Mcgregor [16] proposed a depth-first search method, while more advanced pruning techniques are later proposed in [3], [14]. Another strategy is first constructing a product graph of the two input graphs, and then computing the maximum clique of the product graph [12], [20]. These techniques cannot be applied to GED computation due to the inherently different problem definition.

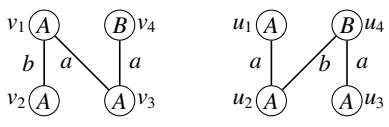
(4) Other Graph Comparison Methods. Comparing graphs based on their low-dimensional embeddings has also been studied. For example, the embedding of a graph can be computed from the eigenvalues of its associated matrices (*e.g.*, adjacency matrix or the Laplacian) [10], [25] or from personalized PageRank values [6]. However, these methods are mainly used for graph classification tasks instead of graph similarity search, and moreover, they cannot handle vertex/edge labels as we do in this paper. Recently, using graph neural networks to learn embeddings for graphs that approximately preserve their GED values is also studied [19]. However, these methods are heuristic in nature without any approximation guarantees, and moreover, edge labels are not handled in [19].

II. Preliminaries

In this paper, we focus on labeled and undirected simple graphs[2] $g = (V(g), E(g), l)$, where $V(g)$ is a vertex set, $E(g)$ is an edge set, and $l : V(g) \cup E(g) \to \Sigma$ is a labelling function that assigns each vertex and/or edge a label from the label set 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. Given a vertex subset $S \subseteq V(g)$, the subgraph of g induced by S is $g_S = (S, \{(u, u') \in E(g) \mid u, u' \in S\}, l)$. For presentation simplicity, we simply refer to a labeled and undirected graph as a graph.

The graph edit distance is defined based on *graph edit operations* that transform graphs. Specifically, there are six (graph) 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 $\mathsf{ged}(q, g)$, is the minimum number of edit operations that can transform q into g.*



(a) Graph q (b) Graph g

Fig. 2. Sample graphs

[2]Note that, all our techniques can straightforwardly handle directed graphs

TABLE I
Frequently used notations

Notation	Description
$\mathbb{1}_\phi$	The indicator function that equals 1 if the expression ϕ evaluates to `true` and 0 otherwise
$\sqcup$, $\sqcap$	Multi-set union and intersection
$\Upsilon(S_1, S_2)$	Edit distance between multi-sets S_1 and S_2, *i.e.*, $\Upsilon(S_1, S_2) = \max\{\lvert S_1\rvert, \lvert S_2\rvert\} - \lvert S_1 \sqcap S_2\rvert$
$\mathsf{ged}(q, g)$	The GED between graphs q and g
$\mathcal{T}$	The search tree of all mappings from $V(q)$ to $V(g)$
f, h	(Partial) mapping from $V(q)$ to $V(g)$
$f(v)$	The vertex of $V(g)$ to which $v \in V(q)$ maps
edc_f	The editorial cost of a full mapping f
mc_f	The mapping cost of a (partial) mapping f
lb_f	Lower bound of the editorial costs of all full mappings that extend f
q_f	The subgraph of q induced by vertices of f
$q_{\bar{f}}$	The remaining subgraph of q by removing q_f
$\mathcal{F}(q_{\bar{f}}, g_{\bar{f}})$	The set of all full mappings from $V(q_{\bar{f}})$ to $V(g_{\bar{f}})$
$f \oplus \sigma$	Concatenation of f and σ where $\sigma \in \mathcal{F}(q_{\bar{f}}, g_{\bar{f}})$
$l(v)$	Label of vertex u
$l(v, v')$	Label of edge (v, v')
$L_V(q_{\bar{f}})$	Multi-set of vertex labels of $q_{\bar{f}}$
$L_E(q_{\bar{f}})$	Multi-set of edge labels of $q_{\bar{f}}$
$L_{E_I}(q_{\bar{f}})$	Multi-set of labels of inner edges of $q_{\bar{f}}$
$L_E(v)$	Multi-set of labels of v's adjacent edges
$L_{E_C}(v)$	Multi-set of labels of v's cross adjacent edges
$L_{E_I}(v)$	Multi-set of labels of v's inner adjacent edges

Consider the two graphs q and g in Figure 2, where vertex labels are illustrated inside circles (*i.e.*, A, B) 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 edge (v_1, v_2) from b to a, (2) delete edge (v_1, v_3), and (3) insert edge (v_2, v_4) with label b. Thus, the GED between q and g is at most 3.

Problem Statement. Given two graphs, q and g, and a threshold τ, we study the problem of *GED verification* that outputs `true` if $\mathsf{ged}(q, g) \le \tau$ and `false` otherwise.

As demonstrated in the Introduction, GED verification is a critical and fundamental operation in graph similarity search. Our techniques can also be applied to the problem of *GED computation* that computes the exact value of $\mathsf{ged}(q, g)$.

As $\mathsf{ged}(q, g) = \mathsf{ged}(g, q)$ [24], we can assume that $|V(q)| \le |V(g)|$; otherwise, we can simply swap q and g. For presentation simplicity, ***we further assume that*** $|V(q)| = |V(g)|$ when discussing our techniques in the remainder of the paper, since we can add $|V(g)| - |V(q)|$ dummy vertices to q [7] if $|V(q)| < |V(g)|$. Note that, we have the case of $|V(q)| \neq |V(g)|$ in our experimental studies.

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

III. State-of-the-art Approach AStar-LSa

The state-of-the-art approach AStar-LSa [7] computes $\mathsf{ged}(q, g)$ by enumerating (vertex) mappings from $V(q)$ to $V(g)$. Each mapping $f : V(q) \to V(g)$ induces an **editorial**

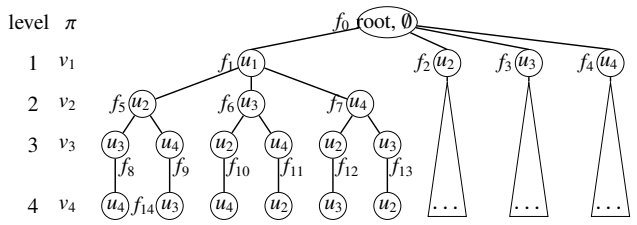


Fig. 3. Search tree $\mathcal{T}$

cost, denoted $\mathsf{edc}_f(q, g)$, which is the number of edit operations required to transform q into g by obeying the mapping (*i.e.*, $v \in V(q)$ maps to $f(v) \in V(g)$). The editorial cost of a mapping can be computed in time linear to the size of q and g [7]. For example, the editorial cost of the mapping $f = \{v_1 \mapsto u_1, v_2 \mapsto u_2, v_3 \mapsto u_3, v_4 \mapsto u_4\}$ for the graphs in Figure 2 is 3: change the label of edge (v_1, v_2) from b to a, delete edge (v_1, v_3), and insert edge (v_2, v_4) with label b.

The GED between q and g then equals the minimum editorial cost among all mappings from $V(q)$ to $V(g)$ [7]. For example, Figure 3 presents all the mappings from $V(q)$ to $V(g)$ in a prefix-shared manner for the matching order $\pi = (v_1, \ldots, v_{|V(q)|})$ of $V(q)$. This results in a **search tree** $\mathcal{T}$. Specifically, each node of $\mathcal{T}$ at level i 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 $V(g)$. For example, node f_5 represents the partial mapping $\{v_1 \mapsto u_1, v_2 \mapsto u_2\}$, and node f_8 extends its parent f_5 by additionally mapping v_3 to u_3.

However, there is an exponential (factorial to be exact) number of mappings in the search tree $\mathcal{T}$, as computing the exact GED is NP-hard [27]. For efficient GED verification/computation, AStar-LSa [7] conducts a pruned best-first search on the search tree $\mathcal{T}$, by exploiting lower bounds of partial mappings for prioritizing as well as for pruning.

Definition 3.1: *Each (partial) mapping f has a **mapping cost**, denoted $\mathsf{mc}_f(q, g)$ and abbreviated as mc_f, which equals $\mathsf{edc}_f(q_f, g_f)$; here, q_f and g_f, respectively, are the subgraphs of q and g induced by vertices in f.*

Definition 3.2: *The **lower bound cost** of a (partial) mapping f from $V(q)$ to $V(g)$, denoted $\mathsf{lb}_f(q, g)$ and abbreviated as lb_f, is a value that is at least the mapping cost mc_f of f and at most the minimum editorial cost among all full mappings that extend f.*

The framework of the AStar search paradigm, which is used by AStar-LSa, for GED verification is shown in Algorithm 1. A priority queue Q is used to store the search frontier which is initialized by the root of the search tree $\mathcal{T}$ (Line 2). Each entry of Q consists of a partial mapping f, its level i and its parent pa in $\mathcal{T}$, and its lower bound lb_f. The algorithm iteratively pops from Q the top entry $(f, i, pa, \mathsf{lb}_f)$ (*i.e.*, with the minimum lb_f) (Line 4), and extends it by computing the lower bound lb_h for every child h of f (Line 5). If there is a child h that is a full mapping (*i.e.*, $|h| = |V(q)|$) and has lower bound at most τ, then the algorithm returns `true` (Line 7).

Algorithm 1: [7] AStar(q, g, τ)

Output: `true` if $\mathsf{ged}(q, g) \le \tau$, and `false` otherwise

1 Compute a matching order $\pi = (v_1, \ldots, v_{|V(q)|})$ of $V(q)$;
2 $Q \leftarrow \{(\emptyset, 0, nil, 0)\}$; `/* Push the root of the search tree into the priority queue Q */`
3 **while** $Q \neq \emptyset$ **do**
4 $\quad (f, i, pa, \mathsf{lb}_f) \leftarrow$ pop the top entry from Q; `/* Lines 5-8 extend f by mapping` v_{i+1} `*/`
5 $\quad$ Compute the lower bound cost lb_h for each child h of f;
6 $\quad$ **for each** *child h of f s.t.* $\mathsf{lb}_h \le \tau$ **do**
7 $\quad\quad$ **if** $i + 1 = |V(q)|$ **then return** `true`;
8 $\quad\quad$ **else** Push $(h, i+1, f, \mathsf{lb}_h)$ into Q;
9 **return** `false`;

Otherwise, all such children with lower bounds at most τ are pushed into Q (Line 8). Note that for space consideration, each partial mapping f is not stored in its entirety in Q, but only stores the vertex $u \in V(g)$ to which v_i maps in f where $i = |f|$; the mapping of other vertices v_j for $j < i$ can be obtained from its parent pa, its parent's parent, and so on.

Lower Bound Estimation. To compute the lower bound of a mapping f, q is decomposed into two parts: q_f – the subgraph of q induced by vertices in f, and $q_{\bar{f}}$ – the remaining subgraph of q. Note that $q_{\bar{f}}$ contains none of the vertices of q_f but includes edges that have exactly one end-point in q_f. That is, $q_{\bar{f}}$ contains both **inner edges** whose both end-points are in $q_{\bar{f}}$, and **cross edges** between vertices of $q_{\bar{f}}$ and vertices of q_f. Similarly, g is decomposed into g_f and $g_{\bar{f}}$.

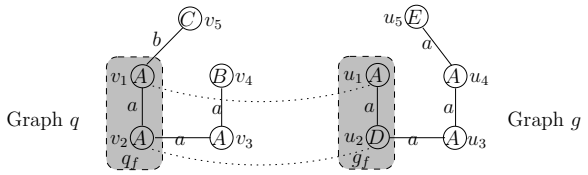


Fig. 4. Lower bound estimation

Example 3.1: *Consider the partial mapping $f = \{v_1 \mapsto u_1, v_2 \mapsto u_2\}$ for the graphs q and g in Figure 4. q_f and g_f are the parts in the shadowed rectangle, while $q_{\bar{f}}$ and $g_{\bar{f}}$ are the remaining parts; specifically, $q_{\bar{f}}$ consists of three vertices $\{v_3, v_4, v_5\}$, one inner edge $\{(v_3, v_4)\}$ and two cross edges $\{(v_5, v_1), (v_3, v_2)\}$.* □

Let $L_V(q_{\bar{f}})$ and $L_V(g_{\bar{f}})$ be the multi-sets of vertex labels of $q_{\bar{f}}$ and $g_{\bar{f}}$, respectively. Let $L_{E_I}(q_{\bar{f}})$ and $L_{E_I}(g_{\bar{f}})$ be the multi-sets of labels of inner edges of $q_{\bar{f}}$ and $g_{\bar{f}}$, respectively. Let $L_{E_C}(v)$ be the multi-set of labels of v's cross adjacent edges. AStar-LSa uses the anchor-aware label set-based lower bound $\mathsf{lb}^{\mathsf{LSa}}$, which exploits the information of the mapped vertices of q_f, called *anchored vertices*.

Definition 3.3: *[7] The **anchor-aware label set-based lower bound** of a mapping f is*

$$\mathsf{lb}_f^{\mathsf{LSa}} := \mathsf{mc}_f + \mathsf{LSa}_f(q_{\bar{f}}, g_{\bar{f}}) \quad (1)$$

$$\mathsf{LSa}_f(q_{\bar{f}}, g_{\bar{f}}) := \Upsilon(L_V(q_{\bar{f}}), L_V(g_{\bar{f}})) + \Upsilon(L_{E_I}(q_{\bar{f}}), L_{E_I}(g_{\bar{f}})) + \textstyle\sum_{v \in V(q_f)} \Upsilon(L_{E_C}(v), L_{E_C}(f(v))) \quad (2)$$

Here $\Upsilon(\cdot,\cdot)$ *denotes the edit distance between two multi-sets and* $\Upsilon(S_1, S_2) = \max\{|S_1|, |S_2|\} - |S_1 \sqcap S_2|$.

Example 3.2: *For the partial mapping* f *in Example 3.1, we have* $L_V(q_{\bar{f}}) = \{A, B, C\}$, $L_V(g_{\bar{f}}) = \{A, A, E\}$, $L_{E_I}(q_{\bar{f}}) = \{a\}$, $L_{E_I}(g_{\bar{f}}) = \{a, a\}$, $L_{E_C}(v_1) = \{b\}, L_{E_C}(v_2) = \{a\}$, $L_{E_C}(u_1) = \emptyset$, *and* $L_{E_C}(u_2) = \{a\}$. *Thus,* $\mathtt{LSa}_f(q_{\bar{f}}, g_{\bar{f}}) = 2 + 1 + 1 = 4$, *and* $\mathsf{lb}_f^{\mathtt{LSa}} = 5$. □

Let $\mathcal{T}_{\leq x}^{\mathtt{LSa}}$ be the set of non-leaf nodes/partial mappings in $\mathcal{T}$ whose lower bounds regarding $\mathsf{lb}^{\mathtt{LSa}}$ are no larger than x, and $|\mathcal{T}_{\leq x}^{\mathtt{LSa}}|$ be its cardinality. The time complexity of AStar-LSa is $O\left(\min\left\{|\mathcal{T}_{\leq \tau}^{\mathtt{LSa}}|, |\mathcal{T}_{\leq \mathsf{ged}(q,g)}^{\mathtt{LSa}}|\right\} \times (|E(q)| + |E(g)|)\right)$, the space complexity is $O\left(\min\left\{|\mathcal{T}_{\leq \tau}^{\mathtt{LSa}}|, |V(g)| \cdot |\mathcal{T}_{\leq \mathsf{ged}(q,g)}^{\mathtt{LSa}}|\right\}\right)$ [7].

Note that, Algorithm 1 can be modified to compute the exact $\mathsf{ged}(q, g)$, by setting $\tau = \infty$, removing Line 7, and terminating the algorithm if f popped at Line 4 is a full mapping, where the lower bound of this f then equals $\mathsf{ged}(q, g)$. The above time complexity and space complexity still hold.

IV. A Tighter Lower Bound Estimation

It is shown in [7] that AStar-LSa significantly outperforms the previous algorithms, and the efficiency of AStar-LSa mainly comes from two innovations. Firstly, the anchor-aware label set-based lower bound $\mathsf{lb}^{\mathtt{LSa}}$ used in AStar-LSa is tighter than the label set-based lower bound $\mathsf{lb}^{\mathtt{LS}}$ used in previous algorithms. Secondly, AStar-LSa proposes an efficient algorithm for computing the lower bound costs of all children of a partial mapping in totally linear time. Nevertheless, we observe in our experiments that the main memory consumption of AStar-LSa increases very fast when either the graph size or the threshold τ becomes larger. This limits AStar-LSa from scaling to larger graphs or larger threshold values.

As the main memory consumption of the AStar search paradigm (*i.e.*, Algorithm 1) is inversely related to the tightness of the lower bound estimation, in this section we propose an anchor-aware branch match-based lower bound $\mathsf{lb}^{\mathtt{BMa}}$ for computing a tighter lower bound than $\mathsf{lb}^{\mathtt{LSa}}$. We present the lower bound $\mathsf{lb}^{\mathtt{BMa}}$ in Section IV-A, discuss its computation in Section IV-B, and finally compare it with $\mathsf{lb}^{\mathtt{LSa}}$ in Section IV-C.

A. Anchor-Aware Branch Match-based Lower Bound $\mathsf{lb}^{\mathtt{BMa}}$

Before presenting the lower bound $\mathsf{lb}^{\mathtt{BMa}}$, we first describe a simpler but looser lower bound, called the branch match-based lower bound $\mathsf{lb}^{\mathtt{BM}}$, to illustrate the main ideas. The lower bound is based on the concept of branch. The *branch* structure 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. For example, for the graph q in Figure 4, $B(v_3) = (A, \{a, a\})$. Based on the branch structure $B(v)$ of $v \in q_{\bar{f}}$ and the branch structure $B(u)$ of $u \in g_{\bar{f}}$, the cost of mapping v to u is defined as

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

where $\mathbb{1}_\phi$ is an indicator function that equals 1 if the expression ϕ evaluates `true` and 0 otherwise. Note that, $\Upsilon(\cdot,\cdot)$ is multiplied by a coefficient $\frac{1}{2}$; this is because each edge $(v, v') \in E(q_{\bar{f}})$ is considered twice: once in $B(v)$ and once in $B(v')$. Then, the branch match-based lower bound of a mapping f, denoted $\mathsf{lb}_f^{\mathtt{BM}}$, is defined as mc_f plus the minimum cost of mapping the set of branch structures of $q_{\bar{f}}$ to the set of branch structures of $g_{\bar{f}}$, *i.e.*,

$$\mathsf{lb}_f^{\mathtt{BM}} := \mathsf{mc}_f + \mathtt{BM}_f(q_{\bar{f}}, g_{\bar{f}}) \quad (3)$$

$$\mathtt{BM}_f(q_{\bar{f}}, g_{\bar{f}}) := \min_{\sigma \in \mathcal{F}(q_{\bar{f}}, g_{\bar{f}})} \sum_{v \in V(q_{\bar{f}})} \lambda^{\mathtt{BM}}(v, \sigma(v)) \quad (4)$$

where $\mathcal{F}(q_{\bar{f}}, g_{\bar{f}})$ denotes the set of all full mappings from vertices of $q_{\bar{f}}$ to vertices of $g_{\bar{f}}$, and $\sigma(v)$ is the vertex of $V(g_{\bar{f}})$ to which v maps by σ.

Example 4.1: *For the partial mapping* f *in Example 3.1,* $B(v_3) = (A, \{a, a\})$ *and* $B(u_4) = (A, \{a, a\})$*; thus,* $\lambda^{\mathtt{BM}}(v_3, u_4) = 0$. *It can be verified that* $\mathtt{BM}_f(q_{\bar{f}}, g_{\bar{f}}) = 3$ *and* $\mathsf{lb}_f^{\mathtt{BM}} = 4$. □

Note that, the branch match technique has been used in [30] for computing a *global* lower bound of $\mathsf{ged}(q, g)$. Thus, the correctness of $\mathsf{lb}_f^{\mathtt{BM}}$ can be proved in a similar way to the proofs in [30]; we omit the details. However, it is worth pointing out that the branch match technique has not been utilized in the literature for computing lower bounds for *partial* mappings as we do in this paper, and thus has not been utilized for GED verification/computation. Moreover, the anchor-aware branch match-based lower bound that we will present next is new.

Anchor-Aware Branch Match-based Lower Bound $\mathsf{lb}^{\mathtt{BMa}}$. By comparing the lower bound computed in Example 4.1 with that in Example 3.2, we see that $\mathsf{lb}_f^{\mathtt{BM}} < \mathsf{lb}_f^{\mathtt{LSa}}$ for this f; note that, for lower bound estimation, the larger the better. Thus, $\mathsf{lb}^{\mathtt{BM}}$ is not tighter than $\mathsf{lb}^{\mathtt{LSa}}$; this is also demonstrated by our experiments in Section VI-B. The main reason is that $\mathsf{lb}^{\mathtt{BM}}$ completely ignored the information of the anchored vertices (*i.e.*, mapped vertices). Motivated by this, we propose an anchor-aware branch match-based lower bound $\mathsf{lb}^{\mathtt{BMa}}$ which improves $\mathsf{lb}^{\mathtt{BM}}$. We first revise the definition of branch structure by also encoding the information of anchored vertices.

Definition 4.1: *Our revised branch structure of a vertex* $v \in V(q_{\bar{f}})$ *regarding* f *is* $B'_f(v) = (l(v), L_{E_I}(v), \bigcup_{v' \in V(q_f)} \{(f(v'), l(v, v'))\})$*, where* $L_{E_I}(v)$ *denotes the multi-set of labels of* v*'s inner adjacent edges, and* $l(v, v') = \perp$ *if there is no edge between* v *and* v'.

That is, we explicitly encode each anchored vertex v' and its connection $l(v, v')$ to v in the revised branch structure. For example, for the graph q in Figure 4, $B'_f(v_3) = (A, \{a\}, \{(u_1, \perp), (u_2, a)\})$. The revised branch structures for vertices of $g_{\bar{f}}$ are defined similarly.

Given $B'_f(v)$ and $B'_f(u)$ for $v \in V(q_{\bar{f}})$ and $u \in V(g_{\bar{f}})$, the cost of mapping v to u regarding the anchored vertices in f is defined as the sum of the edit distances between the three corresponding components of $B'_f(v)$ and $B'_f(u)$, *i.e.*,

$$\lambda_f^{\mathtt{BMa}}(v, u) := \mathbb{1}_{l(v) \neq l(u)} + \tfrac{1}{2} \times \Upsilon(L_{E_I}(v), L_{E_I}(u)) + \sum_{v' \in V(q_f)} \mathbb{1}_{l(v,v') \neq l(u, f(v'))}$$

where the label of a non-existence edge is defined as $\perp$. Intuitively, $\lambda_f^{\mathtt{BMa}}(v, u)$ equals the minimum cost to edit $v \in V(q_{\bar{f}})$

and its adjacent edges to be the same as $u \in V(g_{\bar{f}})$ and u's adjacent edges, subject to the constraint that *the edge connecting v to an anchored vertex $v' \in V(q_f)$ must map to the edge $(u, f(v'))$ and vice versa.*

Definition 4.2: *The **anchor-aware branch match-based lower bound** of a partial mapping f is defined as*

$$\mathsf{lb}_f^{\mathtt{BMa}} := \mathsf{mc}_f + \mathtt{BMa}_f(q_{\bar{f}}, g_{\bar{f}}) \quad (5)$$

$$\mathtt{BMa}_f(q_{\bar{f}}, g_{\bar{f}}) := \min_{\sigma \in \mathcal{F}(q_{\bar{f}}, g_{\bar{f}})} \sum_{v \in V(q_{\bar{f}})} \lambda_f^{\mathtt{BMa}}(v, \sigma(v)) \quad (6)$$

Lemma 4.1: *For any (partial) mapping f from $V(q)$ to $V(g)$, $\mathsf{lb}_f^{\mathtt{BMa}}$ is a lower bound cost of f.*

Proof: It is easy to verify that for each $\sigma \in \mathcal{F}(q_{\bar{f}}, g_{\bar{f}})$, the concatenation of f and σ, denoted $f \oplus \sigma$, is a full mapping from $V(q)$ to $V(g)$. Thus, it suffices to prove that for every $\sigma \in \mathcal{F}(q_{\bar{f}}, g_{\bar{f}})$, the following inequality holds:

$$\mathsf{mc}_f + \sum_{v \in V(q_{\bar{f}})} \lambda_f^{\mathtt{BMa}}(v, \sigma(v)) \leq \mathsf{edc}_{f \oplus \sigma} \quad (7)$$

This is because $\mathsf{lb}_f^{\mathtt{BMa}}$ minimizes the left hand side of the inequality among all $\sigma \in \mathcal{F}(q_{\bar{f}}, g_{\bar{f}})$.

Let's consider a specific $\sigma \in \mathcal{F}(q_{\bar{f}}, g_{\bar{f}})$. Note that

$$\begin{aligned}\mathsf{edc}_{f\oplus\sigma} =& \mathsf{mc}_f + \sum_{v \in V(q_{\bar{f}})} \sum_{v' \in V(q_f)} \mathbb{1}_{l(v,v') \neq l(\sigma(v), f(v'))} \\ &+ \sum_{v \in V(q_{\bar{f}})} \mathbb{1}_{l(v) \neq l(\sigma(v))} \\ &+ \tfrac{1}{2} \sum_{v \in V(q_{\bar{f}})} \sum_{v' \in V(q_{\bar{f}}) \setminus \{v\}} \mathbb{1}_{l(v,v') \neq l(\sigma(v), \sigma(v'))}\end{aligned}$$

where the first term accounts for the editorial cost of f mapping $V(q_f)$ to $V(g_f)$, the second term accounts for the editorial cost for mapping the cross edges between $V(q_f)$ and $V(q_{\bar{f}})$ to the cross edges between $V(g_f)$ and $V(g_{\bar{f}})$, and the sum of the last two terms accounts for the editorial cost of σ mapping $V(q_{\bar{f}})$ to $V(g_{\bar{f}})$.

By plugging in the equation of $\lambda_f^{\mathtt{BMa}}(v, \sigma(v))$ into Inequality (7), the left hand side of Inequality (7) becomes

$$\begin{aligned}&\mathsf{mc}_f + \sum_{v \in V(q_{\bar{f}})} \sum_{v' \in V(q_f)} \mathbb{1}_{l(v,v') \neq l(\sigma(v), f(v'))} \\ &+ \sum_{v \in V(q_{\bar{f}})} \mathbb{1}_{l(v) \neq l(\sigma(v))} + \tfrac{1}{2} \sum_{v \in V(q_{\bar{f}})} \Upsilon(L_{E_I}(v), L_{E_I}(\sigma(v)))\end{aligned}$$

Thus, it suffices to prove that for every $v \in V(q_{\bar{f}})$, the following inequality holds

$$\Upsilon(L_{E_I}(v), L_{E_I}(\sigma(v))) \leq \sum_{v' \in V(q_{\bar{f}}) \setminus \{v\}} \mathbb{1}_{l(v,v') \neq l(\sigma(v), \sigma(v'))}.$$

As $L_{E_I}(v) = \{l(v, v') \mid v' \in V(q_{\bar{f}}) \setminus \{v\}, (v, v') \in E(q)\}$, and $L_{E_I}(\sigma(v)) = \{l(\sigma(v), \sigma(v')) \mid v' \in V(q_{\bar{f}}) \setminus \{v\}, (\sigma(v), \sigma(v')) \in E(g)\}$, the last inequality holds. Thus, the lemma follows. □

Example 4.2: *For the partial mapping f in Example 3.1, the revised branch structures are shown in the dotted rectangles in Figure 5, where non-existence edges (*i.e.*, with label $\perp$) are omitted. Note that, here, we use $\star$ to denote a free vertex which can map to any free vertex, and thus $L_{E_I}(v)$ is represented as $\{(\star, \alpha) \mid \alpha \in L_{E_I}(v)\}$. The cost between two vertices are illustrated, in the middle, on the solid edge connecting the vertices. In particular, $B'_f(v_3) = (A, \{a\}, \{(u_2, a)\})$ and $B'_f(u_4) = (A, \{a, a\}, \{\})$ after omitting non-existence edges; thus,*

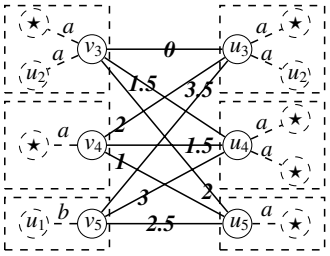


Fig. 5. Anchor-aware branch match-based lower bound

$\lambda_f^{\mathtt{BMa}}(v_3, u_4) = 1.5$. *It can be verified that* $\mathtt{BMa}_f(q_{\bar{f}}, g_{\bar{f}}) = 4$ *and* $\mathsf{lb}_f^{\mathtt{BMa}} = 5$. □

By comparing Example 4.2 with Example 4.1, we see that $\mathsf{lb}_f^{\mathtt{BMa}} > \mathsf{lb}_f^{\mathtt{BM}}$ for this f. More generally, it can be easily verified that $\lambda_f^{\mathtt{BMa}}(v, u) \geq \lambda^{\mathtt{BM}}(v, u)$ for every $v \in V(q_{\bar{f}})$ and $u \in V(g_{\bar{f}})$. Consequently, $\mathsf{lb}_f^{\mathtt{BMa}} \geq \mathsf{lb}_f^{\mathtt{BM}}$ holds for every mapping f. That is, $\mathsf{lb}^{\mathtt{BMa}}$ is tighter than $\mathsf{lb}^{\mathtt{BM}}$.

B. Computing Lower Bound Cost $\mathsf{lb}_f^{\mathtt{BMa}}$

To compute the lower bound $\mathsf{lb}_f^{\mathtt{BMa}}$, we need to find the mapping $\sigma \in \mathcal{F}(q_{\bar{f}}, g_{\bar{f}})$ that has the lowest cost (see Equation (6)). This is exactly the minimum cost perfect matching problem, and can be solved by the classic Hungarian algorithm [8].

Recall that, Algorithm 1 needs to compute the lower bound cost for all children of a partial mapping f (see Line 5). We conduct this for our lower bound $\mathsf{lb}^{\mathtt{BMa}}$ by computing the lower bound cost for each child h of f independently. The pseudocode is shown in Algorithm 2. We first obtain the mapping h from f by additionally mapping v_{i+1} to a vertex u' of $V(g_{\bar{f}})$, where $i = |f|$ (Line 2). Then, we construct an edge-weighted complete bipartite graph consisting of vertices of $q_{\bar{h}}$ on one side and vertices of $g_{\bar{h}}$ on the other side (Lines 3–5), where the cost of edge (v, u) in the bipartite graph is computed as $\lambda_h^{\mathtt{BMa}}(v, u)$. For example, Figure 5 shows the bipartite graph of the mapping $h = \{v_1 \mapsto u_1, v_2 \mapsto u_2\}$ for the graphs in Figure 4. Finally, we compute a minimum cost perfect matching σ^* in the edge-weighted bipartite graph by the Hungarian algorithm (Line 6), and calculate the lower bound cost as $\mathsf{lb}_h^{\mathtt{BMa}} = \mathsf{mc}_h + \sum_{v \in q_{\bar{h}}} \lambda_h^{\mathtt{BMa}}(v, \sigma^*(v))$ (Line 7).

Algorithm 2: Compute lower bound for all f's children regarding $\mathsf{lb}^{\mathtt{BMa}}$

Input: Graphs q and g, and a partial mapping f
Output: Lower bound $\mathsf{lb}_h^{\mathtt{BMa}}$ for every child h of f

1 **for each** vertex $u' \in V(g_{\bar{f}})$ **do**
2 $h \leftarrow f \oplus \{v_{i+1} \mapsto u'\}$;
3 **for each** vertex $v \in q_{\bar{h}}$ **do**
4 **for each** vertex $u \in g_{\bar{h}}$ **do**
5 $\lambda_h^{\mathtt{BMa}}(v, u) \leftarrow$ the cost of mapping v to u regarding h;
6 $\sigma^* \leftarrow$ the minimum cost perfect matching between $V(q_{\bar{h}})$ and $V(g_{\bar{h}})$ based on costs $\lambda_h^{\mathtt{BMa}}(v, u)$;
7 $\mathsf{lb}_h^{\mathtt{BMa}} \leftarrow \mathsf{mc}_h + \sum_{v \in q_{\bar{h}}} \lambda_h^{\mathtt{BMa}}(v, \sigma^*(v))$;

Lemma 4.2: *Algorithm 2 correctly computes the lower bound for all children of f in $O\left((|V(q)| + |V(g)|)^4\right)$ total time.*

Proof: The correctness directly follows from the above discussions. Regarding the time complexity, firstly Line 6 takes $O\left((|V(q)| + |V(g)|)^3\right)$ time [8]. Secondly, Lines 2–5 can also be conducted in $O\left((|V(q)| + |V(g)|)^3\right)$ time. Thirdly, Line 7 take $O(|E(q)| + |E(g)|)$ time. Finally, the time complexity of Algorithm 2 follows from the fact that Lines 2–7 are run for $|V(g_{\bar{f}})|$ times. □

C. Comparing $\mathsf{lb}^{\mathtt{BMa}}$ with $\mathsf{lb}^{\mathtt{LSa}}$

By replacing the lower bound lb in Algorithm 1 with $\mathsf{lb}^{\mathtt{BMa}}$, we obtain our algorithm AStar-BMa. Let $\mathcal{T}^{\mathtt{BMa}}_{\leq x}$ be the set of non-leaf nodes/partial mappings in $\mathcal{T}$ whose lower bounds regarding $\mathsf{lb}^{\mathtt{BMa}}$ are no larger than x. Then, the time and space complexities of AStar-BMa, as shown in the theorem below, directly follow that of AStar-LSa and Lemma 4.2.

Theorem 4.1: *The time complexity of* AStar-BMa *is* $O\left(\min\left\{|\mathcal{T}^{\mathtt{BMa}}_{\leq \tau}|, |\mathcal{T}^{\mathtt{BMa}}_{\leq \mathsf{ged}(q,g)}|\right\} \times (|V(q)| + |V(g)|)^4\right)$. *The space complexity is* $O\left(\min\left\{|\mathcal{T}^{\mathtt{BMa}}_{\leq \tau}|, |V(g)| \cdot |\mathcal{T}^{\mathtt{BMa}}_{\leq \mathsf{ged}(q,g)}|\right\}\right)$.

As we will prove shortly in Lemma 4.3 that $\mathsf{lb}^{\mathtt{BMa}}_f \geq \mathsf{lb}^{\mathtt{LSa}}_f$ holds for any mapping f, we have $\mathcal{T}^{\mathtt{BMa}}_{\leq x} \subseteq \mathcal{T}^{\mathtt{LSa}}_{\leq x}$ for any x. Thus, AStar-BMa has a smaller space complexity than AStar-LSa. However, regarding time complexity, there is no clear winner between the two algorithms, as AStar-BMa computes a tighter lower bound in a higher time complexity.

Lemma 4.3: *For any mapping f, we have $\mathsf{lb}^{\mathtt{BMa}}_f \geq \mathsf{lb}^{\mathtt{LSa}}_f$.*

Proof: By comparing Equation (6) with Equation (2), we see that it suffices to prove that for every mapping $\sigma \in \mathcal{F}(q_{\bar{f}}, g_{\bar{f}})$, the following holds:

$$\sum_{v \in V(q_{\bar{f}})} \lambda^{\mathtt{BMa}}_f(v, \sigma(v)) \geq \Upsilon(L_V(q_{\bar{f}}), L_V(g_{\bar{f}})) + \Upsilon(L_{E_I}(q_{\bar{f}}), L_{E_I}(g_{\bar{f}})) + \sum_{v' \in V(q_f)} \Upsilon(L_{E_C}(v'), L_{E_C}(f(v'))) \quad (8)$$

where $\mathtt{BMa}_f(q_{\bar{f}}, g_{\bar{f}})$ equals the minimum of the left hand side among all $\sigma \in \mathcal{F}(q_{\bar{f}}, g_{\bar{f}})$, and the right hand side of the inequality is $\mathtt{LSa}_f(q_{\bar{f}}, g_{\bar{f}})$. Recall that

$$\begin{aligned} &\sum_{v \in V(q_{\bar{f}})} \lambda^{\mathtt{BMa}}_f(v, \sigma(v)) \\ &= \sum_{v \in V(q_{\bar{f}})} \mathbb{1}_{l(v) \neq l(\sigma(v))} + \tfrac{1}{2} \sum_{v \in V(q_{\bar{f}})} \Upsilon(L_{E_I}(v), L_{E_I}(\sigma(v))) \\ &\quad + \sum_{v \in V(q_{\bar{f}})} \sum_{v' \in V(q_f)} \mathbb{1}_{l(v,v') \neq l(\sigma(v), f(v'))} \end{aligned} \quad (9)$$

We compare the three components of the right hand side of Inequality (8) with that of Equation (9) one-by-one. First, as $\bigsqcup_{v \in V(q_{\bar{f}})} \{l(v)\} = L_V(q_{\bar{f}})$ and $\bigsqcup_{v \in V(q_{\bar{f}})} \{l(\sigma(v))\} = L_V(g_{\bar{f}})$, thus

$$\sum_{v \in V(q_{\bar{f}})} \mathbb{1}_{l(v) \neq l(\sigma(v))} \geq \Upsilon(L_V(q_{\bar{f}}), L_V(g_{\bar{f}}))$$

Second, as $\bigsqcup_{v \in V(q_{\bar{f}})} L_{E_I}(v) = L_{E_I}(q_{\bar{f}}) \sqcup L_{E_I}(q_{\bar{f}})$ and $\bigsqcup_{v \in V(q_{\bar{f}})} L_{E_I}(\sigma(v)) = L_{E_I}(g_{\bar{f}}) \sqcup L_{E_I}(g_{\bar{f}})$, we have

$$\tfrac{1}{2} \sum_{v \in V(q_{\bar{f}})} \Upsilon(L_{E_I}(v), L_{E_I}(\sigma(v))) \geq \Upsilon(L_{E_I}(q_{\bar{f}}), L_{E_I}(g_{\bar{f}}))$$

Third, as $\sum_{v \in V(q_{\bar{f}})} \mathbb{1}_{l(v,v') \neq l(\sigma(v), f(v'))} \geq \Upsilon(L_{E_C}(v'), L_{E_C}(f(v')))$ holds for every vertex $v' \in V(q_f)$, we have

$$\begin{aligned} &\sum_{v \in V(q_{\bar{f}})} \sum_{v' \in V(q_f)} \mathbb{1}_{l(v,v') \neq l(\sigma(v), f(v'))} \\ &\geq \sum_{v' \in V(q_f)} \Upsilon(L_{E_C}(v'), L_{E_C}(f(v'))) \end{aligned}$$

Thus, the lemma holds. □

Another advantage of the lower bound $\mathsf{lb}^{\mathtt{BMa}}$ is that for a partial mapping f, if the vertices of $q_{\bar{f}}$ form an independent set (*i.e.*, have no inner edges), then $\mathsf{lb}^{\mathtt{BMa}}_f$ equals (rather than lower bounds) the minimum editorial cost among all full mappings that extend f, as proved in the lemma below. Thus, in such cases, we do not need to expand the partial mapping f to full mappings, and we can stop early.

Lemma 4.4: *For a partial mapping f, if the vertices of $q_{\bar{f}}$ form an independent set, then $\mathsf{lb}^{\mathtt{BMa}}_f$ equals the minimum editorial cost among all full mappings that extend f.*

The proof can be found in Appendix A.

V. Trading Tightness for Efficiency

We observe in our experimental studies (see Section VI) that AStar-BMa computes a much tighter lower bound and thus has a much smaller search space and main memory consumption than AStar-LSa. However, AStar-BMa may run slower than AStar-LSa. This is mainly due to the high time complexity of Algorithm 2 for computing the lower bounds $\mathsf{lb}^{\mathtt{BMa}}$.

In this section, we trade the tightness of $\mathsf{lb}^{\mathtt{BMa}}$ for a more efficient lower bound computation which will result in an improved overall performance. We present our optimized lower bound $\mathsf{lb}^{\mathtt{BMao}}$ in Section V-A, discuss the time and space complexity of AStar-BMao in Section V-B, and finally propose some optimizations in Section V-C.

A. Optimized Anchor-Aware Branch Match-based Lower Bound $\mathsf{lb}^{\mathtt{BMao}}$

For a more efficient lower bound computation, we treat the last mapped vertex of a partial mapping as a non-anchored vertex instead of an anchored vertex. Specifically, let $h = f \oplus \{v_{i+1} \mapsto u\}$ be a child of f, we treat v_{i+1} as a non-anchored vertex when computing the lower bound of h.

Definition 5.1: *The **optimized anchor-aware branch match-based lower bound** of a partial mapping $f \oplus \{v_{i+1} \mapsto u\}$ is defined as*

$$\mathsf{lb}^{\mathtt{BMao}}_{f \oplus \{v_{i+1} \mapsto u\}} := \mathsf{mc}_f + \mathtt{BMa}^{v_{i+1} \mapsto u}_f(q_{\bar{f}}, g_{\bar{f}}) \quad (10)$$

where $\mathtt{BMa}^{v_{i+1} \mapsto u}_f(q_{\bar{f}}, g_{\bar{f}})$ is similar to $\mathtt{BMa}_f(q_{\bar{f}}, g_{\bar{f}})$ except that v_{i+1} is restricted to map to u. More specifically,

$$\mathtt{BMa}^{v_{i+1} \mapsto u}_f(q_{\bar{f}}, g_{\bar{f}}) := \min_{\sigma \in \mathcal{F}(q_{\bar{f}}, g_{\bar{f}}): \sigma(v_{i+1}) = u} \sum_{v \in V(q_{\bar{f}})} \lambda^{\mathtt{BMa}}_f(v, \sigma(v))$$

Following from the proof of Lemma 4.1, it is easy to see that $\mathsf{lb}^{\mathtt{BMao}}_{f \oplus \{v_{i+1} \mapsto u\}}$ is a lower bound of $f \oplus \{v_{i+1} \mapsto u\}$. The main advantage of treating v_{i+1} as a non-anchored vertex in computing $\mathsf{lb}^{\mathtt{BMao}}_{f \oplus \{v_{i+1} \mapsto u\}}$ is that the branch structures of vertices of $q_{\bar{f}}$ and $g_{\bar{f}}$ as defined in Definition 4.1 are the same for different u. As a result, we can share computation between computing $\mathsf{lb}^{\mathtt{BMao}}_{f \oplus \{v_{i+1} \mapsto u\}}$ and computing $\mathsf{lb}^{\mathtt{BMao}}_{f \oplus \{v_{i+1} \mapsto u'\}}$.

A naive algorithm that uses the same strategy as Algorithm 2 (*i.e.*, compute the lower bound for each child h of f independently) would still have the same time complexity as that of Algorithm 2. To efficiently compute the lower bound cost for all children of f regarding $\mathsf{lb}^{\mathtt{BMao}}$, we use a different strategy as shown in Algorithm 3. We first construct the edge-weighted

Algorithm 3: Compute lower bound for all f's children regarding $\mathsf{lb}^{\mathtt{BMao}}$

Input: Graphs q and g, and a partial mapping f
Output: Lower bound $\mathsf{lb}_h^{\mathtt{BMao}}$ for every child h of f

1 **for each** vertex $v \in q_{\bar{f}}$ **do**
2 **for each** vertex $u \in g_{\bar{f}}$ **do**
3 Compute the cost $\lambda_f^{\mathtt{BMa}}(v,u)$ of mapping v to u regarding f;
4 **for each** $j \leftarrow 1$ to $|V(g_{\bar{f}})|$ **do**
5 $\sigma^* \leftarrow$ the minimum cost perfect matching between $V(q_{\bar{f}})$ and $V(g_{\bar{f}})$ based on costs $\lambda_f^{\mathtt{BMa}}(v,u)$;
6 $u \leftarrow$ the vertex to which v_{i+1} maps in σ^*;
7 $h \leftarrow f \oplus \{v_{i+1} \mapsto u)\}$;
8 $\mathsf{lb}_h^{\mathtt{BMao}} \leftarrow \mathsf{mc}_f + \sum_{v \in q_{\bar{f}}} \lambda_f^{\mathtt{BMa}}(v, \sigma^*(v))$;
9 $\lambda_f^{\mathtt{BMa}}(v_{i+1}, u) \leftarrow +\infty$;

complete bipartite graph between $V(q_{\bar{f}})$ and $V(g_{\bar{f}})$ (Lines 1–3); note that, v_{i+1} is included into the bipartite graph. Then, instead of first fixing h and then computing the lower bound cost of h, we first compute minimum cost perfect matching σ^* in the current bipartite graph (Line 5) and then assign the cost as a lower bound of $f \oplus \{v_{i+1} \mapsto \sigma^*(v_{i+1})\}$ (Lines 6–8). In order to be able to compute the lower bound cost for other children of f, we set the weight of the edge $(v_{i+1}, \sigma^*(v_{i+1}))$ in the bipartite graph as $+\infty$ (Line 9), *i.e.*, we remove this edge from the bipartite graph.

Lemma 5.1: *Algorithm 3 correctly computes the lower bound for all children of a partial mapping f regarding $\mathsf{lb}^{\mathtt{BMao}}$ in $O\left((|V(q)| + |V(g)|)^3\right)$ total time.*

Proof: Firstly, it is easy to see that each vertex of $g_{\bar{f}}$ will be selected at Line 6 exactly once. Thus, Line 7 enumerates all children of f. Secondly, we prove that Line 8 computes the lower bound for child h of f by contradiction. Suppose that $\mathsf{lb}_h^{\mathtt{BMao}}$ computed at Line 8 is not the lower bound of h as defined by Definition 5.1; that is, there is another mapping σ from $V(q_{\bar{f}})$ to $V(g_{\bar{f}})$ such that $\sigma(v_{i+1}) = u$ and $\sum_{v \in q_{\bar{f}}} \lambda_f^{\mathtt{BMa}}(v, \sigma(v)) < \sum_{v \in q_{\bar{f}}} \lambda_f^{\mathtt{BMa}}(v, \sigma^*(v))$. This contradicts that σ^* is a minimum cost perfect matching as computed at Line 5. Thus, Algorithm 3 correctly computes the lower bound for all children of f regarding $\mathsf{lb}^{\mathtt{BMao}}$.

For the time complexity, Lines 1–3 can be conducted in $O\left((|V(q)| + |V(g)|)^3\right)$ time. Lines 6–9 take $O(|E(q)| + |E(g)|)$ time. The most time-critical part is Line 5. Note that, the first invocation of Line 5 takes $O\left((|V(q)| + |V(g)|)^3\right)$ time [8], and then each of the subsequent invocations can be conducted in $O\left((|V(q)| + |V(g)|)^2\right)$ time [13]. Thus, Algorithm 3 runs in $O\left((|V(q)| + |V(g)|)^3\right)$ total time. □

The improved time complexity of $\mathsf{lb}^{\mathtt{BMao}}$ comes from the fact that the lower bound computations of different children of f are shared, such that each subsequent minimum cost perfect matching can be obtained in $O\left((|V(q)| + |V(g)|)^2\right)$ time. It is worth mentioning that the time complexity of lower bound computation regarding $\mathsf{lb}^{\mathtt{BMa}}$ cannot be improved in a similar way. This is because the branch structures for different children are different, and thus the edge weights in the bipartite graph constructed for one child h may be completely different from that for another child h'.

B. Time and Space Complexity of AStar-BMao

By replacing the lower bound lb in Algorithm 1 with $\mathsf{lb}^{\mathtt{BMao}}$, we have the algorithm AStar-BMao. Let $\mathcal{T}_{\leq x}^{\mathtt{BMao}}$ be the set of non-leaf nodes/partial mappings in $\mathcal{T}$ whose lower bounds regarding $\mathsf{lb}^{\mathtt{BMao}}$ are no larger than x. The time and space complexities of AStar-BMao, as shown in the theorem below, directly follow from that of AStar-LSa and Lemma 5.1.

Theorem 5.1: *The time complexity of* AStar-BMao *is* $O\left(\min\left\{|\mathcal{T}_{\leq \tau}^{\mathtt{BMao}}|, |\mathcal{T}_{\leq \mathsf{ged}(q,g)}^{\mathtt{BMao}}|\right\} \times (|V(q)| + |V(g)|)^3\right)$. *The space complexity is* $O\left(\min\left\{|\mathcal{T}_{\leq \tau}^{\mathtt{BMao}}|, |V(g)| \cdot |\mathcal{T}_{\leq \mathsf{ged}(q,g)}^{\mathtt{BMao}}|\right\}\right)$.

According to the definitions, we intuitively have $\mathsf{lb}_{f\oplus\{v_{i+1}\mapsto u\}}^{\mathtt{BMao}} \leq \mathsf{lb}_{f\oplus\{v_{i+1}\mapsto u\}}^{\mathtt{BMa}}$. Thus, AStar-BMao has a larger space complexity than AStar-BMa. Nevertheless, we prove in the lemma below that the gap is small.

Lemma 5.2: $|\mathcal{T}_{\leq x}^{\mathtt{BMao}}| \leq |V(g)| \times |\mathcal{T}_{\leq x}^{\mathtt{BMa}}|$ *holds for every x.*

Proof: Recall that $\mathsf{lb}_f^{\mathtt{BMa}} = \mathsf{mc}_f + \mathtt{BMa}_f(q_{\bar{f}}, g_{\bar{f}})$. By comparing the definitions of $\mathtt{BMa}_f(q_{\bar{f}}, g_{\bar{f}})$ and $\mathtt{BMa}_f^{v_{i+1}\mapsto u}(q_{\bar{f}}, g_{\bar{f}})$, we see that

$$\mathsf{lb}_f^{\mathtt{BMa}} = \min_{u \in g_{\bar{f}}} \mathsf{lb}_{f\oplus\{v_{i+1}\mapsto u\}}^{\mathtt{BMao}}.$$

Consequently, for every child h of f in the search tree $\mathcal{T}$, we have $\mathsf{lb}_h^{\mathtt{BMao}} \geq \mathsf{lb}_f^{\mathtt{BMa}}$; recall that for lower bound, the larger the better. As a result, for each partial mapping $h \in \mathcal{T}_{\leq x}^{\mathtt{BMao}}$, its parent must be in $\mathcal{T}_{\leq x}^{\mathtt{BMa}}$. Thus, the lemma holds. □

From Lemma 5.2, we also know that the time complexity of AStar-BMao is no larger than that of AStar-BMa.

C. Early Stopping and Maintaining an Upper Bound

In this subsection, we propose two optimization techniques for AStar-BMao. Firstly, it is easy to see that Algorithm 3 computes the lower bounds for all children of f in non-decreasing order regarding their lower bound costs. Thus, if $\mathsf{lb}_h^{\mathtt{BMao}}$ computed for the child h is larger than the input threshold τ, then we can skip the lower bound computation for the remaining children of f as they are all guaranteed to be larger than τ. We call this optimization *early stopping*.

Secondly, it is easy to see that $f \oplus \sigma^*$ is a full mapping from $V(q)$ to $V(g)$, where σ^* is either computed in Algorithm 2 or in Algorithm 3. Thus, we can obtain an upper bound ub of $\mathsf{ged}(q,g)$ based on the editorial cost of $f \oplus \sigma^*$. For the problem of GED verification, we can directly return `true` if this upper bound is no larger than τ. For the problem of GED computation, we can maintain ub as the smallest value among all the computed upper bounds; ub can be used for early stopping as described above and also for reducing main memory consumption, as we do not need to push into Q partial mappings whose lower bounds are no smaller than ub. Note that this optimization applies to all of our algorithms, and is incorporated into all of our algorithms by default.

VI. Experiments

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

- AStar-LSa: the state-of-the-art algorithm proposed in [7].
- AStar-BM: AStar (Algorithm 1) incorporated with the lower bound lb^{BM} described in Section IV.
- AStar-BMa: AStar (Algorithm 1) incorporated with the lower bound lb^{BMa} proposed in Section IV.
- AStar-BMao: AStar incorporated with the optimized lower bound lb^{BMao} proposed in Section V as well as all the optimizations in Section V-C.
- AStar-SMa: AStar incorporated with the lower bound lb^{SMa} that replaces the branch structure of lb^{BMa} with the star structure proposed in [27] (see Appendix for details).

All our algorithms are implemented in C++ based on the source code of AStar-LSa; specifically, we only replaced the lower bound estimation function of AStar-LSa.[3] We do not compare with other earlier algorithms such as CSI_GED [9] and Inves [11], as they have been shown to be outperformed by AStar-LSa in [7]. All algorithms run in main memory. All experiments, except the one about parallel graph similarity search in Section VI-A, are run in single-thread mode and conducted on a machine with an Intel Core i7-8700 3.2GHz CPU and 64GB main memory running Ubuntu.

TABLE II
Statistics of real graph datasets ($|\Sigma_V|$: number of distinct vertex labels, $|\Sigma_E|$: number of distinct edge labels)

Datasets	$\lvert\mathcal{D}\rvert$	$\lvert V\rvert$			$\lvert E\rvert$			$\lvert\Sigma_V\rvert$	$\lvert\Sigma_E\rvert$
		max	avg	std	max	avg	std		
AIDS	42,689	222	25.60	12.19	247	27.53	13.30	66	3
PubChem	23,903	88	48.33	9.34	92	50.82	9.87	10	3
Cancer	32,557	229	26.33	11.78	236	28.32	12.97	68	3
Linux	24,183	15	9.44	2.98	26	8.90	3.58	1	1

Datasets. Same as the existing works, we use both real graph datasets and synthetic graph datasets to evaluate the algorithms. We use two widely-used real graph datasets [9], [11], [29], AIDS and PubChem. AIDS is an antivirus screen chemical compound dataset[4], and PubChem is a chemical compound dataset[5]. In addition, we also use another two real datasets, Cancer and Linux. Cancer is a human tumor cell line screen dataset[6], and Linux is a program dependence graph dataset generated from the Linux kernel procedure; the Linux dataset is obtained from [19]. Statistics of the four real datasets are shown in Table II, where $|\mathcal{D}|$ is the number of graphs. max /avg/std $|V|$ are respectively the maximum, average and standard deviation for the graphs' vertex numbers. max /avg/std $|E|$ are respectively the maximum, average and standard deviation for the graphs' edge numbers. $|\Sigma_V|$ is the number of *distinct* vertex labels, and $|\Sigma_E|$ is the number of *distinct* edge labels.

We also generate synthetic random graphs G_R by the graph generator GraphGen[7]. Specifically, we generate five groups of random graphs G_R, where the number of vertices for each graph is chosen from {64, 128, 256, 512, 1024}. 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 {2, 5, 10, 20, 40}. Each graph generated by GraphGen has an edge density $\frac{2|E|}{|V|\times(|V|-1)}$ of 20%, and has five distinct vertex labels and two distinct edge labels, similar to that used in [7], [9].

Evaluation Metrics. For each testing, we record the processing time, search space, and main memory usage. The search space is defined as the number of invocations of Line 5 of Algorithm 1, *i.e.*, the number of lower bound computations for all children of a partial mapping. The reported memory usage is "the maximum resident set size of the process during its lifetime", as measured by the command /usr/bin/time[8].

A. Results for Graph Similarity Search

We first evaluate AStar-BMao against the state-of-the-art algorithm AStar-LSa for index-free graph similarity search. For each of the four real graph datasets, we randomly select 100 graphs from the corresponding datasets as query graphs. The number of vertices in the query graphs ranges from 10 to 63 for AIDS, from 27 to 80 for PubChem, from 10 to 101 for Cancer, and from 4 to 15 for Linux.

Same as existing works [7], [11], [29], we first apply label filter, denoted LabelF, to filter out unpromising data graphs. That is, given a query graph q, we compute the label set-based lower bound between q and each data graph g in the database; if the lower bound between q and g is larger than τ, then g definitely is not similar to q and is pruned. The remaining candidates are verified by either AStar-BMao or AStar-LSa. Note that, the time of LabelF is included in our reported time.

The aggregated results for 100 queries are shown in Figure 6. When τ increases, the search space and memory consumption of AStar-LSa increase dramatically, and much faster than AStar-BMao. In particular, the peak memory consumption of AStar-LSa on PubChem for $\tau = 11$ is 46.5GB, while that of AStar-BMao is only 288MB. This is the result of the much smaller search space of AStar-BMao compared with AStar-LSa. This also indicates that the lower bound lb^{BMao} is tighter than lb^{LSa}, as the search space is inversely related to the tightness of the lower bound.

Regarding the running time, AStar-LSa runs faster than AStar-BMao for small τ. This is because, for small τ, the search spaces between AStar-LSa and AStar-BMao do not differ significantly, while the lower bound computation of lb^{LSa} is much faster than lb^{BMao}. Nevertheless, for $\tau \geq 5$ on AIDS, $\tau \geq 7$ on PubChem, $\tau \geq 5$ on Cancer, and all values of τ on Linux, AStar-BMao runs faster than AStar-LSa. Note that,

[3]The source code will be released along the publication of the paper.
[4]https://cactus.nci.nih.gov/download/nci/AID2DA99.sdz
[5]http://pubchem.ncbi.nlm.nih.gov: Compound_000975001_001000000.sdf
[6]https://cactus.nci.nih.gov/download/nci/CAN2DA99.sdz
[7]http://www.cse.cuhk.edu.hk/~jcheng/graphgen1.0.zip
[8]http://man7.org/linux/man-pages/man1/time.1.html

τ	Results	AStar-LSa			AStar-BMao		
		Time(s)	Mem	SS	Time(s)	Mem	SS
1	135	0.10	36M	8.9×10^3	0.13	36M	4.3×10^3
3	213	0.65	36M	2.1×10^5	0.78	36M	4.3×10^4
5	480	7.5	36M	4.7×10^6	5.4	36M	4.3×10^5
7	1,852	109	47M	8.3×10^7	49	37M	5.0×10^6
9	9,220	1,621	252M	1.1×10^9	563	47M	5.8×10^7
11	38,425	20,891	4.3G	1.4×10^{10}	5,756	299M	5.5×10^8

(a) AIDS

τ	Results	AStar-LSa			AStar-BMao		
		Time(s)	Mem	SS	Time(s)	Mem	SS
1	183	0.18	35M	6.6×10^4	0.36	35M	2.5×10^4
3	243	0.65	41M	2.1×10^5	2.02	36M	1.0×10^5
5	358	9.6	52M	4.7×10^6	14.4	36M	5.1×10^5
7	529	205	1.7G	1.0×10^8	158	37M	5.6×10^6
9	931	2,807	9.7G	1.3×10^9	1,662	50M	5.9×10^7
11	1,707	43,492	46.5G	2.0×10^{10}	18,099	288M	6.4×10^8

(b) PubChem

τ	Results	AStar-LSa			AStar-BMao		
		Time(s)	Mem	SS	Time(s)	Mem	SS
1	122	0.092	29M	7.9×10^3	0.119	29M	4.0×10^3
3	212	0.641	29M	2.1×10^5	0.848	29M	4.0×10^4
5	406	7.6	34M	4.5×10^6	7.0	30M	4.2×10^5
7	1,187	102	77M	7.4×10^7	52	35M	4.5×10^6
9	4,952	1,483	80M	1.1×10^9	514	35M	4.8×10^7
11	20,811	19,249	1.7G	1.3×10^{10}	5,154	65M	4.7×10^8

(c) Cancer

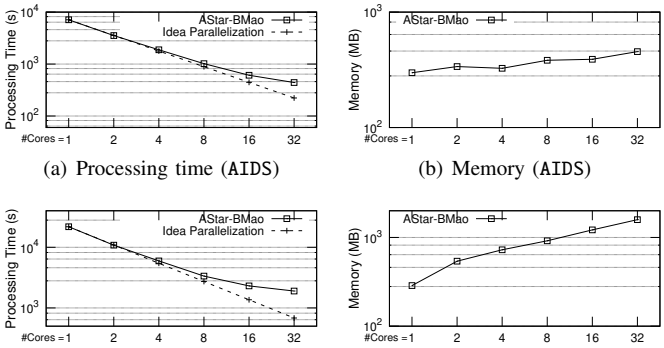
τ	Results	AStar-LSa			AStar-BMao		
		Time(s)	Mem	SS	Time(s)	Mem	SS
1	107,107	273	17M	4.4×10^8	5.8	12M	2.1×10^6
3	404,564	5,897	152M	8.3×10^9	96	12M	3.6×10^7
5	799,806	41,575	1.3G	5.0×10^{10}	1,048	19M	3.8×10^8
7	1,176,423	156,706	7.8G	1.7×10^{11}	5,347	53M	1.8×10^9

(d) Linux

Fig. 6. Aggregated results for graph similarity search with 100 queries (SS: Search Space)

these τ values are not too large for typical queries, as the aggregated number of results for 100 queries on AIDS is 480 for $\tau = 5$, on PubChem is 529 for $\tau = 7$, and on Cancer is 406 for $\tau = 5$ (see the second columns of the tables in Figure 6). Moreover, as the running time increases along with τ, it is more meaningful to reduce the running time for large τ, *e.g.*, AStar-BMao reduces the aggregated running time on PubChem for $\tau = 11$ from 12 hours to 5 hours. By comparing the different tables, we see that Cancer has similar query performance as AIDS, and Linux is the hardest to process due to lack of vertex and edge labels. Thus, we didn't run Linux for $\tau = 9$ and 11, and we exclude Cancer and Linux in the remaining testings.

Parallel Graph Similarity Search. We can reduce the running time of index-free algorithms, *e.g.*, AStar-BMao, for graph similarity search by utilizing multiple CPU cores. It is straightforward to parallelize these algorithms, *i.e.*, we can verify ged(q, g) simultaneously for multiple data graphs g. Specifically, we use openMP to parallelize the "for loop" for iterating though all data graphs in a database. The preliminary results by varying the number of CPU cores from 1 to 32 are shown in Figure 7; this set of experiments is conducted on a different machine with an Intel(R) Xeon(R) Platinum 8160 CPU@2.10GHz and 48 physical CPU cores. In Figure 7, we



(a) Processing time (AIDS) (b) Memory (AIDS)

(c) Processing time (PubChem) (d) Memory (PubChem)

Fig. 7. Parallel graph similarity search by using multiple CPU cores

also show the processing time that would be achieved by an idea parallelization, *i.e.*, the processing time when running on a single core divided by the number of cores. AStar-BMao achieves almost linear speedup. The memory consumption increases as there are multiple concurrently running copies of GED verification; nevertheless, this is affordable as the memory footprint of AStar-BMao is small. Note that, here we only exploit the intra-query parallelization between GED verifications. By exploiting inter-query parallelization and the parallelization within a single GED verification, it is anticipated that the running time can be further improved; we leave a detailed exploration of parallelization to our future work.

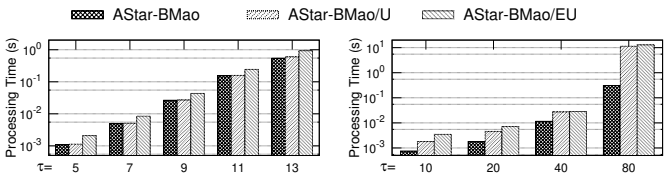
B. Results for GED Verification

To conduct a more detailed analysis of our algorithms, we generate graph pairs for GED verification as follows. 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. The parameters of the resulting groups are shown in Table III, where $|V|$ is the (approximate) size of graphs in the group, and ged is the GED for the graph pairs in the group. Finally, 20 graph pairs are randomly sampled from each group. For GED verification, we union all the obtained groups for each specific $|V|$ (*i.e.*, union the groups corresponding to different ged values), and *report the average processing time for each query* τ. Default values of $|V|$ are in boldface in Table III.

TABLE III
PARAMETER FOR GROUPS OF GRAPH PAIRS

Datasets	$\|V\|$	ged
AIDS	{20, **30**, 40, 50, 60}	{5, 6, 7, 8, . . . , 13, 14}
PubChem	{20, **30**, 40, 50, 60}	{5, 6, 7, 8, . . . , 13, 14}
G_R	{**64**, 128, 256, 512, 1024}	{10, 20, 40, 80}

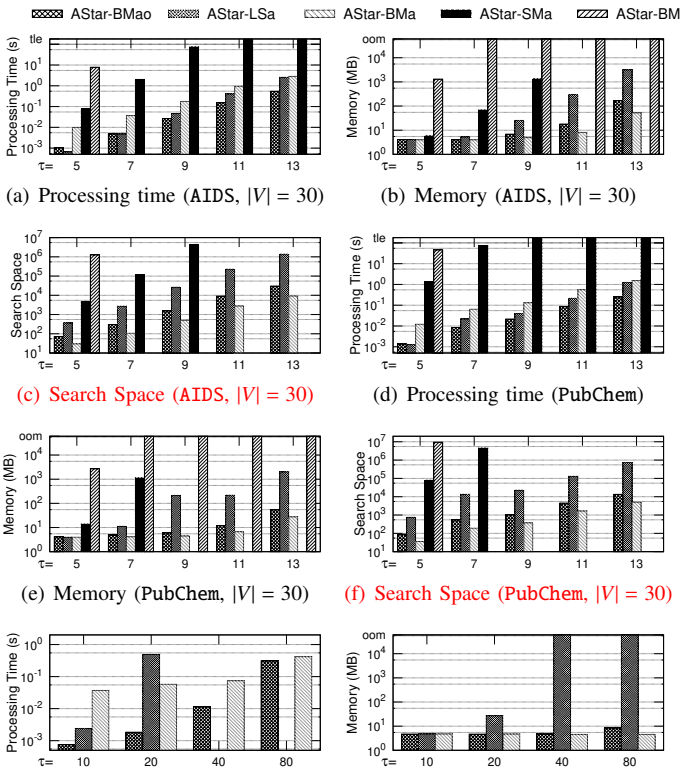
Evaluate Our Optimization Techniques. In this testing, we evaluate the effect of our two optimization techniques proposed in Section V-C: early stopping and maintaining an upper bound. We additionally implemented AStar-BMao/U which is AStar-BMao without upper bounding, and AStar-BMao/EU which is AStar-BMao without early stopping and without upper bounding. The results on AIDS and G_R are shown in Figure 8,



(a) AIDS ($|V| = 30$) (b) G_R ($|V| = 64$)

Fig. 8. Evaluate our optimization techniques (vary τ)

while the result on PubChem is similar and is omitted due to limit of space. We can see that both early stopping and upper bounding improve the efficiency. Specifically, AStar-BMao/U outperforms AStar-BMao/EU on AIDS due to incorporating the early stopping optimization that stops Algorithm 3 once the lower bound computed at Line 8 is larger than τ. AStar-BMao outperforms AStar-BMao/U on G_R due to the upper bounding optimization that immediately returns true once an upper bound, computed as the editorial cost of a heuristic mapping from $V(q)$ to $V(g)$, is no larger than τ. Thus, we adopt both optimizations in the following experiments; note that all our algorithms adopt the upper bounding optimization.



(a) Processing time (AIDS, $|V| = 30$) (b) Memory (AIDS, $|V| = 30$)

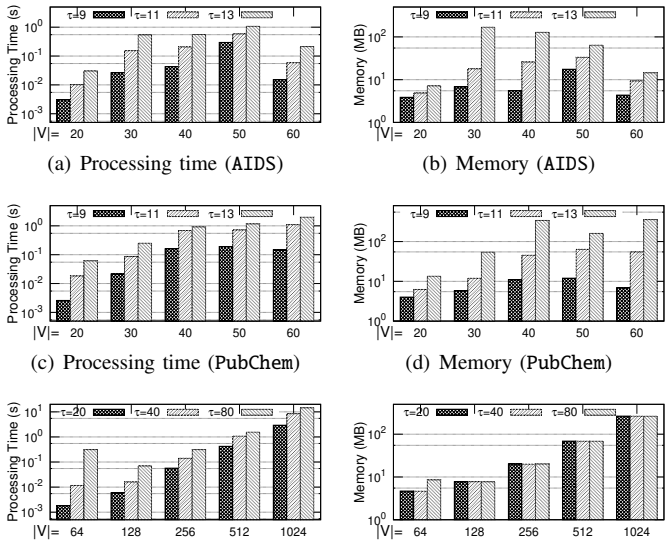
(c) Search Space (AIDS, $|V| = 30$) (d) Processing time (PubChem)

(e) Memory (PubChem, $|V| = 30$) (f) Search Space (PubChem, $|V| = 30$)

(g) Processing time (G_R, $|V| = 64$) (h) Memory (G_R, $|V| = 64$)

Fig. 9. Evaluate different lower bounds (vary τ)

Evaluate Different Lower Bounds. In this testing, we run AStar-BMao against AStar-BMa, AStar-BM, AStar-SMa, and AStar-LSa to evaluate the effect of different lower bounds. Note that, all these algorithms are implemented based on the same codebase, and they differ only in lower bound estimation. The results by varying τ are shown in Figure 9. We can see that AStar-BM has the largest memory footprint and search space and easily runs out-of-memory, and AStar-SMa has the second largest memory footprint and search space and cannot finish within 10 hours for $\tau \geq 9$ (denoted as tle in the corresponding plots), due to the loose lower bounds lb^{BM} and lb^{SMa}; thus, we do not run these two algorithms on G_R. AStar-BMa has the smallest memory footprint and search space due to computing the tightest lower bound among them. Nevertheless, AStar-BMa runs slower than AStar-LSa on AIDS and PubChem, this is because AStar-BMa computes the lower bounds much slower than AStar-LSa. Overall, AStar-BMao slightly increases the memory consumption and search space compared with AStar-BMa, but runs faster than both AStar-BMa and AStar-LSa. AStar-LSa runs out-of-memory on G_R for $\tau \geq 40$. We also observe that search space and memory consumption correlate well to each other.



(a) Processing time (AIDS) (b) Memory (AIDS)

(c) Processing time (PubChem) (d) Memory (PubChem)

(e) Processing time (G_R) (f) Memory (G_R)

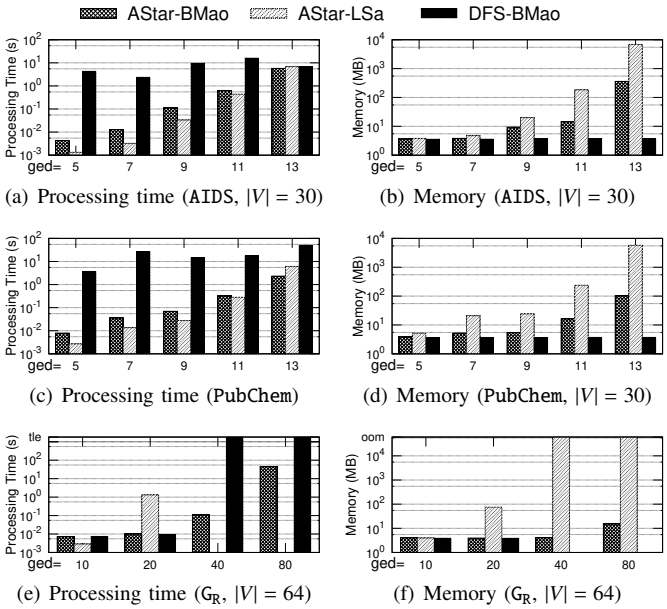
Fig. 10. Scalability testing of AStar-BMao (vary $|V|$)

Scalability Testing of AStar-BMao. The results of scalability testing of AStar-BMao by varying $|V|$ and for different τ values are shown in Figure 10. Same as the above testings, for each $|V|$, we union all the groups corresponding to that particular $|V|$, and report the average processing time for each query τ. We can see that AStar-BMao scales well in terms of both processing time and main memory consumption, for large graph sizes and for large threshold values.

C. Results for GED Computation

In this subsection, we compare AStar-BMao against AStar-LSa and DFS-BMao for exact GED computation. DFS-BMao is a variant of AStar-BMao that traverses the search tree $\mathcal{T}$ in a depth-first manner, and is implemented in the same way as the DFS-LSa algorithm in [7]; more detailed discussion about these two search paradigms can be found in [7]. We compute GED for graph pairs in the five groups of AIDS and PubChem with $|V| = 30$ corresponding to ged $= 5, 7, 9, 11, 13$, and in the four groups of G_R with $|V| = 64$ corresponding to ged $= 10, 20, 40, 80$, as obtained in Section VI-B. Each group

contains 20 graph pairs, and we report the average processing time. Note that, although all graph pairs in the same group share a GED, the algorithms are unaware of the GED values.



(a) Processing time (AIDS, $|V| = 30$) (b) Memory (AIDS, $|V| = 30$)

(c) Processing time (PubChem) (d) Memory (PubChem, $|V| = 30$)

(e) Processing time (G_R, $|V| = 64$) (f) Memory (G_R, $|V| = 64$)

Fig. 11. GED computation by varying groups

The results are shown in Figure 11, where we report both the processing time and memory usage. We see that AStar-BMao runs slower than AStar-LSa for small ged but runs faster than AStar-LSa for large ged. More importantly, the memory usage of AStar-LSa increases very fast and much faster than AStar-BMao, which results in AStar-LSa running out-of-memory on G_R for ged ≥ 40. Thus, AStar-BMao scales much better than AStar-LSa. On the other hand, DFS-BMao consistently runs slower than AStar-BMao, due to the large search space of the depth-first search paradigm [7]; note that DFS-BMao does not finish within 10 hours for the 20 graph pairs in the groups of G_R for $\tau \geq 40$.

VII. Conclusion

In this paper, we proposed a tighter lower bound estimation for GED verification and computation, as well as efficient algorithms for computing the lower bounds. Extensive performance studies demonstrated the effectiveness and efficiency of our techniques. In particular, our AStar-BMao algorithm outperforms the state-of-the-art algorithm AStar-LSa in terms of both processing time and main memory consumption, for both graph similarity search and GED verification/computation. One possible direction of future work is to design better lower bound estimation techniques.

References

[1] full version: https://lijunchang.github.io/pdf/ged-icde21-tr.pdf.

[2] 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.

[3] 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.

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

[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] Felix Borutta, Julian Busch, Evgeniy Faerman, Adina Klink, and Matthias Schubert. Structural graph representations based on multiscale local network topologies. In *Proc. of WI'19*, pages 91–98, 2019.

[7] Lijun Chang, Xing Feng, Xuemin Lin, Lu Qin, Wenjie Zhang, and Dian Ouyang. Speeding up ged verification for graph similarity search. In *Proc. of ICDE'20*, 2020.

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

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

[10] Shengmin Jin and Reza Zafarani. The spectral zoo of networks: Embedding and visualizing networks with spectral moments. In *Proc. of KDD'20*, pages 1426–1434, 2020.

[11] Jongik Kim, Dong-Hoon Choi, and Chen Li. Inves: Incremental partitioning-based verification for graph similarity search. In *Proc. of EDBT'19*, 2019.

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

[13] G. Ayorkor Korsah, Anthony (Tony) Stentz, and M Bernardine Dias. The dynamic hungarian algorithm for the assignment problem with changing costs. Technical Report CMU-RI-TR-07-27, Carnegie Mellon University, Pittsburgh, PA, July 2007.

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

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

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

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

[18] 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.

[19] Zongyue Qin, Yunsheng Bai, and Yizhou Sun. Ghashing: Semantic graph hashing for approximate similarity search in graph databases. In *Proc. of KDD'20*, pages 2062–2072, 2020.

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

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

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

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

[24] 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.

[25] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alexander M. Bronstein, and Emmanuel Müller. Netlsd: Hearing the shape of a graph. In *Proc. of KDD'18*, pages 2347–2356, 2018.

[26] 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.

[27] 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.

[28] Xiang Zhao, Chuan Xiao, Xuemin Lin, Wei Wang, and Yoshiharu Ishikawa. Efficient processing of graph similarity queries with edit distance constraints. *VLDB J.*, 22(6):727–752, 2013.

[29] Xiang Zhao, Chuan Xiao, Xuemin Lin, Wenjie Zhang, and Yang Wang. Efficient structure similarity searches: a partition-based approach. *VLDB J.*, 27(1), 2018.

[30] 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.

A. Proof of Lemma 4.4

We prove the lemma by showing that, if the vertices of $q_{\bar{f}}$ form an independent set, then for every mapping $\sigma \in \mathcal{F}(q_{\bar{f}}, g_{\bar{f}})$, we have $\mathsf{edc}_{f\oplus\sigma}(q,g) = \mathsf{mc}_f + \sum_{v\in V(q_{\bar{f}})} \lambda_f^{\mathtt{BMa}}(v,\sigma(v))$. Firstly, let's assume that the vertices of $g_{\bar{f}}$ also form an independent set. It is easy to verify that $\mathsf{edc}_{f\oplus\sigma}(q,g) = \mathsf{mc}_f + \sum_{v\in V(q_{\bar{f}})} \lambda_f^{\mathtt{BMa}}(v,\sigma(v))$. Secondly, if the vertices of $g_{\bar{f}}$ do not form an independent set, then $\mathsf{edc}_{f\oplus\sigma}(q,g)$ equals the number of inner edges of $g_{\bar{f}}$ plus $\mathsf{edc}_{f\oplus\sigma}(q,g')$ where g' is the resulting graph of g by removing all inner edges of $g_{\bar{f}}$; that is, $\mathsf{edc}_{f\oplus\sigma}(q,g) = \mathsf{mc}_f + \sum_{v\in V(q_{\bar{f}})} \lambda_f^{\mathtt{BMa}}(v,\sigma(v))$, by noting that $L_{E_I}(q_{\bar{f}}) = \emptyset$ and $\bigsqcup_{u\in V(g_{\bar{f}})} L_{E_I}(u) = L_{E_I}(g_{\bar{f}}) \sqcup L_{E_I}(g_{\bar{f}})$. Thus the lemma holds.

B. Anchor-aware Star Match-based Lower Bound

The star structure has been used in the literature for computing the lower bound of GED between two graphs without edge labels [27]. We extend it to handle edge labels as follows.

Definition B.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 neighbors.*

Based on the star structures $S(v)$ and $S(u)$, we define the cost of mapping $v \in q_{\bar{f}}$ to $u \in g_{\bar{f}}$ as,

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

Thus, $\lambda^{\mathtt{SM}}(v,u) = \lambda^{\mathtt{BM}}(v,u) + \Upsilon(L_V(v), L_V(u))$. The star match-based lower bound [27] is,

$$\mathtt{SM}_f(q_{\bar{f}}, g_{\bar{f}}) := \frac{\min_{\sigma\in\mathcal{F}(q_{\bar{f}},g_{\bar{f}})} \sum_{v\in V(q_{\bar{f}})} \lambda^{\mathtt{SM}}(v,\sigma(v))}{\max\{4,\Delta(q_{\bar{f}})+1,\Delta(g_{\bar{f}})+1\}}$$

where $\Delta(q_{\bar{f}})$ and $\Delta(g_{\bar{f}})$ denote the maximum vertex degree in $q_{\bar{f}}$ and $g_{\bar{f}}$, respectively.

Anchor-aware Star Match-based Lower Bound. Similarly, by exploiting the information of anchored vertices, we revise the star structure to define the cost of mapping $v \in q_{\bar{f}}$ to $u \in g_{\bar{f}}$ as,

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

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

$$\mathsf{lb}_f^{\mathtt{SMa}} := \mathsf{mc}_f + \mathtt{SMa}_f(q_{\bar{f}}, g_{\bar{f}})$$

$$\mathtt{SMa}_f(q_{\bar{f}}, g_{\bar{f}}) := \frac{\min_{\sigma\in\mathcal{F}(q_{\bar{f}},g_{\bar{f}})} \sum_{v\in V(q_{\bar{f}})} \lambda_f^{\mathtt{SMa}}(v,\sigma(v))}{\max\{4,\Delta(q_{\bar{f}})+1,\Delta(g_{\bar{f}})+1\}}$$

It can be easily verified that $\lambda_f^{\mathtt{SMa}}(v,u) \geq \lambda^{\mathtt{SM}}(v,u)$, and thus we have $\mathtt{SMa}_f(q_{\bar{f}}, g_{\bar{f}}) \geq \mathtt{SM}_f(q_{\bar{f}}, g_{\bar{f}})$.

Lemma B.1: *For a partial mapping f, we have $\mathsf{lb}_f^{\mathtt{BMa}} \geq \mathsf{lb}_f^{\mathtt{SMa}}$ if $\mathtt{BMa}_f(q_{\bar{f}}, g_{\bar{f}}) \geq |V(q_{\bar{f}})|$.*

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

$$\arg\min_{\sigma'\in\mathcal{F}(q_{\bar{f}},g_{\bar{f}})} \textstyle\sum_{v\in q_{\bar{f}}} \lambda_f^{\mathtt{BMa}}(v,\sigma'(v)).$$

Then, we have

$$\begin{aligned}& d\times(\mathtt{BMa}_f(q_{\bar{f}}, g_{\bar{f}}) - \mathtt{SMa}_f(q_{\bar{f}}, g_{\bar{f}})) \\ =& (d\times\min_{\sigma'\in\mathcal{F}(q_{\bar{f}},g_{\bar{f}})} \textstyle\sum_{v\in q_{\bar{f}}} \lambda_f^{\mathtt{BMa}}(v,\sigma'(v))) \\ & - \min_{\sigma''\in\mathcal{F}(q_{\bar{f}},g_{\bar{f}})} \textstyle\sum_{v\in q_{\bar{f}}} \lambda_f^{\mathtt{SMa}}(v,\sigma''(v)) \\ \geq& (d\times\textstyle\sum_{v\in q_{\bar{f}}} \lambda_f^{\mathtt{BMa}}(v,\sigma(v))) - \sum_{v\in q_{\bar{f}}} \lambda_f^{\mathtt{SMa}}(v,\sigma(v)) \\ =& \textstyle\sum_{v\in q_{\bar{f}}} (d\times\lambda_f^{\mathtt{BMa}}(v,\sigma(v)) - \lambda_f^{\mathtt{SMa}}(v,\sigma(v)))\end{aligned}$$

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

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

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

$$\begin{aligned}& d\times(\mathtt{BMa}_f(q_{\bar{f}}, g_{\bar{f}}) - \mathtt{SMa}_f(q_{\bar{f}}, g_{\bar{f}})) \\ \geq& \textstyle\sum_{v\in q_{\bar{f}}} (d\times\lambda_f^{\mathtt{BMa}}(v,\sigma(v)) - \lambda_f^{\mathtt{SMa}}(v,\sigma(v))) \\ \geq& (d-1)\times\textstyle\sum_{v\in q_{\bar{f}}} (\lambda_f^{\mathtt{BMa}}(v,\sigma(v)) - 1) \\ =& (d-1)\times(\mathtt{BMa}_f(q_{\bar{f}}, g_{\bar{f}}) - |V(q_{\bar{f}})|)\end{aligned}$$

Therefore, if $\mathtt{BMa}_f(q_{\bar{f}}, g_{\bar{f}}) \geq |V(q_{\bar{f}})|$, then we have $\mathtt{BMa}_f(q_{\bar{f}}, g_{\bar{f}}) \geq \mathtt{SMa}_f(q_{\bar{f}}, g_{\bar{f}})$. □

Note that, the above lemma is conservative, while in practice, $\mathsf{lb}_f^{\mathtt{SMa}}$ is even smaller than $\mathsf{lb}_f^{\mathtt{LSa}}$ as verified by our experiments in Section VI-B. 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 cost $\lambda_f^{\mathtt{SMa}}(v,u)$ has to be normalized by a large factor of $\max\{4, \Delta(q_{\bar{f}})+1, \Delta(g_{\bar{f}})+1\}$.