"© 2020 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works."

Speeding Up GED Verification for Graph Similarity Search

Lijun Chang[§], Xing Feng[†], Xuemin Lin[†], Lu Qin[‡], Wenjie Zhang[†], Dian Ouyang[§]

[§]The University of Sydney [†]University of New South Wales [‡]University of Technology Sydney

{lijun.chang,dian.ouyang}@sydney.edu.au, {lxue,zhangw}@cse.unsw.edu.au, lu.qin@uts.edu.au

***Abstract*—Graph similarity search retrieves from a database all graphs whose edit distance (GED) to a query graph is within a threshold. As GED computation is NP-hard, the existing works adopt the filtering-and-verification paradigm to reduce the number of GED verifications, and they mainly focus on designing filtering techniques while using the now out-dated algorithm $\mathsf{A^*GED}$ for verification. In this paper, we aim to speed up GED verification, which is orthogonal to the index structures used in the filtering phase. We propose a best-first search algorithm $\mathsf{AStar^+}$-LSa which improves $\mathsf{A^*GED}$ by (1) reducing memory consumption, (2) tightening lower bound estimation, and (3) improving the time complexity for lower bound computation. We formally show that $\mathsf{AStar^+}$-LSa has a lower space and time complexity than $\mathsf{A^*GED}$. We further modify $\mathsf{AStar^+}$-LSa into a depth-first search algorithm to contrast these two search paradigms, and we extend our algorithms for exact GED computation. We conduct extensive empirical studies on real graph datasets, and show that our algorithm $\mathsf{AStar^+}$-LSa outperforms the state-of-the-art algorithms by several orders of magnitude for both GED verification and GED computation.**

I. Introduction

Graph model is ubiquitous and has been used to model the relationship/interaction between entities in a wide spectrum of applications, such as chemical compounds and biological structures. With the proliferation of graph data, one of the fundamental query types is *graph search*, which aims to retrieve all occurrences of a query graph in a graph database consisting of thousands or millions of small- and medium-sized graphs. Due to erroneous data entry, data noise, and/or the nature of the applications, it is not unusual that exact graph search returns no or very few results and thus does not serve the applications well. As a result, a recent trend is to find from the database all graphs that are *similar* to the query graph [8], [10], [11], [18], [21], [22], [23].

Various (dis-)similarity measures have been studied in the literature, such as graph edit distance [8], [17], maximum common subgraph [4], [7], and the number of miss-matching edges [24]. Among them, graph edit distance (GED) has been popularly adopted by the existing works on graph search queries [8], [10], [11], [18], [21], [22], [23]. This is because GED is a metric 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*. Thus, GED gives the minimum amount of distortion needed to transform one graph into the other. Note that $\mathsf{ged}(q, g) = \mathsf{ged}(g, q)$.

Given a graph database $\mathcal{D}$, a query graph q and a threshold τ, the problem of *graph similarity search* is to find all graphs in $\mathcal{D}$ whose GED to q is within the threshold τ, *i.e.*, $\mathsf{result} = \{g \in \mathcal{D} \mid \mathsf{ged}(q, g) \leq \tau\}$. As computing GED (as well as other graph similarity measures) is NP-hard [19], the existing works adopt the *filtering-and-verification* paradigm to reduce the number of GED verifications [22]. That is, an offline-constructed index is probed online to generate a set of candidate graphs $\mathsf{cand} \subseteq \mathcal{D}$ such that $\mathsf{result} \subseteq \mathsf{cand}$, *i.e.*, no true result is filtered; this is the *filtering* phase. Then, the GED between q and each candidate graph $g \in \mathsf{cand}$ is verified to identify the true results; this is the *verification* phase. The existing studies mainly focus on designing effective index structures — such as q-gram-based index [21], star structure-based index [18], and subgraph-based index [11], [22] — aiming to reduce the size of cand, while using the now out-dated algorithm $\mathsf{A^*GED}$ [15] for GED verification.

GED Verification. In this paper, we aim to speed up GED verification for graph similarity search, which is *orthogonal* to the index structures used in the filtering phase and is the *bottleneck* in the existing graph similarity search algorithms. Besides the application in graph similarity search, GED verification/computation is also a key building block in graph classification [12] and graph clustering [16]. In addition, it also assists to identify functionally related enzyme clusters in biochemistry [13], to compare electroencephalogram in medicine [2], and to retrieve similar objects in videos [6].

The existing GED algorithms are *branch-and-bound* algorithms [1], [3], [8], [14], [15]. They enumerate mappings from vertices of q' to vertices of g', where q' and g' are obtained from q and g, respectively, by *adding dummy vertices* to cope with vertex insertion/deletion. Specifically, if a dummy vertex of q' is mapped to a non-dummy vertex u of g' in a mapping f, then it means that u is *inserted* into q to match g regarding mapping f. Similarly, mapping a non-dummy vertex v of q' to a dummy vertex of g' corresponds to *deleting* vertex v from q to match g. Then, $\mathsf{ged}(q, g)$ equals the minimum cost among all the mappings from $V(q')$ to $V(g')$.

As there is an exponential number of mappings, these mappings are implicitly organized into a prefix-shared search tree $\mathcal{T}$ (see Figure 2) such that *branches* of $\mathcal{T}$ can be pruned based on lower *bound* computations. Specifically, each node

TABLE I: A summary of the existing vertex mapping-based algorithms A*GED and DF_GED, and our algorithms AStar$^+$-LS, DFS$^+$-LS, AStar$^+$-LSa, and DFS$^+$-LSa

Algorithms	Add dummy vertices	Search strategy	Lower bound estimation	Time complexity of extending a partial mapping
A*GED [14], [15]	Yes	Best-first search	Label set-based	$O(\lvert V(g)\rvert \times (\lvert E(q)\rvert + \lvert E(g)\rvert))$
DF_GED [1], [3]	Yes	Depth-first search	Label set-based	$O(\lvert V(g)\rvert \times (\lvert E(q)\rvert + \lvert E(g)\rvert))$
AStar$^+$-LS	No	Best-first search	Label set-based	$O(\lvert E(q)\rvert + \lvert E(g)\rvert)$
DFS$^+$-LS	No	Depth-first search	Label set-based	$O(\lvert E(q)\rvert + \lvert E(g)\rvert)$
AStar$^+$-LSa	No	Best-first search	Anchor-aware label set-based	$O(\lvert E(q)\rvert + \lvert E(g)\rvert)$
DFS$^+$-LSa	No	Depth-first search	Anchor-aware label set-based	$O(\lvert E(q)\rvert + \lvert E(g)\rvert)$

of $\mathcal{T}$ corresponds to a partial mapping f that is shared by all full mappings that are its descendants in $\mathcal{T}$. The lower bound cost of f, denoted lb_f, is computed as a lower bound of the costs of all full mappings that are descendants of f, and the branch rooted at f is pruned if $\mathsf{lb}_f > \tau$. There are two key components in branch-and-bound algorithms.

- **Search Strategy:** the first GED algorithm, A*GED [14], [15], adopts the *best-first search* strategy, while recent algorithms, DF_GED [1], [3] and CSI_GED [8], suggest conducting a *depth-first search*.
- **Lower Bound Estimation:** A*GED and DF_GED use *label set-based lower bound*, while CSI_GED uses *degree-based lower bound*.

It is reported in [8] that CSI_GED outperforms A*GED by over two orders of magnitude. However, it is unclear whether this is due to the different search strategies or the different lower bound estimations or anything else, used by CSI_GED.

Contributions. A summary of our algorithms against the existing algorithms is shown in Table I. Our main contributions are summarized as follows.

① *We show that* $\mathsf{ged}(q, g)$ *can be computed via enumerating mappings from* $V(q)$ *to* $V(g)$ *without adding dummy vertices (Section III).* Thus, the total number of mappings is $\approx |V(g)|^{|V(q)|}$. In contrast, the existing algorithms as discussed above need to enumerate mappings from $V(q')$ to $V(g')$, and there are approximately $(|V(q)| + |V(g)|)^{|V(q)|+|V(g)|}$ mappings in total as $|V(q')| = |V(g')| = |V(q)| + |V(g)|$. We significantly improve the worst-case behaviour of GED algorithms.

② *We propose a best-first search algorithm* AStar$^+$-LSa *for efficient GED verification (Section IV).* AStar$^+$-LSa improves the existing best-first search algorithm A*GED from the following aspects. Firstly, AStar$^+$-LSa reduces the memory consumption by storing each partial mapping (*i.e.*, search state) in constant memory space. Secondly, AStar$^+$-LSa reduces the search space by using a tighter lower bound estimation. Thirdly, we propose an efficient algorithm to extend a partial mapping (*i.e.*, compute the lower bound cost for all of its children) in linear time, which improves the time complexity of the existing techniques by a factor of $|V(g)|$ (see the last column of Table I). As a result, AStar$^+$-LSa has both a lower space and a lower time complexity than A*GED.

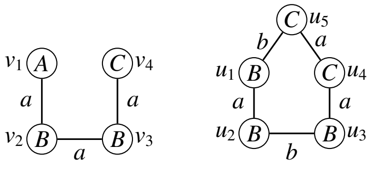
③ *We modify* AStar$^+$-LSa *into a depth-first search algorithm* DFS$^+$-LSa *to contrast these two search paradigms, and we extend our algorithms for GED computation (Section V).* DFS$^+$-LSa differs from AStar$^+$-LSa only in the definition of priority for the queues used by these two algorithms. We quantitatively show that DFS$^+$-LSa has a lower time complexity than the existing depth-first search algorithm DF_GED. We also show that AStar$^+$-LSa in general has a smaller search space and thus lower time complexity than DFS$^+$-LSa, which becomes more profound when conducting exact GED computation.

④ *We conduct extensive performance studies and have the following findings (Section VII).* (1) Our algorithm AStar$^+$-LSa outperforms the state-of-the-art algorithms CSI_GED [8] and Inves [10] for index-free graph similarity search (*i.e.*, directly verifying GED) by up-to two orders of magnitude, and outperforms CSI_GED for GED computation by up-to four orders of magnitude. (2) The state-of-the-art index-based filtering algorithm Pars [20], [22] for graph similarity search has limited effectiveness; that is, the improvement of AStar$^+$-LSa by using Pars for filtering will be at most 52% than directly verifying all graphs by AStar$^+$-LSa. (3) Our best-first search algorithm AStar$^+$-LSa runs faster than its depth-first search variant DFS$^+$-LSa (which outperforms both CSI_GED and DF_GED) for both GED verification and GED computation; this invalidates the recent claims in [3], [8] that depth-first search is more suitable than best-first search for GED verification/computation.

II. Preliminaries

In this paper, we focus our discussions on labeled and undirected simple graphs while our techniques can be straightforwardly extended to handle other types of graphs. A labeled and undirected graph is represented by $g = (V(g), E(g), l)$, where $V(g)$ is the set of vertices, $E(g)$ is the set of edges, and $l : V(g) \cup E(g) \to \Sigma$ is a labelling function that assigns each vertex and/or edge a label from Σ. Specifically, $l(u)$ and $l(u, u')$ are the label of vertex u and the label of edge (u, u'), respectively. The number of vertices and the number of edges in g are denoted by $|V(g)|$ and $|E(g)|$, respectively. Given a vertex subset $S \subseteq V(g)$, the subgraph of g induced by S is $g_S = (S, \{(u, u') \in E(g) \mid u, u' \in S\}, l)$. 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*), inserting/deleting an edge between two vertices (*edge insertion* and *edge deletion*), and changing the label of a vertex/edge (*vertex relabeling* and *edge relabeling*).



(a) Graph q (b) Graph g

Fig. 1: Sample graphs

Definition 2.1: *The **graph edit distance** (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.*

Note that, $\mathsf{ged}(q,g) = \mathsf{ged}(g,q)$ and $\mathsf{ged}(\cdot,\cdot)$ is a metric [17]. Consider the two graphs q and g in Figure 1, 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 [19].

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

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 II.

TABLE II: Frequently used notations

Notation	Description
$\mathsf{ged}(q,g)$	The GED between graphs q and g
$\Upsilon(S_1,S_2)$	Edit distance between multi-sets S_1 and S_2, *i.e.*, $\Upsilon(S_1,S_2) = \max\{\|S_1\|,\|S_2\|\} - \|S_1 \cap S_2\|$
$\mathcal{T}$	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
$f^{-}(u)$	The vertex of $V(q)$ that maps to $u \in V(g)$
mc_f	The mapping cost of a (possibly partial) mapping f
edc_f	The editorial cost of a full 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
$L_V(q_{\bar{f}})$	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 $q_{\bar{f}}$
$L_{E_C}(v)$	Multi-set of labels of v's cross adjacent edges

III. Compute GED via Vertex Mapping

As GED is a metric, any of the two graphs can be regarded as q. **We choose q to be the graph with fewer vertices.** We prove in Lemma 3.1 that there then is no vertex deletion in the optimal sequence (*i.e.*, the sequence with the minimum number) of edit operations that transform q into g.

Lemma 3.1: *Given graphs q and g with $|V(q)| \leq |V(g)|$, there is no vertex deletion in the optimal sequence of edit operations that transform q into g.*

Proof: We prove the lemma by contradiction. Assume there is such a sequence of edit operations, $P = (o_1, \ldots, o_i, \ldots, o_n)$ of length $n = \mathsf{ged}(q,g)$, that contains a vertex deletion. Without loss of generality, let o_i be the operation of deleting vertex v from q. Then, there must also exist a vertex insertion operation since $|V(q)| \leq |V(g)|$; let o_j be the operation of inserting vertex v' with label a, where j can be either smaller or larger than i. Now consider another sequence P' of edit operations that differs from P by removing o_i and changing o_j from vertex insertion to vertex relabeling (*i.e.*, change the label of v to a); note that, we may also need to replace the occurrences of v' in P with v. It is easy to verify that $|P'| = |P| - 1$ and P' also transforms q into g, which contradicts that P is optimal. Thus, the lemma holds. □

Following Lemma 3.1, we do not need to add dummy vertices to g for encoding vertex deletion from q. Recall that existing algorithms need to add dummy vertices to both q and g to cope with vertex insertion/deletion (see Section I).

Computing GED via Enumerating Vertex Mappings. It can be verified that if $|V(q)| < |V(g)|$, then $\mathsf{ged}(q,g) = \mathsf{ged}(q',g)$ where q' is obtained from q by adding $|V(g)| - |V(q)|$ isolated dummy vertices with label $\perp \notin \Sigma$. Thus, we can assume that $|V(q)| = |V(g)|$; we will remove this assumption shortly. Then, we can prove in a similar way to the proof of Lemma 3.1 that there is no vertex deletion nor vertex insertion in the optimal sequence of edit operations that transform q into g. That is, *we only need to consider four edit operations,* i.e.*, edge insertion/deletion, and vertex/edge relabeling*. As a result, 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 simplicity, we refer to one-to-one mapping as mapping in the following. We define the editorial cost of a full mapping (*i.e.*, involving all vertices of q) as follows.

Definition 3.1: *For a full mapping f from $V(q)$ to $V(g)$, the **editorial cost** of f, denoted $\mathsf{edc}_f(q,g)$ and abbreviated as edc_f, is the number of edit operations that are required to transform q into g by obeying the mapping f (*i.e.*, $v \in V(q)$ maps to $f(v) \in V(g)$).*

The editorial cost of a full mapping from $V(q)$ to $V(g)$ can be computed in $O(|E(q)| + |E(g)|)$ time;[1] such an algorithm is shown in Algorithm 1. Following the above discussions, the GED can be computed from mappings as follows.

Lemma 3.2: *The GED between q and g equals the minimum editorial cost among all full mappings $\mathcal{F}(q,g)$ from $V(q)$ to $V(g)$; that is, $\mathsf{ged}(q,g) = \min_{f \in \mathcal{F}(q,g)} \mathsf{edc}_f$.*

Thus, $\mathsf{ged}(q,g)$ can be computed by enumerating all full mappings from $V(q)$ to $V(g)$ and computing their editorial

[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 1: EditorialCost

```
Input: Graphs $q$ and $g$ with $|V(q)| = |V(g)|$, and a full mapping
       $f$ from $V(q)$ to $V(g)$
edc_f ← 0;
/* Vertex relabeling                                         */
for each vertex $v \in V(q)$ do
    if $l(v) \neq l(f(v))$ then edc_f ← edc_f + 1;
/* Edge deletion or relabeling                               */
for each edge $(v, v') \in E(q)$ do
    if $(f(v), f(v')) \notin E(g)$ or $l(v, v') \neq l(f(v), f(v'))$ then
        edc_f ← edc_f + 1;
/* Edge insertion                                            */
for each edge $(u, u') \in E(g)$ do
    if $(f^{-}(u), f^{-}(u')) \notin E(q)$ then edc_f ← edc_f + 1;
return edc_f;
```

costs. Then, the minimum editorial cost is the result. It is worth mentioning that, however, there is an exponential number of full mappings from $V(q)$ to $V(g)$.

Removing the Assumption of $|V(q)| = |V(g)|$**.** Now, we show that even if $|V(q)| < |V(g)|$, we still do not need to add dummy vertices to q. Note that, if $|V(q)| < |V(g)|$, then a full mapping may not involve all vertices of g. We define the mapping cost of a (possibly partial) mapping f by considering only the subgraphs q_f and g_f of q and g that are induced by vertices of f, respectively.

Definition 3.2: *For a (possibly partial) mapping f from $V(q)$ to $V(g)$, the* ***mapping cost*** *of f, denoted $\mathsf{mc}_f(q, g)$ and abbreviated as mc_f, is the number of edit operations required to transform q_f into g_f by obeying the mapping f.*

Note that, if f is a full mapping and $|V(q)| = |V(g)|$, then $\mathsf{mc}_f = \mathsf{edc}_f$; otherwise $\mathsf{mc}_f \neq \mathsf{edc}_f$. In particular, if f is a full mapping and $|V(q)| < |V(g)|$, then $\mathsf{edc}_f = \mathsf{mc}_f + (|V(g)| + |E(g)| - |V(g_f)| - |E(g_f)|)$, *i.e.*, edc_f equals mc_f plus the number of vertices and edges of g that are not in g_f. For example, consider the full mapping $f = \{v_1 \mapsto u_1, v_2 \mapsto u_2, v_3 \mapsto u_3, v_4 \mapsto u_4\}$ for graphs q and g in Figure 1, an edit operation is needed to relabel vertex v_1 to B, and an edit operation is needed to relabel edge (v_2, v_3) to b; thus, $\mathsf{mc}_f = 2$. It can be verified that $\mathsf{edc}_f = \mathsf{mc}_f + 3$ as it needs another three edit operations to transform q into g, *i.e.*, insert vertex v_5 and add edges (v_1, v_5) and (v_4, v_5) into q.

Based on the above discussions, it is easy to verify that Lemma 3.2 still holds even if $|V(q)| < |V(g)|$. Thus, dummy vertices are not needed for q either. As a result, the total number of full mappings is approximately $|V(g)|^{|V(q)|}$. Note that, although vertex mapping has been used in the existing GED algorithms (*e.g.*, in [9], [14], [15], [1], [3]), they all need to add dummy vertices to enlarge q and g to q' and g' (see Section I) such that $|V(q')| = |V(g')| = |V(q)| + |V(g)|$ to cope with possible vertex insertion and deletion of q. That is, they may enumerate approximately $(|V(q)| + |V(g)|)^{|V(q)|+|V(g)|}$ full mappings in the worst-case. We significantly reduce the worst-case number of full mappings, and thus improve the worst-case behaviour of GED algorithms.

IV. Our AStar^{+}-LSa Approach

In this section, we propose a best-first search algorithm AStar^{+}-LSa for efficient GED verification. We present our AStar^{+} search framework in Section IV-A, and our LSa lower bound estimation technique in Section IV-B. We analyze the time and space complexity of AStar^{+}-LSa in Section IV-C.

A. Our AStar^{+} Search Framework

Given graphs q and g, the set of all full mappings from $V(q)$ to $V(g)$ — according to a matching order $\pi = (v_1, \ldots, v_{|V(q)|})$ of $V(q)$ — can be compactly represented in a prefix-shared **search tree** $\mathcal{T}$, see Figure 2. A node at level i of $\mathcal{T}$ represents a partial mapping from $(v_1, \ldots, v_i)$ to $V(g)$, which *extends* the partial mapping of its parent (at level $i - 1$) by additionally mapping v_i to a vertex of $V(g)$. All the full mappings are at level $|V(q)|$ of $\mathcal{T}$. To distinguish nodes of the search tree $\mathcal{T}$ from vertices of graphs q and g, we refer to the former as nodes. Based on the parent-child relationships of nodes in $\mathcal{T}$, parent-child (as well as ancestor-descendant) relationships are also defined for mappings.

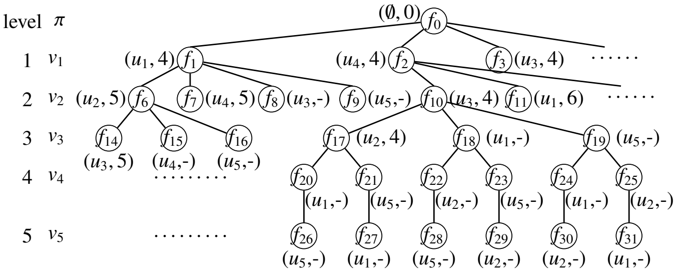


Fig. 2: The search tree $\mathcal{T}$ for computing $\mathsf{ged}(q, g)$ for graphs in Figure 3: f_i is a partial mapping, and beside f at level j is a pair (u, lb_f) where $u \in V(g)$ is the vertex to which v_j maps and lb_f is a lower bound of f

Figure 2 shows a snippet of the search tree $\mathcal{T}$ for the graphs q and g in Figure 3. The root node of $\mathcal{T}$ is at level 0, and represents an empty mapping $\emptyset$. Beside each node f at level i in $\mathcal{T}$, we show two values: the vertex $u \in V(g)$ to which v_i maps in the mapping f, and a lower bound cost lb_f of f which is defined in Definition 4.1; note that, we use f to denote both a mapping and its corresponding node in $\mathcal{T}$. The vertex to which v_j maps in the mapping f for $j < i$ can be obtained from the corresponding ancestor of f at level j 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, and f_6's parent is $f_1 = \{v_1 \mapsto u_1\}$.

To avoid enumerating all full mappings which are of exponential quantity, we compute a lower bound cost for a partial mapping for the purpose of pruning, as follows.

Definition 4.1: *The* ***lower bound cost of a 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 (*i.e.*, are descendants of) f.*

Algorithm 2: $\mathsf{AStar}^+(q, g, \tau)$

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

Thus, for GED verification, we can prune the search branch rooted at f (*i.e.*, all mappings that extend f) if lb_f is larger than the threshold τ. Based on this, the pseudocode of conducting a pruned best-first search on $\mathcal{T}$ for GED verification is shown in Algorithm 2, denoted AStar^+. We first compute a matching order of $V(q)$ which defines the search tree $\mathcal{T}$, and let it be $\pi = (v_1, \ldots, v_{|V(q)|})$ (Line 1). To conduct a best-first search on $\mathcal{T}$, we use a priority queue Q to maintain the search frontier which is initialized by the root of $\mathcal{T}$ (Line 2). Each entry of Q stores a partial mapping f to be extended, its level i and its parent pa in $\mathcal{T}$, its mapping cost mc_f and its lower bound cost lb_f. As long as Q is not empty (Line 3), we pop the top entry $(f, i, pa, \mathsf{mc}_f, \mathsf{lb}_f)$ (*i.e.*, with the minimum lb_f) from Q (Line 4), and then extend f by computing the lower bound cost lb_h for each child h of f (Line 5). For each child h with $\mathsf{lb}_h \leq \tau$, we push it into Q if it is not a full mapping (*i.e.*, $|h| = i + 1 < |V(q)|$) (Line 8), and we otherwise return `true` indicating that $\mathsf{ged}(q, g) \leq \tau$ (Line 7). If the algorithm does not return `true`, then we finally return `false` indicating that $\mathsf{ged}(q, g) > \tau$ (Line 9). It is easy to see that during the execution of the algorithm, we maintain the invariant that all mappings in Q are *partial mappings* and have *lower bound cost no larger than* τ.

Theorem 4.1: *Algorithm 2 correctly verifies the GED between graphs q and g, if* $\mathsf{lb}_f = \mathsf{edc}_f$ *for every full mapping f.*

Proof: It is easy to see that if Algorithm 2 returns `true` at Line 7, then it must be the case that $\mathsf{ged}(q, g) \leq \tau$ as $\mathsf{lb}_f = \mathsf{edc}_f$ for every full mapping f. The only possible case that Algorithm 2 goes wrong is that it returns `false` while $\mathsf{ged}(q, g) \leq \tau$. We prove that this cannot happen. From Lemma 3.2, we know that $\mathsf{ged}(q, g)$ equals the minimum editorial cost among all full mappings from $V(q)$ to $V(g)$. Thus, there is a full mapping f^* with editorial cost $\mathsf{ged}(q, g)$. Then, the lower bounds of all ancestors of f^* in $\mathcal{T}$ will be no larger than $\mathsf{ged}(q, g) \leq \tau$, and thus all ancestors of f^* will be pushed into Q at Line 8. Consequently, the parent of f^* will be popped from Q at Line 4, the lower bound cost of f^* will be computed as $\mathsf{ged}(q, g) \leq \tau$, and the algorithm will return `true` at Line 7. Thus, the theorem holds. □

Compared with Existing Best-First Search Framework. Although the existing algorithm $\mathsf{A^*GED}$ [14], [15] also uses the best-first search strategy, there are two features that distinguish AStar^+ (Algorithm 2) from $\mathsf{A^*GED}$. Firstly, AStar^+ reduces the memory consumption by storing each partial mapping (*i.e.*, search state) in a constant memory space, see the description of Q. Secondly, AStar^+ computes the lower bound cost for all children of a partial mapping at the same time (see Line 5), which enables an improved time complexity (see Section IV-B). As a result, AStar^+ has both a lower space and a lower time complexity than $\mathsf{A^*GED}$, see Section IV-C for the analysis. Moreover, we have shown in Section III that AStar^+ does not need to add dummy vertices to q or g, while $\mathsf{A^*GED}$ needs to which increases the search space.

A Frequency-Aware Matching Order. For completeness, we briefly describe our matching order selection algorithm in the following. It is mainly based on two intuitions: (1) a connected matching order is preferred; and (2) infrequent part of a graph should be mapped first. 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 its frequency in g for the corresponding vertex label or edge label. 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 append, to the end of π, the vertex that has the largest total weight for the vertex and its adjacent edges to vertices of π.

B. *Efficient* LSa *Lower Bound Computation*

In the following, we first introduce the anchor-aware label set-based lower bound LSa in Section IV-B1, and then propose linear-time algorithms for computing the lower bound cost for all children of a partial mapping in Section IV-B2

1) Anchor-aware Label Set-based Lower Bound: Given a partial mapping f, we decompose q into two parts: the subgraph q_f of q induced by vertices of f, and the remaining subgraph of q, denoted $q_{\bar{f}}$. Similarly, we decompose g into g_f and $g_{\bar{f}}$. 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.

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 3.* 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 $L_V(g_{\bar{f}})$ denote the multi-sets of vertex labels of $q_{\bar{f}}$ and $g_{\bar{f}}$, respectively, and let $L_E(q_{\bar{f}})$ and $L_E(g_{\bar{f}})$ denote the multi-sets of edge labels. The lower bound that has been extensively used in the existing algorithms $\mathsf{A^*GED}$ [14], [15] and $\mathsf{DF\_GED}$ [1], [3] is defined as follows.

Definition 4.2: *[3] The* ***label set-based lower bound*** *of a mapping f is* $\mathsf{lb}^{\mathsf{LS}}_f = \mathsf{mc}_f + \mathsf{LS}_f(q_{\bar{f}}, g_{\bar{f}})$ *where* $\mathsf{LS}_f(q_{\bar{f}}, g_{\bar{f}})$ *is the difference between vertex labels of* $q_{\bar{f}}$ *and* $g_{\bar{f}}$ *plus the difference between edge labels of* $q_{\bar{f}}$ *and* $g_{\bar{f}}$,

$$\mathsf{LS}_f(q_{\bar{f}}, g_{\bar{f}}) = \Upsilon\left(L_V(q_{\bar{f}}), L_V(g_{\bar{f}})\right) + \Upsilon\left(L_E(q_{\bar{f}}), L_E(g_{\bar{f}})\right) \quad (1)$$

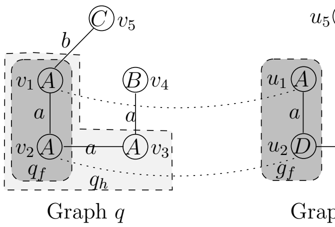


Fig. 3: Graphs q and g

Here $\Upsilon(\cdot,\cdot)$ *denotes the edit distance between two multi-sets and* $\Upsilon(S_1, S_2) = \max\{|S_1|, |S_2|\} - |S_1 \cap S_2|$. [2]

Example 4.2: *For the partial mapping f in Example 4.1, $L_V(q_{\bar{f}}) = \{A, B, C\}$, $L_V(g_{\bar{f}}) = \{A, A, E\}$, $L_E(q_{\bar{f}}) = \{a, a, b\}$, and $L_E(g_{\bar{f}}) = \{a, a, a\}$. Thus,* $\mathsf{LS}_f(q_{\bar{f}}, g_{\bar{f}}) = 2 + 1 = 3$. *As* $\mathsf{mc}_f = 1$, *we have* $\mathsf{lb}^{\mathsf{LS}}_f = 4$. □

Recently, Kim et al [10] and we (in our early draft [5]) independently proposed the anchor-aware label set-based lower bound, by making use of the information of the mapped vertices, called *anchored vertices*, of q_f.[3] The general idea is that 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, as each anchored vertex v in q_f must map to $f(v) \in V(g_f)$.

Definition 4.3: *[5], [10] The **anchor-aware label set-based lower bound** of a mapping f is* $\mathsf{lb}^{\mathsf{LSa}}_f = \mathsf{mc}_f + \mathsf{LSa}_f(q_{\bar{f}}, g_{\bar{f}})$

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

where $L_{E_C}(v)$ is the multi-set of labels of v's cross adjacent edges, and $L_{E_I}(q_{\bar{f}})$ and $L_{E_I}(g_{\bar{f}})$ are the multi-sets of labels of inner edges of $q_{\bar{f}}$ and $g_{\bar{f}}$, respectively.

Example 4.3: *For the partial mapping f in Example 4.1, we have $L_{E_I}(q_{\bar{f}}) = \{a\}$, $L_{E_I}(g_{\bar{f}}) = \{a, a\}$, $L_{E_C}(v_1) = \{b\}$, $L_{E_C}(v_2) = \{a\}$, $L_{E_C}(u_1) = \emptyset$, and $L_{E_C}(u_2) = \{a\}$. Thus,* $\mathsf{LSa}_f(q_{\bar{f}}, g_{\bar{f}}) = 2 + 1 + 1 = 4$, *and* $\mathsf{lb}^{\mathsf{LSa}}_f = 5 > \mathsf{lb}^{\mathsf{LS}}_f$. □

We prove in Lemma 4.1 that $\mathsf{lb}^{\mathsf{LSa}}_f \geq \mathsf{lb}^{\mathsf{LS}}_f$ holds in general. Note that, for lower bound estimation, the larger the tighter.

Lemma 4.1: *For any mapping f, we have* $\mathsf{lb}^{\mathsf{LSa}}_f \geq \mathsf{lb}^{\mathsf{LS}}_f$.

Proof: We can see that $L_{E_I}(q_{\bar{f}})$ and $L_{E_C}(v)$ for anchored vertices $v \in q_f$ refine $L_E(q_{\bar{f}})$, and $L_{E_I}(g_{\bar{f}})$ and $L_{E_C}(u)$ for $u \in g_f$ refine $L_E(g_{\bar{f}})$. That is, $L_E(q_{\bar{f}}) = L_{E_I}(q_{\bar{f}}) \cup \Big(\bigcup_{v \in q_f} L_{E_C}(v)\Big)$ and $L_E(g_{\bar{f}}) = L_{E_I}(g_{\bar{f}}) \cup \Big(\bigcup_{u \in g_f} L_{E_C}(u)\Big)$. Thus, $\Upsilon\Big(L_{E_I}(q_{\bar{f}}), L_{E_I}(g_{\bar{f}})\Big) + \sum_{v \in q_f} \Upsilon\Big(L_{E_C}(v), L_{E_C}(f(v))\Big) \geq \Upsilon\Big(L_E(q_{\bar{f}}), L_E(g_{\bar{f}})\Big)$ and the lemma holds. □

It is easy to verify that, for full mappings f, we have $\mathsf{lb}^{\mathsf{LS}} = \mathsf{lb}^{\mathsf{LSa}} = \mathsf{edc}_f$. Thus, Algorithm 2 correctly verifies GED if any one of these two lower bounds is used.

2) Linear-time Lower Bound Computation: For extending a partial mapping f in Algorithm 2, we need to compute the lower bound cost for all children of f. Assuming that f is at level i of the search tree $\mathcal{T}$ (*i.e.*, f maps vertices $v_1, \ldots, v_i$ of q to g_f), then f has $|V(g_{\bar{f}})|$ children and each child extends f by mapping v_{i+1} to a vertex of $g_{\bar{f}}$. We use $f \cup \{v_{i+1} \mapsto u\}$ to denote a child of f. In the following, we propose techniques to efficiently compute these lower bound costs, and for presentation simplicity we focus our discussions for the label set-based lower bound $\mathsf{lb}^{\mathsf{LS}}_f$.

A straightforward approach is to *independently* compute the lower bound cost $\mathsf{lb}^{\mathsf{LS}}_h$ for each child h of f, which is the strategy adopted by the existing algorithms $\mathsf{A^*GED}$ [14], [15] and $\mathsf{DF\_GED}$ [1], [3]. As computing $\mathsf{lb}^{\mathsf{LS}}_h$ for a specific child takes $O(|E(q)| + |E(g)|)$ time, the time complexity of this straightforward approach for computing the lower bound cost for all children of f is $O(|V(g)| \times (|E(q)| + |E(g)|))$. This is inefficient in viewing that such computation needs to be conducted for a lot of partial mappings.

To improve the efficiency for extending a partial mapping, we propose to first online build a data structure in linear time such that the lower bound cost of $f \cup \{v_{i+1} \mapsto u\}$ can be obtained in $O(d(u))$ time, where $d(u)$ is the degree of u in g. Let h be $f \cup \{v_{i+1} \mapsto u\}$. The data structure simply maintains, for each vertex label and each edge label, the difference between its number of occurrences in $q_{\bar{h}}$ and $g_{\bar{f}}$. Specifically,

- for each vertex label $A \in L_V(g_{\bar{f}}) \cup L_V(q_{\bar{h}})$, we store a *count* $\mathsf{cnt}_V(A)$ which equals the number of occurrences of A in $g_{\bar{f}}$ minus the number of its occurrences in $q_{\bar{h}}$;
- for each edge label $a \in L_E(g_{\bar{f}}) \cup L_E(q_{\bar{h}})$, we store a *count* $\mathsf{cnt}_E(a)$ which equals the number of occurrences of a in $g_{\bar{f}}$ minus the number of its occurrences in $q_{\bar{h}}$.

We can see that, the data structure is independent to u, the vertex that v_{i+1} maps to in h, as $q_{\bar{h}}$ is not related to u.

TABLE III: Data structure (Vlabel: Vertex label)

Vlabel	$\mathsf{cnt}_V(\cdot)$	Vlabel	$\mathsf{cnt}_V(\cdot)$	Edge label	$\mathsf{cnt}_E(\cdot)$
A	2	C	-1	a	2
B	-1	E	1	b	-1

Example 4.4: *Consider the graphs q and g in Figure 3 and $f = \{v_1 \mapsto u_1, v_2 \mapsto u_2\}$. The data structure constructed is shown in Table III, where $L_V(g_{\bar{f}}) = \{A, A, E\}$, $L_V(q_{\bar{h}}) = \{B, C\}$, $L_E(g_{\bar{f}}) = \{a, a, a\}$, and $L_E(q_{\bar{h}}) = \{a, b\}$.* □

Based on the data structure, we now show how to compute the lower bound $\mathsf{lb}^{\mathsf{LS}}_h$ of h in $O(d(u))$ time. Recall that, $h = f \cup \{v_{i+1} \mapsto u\}$ and

$$\mathsf{lb}^{\mathsf{LS}}_h = \mathsf{mc}_h + \Upsilon\left(L_V(q_{\bar{h}}), L_V(g_{\bar{h}})\right) + \Upsilon\left(L_E(q_{\bar{h}}), L_E(g_{\bar{h}})\right)$$

We discuss the three components one-by-one.

Firstly, mc_h can be obtained from mc_f by adding the cost of editing edges of $E(v_{i+1}, q_f)$ to map to edges of $E(u, g_f)$ according to the mapping h, where $E(v_{i+1}, q_f)$ denotes the edges between v_{i+1} and q_f and $E(u, g_f)$ is defined similarly. Specifically, we have

$$\mathsf{mc}_h = \mathsf{mc}_f + \mathbb{I}_{l(v_{i+1}) \neq l(u)} + |E(v_{i+1}, q_f)| + |E(u, g_f)| - c_1 - 2 \cdot c_2$$

[2] Note that, all unions ∪ and intersections ∩ in this paper follow the multi-set-based semantics.

[3] Note that, techniques for efficiently computing the anchor-aware label set-based lower bound for all children of a partial mapping are not discussed in [10].

where $\mathbb{I}_\phi$ is an indicator function that evaluates to 1 if ϕ is true and evaluates to 0 otherwise, c_1 is the number of matched edges with *different labels* between $E(v_{i+1}, q_f)$ and $E(u, g_f)$ (*i.e.*, edge relabeling is required), and c_2 is the number of matched edges with *the same labels* between $E(v_{i+1}, q_f)$ and $E(u, g_f)$ (*i.e.*, no edit operation is required). For example, continuing Example 4.4 and $u = u_3$, then $\mathbb{I}_{l(v_3) \neq l(u_3)} = 0$, $E(v_3, q_f) = \{(v_2, v_3)\}$ and $E(u_3, g_f) = \{(u_2, u_3)\}$. As (u_2, u_3) matches with (v_2, v_3) with the same edge label, we have $c_1 = 0$ and $c_2 = 1$. Thus, $\mathsf{mc}_h = \mathsf{mc}_f + 0$. Note that, among the above quantities, mc_f is directly obtained from Algorithm 2 (see Line 4) and $|E(v_{i+1}, q_f)|$ can be precomputed when building the data structure online, while $|E(u, g_f)|, c_1$ and c_2 can be computed online in $\mathcal{O}(d(u))$ time. Consequently, mc_h can be computed online in $\mathcal{O}(d(u))$ time.

Secondly, recall that $\Upsilon(L_V(q_{\bar{h}}), L_V(g_{\bar{h}}))$ equals

$$\max\{|L_V(q_{\bar{h}})|, |L_V(g_{\bar{h}})|\} - \left|L_V(q_{\bar{h}}) \cap L_V(g_{\bar{h}})\right|.$$

As $L_V(g_{\bar{h}})$ is obtained from $L_V(g_{\bar{f}})$ by removing the label $l(u)$, we have $\max\{|L_V(q_{\bar{h}})|, |L_V(g_{\bar{h}})|\} = \max\{|L_V(q_{\bar{h}})|, |L_V(g_{\bar{f}})| - 1\}$. Moreover, $|L_V(q_{\bar{h}}) \cap L_V(g_{\bar{h}})|$ differs from $|L_V(q_{\bar{h}}) \cap L_V(g_{\bar{f}})|$ by at most one which happens exactly when the number of occurrences of the vertex label $l(u)$ in $g_{\bar{f}}$ is no more than its number of occurrences in $q_{\bar{h}}$ (*i.e.*, when $\mathsf{cnt}_V(l(u)) \leq 0$). Thus, we have

$$\Upsilon(L_V(q_{\bar{h}}), L_V(g_{\bar{h}})) = \max\left\{|L_V(q_{\bar{h}})|, |L_V(g_{\bar{f}})| - 1\right\} - \left(|L_V(q_{\bar{h}}) \cap L_V(g_{\bar{f}})| - \mathbb{I}_{\mathsf{cnt}_V(l(u)) \leq 0}\right) \quad (3)$$

For example, continuing Example 4.4 and $u = u_3$, then $|L_V(q_{\bar{h}})| = 2$, $|L_V(g_{\bar{f}})| = 3$, $L_V(q_{\bar{h}}) \cap L_V(g_{\bar{f}}) = \emptyset$, and $\mathsf{cnt}_V(l(u_3)) = 2$. We have $\Upsilon(L_V(q_{\bar{h}}), L_V(g_{\bar{h}})) = 2 - 0 = 2$. Note that, among the above quantities, $|L_V(q_{\bar{h}})|$, $|L_V(g_{\bar{f}})|$, and $|L_V(q_{\bar{h}}) \cap L_V(g_{\bar{f}})|$ can be precomputed online when building the data structure. Consequently, $\Upsilon(L_V(q_{\bar{h}}), L_V(g_{\bar{h}}))$ can be computed online in constant time.

Similarly, we have

$$\Upsilon(L_E(q_{\bar{h}}), L_E(g_{\bar{h}})) = \max\left\{|L_E(q_{\bar{h}})|, |L_E(g_{\bar{f}})| - |E(u, g_f)|\right\} - \left(|L_E(q_{\bar{h}}) \cap L_E(g_{\bar{f}})| - c_E\right) \quad (4)$$

where c_E is the decrease in $|L_E(q_{\bar{h}}) \cap L_E(g_{\bar{f}})|$ when labels of edges $E(u, g_{\bar{f}})$ are removed from $L_E(g_{\bar{f}})$. Note that, c_E can be computed from the data structure in $\mathcal{O}(d(u))$ time. Consequently, $\Upsilon(L_E(q_{\bar{h}}), L_E(g_{\bar{h}}))$ can be computed online in $\mathcal{O}(d(u))$ time. For example, continuing Example 4.4 and $u = u_3$, then $L_E(q_{\bar{h}}) = \{a, b\}$, $L_E(g_{\bar{f}}) = \{a, a, a\}$, and $E(u_3, g_f) = \{(u_2, u_3)\}$ where $l(u_2, u_3) = \{a\}$. Thus, $c_E = 0$ and $\Upsilon(L_E(q_{\bar{h}}), L_E(g_{\bar{h}})) = 2 - (1 - 0) = 1$.

Based on the above ideas, the pseudocode of our algorithm for computing the lower bound costs of all children of f is shown in Algorithm 3. Line 1 constructs the data structure in linear time, Lines 4–9 compute mc_h in $\mathcal{O}(d(u))$ time, Line 10 computes $\Upsilon(L_V(q_{\bar{h}}), L_V(g_{\bar{h}}))$ in constant time, Lines 11-15 compute $\Upsilon(L_E(q_{\bar{h}}), L_E(g_{\bar{h}}))$ in $\mathcal{O}(d(u))$ time, and Line 16 sums these values to get $\mathsf{lb}_h^{\mathsf{LS}}$ in constant time. The following lemma immediately follows from the above discussions.

Algorithm 3: Compute lower bound for all f's children

```
Input: Graphs $q$ and $g$, a partial mapping $f$, and $\mathsf{mc}_f$
Output: Lower bound cost $\mathsf{lb}_h^{\mathsf{LS}}$ for all children $h$ of $f$
1  Construct the data structure $\mathsf{cnt}_V(\cdot)$ and $\mathsf{cnt}_E(\cdot)$ for $q_{\bar{h}}$ and $g_{\bar{f}}$;
2  for each vertex $u \in V(g_{\bar{f}})$ do
3      $h \leftarrow f \cup \{v_{i+1} \mapsto u\}$;
4      $c_1 \leftarrow 0, c_2 \leftarrow 0$;
5      for each edge $(u, u') \in E(u, g_f)$ (i.e., between $u$ and $g_f$) do
6          if $(v_{i+1}, f^-(u')) \in E(q)$ then
7              if $l(u, u') \neq l(v_{i+1}, f^-(u'))$ then $c_1 \leftarrow c_1 + 1$;
8              else $c_2 \leftarrow c_2 + 1$;
9      $\mathsf{mc}_h \leftarrow \mathsf{mc}_f + \mathbb{I}_{l(v_{i+1}) \neq l(u)} + |E(v_{i+1}, q_f)| + |E(u, g_f)| - c_1 - 2 \cdot c_2$;
10     Calculate $\Upsilon(L_V(q_{\bar{h}}), L_V(g_{\bar{h}}))$ by Equation (3);
11     $c_E \leftarrow 0$;
12     for each edge $(u, u') \in E(u, g_f)$ do
13         $a \leftarrow l(u, u')$; $\mathsf{cnt}_E(a) \leftarrow \mathsf{cnt}_E(a) - 1$;
14         if $\mathsf{cnt}_E(a) < 0$ then $c_E \leftarrow c_E - 1$;
15     Calculate $\Upsilon(L_E(q_{\bar{h}}), L_E(g_{\bar{h}}))$ by Equation (4);
16     $\mathsf{lb}_h^{\mathsf{LS}} \leftarrow \mathsf{mc}_h + \Upsilon(L_V(q_{\bar{h}}), L_V(g_{\bar{h}})) + \Upsilon(L_E(q_{\bar{h}}), L_E(g_{\bar{h}}))$;
       /* The following lines restore $\mathsf{cnt}_E(\cdot)$                */
17     for each edge $(u, u') \in E(u, g_f)$ do
18         $a \leftarrow l(u, u')$; $\mathsf{cnt}_E(a) \leftarrow \mathsf{cnt}_E(a) + 1$;
```

Lemma 4.2: *Algorithm 3 correctly computes the lower bound cost for all children of f regarding $\mathsf{lb}^{\mathsf{LS}}$ in $\mathcal{O}(|E(q)| + |E(g)|)$ total time.*

Similar to Algorithm 3, we can compute the lower bound cost for all children of f regarding $\mathsf{lb}^{\mathsf{LSa}}$ in $\mathcal{O}(|E(q)| + |E(g)|)$ total time. Note that, in order to efficiently compute $\sum_{v \in q_f} \Upsilon(L_{E_C}(v), L_{E_C}(f(v)))$ in $\mathcal{O}(d(u))$ time, we will also need to maintain such a data structure $\mathsf{cnt}_E(\cdot)$ for every vertex of q_f. We omit the details.[4]

C. *Analysis of* AStar$^+$-LSa

Let $\mathcal{T}_{\leq x}$ be the set of non-leaf nodes/partial mappings in $\mathcal{T}$ whose lower bound costs are no larger than a threshold x, and $|\mathcal{T}_{\leq x}|$ be its cardinality. Note that, $\mathcal{T}_{\leq x_1} \subseteq \mathcal{T}_{\leq x_2}$ if $x_1 \leq x_2$. Then, we have the following lemma.

Lemma 4.3: *The total numbers of partial mappings that were ever popped out the priority queue Q and pushed into Q, when running* AStar$^+$, *are* $\min\{|\mathcal{T}_{\leq \tau}|, |\mathcal{T}_{\leq \mathsf{ged}(q,g)}|\}$ *and* $\min\{|\mathcal{T}_{\leq \tau}|, |V(g)| \cdot |\mathcal{T}_{\leq \mathsf{ged}(q,g)}|\}$, *respectively.*

Proof: Firstly, if $\tau < \mathsf{ged}(q, g)$, then q is dissimilar to g and Algorithm 2 terminates when $Q = \emptyset$. Then, every partial mapping whose lower bound cost is no larger than τ must be pushed into Q and then popped out from Q. Moreover, only the partial mappings whose lower bound costs are no larger than τ are pushed into Q (see Line 6). Thus, the sets of partial mappings that are pushed into and popped out from Q are exactly $\mathcal{T}_{\leq \tau}$.

Secondly, if $\mathsf{ged}(q, g) \leq \tau$, then Algorithm 2 will return `true` at Line 7. Let f be the partial mapping popped from Q at Line 4 when the algorithm terminates at Line 7. Then,

[4] Details can be found in our C++ source code, which will be available in GitHub along the publication of the paper.

the lower bound cost of f must be no larger than $\mathsf{ged}(q,g)$. Thus, every partial mapping that is popped from Q must have a lower bound cost no larger than that of f, due to the strategy of always popping from Q the partial mapping with the smallest lower bound cost. Consequently, the set of partial mappings that are popped out from Q is a subset of $\mathcal{T}_{\leq \mathsf{ged}(q,g)}$. For each partial mapping that is popped from Q, we push at most $|V(g)|$ new partial mappings into Q at Line 8 as each partial mapping has at most $|V(g)|$ children. Moreover, all partial mappings that were pushed into Q have lower bound costs $\leq \tau$ (Line 6). Thus, the lemma holds. □

Note that, the above two lemmas hold for any lower bound estimation. In the following, we use the superscripts LS and LSa to denote the two lower bounds discussed in Section IV-B1. Recall that, the tighter the lower bound estimation, the smaller the size of $\mathcal{T}_{\leq x}$. Thus, following from Lemma 4.1, we have $\mathcal{T}^{\mathsf{LSa}}_{\leq x} \subseteq \mathcal{T}^{\mathsf{LS}}_{\leq x}$ for any x.

Space Complexity of $\mathsf{AStar^{+}\text{-}LSa}$**.** We prove the space complexity of $\mathsf{AStar^{+}\text{-}LSa}$ by the theorem below.

Theorem 4.2: *The space complexity of* $\mathsf{AStar^{+}\text{-}LSa}$ *is* $O\left(\min\left\{|\mathcal{T}^{\mathsf{LSa}}_{\leq\tau}|, |V(g)| \cdot |\mathcal{T}^{\mathsf{LSa}}_{\leq \mathsf{ged}(q,g)}|\right\}\right)$.

Proof: The space consumption of $\mathsf{AStar^{+}\text{-}LSa}$ is dominated by the priority queue Q. Firstly, we have proved in Lemma 4.3 that the total number of partial mappings that were ever pushed into Q is $\min\{|\mathcal{T}^{\mathsf{LSa}}_{\leq\tau}|, |V(g)| \cdot |\mathcal{T}^{\mathsf{LSa}}_{\leq \mathsf{ged}(q,g)}|\}$. Secondly, the information of each partial mapping (*i.e.*, each entry in Q) is 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 other information — such as its level number i, its parent pa, its mapping cost mc_f, and its lower bound cost $\mathsf{lb}^{\mathsf{LSa}}_f$ — each take constant space. Thus, the space complexity of $\mathsf{AStar^{+}\text{-}LSa}$ follows. □

The existing best-first search algorithm $\mathsf{A^{*}GED}$ uses the label set-based lower bound $\mathsf{lb}^{\mathsf{LS}}$, and explicitly stores the partial mappings in Q where each partial mapping takes space $O(|V(q)|)$. Thus, the space complexity of $\mathsf{A^{*}GED}$ is $O\left(|V(q)| \times \min\left\{|\mathcal{T}^{\mathsf{LS}}_{\leq\tau}|, |V(g)| \cdot |\mathcal{T}^{\mathsf{LS}}_{\leq \mathsf{ged}(q,g)}|\right\}\right)$, which is at least $|V(q)|$ times larger than that of $\mathsf{AStar^{+}\text{-}LSa}$ by noting the following two facts. (1) $|\mathcal{T}^{\mathsf{LS}}_{\leq x}| \geq |\mathcal{T}^{\mathsf{LSa}}_{\leq x}|$ for any x. (2) $\mathsf{A^{*}GED}$ adds dummy vertices to q and g (see Section I), which further enlarges the search tree $\mathcal{T}$.

Time Complexity of $\mathsf{AStar^{+}}$**.** We define the *search space* of $\mathsf{AStar^{+}\text{-}LSa}$ as the number of partial mappings that are extended. Then, following Lemma 4.3, the search space of $\mathsf{AStar^{+}\text{-}LSa}$ is $\min\left\{|\mathcal{T}^{\mathsf{LSa}}_{\leq\tau}|, |\mathcal{T}^{\mathsf{LSa}}_{\leq \mathsf{ged}(q,g)}|\right\}$. We prove the time complexity of $\mathsf{AStar^{+}\text{-}LSa}$ by the following theorem.

Theorem 4.3: *The time complexity of* $\mathsf{AStar^{+}\text{-}LSa}$ *is* $O\Big(\min\left\{|\mathcal{T}^{\mathsf{LSa}}_{\leq\tau}|, |\mathcal{T}^{\mathsf{LSa}}_{\leq \mathsf{ged}(q,g)}|\right\} \times (|E(q)| + |E(g)| + |V(q)| \cdot \log |V(g)|)\Big)$.

Proof: Following from Lemma 4.3, $\mathsf{AStar^{+}\text{-}LSa}$ runs for at most $\min\{|\mathcal{T}^{\mathsf{LSa}}_{\leq\tau}|, |\mathcal{T}^{\mathsf{LSa}}_{\leq \mathsf{ged}(q,g)}|\}$ iterations (*i.e.*, Lines 4–8). Each iteration consists of one pop operation (Line 4) and at most $|V(g)|$ push operations (Line 8) of Q, and one invocation of lower bound computation (Line 5). The time complexity of pop is $O(\log|Q|)$ which is $O(|V(q)| \cdot \log|V(g)|)$ as $|Q| \leq |V(g)|^{|V(q)|}$, the time complexity of push is $O(1)$, and the time complexity of lower bound computation is $O(|E(q)| + |E(g)|)$. Thus, the theorem holds. □

As $|V(q)| \cdot \log|V(g)|$ usually is smaller than $|E(q)| + |E(g)|$, we can simply regard the time complexity of $\mathsf{AStar^{+}\text{-}LSa}$ as $O\Big(\min\left\{|\mathcal{T}^{\mathsf{LSa}}_{\leq\tau}|, |\mathcal{T}^{\mathsf{LSa}}_{\leq \mathsf{ged}(q,g)}|\right\} \times (|E(q)| + |E(g)|)\Big)$. As $\mathsf{A^{*}GED}$ extends a partial mapping in $O(|V(g)| \cdot (|E(q)| + |E(g)|))$ time, the time complexity of $\mathsf{A^{*}GED}$ is $O\Big(\min\left\{|\mathcal{T}^{\mathsf{LS}}_{\leq\tau}|, |\mathcal{T}^{\mathsf{LS}}_{\leq \mathsf{ged}(q,g)}|\right\} \times |V(g)| \times (|E(q)| + |E(g)|)\Big)$, which is at least $|V(g)|$ times larger than that of $\mathsf{AStar^{+}\text{-}LSa}$.

V. Extensions

In this section, we modify our best-first search algorithm $\mathsf{AStar^{+}\text{-}LSa}$ into a depth-first search algorithm $\mathsf{DFS^{+}\text{-}LSa}$ to contrast these two search paradigms in Section V-A, and then extend our algorithms for GED computation in Section V-B.

A. Our $\mathsf{DFS^{+}\text{-}LSa}$ *Approach*

Algorithm 2 works in a depth-first search fashion if at Line 4, we pop from the priority queue Q the partial mapping that has the largest level number in the search tree $\mathcal{T}$; if there is a tie, then we prefer the one with a smaller lower bound cost. As the level numbers are integers in the range from 1 to $|V(q)|$, we can simulate the priority queue by an array of size $|V(q)|$. We denote this variant of Algorithm 2 as $\mathsf{DFS^{+}}$. By combining $\mathsf{DFS^{+}}$ with our lower bound estimation algorithm for LSa, we have the algorithm $\mathsf{DFS^{+}\text{-}LSa}$.

Analysis of $\mathsf{DFS^{+}\text{-}LSa}$**.** The space complexity of $\mathsf{DFS^{+}\text{-}LSa}$ is $O(|V(q)| \times |V(g)|)$. Firstly, the number of distinct levels is $|V(q)|$. Secondly, for each level, there can have up-to $|V(g)|$ partial mappings stored in the priority queue at each moment. Thus, the total number of partial mappings in the priority queue at each moment is at most $|V(q)| \times |V(g)|$, where each partial mapping takes constant space.

Let $\mathcal{T}_{\mathsf{DFS}}$ be the set of partial mappings that are extended by $\mathsf{DFS^{+}}$, which is also the search space of $\mathsf{DFS^{+}}$. Then, the time complexity of $\mathsf{DFS^{+}\text{-}LSa}$ is $O(|\mathcal{T}^{\mathsf{LSa}}_{\mathsf{DFS}}| \times (|E(q)| + |E(g)|))$.[5] Note that if $\tau < \mathsf{ged}(q,g)$, then it can be proved in a similar way to the proof of Lemma 4.3 that $\mathcal{T}_{\mathsf{DFS}} = \mathcal{T}_{\leq\tau}$. However, for the case $\tau \geq \mathsf{ged}(q,g)$, it is possible that $|\mathcal{T}_{\mathsf{DFS}}|$ is larger than $|\mathcal{T}_{\leq \mathsf{ged}(q,g)}|$, and it is also possible that $|\mathcal{T}_{\mathsf{DFS}}| < |\mathcal{T}_{\leq \mathsf{ged}(q,g)}|$.

Compared with the Existing Depth-First Search Algorithms. The strategy of depth-first search has been used in the existing GED algorithms $\mathsf{DF\_GED}$ [1], [3] and $\mathsf{CSI\_GED}$ [8]. $\mathsf{DF\_GED}$ is similar to $\mathsf{A^{*}GED}$ but uses depth-first search. Thus, the time complexity of $\mathsf{DF\_GED}$ is $O(|V(g)| \times |\mathcal{T}^{\mathsf{LS}}_{\mathsf{DFS}}| \times (|E(q)| + |E(g)|))$, which is at least $|V(g)|$ times larger than that of our algorithm $\mathsf{DFS^{+}\text{-}LSa}$.

On the other hand, $\mathsf{CSI\_GED}$ enumerates edge mappings rather than vertex mappings. Moreover, $\mathsf{CSI\_GED}$ uses a different lower bound, the *degree-based lower bound* $\mathsf{lb}^{\mathsf{DE}}$,

[5] Note that, for simplicity, we here ignore the time for breaking ties.

which is similar to Definition 4.3 but revises $\Upsilon(S_1, S_2)$ to be $\big||S_1| - |S_2|\big|$. Due to entirely ignoring the edge labels of $q_{\bar{f}}$ and $g_{\bar{f}}$, $\mathsf{lb}_f^{\mathtt{DE}}$ is no larger than and could be much smaller than $\mathsf{lb}_f^{\mathtt{LSa}}$. Our experimental results in Section VII show that DFS$^+$-LSa significantly outperforms CSI_GED.

Compared with Our Best-First Search AStar$^+$-LSa. It is obvious that best-first search algorithms have a larger and usually much larger space complexity than depth-first search algorithms. As a result, a best-first search algorithm may run out-of-memory if it has a very large search space, which is the case for A*GED. This motivates the recent algorithms DF_GED and CSI_GED to adopt the depth-first search paradigm. We show in the following that best-first search and depth-first search have different behaviours for similar and dissimilar graph pairs, for GED verification.

- If q and g are dissimilar (*i.e.*, $\mathsf{ged}(q, g) > \tau$), then the search spaces of AStar$^+$-LSa and DFS$^+$-LSa are the same, *i.e.*, $\mathcal{T}_{\leq\tau} = \mathcal{T}_{\mathsf{DFS}}$ as discussed above. Thus, they will perform similarly for dissimilar graph pairs, with AStar$^+$-LSa being slightly slower due to the overhead of priority queue.
- If $\mathsf{ged}(q, g) \leq \tau$, then the search space of AStar$^+$-LSa is a subset of $\mathcal{T}^{\mathtt{LSa}}_{\leq \mathsf{ged}(q,g)}$ (see Lemma 4.3), while DFS$^+$-LSa may extend any partial mapping in $\mathcal{T}_{\leq\tau} \supseteq \mathcal{T}^{\mathtt{LSa}}_{\leq \mathsf{ged}(q,g)}$ and terminates once finding a full mapping with editorial cost at most τ. Our experimental results in Section VII-C show that AStar$^+$-LSa performs better than DFS$^+$-LSa for similar graph pairs.

Moreover, AStar$^+$-LSa largely solves the out-of-memory issue of A*GED by (1) not adding dummy vertices to q or g, (2) storing each search state in constant memory, and (3) using a tighter lower bound estimation. Thus, AStar$^+$-LSa is suitable for GED verification.

B. GED Computation

Both our algorithms AStar$^+$-LSa and DFS$^+$-LSa can be easily extended for computing the GED value. For example, we modify AStar$^+$-LSa for GED computation as follows: set τ to be ∞, remove Line 7 from Algorithm 2, and terminate the algorithm when the mapping popped at Line 4 is a full mapping. Then, the editorial cost of this mapping is $\mathsf{ged}(q, g)$.

In the following, we show that AStar$^+$-LSa is likely to have a smaller search space (and thus time complexity) than DFS$^+$-LSa for GED computation. Firstly, the search space of AStar$^+$-LSa is a subset of $\mathcal{T}^{\mathtt{LSa}}_{\leq \mathsf{ged}(q,g)}$ (see Lemma 4.3). Secondly, we have the following for the search space of DFS$^+$-LSa.

1) It can be verified that every partial mapping whose lower bound cost is smaller than $\mathsf{ged}(q, g)$ must be extended by DFS$^+$-LSa, *i.e.*, $\mathcal{T}_{\mathsf{DFS}} \supseteq \mathcal{T}_{<\mathsf{ged}(q,g)}$.
2) DFS$^+$-LSa usually also extends many partial mappings whose lower bound costs are larger than $\mathsf{ged}(q, g)$, due to the depth-first search strategy.

Our empirical studies in Section VII-D show that DFS$^+$-LSa has a significantly larger search space than AStar$^+$-LSa. Thus, AStar$^+$-LSa is better than DFS$^+$-LSa for GED computation.

VI. Related Work

Graph Similarity Search. GED-based graph similarity search has been studied in [11], [18], [20], [21], [22], [23]. All these works focus on designing effective index structures — such as q-gram-based index [21], star structure-based index [18], and subgraph-based index [11], [20], [22] — to filter out as many false-positive graphs (*i.e.*, dissimilar to the query graph) as possible, while all remaining candidates are verified by the now out-dated algorithm A*GED. In this paper, we propose a significantly improved GED verification algorithm AStar$^+$-LSa, which is index-free and can be incorporated to speed up GED verification for any index-based graph similarity search algorithm.

GED Verification and Computation. The notion of GED is defined in [17], and is proved to be NP-hard in [19]. A best-first search algorithm A*GED is developed in [15], and depth-first search algorithms DF_GED [1], [3] and CSI_GED [8] are recently proposed and shown to outperform A*GED. All these algorithms can compute as well as verify GED between two graphs. Recently, Inves [10] conducts online graph partitioning-based filtering for GED verification, which extends and enhances the offline graph partitioning-based index method in Pars [22]. In this paper, we propose a best-first search algorithm AStar$^+$-LSa which outperforms all existing algorithms by several orders of magnitude for both GED verification and GED computation.

VII. Experiments

In this section, we conduct extensive empirical studies aiming to answer the following major questions.

1) *How does* AStar$^+$-LSa *perform compared to the existing index-free algorithms* CSI_GED *and* Inves *for graph similarity search?*
   We show in **Eval-I** that AStar$^+$-LSa outperforms both CSI_GED and Inves by up-to two orders of magnitude for graph similarity search (Section VII-B).
2) *Are the existing index-based filtering techniques such as* Pars *useful for graph similarity search?*
   We show in **Eval-II** that for our index-free algorithm AStar$^+$-LSa, the potential improvement brought by the filtering technique of Pars is at most 52% (Section VII-B).
3) *How does* AStar$^+$-LSa *perform compared to* CSI_GED *for exact GED computation?*
   We show in **Eval-VI** that AStar$^+$-LSa outperforms CSI_GED by up-to four orders of magnitude for GED computation (Section VII-D).
4) *Is best-first search suitable for GED verification and/or computation?*
   We show in **Eval-IV** and **Eval-VII** that AStar$^+$-LSa outperforms its depth-first search variant DFS$^+$-LSa for both GED verification and GED computation.

A. Experimental Setting

We compare the following algorithms.

- **Our Algorithms.** We implemented our best-first search algorithms AStar$^+$-LSa and AStar$^+$-LS, and our depth-first search algorithm DFS$^+$-LSa.
- **Existing Algorithms.** We obtained the binary code of CSI_GED from the authors of [8], the source code of Pars from the authors of [22], and the source code of Inves from https://github.com/JongikKim/Inves.

The algorithms are implemented in C/C++ and compiled with GNU GCC with the -O3 flag.

Real Graphs. We evaluate the algorithms on two sets of widely-used real graphs [8], [10], [22]: AIDS and PubChem. AIDS is an antivirus screen chemical compound dataset published by the Developmental Therapeutics Program at NCI/NIH[6], and contains 42,689 graphs. PubChem is a chemical compound dataset[7], and contains 23,903 graphs. Statistics of the two datasets are illustrated in Table IV, which shows the number $|\mathcal{D}|$ of graphs, the average number of vertices, the average number of edges, the number of distinct vertex labels (#vlabels), and the number of distinct edge labels (#elabels).

TABLE IV: Statistics of real graphs

Database $\mathcal{D}$	$\lvert\mathcal{D}\rvert$	Avg $\lvert V\rvert$	Avg $\lvert E\rvert$	#vlabels	#elabels
AIDS	42,689	25.6	27.5	66	3
PubChem	23,903	48.3	50.8	10	3

Synthetic Graphs. We also generate synthetic graphs by the graph generator GraphGen[8], to evaluate the scalability of our algorithm AStar$^+$-LSa. We generate 10 groups of random graphs G_R, with the number of vertices 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 of 20%, 5 distinct vertex labels, and 2 distinct edge labels, similar to that used in [8].

Evaluation Metrics. For each testing, we report the processing time of the algorithms. In addition, we also report the search space for our algorithms, which is defined as the number of invocations of Line 5 of Algorithm 2 and roughly estimates the space consumption of our AStar$^+$ algorithms. All experiments are conducted on a machine with an Intel Core-i7 3.20GHz CPU and 64GB main memory.

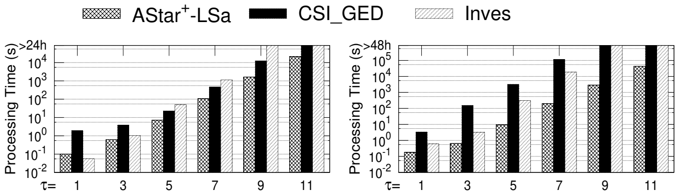
B. Results for Graph Similarity Search

Our first set of experiments is to evaluate the algorithms for graph similarity search. For both AIDS and PubChem, we randomly sample 100 graphs as query graphs. Most of the query graphs for AIDS have vertex numbers in the range from 10 to 46, with one query graph containing 59 vertices and another query graph containing 63 vertices. Most of the query graphs for PubChem have vertex numbers in the range from 27 to 65, with one query graph containing 80 vertices.

TABLE V: Number of candidates generated by label filter (LabelF) and Inves v.s. number of true results

	AIDS			PubChem		
τ	LabelF	Inves	Results	LabelF	Inves	Results
1	1,165	165	135	551	199	183
3	26,456	7,866	213	8,496	668	243
5	138,139	111,419	480	52,804	13,149	358
7	354,983	342,073	1,852	152,299	98,496	529
9	632,786	-	9,220	291,524	-	931
11	929,581	-	38,425	447,103	-	1,707

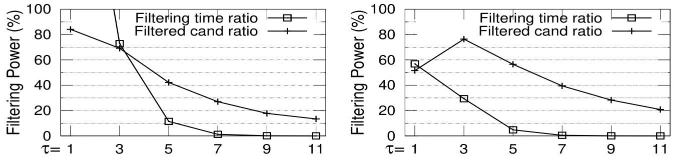
Same as Inves [10] and Pars [22], we also firstly apply label filter, denoted LabelF, to filter out unpromising graphs in linear time for our algorithms; note that, the running time of LabelF is included in the reported time for our algorithms. The number of candidate graphs obtained by LabelF and Inves, and the number of true results are shown in Table V; note that, Inves conducts additional index-free filtering and then verifies the remaining candidates. **All the reported results in this subsection are aggregates for** 100 **queries**.



(a) AIDS (b) PubChem

Fig. 4: Against CSI_GED and Inves for 100 queries

Eval-I: Against Existing Index-free Algorithms CSI_GED **and** Inves**.** The results of evaluating AStar$^+$-LSa against CSI_GED and Inves for $\tau = 1, 3, 5, 7, 9, 11$ are shown in Figure 4. Inves outperforms CSI_GED if the online graph partitioning of Inves filters a large portion of the candidate graphs (*e.g.*, on AIDS for $\tau = 1, 3$ and on PubChem for $\tau \leq 7$, see Table V), and is outperformed by CSI_GED otherwise. Nevertheless, AStar$^+$-LSa consistently outperforms CSI_GED and Inves, and the improvements can be up-to two orders of magnitude (*e.g.*, on PubChem for $\tau = 7$). We also observe that the online graph partitioning-based filtering of Inves already takes longer time than AStar$^+$-LSa on PubChem for $\tau \leq 5$.



(a) AIDS (b) PubChem

Fig. 5: Filtering effectiveness of Pars

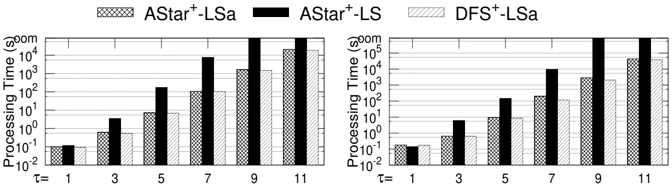
Eval-II: Against Existing Index-based Algorithm Pars**.** We evaluate the effectiveness of the index-based filtering of Pars compared with our index-free algorithm AStar$^+$-LSa. In Figure 5, we show the *filtering time ratio* r_F which is the ratio of the filtering time of Pars to the total running time of AStar$^+$-LSa, and the *filtered candidate ratio* r_C which is the ratio of the number of candidates filtered by Pars to the total number of candidates generated by LabelF. We can see that r_F is large for small τ (specifically, $r_F \geq 30\%$ for $\tau \leq 3$), and

[6] https://cactus.nci.nih.gov/download/nci/AID2DA99.sdz

[7] http://pubchem.ncbi.nlm.nih.gov: Compound_000975001_001000000.sdf

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

r_C is small for large τ (specifically, $r_C \leq 60\%$ for $\tau \geq 5$ and $r_C \leq 40\%$ for $\tau \geq 7$). Assume that all the candidates take equal time to verify, and let T be the running time of AStar$^+$-LSa. Then, the running time of Pars after incorporating AStar$^+$-LSa for GED verification will be $(r_F + 1 - r_C) \times T$, which is at least $0.69 \times T$ for AIDS and at least $0.48 \times T$ for PubChem as shown in Figure 5. Thus, the filtering effectiveness of Pars is very limited compared to our index-free algorithm AStar$^+$-LSa. Note that, as shown in [20], [10], other index methods have higher r_F and lower r_C than Pars; thus, they are expected to have even lower filtering effectiveness than Pars. Regarding the total processing time, Pars is 4, 25, and 338 times slower than AStar$^+$-LSa on AIDS for $\tau = 1, 3, 5$, respectively, due to using the out-dated algorithm A*GED for GED verification.



(a) AIDS (b) PubChem

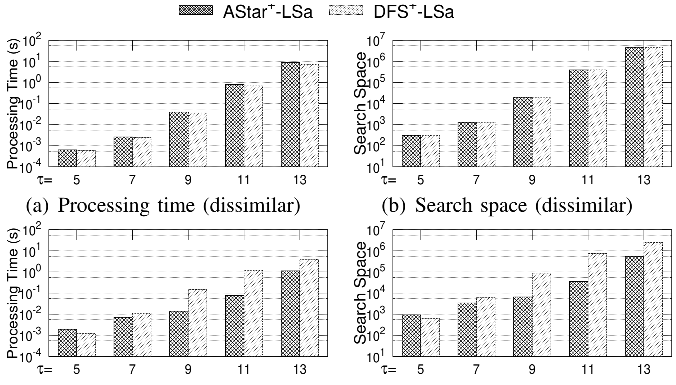
Fig. 6: Evaluate our algorithms for 100 queries

Eval-III: Evaluate Our Algorithms. The results of evaluating AStar$^+$-LSa, AStar$^+$-LS, and DFS$^+$-LSa are shown in Figure 6. Firstly, AStar$^+$-LSa performs similarly to DFS$^+$-LSa; this is because most candidates are dissimilar to the query graph (see Table V), and this conforms with our theoretical analysis in Section V-A. Thus, DFS$^+$-LSa also significantly outperforms CSI_GED and Inves for GED verification. Secondly, AStar$^+$-LSa runs much faster than AStar$^+$-LS as a result of the significantly reduced search space by the tighter lower bound estimation lb^{LSa}, and the latter runs out-of-memory (denoted oom) for $\tau \geq 9$. By comparing Figures 4 and 6, we can see that AStar$^+$-LS, which uses the same search strategy and the same lower bound estimation as A*GED, also significantly outperforms CSI_GED on PubChem for $\tau \leq 7$. This is because AStar$^+$-LS improves A*GED by reducing memory consumption (see Section IV-A) and improving the time complexity of lower bound computation (see Section IV-B2).

C. Results for GED Verification

In this subsection, we evaluate the best-first search paradigm with the depth-first search paradigm, and evaluate the scalability of AStar$^+$-LSa, for GED verification. To this end, we generate query graph pairs 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 all graph pairs among this set of graphs into different groups according to their GED values. Thus, each group consists of graph pairs of similar size and the same GED value. Finally, 10 graph pairs are randomly sampled and kept for each group. The reported processing time and search space are the average for each graph pair.

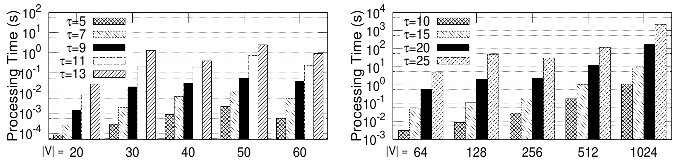
Eval-IV: Evaluate Best-first Search Against Depth-first Search. In this testing, we evaluate the paradigms of AStar$^+$ and DFS$^+$ for GED verification. Specifically, we evaluate AStar$^+$-LSa against DFS$^+$-LSa, which only differ by the search paradigm. Due to much fewer graph pairs than in graph similarity search (see Table V), we vary τ from 5 to 13. The results for dissimilar graph pairs are shown in Figure 7(a) and (b), where the query graph pairs are the ones in the group corresponding to $|V| = 30$ and $\tau \geq 14$. The two algorithms perform similarly by having exactly the same search space, which conforms with our theoretical analysis in Section V-A. The results for similar graph pairs are shown in Figure 7(c) and (d), where the query graph pairs for $\tau = x \in \{5, 7, 9, 11, 13\}$ are the ones in the union of the groups corresponding to $|V| = 30$ and $\tau \leq x$. AStar$^+$-LSa runs slightly faster than DFS$^+$-LSa by having a smaller search space. This invalidates the recent claims in [3], [8] that depth-first search is much better than best-first search for GED verification.



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

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

Fig. 7: Evaluate AStar$^+$ v.s. DFS$^+$ on PubChem



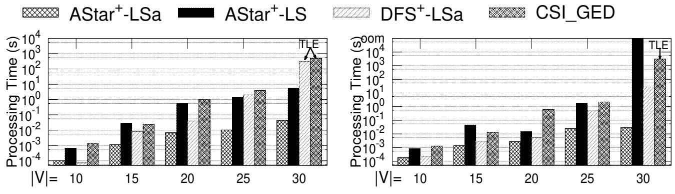
(a) AIDS (b) G$_R$

Fig. 8: Scalability testing of AStar$^+$-LSa

Eval-V: Scalability Testing of AStar$^+$-LSa. The results of evaluating the scalability of AStar$^+$-LSa for GED verification are shown in Figure 8. We choose τ from $\{5, 7, 9, 11, 13\}$ for AIDS and from $\{10, 15, 20, 25\}$ for G$_R$. For a fixed $|V|$, the query graph pairs are the same across different τ values, which are the union of the groups corresponding to these τ. We can see that AStar$^+$-LSa scales well to large graphs.

D. Results for GED Computation

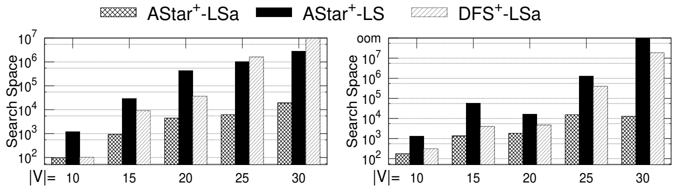
In this subsection, we evaluate our algorithms against the state-of-the-art algorithm CSI_GED for exact GED computation. Note that the existing algorithm Inves [10] does not support GED computation. The query graph pairs are generated in the same way as in Section VII-C. We fix $\mathsf{ged}(q, g) = 9$ and vary $|V|$ from 10 to 30. We set the largest $|V|$ as 30 because CSI_GED fails to process graphs with 30 or more vertices in a reasonable amount of time. Note that, exact GED computation is much harder than GED verification, as the threshold τ given in GED verification prunes a large portion of the search space, especially for depth-first search algorithms. For each query graph pair, we set a timeout of 1 hour (*i.e.*, 3.6×10^3 seconds); if an algorithm does not terminate in 1 hour, then we record the time for this query graph pair as 1 hour and label the algorithm with "TLE" in the plot. The reported results are the averaged result for each graph pair.



(a) AIDS (b) PubChem

Fig. 9: Processing time for GED computation

Eval-VI: Against Existing Algorithm CSI_GED. The running time of AStar$^+$-LSa, AStar$^+$-LS, DFS$^+$-LSa, and CSI_GED on AIDS and PubChem for GED computation by varying $|V|$ is shown in Figure 9. All our algorithms outperform CSI_GED. Firstly, our best-first search algorithm AStar$^+$-LSa outperforms CSI_GED by up-to four orders of magnitude. For example, the average processing time of AStar$^+$-LSa on the PubChem graphs with 30 vertices is less than 0.1 seconds, while CSI_GED cannot finish within 1 hour for most of the query graph pairs. Secondly, our best-first search algorithm AStar$^+$-LS that uses the same search strategy and the same lower bound estimation as A*GED also outperforms CSI_GED. Thirdly, our depth-first search algorithm DFS$^+$-LSa also outperforms CSI_GED, as a result of the tighter lower bound estimation $\mathsf{lb}^{\mathsf{LSa}}$ used in DFS$^+$-LSa.



(a) AIDS (b) PubChem

Fig. 10: Search space for GED computation

Eval-VII: Evaluate Our Algorithms. The search spaces of our algorithms for GED computation are shown in Figure 10, which have the same trend as the running time in Figure 9. Firstly, AStar$^+$-LSa runs much faster than AStar$^+$-LS due to the significantly reduced search space. Recall that, the only difference between these two algorithms lies in the lower bound estimation. Secondly, AStar$^+$-LSa runs much faster than DFS$^+$-LSa due to having a significantly smaller search space, which conforms with our theoretical analysis in Section V-B. Note that, the improvement of AStar$^+$-LSa over DFS$^+$-LSa for GED computation is much more profound than for GED verification in Section VII-C. This further invalidates the claims in [3], [8] that depth-first search is more suitable than best-first search for GED computation.

VIII. CONCLUSION

In this paper, we proposed a significantly improved best-first search algorithm AStar$^+$-LSa for both GED verification and GED computation. AStar$^+$-LSa improves the existing best-first search algorithm A*GED by reducing memory consumption, tightening lower bound estimation, and improving the time complexity for lower bound computation. We theoretically showed that AStar$^+$-LSa has a smaller time and space complexity than A*GED, and AStar$^+$-LSa is better than its depth-first search variant DFS$^+$-LSa which outperforms the existing depth-first search algorithms CSI_GED and DF_GED. Extensive performance studies confirmed the efficiency of AStar$^+$-LSa for both GED verification (*i.e.*, index-free graph similarity search) and GED computation.

REFERENCES

[1] Z. Abu-Aisheh, R. Raveaux, J. Ramel, and P. Martineau. An exact graph edit distance algorithm for solving pattern recognition problems. In *Proc. of ICPRAM'15*, pages 271–278, 2015.

[2] M. Bernard, N. Richard, and J. Paquereau. Functional brain imaging by eeg graph-matching. In *Proc. of EMB'06*, 2006.

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

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

[5] L. Chang, X. Feng, X. Lin, L. Qin, and W. Zhang. Efficient graph edit distance computation and verification via anchor-aware lower bound estimation. *CoRR*, abs/1709.06810, 2017.

[6] F. Chevalier, J. Domenger, J. Benois-Pineau, and M. Delest. Retrieval of objects in video by similarity based on graph matching. *Pattern Recognition Letters*, 2007.

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

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

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

[10] J. Kim, D. Choi, and C. Li. Inves: Incremental partitioning-based verification for graph similarity search. In *Proc. of EDBT'19*, 2019.

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

[12] M. Neuhaus and H. Bunke. Edit distance-based kernel functions for structural pattern classification. *Pattern Recognition*, 2006.

[13] H. Ogata, W. Fujibuchi, S. Goto, and M. Kanehisa. A heuristic graph comparison algorithm and its application to detect functionally related enzyme clusters. *Nucleic acids research*, 2000.

[14] K. Riesen, S. Emmenegger, and H. Bunke. A novel software toolkit for graph edit distance computation. In *Proc. of GbRPR'13*, 2013.

[15] K. Riesen, S. Fankhauser, and H. Bunke. Speeding up graph edit distance computation with a bipartite heuristic. In *Proc. of MLG'07*, 2007.

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

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

[18] X. Wang, X. Ding, A. K. H. Tung, S. Ying, and H. Jin. An efficient graph indexing method. In *Proc. of ICDE'12*, pages 210–221, 2012.

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

[20] X. Zhao, C. Xiao, X. Lin, Q. Liu, and W. Zhang. A partition-based approach to structure similarity search. *PVLDB*, 7(3):169–180, 2013.

[21] X. Zhao, C. Xiao, X. Lin, W. Wang, and Y. Ishikawa. Efficient processing of graph similarity queries with edit distance constraints. *VLDB J.*, 22(6):727–752, 2013.

[22] X. Zhao, C. Xiao, X. Lin, W. Zhang, and Y. Wang. Efficient structure similarity searches: a partition-based approach. *VLDB J.*, 27(1), 2018.

[23] W. Zheng, L. Zou, X. Lian, D. Wang, and D. Zhao. Efficient graph similarity search over large graph databases. *IEEE Trans. Knowl. Data Eng.*, 27(4):964–978, 2015.

[24] G. Zhu, X. Lin, K. Zhu, W. Zhang, and J. X. Yu. Treespan: efficiently computing similarity all-matching. In *Proc. of SIGMOD'12*, 2012.