2017 IEEE 33rd International Conference on Data Engineering

Similarity Search in Graph Databases: A Multi-layered Indexing Approach

Yongjiang Liang Peixiang Zhao
Department of Computer Science
Florida State University
Tallahassee, Florida 32306
liang@cs.fsu.edu zhao@cs.fsu.edu

***Abstract*—We consider in this paper the similarity search problem that retrieves relevant graphs from a graph database under the well-known graph edit distance (GED) constraint. Formally, given a graph database $\mathcal{G} = \{g_1, g_2, \ldots, g_n\}$ and a query graph q, we aim to search the graph $g_i \in \mathcal{G}$ such that the graph edit distance between g_i and q, GED(g_i, q), is within a user-specified GED threshold, τ. In spite of its theoretical significance and wide applicability, the GED-based similarity search problem is challenging in large graph databases due in particular to a large amount of GED computation incurred, which has proven to be NP-hard. In this paper, we propose a *parameterized*, partition-based GED lower bound that can be instantiated into a series of tight lower bounds towards synergistically pruning false-positive graphs from $\mathcal{G}$ before costly GED computation is performed. We design an efficient, *selectivity-aware* algorithm to partition graphs of $\mathcal{G}$ into highly selective subgraphs. They are further incorporated in a cost-effective, multi-layered indexing structure, ML-Index (Multi-Layered Index), for GED lower bound crosschecking and false-positive graph filtering with theoretical performance guarantees. Experimental studies in real and synthetic graph databases validate the efficiency and effectiveness of ML-Index, which achieves up to an order of magnitude speedup over the state-of-the-art method for similarity search in graph databases.**

I. Introduction

Today's highly networked world is facing numerous challenges raised in particular by the abundance of complex and interconnected data, which, without loss of generality, are typically modeled and interpreted as graphs [1], [11]. The proliferation of graphs has sparked a growing interest in enabling efficient access methods and flexible, structure-aware querying capabilities in large graph databases [18], [2], [16]. In order to account for noisy and distorted information arising unavoidably in real-world graphs, and to support rank-based exploration in graph-shaped data, it is essential and highly desirable to enable similarity search in graph databases, the goal of which is to retrieve relevant graphs given a user-specified, graph-structured query. Graph similarity search has found numerous real-world applications in business process management, pattern recognition, drug design, program analysis, and cheminformatics [34], [37], [36], [28], [27], [23].

There has been a rich literature in the modeling and computation of similarity between graphs, such as graph edit distances [15], [21], [32], maximum common subgraphs [23], [9], edge/feature misses [31], [38], [33], [30], graph alignment [25], and graph kernels [19]. In this paper, we consider the similarity search problem defined upon the *graph edit distance* (GED) constraint: given a graph database $\mathcal{G} = \{g_1, g_2, \ldots, g_n\}$, and a query graph q, we aim to find as output $g_i \in \mathcal{G}$ whose graph edit distance *w.r.t.* q, GED(g_i, q), is within a user-specified GED threshold, τ. The graph edit distance, GED(g_i, q), is the minimum number of *graph edit operations* that modify g_i step-by-step to q (or vise versa), and a graph edit operation can be vertex/edge insertion, deletion, or relabeling. Our choice of GED as the underlying graph similarity function is due primarily to its generality and broad applicability [15], [19]: First, GED is a metric applicable to virtually any type of graphs. The intuitive graph edit operations can precisely capture any fine-grained difference concerning both graph structures and contents [13]; Second, GED defines a theoretical framework for graph proximity modeling and quantification, within which many graph similarity measures, such as maximum common subgraphs [7] and edge misses [30], are just its special cases. Therefore, a systematic exploration of the GED-based similarity search problem becomes fundamental to real-world graph databases, and its solution will help address a family of graph similarity search problems that are defined upon other graph similarity constraints.

Unfortunately, the GED computation, GED(g, q), is NP-hard [14], rendering the similarity search problem hard especially in large graph databases. Existing solutions typically adopt a *filtering-verification* approach. In the filtering phase, some GED lower-bounds are employed to pre-prune false-positive graphs from $\mathcal{G}$, and the graphs surviving the filtering phase constitute a candidate set, $\mathcal{C}$. In the verification phase, costly GED computation is performed between q and each candidate graph $g_i \in \mathcal{C}$ in order to find the final answers. Therefore, the search performance is determined primarily by the tightness of GED lower bounds, together with the computational overhead for false-positive graph identification and filtering. Insofar, different GED lower bounds and false-positive pruning techniques have been proposed [15], [34], [37], [35], [28], [27], [32], which typically suffer from the following weaknesses: (1) Existing GED lower bounds have demonstrated limited filtering capabilities without theoretical performance guarantees. As a result, lots of false-positive graphs fail to be identified, thus incurring a large amount of fruitless GED computation; (2) The GED lower-bound evaluation is time-consuming, which imposes another performance bottleneck for similarity search in graph databases.

In this paper, we propose a new, multi-layered graph indexing approach, ML-Index (**M**ulti-**L**ayer **I**ndex), to efficiently addressing the similarity search problem in graph

2375-026X/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDE.2017.129

databases. We consider a parameterized (by a parameter k), partition-based GED lower bound that subsumes the existing best-known partition-based GED bound [34] as a degraded special case (when $k = 1$). More importantly, it will result in a series of instantiated, tighter GED lower bounds for effective false-positive graph identification and pruning. While evaluating these partition-based GED lower bounds, we need to partition each data graph $g_i \in \mathcal{G}$ into a set of variable-size, non-overlapping subgraphs that constitute the basic index features in our indexing solution. However, graph partitioning has proven to be hard [6], and a straightforward, random partitioning scheme oftentimes yields subgraphs with limited pruning capabilities. We thus design an efficient, *selectivity-aware* partitioning method that takes account of selectivity modelling of index features into a quality-aware graph partitioning scheme toward filtering false-positive graphs early and effectively from the graph database. The resultant partitioned subgraphs demonstrate good pruning capabilities, and thus are ideal index features for GED lower bound evaluation.

To ensure tight GED lower bounds with theoretical performance guarantees, we design a multi-layered indexing structure, ML-Index, each layer i of which comprises three key components: (1) $\underline{\text{GED}}_{k_i}$: an instantiated GED lower bound characterized by its parameter, k_i; (2) $\mathcal{P}_i$: a graph partitioning method (*e.g.*, selectivity-aware graph partitioning) that partitions data graphs of $\mathcal{G}$ into index features; (3) the resultant partitions to be used in the evaluation of the GED lower bound, $\underline{\text{GED}}_{k_i}$. Given a data graph $g_i \in \mathcal{G}$, it has to satisfy all the GED lower-bound constraints in different layers of ML-Index before being considered a candidate graph in $\mathcal{C}$. To this end, multiple GED lower bounds, as opposed to a single lower bound considered in previous methods, are jointly crosschecked for false-positive graph detection, and the probability of a false-positive graph not to be identified by ML-Index is exponentially small *w.r.t.* the number of layers of ML-Index. To the best of our knowledge, ML-Index is the first work for similarity search in graph databases with theoretically guaranteed performance. In addition, ML-Index is a generic, multi-layered indexing framework, upon which different GED lower-bounds, partitioning algorithms, and graph indexing mechanisms can be synergetically incorporated and optimized towards high-performance similarity search in large-scale graph databases. The main contributions of ML-Index are summarized as follows,

1) We propose a parameterized, partition-based GED lower bound that can be instantiated into a series of new lower bounds with improved filtering capabilities. Such instantiated lower bounds are further leveraged for collective false-positive graph identification and filtering in ML-Index (Section IV-A);
2) We design an efficient, selectivity-aware graph partitioning algorithm to generate highly selective index features for effective GED lower bound evaluation (Section IV-B);
3) We propose a cost-effective, multi-layered graph indexing approach, ML-Index, that enables false-positive graph filtering with theoretical guaranteed performance (Section IV-C);
4) We perform extensive experimental studies for ML-Index in comparison with the state-of-the-art method, Pars [34], in both real and synthetic graph databases. Experimental results validate the efficiency and effectiveness of ML-Index that achieves up to one order of magnitude speedup over Pars (Section VI).

The remainder of this paper is organized as follows. In Section II, we will brief the related work for similarity search in graph databases. The preliminary concepts and problem definitions will be formulated in Section III. We will discuss in detail our multi-layered indexing solution, ML-Index, in Section IV, and design the index-based similarity search algorithm in Section V. Experimental studies and results will be reported in Section VI, followed by concluding remarks in Section VII.

II. Related Work

The similarity search problem has drawn considerable attention in a wide range of practical, rank-based applications in graph databases [21], [39]. For example, in cheminformatics, molecules are recorded as graphs and compared in an inexact way for new material discovery and synthesis [30], [3]. In pattern recognition and computer vision, graphs representing hand-written symbols, fingerprints, or medical images are matched and retrieved approximately for identity discovery, object detection, and scene identification [10], [4]. In bioinformatics, graph similarity tools are devised for biological pathway enumeration and protein interaction detection [20].

Graph similarity search relies on some graph proximity function that models the similarity of graphs, and the most referenced one is graph edit distance (GED) due in particular to its generality and broad applicability [36], [37], [24], [27], [35], [32], [8], [7]. The GED computation is NP-hard [14], and the state-of-the-art GED method is based on an A* search strategy with bipartite matching heuristics [12]. However, it is only applicable to small-size graphs [15], and thus hard to employ in large-scale graph databases.

Previous studies on GED-based similarity search focused on pruning false-positive graphs before the costly GED computation is performed throughout the whole graph database. Inspired by the *q-gram* concept for string edit distance computation [26], k-AT [27] (k-**A**djacent **T**ree) decomposes each data graph $g \in \mathcal{G}$ into a multiset of *tree-based* q-grams, k-AT trees, for GED estimation. A k-AT tree is a subtree of g encompassing all vertices k-hops away from a given vertex v in g. A count-based GED lower bound is proposed by computing the minimum number of common k-AT trees from $g \in \mathcal{G}$ and the query graph q, respectively. This GED lower bound, however, is loose if g or q has high-degree vertices or the GED threshold, τ, is set large, thus making k-AT applicable only for very sparse graphs. In contrast, *path-based* q-grams, and the corresponding GED lower bounds, are designed to address the limitations of k-AT [35]. However, the exponential number of paths in a graph imposes a significant performance bottleneck for GED lower bound computation. Additionally, paths overlap with each other thus weakening the overall pruning capabilities. SEGOS [28], [32] and b-Tree [37] take advantage of *star structures* (1-hop k-AT trees) and *branch structures*, respectively, as q-grams. Both methods estimate the GED of two graphs based on the edit distance of their star/branch representations using the Hungarian algorithm [17]. Although these methods achieve tighter GED lower bounds than k-AT, the decomposed q-grams still overlap, thus

resulting in index features with limited pruning capabilities. Furthermore, the inability to handle large-degree vertices and large GED thresholds is inherited from k-AT.

To address the deficiencies of the aforementioned approaches, Pars [34] considers variable-size, non-overlapping graph partitions as q-grams for false-positive graph detection and pruning. The partition-based scheme in Pars is not prone to drastic changes of vertex degrees and GED threshold values, and more importantly, there is no feature overlap between partitioned q-grams. That is, any graph edit operation can affect *at most* one graph partition, thus breaking the worst-case assumption that graph edit operations can co-occur at the same region where numerous q-grams are generated. Known as the state-of-the-art solution for similarity search in graph databases, Pars has its own limitations as follows. First, the partition-based GED lower bound in Pars is still not tight, which may result in a huge amount of wasteful GED computation. In this paper, we consider a parameterized GED lower bound that can give rise to a series of instantiated, tighter GED lower bounds than the one in Pars. It turns out that the lower bound of Pars is just a special, degraded case of our parameterized GED lower bound. Second, Pars employs a random graph partitioning strategy for index feature generation. The resultant partitions, however, are of limited selectivity for false-positive graph identification. Although a sophisticated partitioning refinement is designed to regain quality partitions, it relies on the availability of query workload information, and inevitably incurs a large amount of subgraph isomorphism checking and index reorganization, which are time-demanding. In this paper, we consider an efficient, *selectivity-aware* graph partitioning method to generate high-selectivity index features that can help identify more false-positive graphs than Pars. Third, there are no theoretical performance guarantees for Pars and all the other existing similarity search methods. In this paper, we design a multi-layered indexing solution, ML-Index, that incorporates multiple GED lower-bounds for collective false-positive graph filtering with theoretical performance guarantees. To the best of our knowledge, ML-Index is the first work for similarity search with guaranteed performance in real-world, large graph databases.

III. Problem Formulation

Henceforth, we focus on simple, undirected, and labeled graphs, though the proposed approach can be extended to other types of graphs with minor revisions. A graph g is a 4-tuple (V_g, E_g, l_g, Σ), where V_g is a vertex set; $E_g \subseteq V_g \times V_g$ is an edge set; $l_g : V_g \cup E_g \rightarrow \Sigma$ is a labeling function, where Σ is the label set of vertices and edges (the subscript g in the notations will be omitted when the context is clear). In practice, the labels of a graph may represent vertex/edge attributes or contents, such as tags in XML documents, atoms and bonds in chemical compounds, gene ontology (GO) terms in biological networks, or object descriptors of images.

A graph g can be modified by *graph edit operations* including (1) inserting a new, isolated vertex u; (2) inserting a new edge $e = (u, v)$ between existing vertices u and v; (3) deleting an isolated vertex u; (4) deleting an edge $e = (u, v)$; (5) changing the label $l(u)$ of the vertex u; (6) changing the label $l(e)$ of the edge e. Given two graphs g and g', g can be modified step-by-step to g', or vice versa, by a finite sequence of graph edit operations, the *minimum* number of which is referred to as the *graph edit distance* (GED) between g and g', denoted as $\text{GED}(g, g')$. We remark that GED is a metric [13].

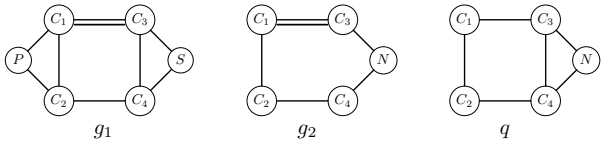


Fig. 1: Two graphs g_1, g_2, and a query graph q. Vertex labels represent atom symbols, and edge labels are either single-bond or double-bond. Subscripts of vertex labels differentiate vertices that share the same label.

Example 1. Figure 1 presents a tiny graph database $\mathcal{G}$ consisting of two graphs, g_1 and g_2, and a query graph q. The graph edit distance between q and g_1, $\text{GED}(q, g_1) = \text{GED}(g_1, q) = 5$, indicates 5 graph edit operations that modify q to g_1: inserting an isolated vertex P, inserting an edge (P, C_1), inserting an edge (P, C_2), relabeling the vertex label N to S, and relabeling the edge label of (C_1, C_3) from single-bond to double-bond. Similarly, $\text{GED}(q, g_2) = 2$. □

We define the problem of similarity search in a graph database, as follows,

Definition 1 (Similarity Search). Given a graph database $\mathcal{G} = \{g_1, g_2, \ldots, g_n\}$, a query graph q, and a GED threshold τ, the similarity search problem is to find as output all the data graphs $g_i \in \mathcal{G}$ such that $\text{GED}(g_i, q) \leq \tau$. □

Example 2. Consider the graph database $\mathcal{G}$ with two graphs g_1 and g_2, a query graph q, as shown in Figure 1, and the GED threshold $\tau = 2$. The graph g_2 is returned as a similar graph to q because $\text{GED}(q, g_2) = 2 \leq \tau$. □

The similarity search problem is NP-hard in that the computation of $\text{GED}(q, g_i)$ is NP-hard [14]. It is thus infeasible to perform pairwise GED computation throughout the whole graph database for similarity search. Instead, we consider a *filtering-verification* approach to addressing this problem. First, we employ some GED lower bound, denoted as $\underline{\text{GED}}(q, g_i)$, in the filtering phase. If $\underline{\text{GED}}(q, g_i) > \tau$, we have

$$\text{GED}(q, g_i) \geq \underline{\text{GED}}(q, g_i) > \tau,$$

so g_i is a false-positive graph, and can be filtered without costly GED verification. Furthermore, the graphs satisfying the GED lower-bound constraint in the filtering phase are put into a candidate set, $\mathcal{C}$, from which the exact GED verification is performed against q. To this end, we have to carry out a number $|\mathcal{C}|$ of exact GED computations to find the truly similar graphs of q. As a consequence, the key to the similarity search problem is to devise tight GED lower bounds and efficient filtering techniques in order to reduce the candidate set size, $|\mathcal{C}|$, which turns out to be the prime goal of our work.

IV. ML-Index

In this section, we detail our multi-layered indexing solution, ML-Index, for similarity search in graph databases. We first introduce a parameterized, partition-based GED lower bound, then discuss an efficient, selectivity-aware graph partitioning method in order to partition graphs into a series of

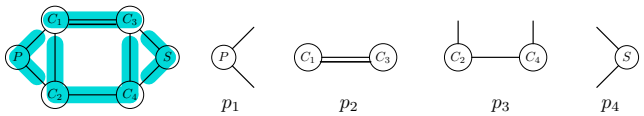


Fig. 2: The graph g_1 in Figure 1 is partitioned into four half-edge graphs: $\mathcal{P}(g_1) = \{p_1, p_2, p_3, p_4\}$.

high-selectivity subgraph partitions, which comprise the main index features of ML-Index for the evaluation of GED lower bounds. Finally, we design and implement ML-Index, upon which multiple partition-based GED lower bounds can be evaluated synergistically for false-positive graph identification and filtering with theoretical performance guarantees.

A. Partition-based GED Lower Bounds

We first define *half-edge* graphs [34] that model and represent graph partitions and also index features of ML-Index.

Definition 2 (Half-edge Graph). A half-edge graph $g = (V_g, E_g, l_g, \Sigma)$, where $E_g \subseteq V_g \times (V_g \cup \{*\})$, is a graph with possible existences of *half edges* $(u, *) \in E_g$, where one incident vertex $u \in V_g$ is a definite vertex, but the other vertex (and its label) of the half edge is not explicitly specified, represented as $*$. □

Definition 3 (Half-edge Subgraph Isomorphism). A half-edge graph g is a subgraph of another graph g', denoted as $g \subseteq g'$, if there exists an injective subgraph isomorphism function $f : V_g \rightarrow V_{g'}$, such that (1) $\forall u \in V_g, f(u) \in V_{g'}$ and $l_g(u) = l_{g'}(f(u))$; (2) $\forall (u, v) \in E_g, (f(u), f(v)) \in E_{g'}$ and $l_g((u, v)) = l_{g'}((f(u), f(v)))$; (3) $\forall (u, *) \in E_g, \exists w \in V_{g'} \backslash f(V_g)$ s.t. $(f(u), w) \in E_{g'}$ and $l_g((u, *)) = l_{g'}((f(u), w))$. □

If $g \subseteq g'$, g is a *half-edge subgraph* of g', or a subgraph of g' for brevity. Intuitively, if $g \subseteq g'$, we say g is *contained* in g', or g' *contains* g. We remark that half-edge subgraph isomorphism is NP-complete in that ordinary graphs without the special vertex $*$ can be regarded as a special case of half-edge graphs, and the subgraph isomorphism problem for ordinary graphs has proven to be NP-complete [14].

Definition 4 (Graph Partitioning). A graph g can be partitioned to a set $\mathcal{P}$ of *collective exhaustive*, *mutually exclusive*, and *non-empty* half-edge graphs as $\mathcal{P}(g) = \{p_i | \bigcup_i V_{p_i} = V_g, \bigcup_i E_{p_i} \subseteq E_g \cup V_g \times \{*\}, p_i \bigcap p_j = \emptyset, \forall i, j, i \neq j\}$. $\mathcal{P}$ is called a *partitioning* of g. □

Example 3. As shown in Figure 2, g_1 is partitioned to four half-edge graphs, p_1, p_2, p_3, and p_4. So $\mathcal{P} = \{p_1, p_2, p_3, p_4\}$ is one partitioning, among others, of g_1. □

Partitioning graphs into half-edge subgraphs has a clear advantage for GED lower bound estimation: given any graph edit operation, it can only affect at most one half-edge graph partition. As a result, we derive the following partition-based GED lower bound:

Theorem 1. Consider a graph g that is partitioned to a set $\mathcal{P}(g)$ of $(\tau + k)$ half-edge graphs, where τ is the GED threshold and k ($k \geq 1$) is an integer parameter. Given the query graph q, if $\text{GED}(g, q) \leq \tau$, there must exist at least k partitions of g, $p_{i_1}, \ldots, p_{i_k} \in \mathcal{P}(g)$, that satisfy $p_{i_l} \subseteq q$ ($1 \leq l \leq k$). □

Proof: Please refer to Appendix A. ■

Consider a data graph $g \in \mathcal{G}$ and a partitioning $\mathcal{P}(g) = \{p_1, \ldots, p_{\tau+k}\}$. If $p_i \subseteq q$, p_i is called a *matching* partition. Otherwise, p_i is a *mismatching* partition. According to Theorem 1, if the number of matching partitions of g *w.r.t.* q is less than k, the graph edit distance, $\text{GED}(g, q)$, must be larger than τ. Therefore, g is a false-positive graph and can be safely pruned without exact GED verification.

Example 4. Consider the graph database $\mathcal{G} = \{g_1, g_2\}$ and the query graph q as shown in Figure 1, and the GED threshold $\tau = 2$. We set $k = 2$ such that g_1 and g_2 are partitioned into $(\tau + k) = 4$ partitions, respectively. Specifically, the partitioning of g_1, $\mathcal{P}(g_1)$, is shown in Figure 2. Because there exist three mismatching partitions *w.r.t.* q: p_1, p_2, and p_4, g_1 is thus a false-positive graph. However, no matter what graph partitioning methods applied on g_2, we can always find at least $k = 2$ matching partitions. Therefore, g_2 is a candidate graph that further needs exact GED verification. □

Theorem 1 provides a parameterized, partition-based GED lower bound that can immediately generate a series of new GED lower bounds by setting k with different values. We remark that $1 \leq k \leq \min_{g \in \mathcal{G}}(|V_g| - \tau)$, and the newly generated GED lower bounds have varied filtering capabilities. When $k = 1$, the instantiated GED lower bound boils down to a special, degraded case [34]. Consider a graph $g \in \mathcal{G}$ which is a false positive *w.r.t.* the query graph q. It is easy to find one $(k = 1)$ matching partition given the $(\tau + 1)$ partitions of g. Once found, g will be treated as a candidate graph by mistake. When $k > 1$, however, g as a false-positive graph will more likely be identified and filtered, as detecting $k > 1$ matching partitions from g becomes less likely than in the case of $k = 1$. This is demonstrated in the following theorem,

Theorem 2. Consider a false-positive graph g ($\text{GED}(q, g) > \tau$) that is partitioned to $\mathcal{P} = \{p_1, \ldots, p_{\tau+1}\}$ and $\mathcal{P}' = \{p'_1, \ldots, p'_{\tau+k}\}$, $k > 1$, respectively. If we assume graph edit operations occur irrespective of g, the probability of the first k partitions of $\mathcal{P}'$ being matching partitions is smaller than the probability of the first one partition of $\mathcal{P}$ being a matching partition. □

Proof: Please refer to Appendix A. ■

If we partition graphs of $\mathcal{G}$ into $(\tau + k)$ partitions, where $k > 1$, it is more likely to identify false-positive graphs from $\mathcal{G}$ than the degraded case of $k = 1$. As a consequence, the generalized GED lower bound can be instantiated into a series of tigher lower bounds, when $k > 1$, with better filtering capabilities for similarity search.

B. Selectivity-aware Graph Partitioning

Given a data graph $g \in \mathcal{G}$, there exist a huge number of ways to partition g into $(\tau + k)$ partitions with varied sizes and structures [6]. If not designed carefully, a partitioning method (*e.g.*, random partitioning) may lead to graph partitions with limited filtering capabilities. Therefore, graph partitioning also plays a critical role in evaluating partitioned-based GED lower bounds for similarity search in graph databases.

Example 5. For the same problem setting as described in Example 4, if g_1 in Figure 1 is partitioned by a random

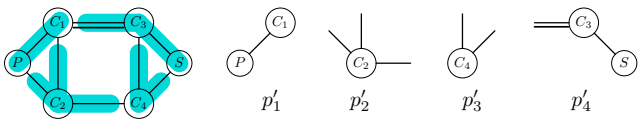


Fig. 3: The graph g_1 in Figure 1 is randomly partitioned into four half-edge graphs, $\mathcal{P}'(g) = \{p'_1, p'_2, p'_3, p'_4\}$, with limited selectivity.

partitioning method, $\mathcal{P}'$, into four half-edge graphs, p'_1, p'_2, p'_3, and p'_4, as shown in Figure 3, we note that both p'_2 and p'_3 are matching partitions *w.r.t.* q, because $p'_2 \subseteq q$ and $p'_3 \subseteq q$. Based on Theorem 1 ($k = 2$), g_1 is a candidate graph, although it is in fact a false-positive graph and should be filtered if the partitioning method $\mathcal{P}$ in Example 3 is employed. □

In order to devise partitioning algorithms that lead to partitions with good filtering capabilities, we define the notion of *selectivity* for half-edge graph partitions. Consider any partitioning scheme $\mathcal{P}$ that partitions a graph g into $(\tau + k)$ partitions, $\mathcal{P} = \{p_1, \ldots, p_{\tau+k}\}$. We design a *selectivity* function $s : p_i \in \mathcal{P} \rightarrow \mathbb{R}_+$ assigning for each partition p_i $(1 \leq i \leq \tau + k)$ a positive value, $s(p_i)$, indicating how *selective* p_i will be as a mismatching partition ($p_i \not\subseteq q$) if g is a false-positive graph. Intuitively, the higher the value of $s(p_i)$ is, the more probably p_i is a mismatching partition, and the false-positive graph g will more easily be identified and filtered from $\mathcal{G}$. To this end, the objective of a *selectivity-aware partitioning* $\mathcal{P}$ is to partition a graph g into $(\tau + k)$ partitions, which have the highest overall selectivity values. However, even for the simplest, balanced bi-partitioning case ($\mathcal{P} = \{p_1, p_2\}$), the number of possible partitionings is

$$\binom{|V_g|}{|V_g|/2} = \frac{|V_g|!}{((|V_g|/2)!)^2} \approx 2^{|V_g|}\sqrt{2/(\pi|V_g|)},$$

and finding the optimal selectivity-aware partitioning is NP-hard, which is polynomially reducible from the l-way graph partitioning problem [5], [14].

In the following, we design an efficient selectivity-aware graph partitioning method. We first model the selectivity of graph partitions based on the following heuristics:

1) **Partition size:** a larger-size partition p_i is more likely to be affected by graph edit operations, thus making p_i a mismatching partition. We thus consider the graph size, $(|V_{p_i}| + |E_{p_i}|)$, as one aspect of the graph selectivity;
2) **Vertex/Edge label frequency:** vertices/edges of p_i with small label frequencies in $\mathcal{G}$ may occur proportionally rarely in the query q. Therefore, a partition p_i containing low-frequency vertices/edges might be a mismatching partition, $p_i \subsetneq q$. We thus consider the *average* vertex/edge label frequencies as another aspect related to the graph selectivity.

To incorporate both factors, we define the selectivity, $s(p_i)$, of a partition p_i as

$$s(p_i) = \frac{|V_{p_i}| + |E_{p_i}|}{\sum_{v \in V_{p_i}} f(l_v)/|V_{p_i}| + \sum_{e \in E_{p_i}} f(l_e)/|E_{p_i}|} \quad (1)$$

where $f(\cdot)$ is the vertex/edge label frequency in $\mathcal{G}$. If there exists a half-edge $(v, *)$ in p_i, its edge label frequency is estimated as

$$f(l_{(v,*)}) = \frac{\sum_{u \in \mathcal{N}(v)} f(l_{(v,u)})}{|\mathcal{N}(v)|} \quad (2)$$

where $\mathcal{N}(v)$ is the set of neighboring vertices of v in $\mathcal{G}$.

Algorithm 1: Selectivity-aware Graph Partitioning

Input: a graph $g \in \mathcal{G}$, GED threshold τ, parameter k
Output: The partitioning $\mathcal{P}(g) = \{p_1, \ldots, p_{\tau+k}\}$

1 **begin**
2 A Boolean vector $B[\cdot] : V_g \rightarrow \{\texttt{true}, \texttt{false}\}$ where $\forall v \in V_g, B[v] \leftarrow \texttt{false}$;
3 $\Gamma \leftarrow \emptyset$;
4 **for** $i \leftarrow 1$ **to** $\tau + k$ **do**
5 Select $v \in V_g$ where $B(v) = \texttt{false}$, $p_i \leftarrow \{v\}$;
6 $B(v) \leftarrow \texttt{true}$;
7 **for** $u \in N(v), B[u] = \texttt{false}, u \notin \Gamma$ **do**
8 $\Gamma \leftarrow \Gamma \cup \{u\}$;
9 **while** $\exists v \in \Gamma$ **do**
10 **for** $i \leftarrow 1$ **to** $\tau + k$ **do**
11 $\Delta_i \leftarrow s(G[p_i \cup \{v\}]) - s(p_i)$;
12 $p_{i^*} \leftarrow G[p_{i^*} \cup \{v\}]$ where $i^* = \arg\max_i \Delta_i$;
13 $B(v) \leftarrow \texttt{true}$;
14 **for** $u \in N(v), B[u] = \texttt{false}, u \notin \Gamma$ **do**
15 $\Gamma \leftarrow \Gamma \cup \{u\}$;
16 **while** $\exists (u, v) \in E_g, u \in p_i, v \in p_j, i \neq j$ **do**
17 $\Delta_i \leftarrow s(p_i \cup (u, *)) - s(p_i)$;
18 $\Delta_j \leftarrow s(p_j \cup (v, *)) - s(p_j)$;
19 **if** $\Delta_i \geq \Delta_j$ **then**
20 $p_i \leftarrow p_i \cup \{(u, *)\}$;
21 **else**
22 $p_j \leftarrow p_j \cup \{(v, *)\}$;
23 **return** $\mathcal{P}(g) = \{p_1, \ldots, p_{\tau+k}\}$;

The selectivity-aware partitioning algorithm is presented in Algorithm 1, which partitions a graph $g \in \mathcal{G}$ into $(\tau + k)$ partitions (half-edge graphs). We first create a Boolean vector $B[\cdot]$ indicating for each vertex $v \in V_g$ whether v has been assigned to some partition, and $B[v]$ is initialized to `false` (Line 2). We maintain another set, Γ, holding the unassigned vertices of g that will be processed immediately (Line 3). Next, we choose $(\tau + k)$ vertices as the initial seeds, which will be expanded to the final $(\tau + k)$ partitions (Lines $4 - 6$). The neighboring vertices, $N(\cdot)$, of these seeds are added to Γ as they will be considered for partition assignment in the next step (Lines $7 - 8$). Henceforth, we examine each vertex $v \in \Gamma$ by evaluating the *selectivity gain* of assigning v to each existing partition p_i, denoted as

$$\Delta_i = s(G[p_i \cup \{v\}]) - s(p_i), \quad 1 \leq i \leq \tau + k \quad (3)$$

where $G[p_i \cup \{v\}]$ denotes the induced subgraph if v and all its induced edges (u, v), $u \in V_{p_i}$ are inserted to the partition p_i (Lines $10 - 11$). The vertex v will be assigned to the partition p_{i^*} which, with the addition of v (and its induced edges), has the largest selectivity gain, Δ_{i^*} (Line 12). After v has been assigned to some partition, all its unassigned neighboring vertices are added to Γ for further inspection

(Lines $14-15$). When all vertices of g have been properly assigned to the $(\tau+k)$ partitions, we then consider the edges straddling different partitions and assign them as half-edges to one of the participant partitions. The principle of assignment is similar: for the edge (u,v) where $u \in V_{p_i}, v \in V_{p_j}, i \neq j$, we assign it to the partition that leads to a larger selectivity gain (Lines $16-22$).

We remark that the vertex/edge label frequencies can be pre-computed by scanning the graph database $\mathcal{G}$ once during the index construction phase. Therefore, the time complexity of Algorithm 1 is $O((\tau+k)|V_g|+|E_g|)$, or simply $O(|V_g|+|E_g|)$ considering $(\tau+k)$ is a small value in similarity search.

C. The Multi-layered Index Structure

In order to filter false positive graphs from the graph database $\mathcal{G}$, we build an index structure upon which partition-based GED lower bounds can be evaluated in an efficient and cost-effective way. For each data graph $g \in \mathcal{G}$, we partition it into $(\tau+k)$ half-edge subgraphs, which constitute the basic index features. In addition, for each partition p, we maintain the following crucial information:

1) **Inverted index:** If p is a partition of the graph $g_i \in \mathcal{G}$, we maintain for p an inverted index, $\mathcal{I}(p)$, containing the graph identifier i as $p \subseteq g_i$. Given a query graph q, if $p \subseteq q$, we can quickly locate all the graphs from $\mathcal{I}(p)$, each of which has p as a matching partition. Furthermore, by exploring all the indexed partitions, we can find graphs of $\mathcal{G}$ that contain *at least* k matching partitions, which constitute the candidate set $\mathcal{C}$ for exact GED verification;
2) **Graph profile:** A common operation we have to perform frequently is to examine if $p \subseteq q$ holds as a matching partition, or $p \nsubseteq q$ as a mismatching pattern otherwise. This half-edge subgraph isomorphism testing is time-consuming in practice. We therefore construct a *graph profile*, $\mathcal{R}(p)$, for the partition p to facilitate this computation. $\mathcal{R}(p)$ maintains the vertex/edge label frequencies of p in a space-efficient histogram. Before examining $p \subseteq q$, their graph profiles, $\mathcal{R}(p)$ and $\mathcal{R}(q)$, are first compared bucket-wise: For each vertex/edge label in $\mathcal{R}(p)$, its frequency should be no more than that of the corresponding vertex/edge label in $\mathcal{R}(q)$, denoted as $\mathcal{R}(p) \preceq \mathcal{R}(q)$. Otherwise, we immediately know that $p \nsubseteq q$, and the costly half-edge subgraph isomorphism computation can be saved.

It is possible two partitions are graph isomorphic with each other. We thus use the canonical DFS code [29] as a unique representation of graph partitions to ensure that two isomorphic partitions will share one index entry represented by their canonical DFS code.

The aforementioned index structure is a conventional, one-layer graph index designed for similarity search. However, it promises limited filtering capabilities due to the following reasons. First, only a single GED lower bound is adopted for false-positive graph filtering. Although we can choose a tight, partition-based GED lower bound (when $k > 1$) and take advantage of high-selectivity index features generated by the selectivity-aware graph partitioning method, there are still many false-positive graphs unidentified from the graph database $\mathcal{G}$. Second, and more importantly, there are no theoretical tightness guarantees for GED lower bounds, which typically lead to unstable, and sometimes poor, similarity search performance in real-world graph databases.

Algorithm 2: ML-Index Construction

Input: Graph database $\mathcal{G}$
Output: The multi-layered index ML-Index

```
1  begin
2      for i ← 1 to L do
3          A hash structure, H_i : p → (I(p), R(p)), is
             initialized as ∅;
4      foreach g ∈ G do
5          for i ← 1 to L do
6              P_i(g) = {p_1^i, ..., p_{τ+k_i}^i};
7              for l ← 1 to τ + k_i do
                   /* Inverted index */
8                  I(p_l^i) ← I(p_l^i) ∪ {g};
                   /* Graph profile */
9                  R(p_l^i) ← a histogram of vertex/edge
                     label frequencies for p_l^i;
10                 H_i(p_l^i) = (I(p_l^i), R(p_l^i));
11     return ML-Index {H_1, ..., H_L};
```

In order to take advantage of multiple GED lower bounds and exert a collective filtering strategy, we design a multi-layered graph indexing structure, ML-Index, to enhance filtering capabilities with theoretical performance guarantees. In ML-Index, we consider L different graph partitioning methods, $\mathcal{P}_1, \ldots, \mathcal{P}_L$, with each $\mathcal{P}_i$ partitioning $g \in \mathcal{G}$ into $(\tau+k_i)$ partitions, $\mathcal{P}_i(g) = \{p_1^i, \ldots, p_{\tau+k_i}^i\}$. To this end, the partitioning method $\mathcal{P}_i$, the instantiated GED lower bound (parameterized by k_i), and the resultant graph partitions (together with their associated inverted indexes and graph profiles) constitute the ith layer of ML-Index. Namely, ML-Index consists of L layers of indexes, with each layer being responsible for the evaluation of the ith GED lower-bound. Given a query graph q, we examine ML-Index layer-by-layer. Specifically, at the ith layer, we evaluate the ith GED lower bound parameterized by k_i, and generate a candidate set $\mathcal{C}_i$. A data graph $g \in \mathcal{G}$ is a candidate graph at the ith layer if and only if there exist at least k_i matching partitions from g:

$$\mathcal{C}_i = \{g | \exists\ p_{l_1}^i \ldots, p_{l_{k_i}}^i \subseteq g \text{ and } q, \quad g \in \mathcal{G}\} \tag{4}$$

where $p_{l_*}^i$ are the partitioned index features at the ith layer of ML-Index. As a result, the final candidate set $\mathcal{C}$ after all L GED lower bounds of ML-Index have been evaluated is

$$\mathcal{C} = \bigcap_{i=1}^{L} \mathcal{C}_i \tag{5}$$

Given a graph $g \in \mathcal{G}$, if it fails in the evaluation of a GED lower-bound at any layer of ML-Index, it must be a false-positive graph, and can be safely filtered without exact GED verification.

Algorithm 2 presents the index construction process for ML-Index with L layers of partition-based indexes. Each layer of ML-Index is a hash structure, $\mathcal{H}_i (1 \leq i \leq L)$, that maintains

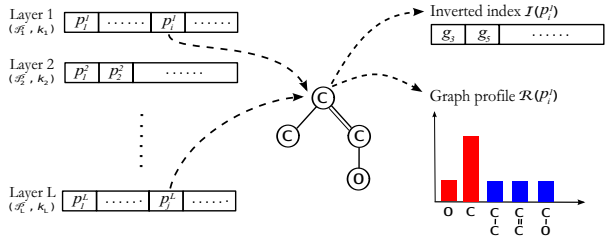


Fig. 4: The multi-layered index structure of ML-Index. Each layer i is featured a partitioning strategy $\mathcal{P}_i$, the partitioned index features $\{p_1^i, p_2^i, \ldots, \}$, and an instantiated GED lower bound parameterized by k_i. Each index feature p_j^i is associated with an inverted index $\mathcal{I}(p_j^i)$ and a graph profile $\mathcal{R}(p_j^i)$.

the correlation between a partition p and the associated inverted index, $\mathcal{I}(p)$, and its graph profile, $\mathcal{R}(p)$ (Lines $2-3$). We start with a sequential scan of the graph database $\mathcal{G}$, and for each data graph $g \in \mathcal{G}$, we partition it using L different partitioning methods[1]. Each partitioning $\mathcal{P}_i$ yields $(\tau + k_i)$ partitions from g, which constitute the index features at the ith layer of ML-Index (Lines $4-6$). For each partition p_l^i $(1 \leq l \leq \tau + k_i)$, we further maintain its inverted index (Line 8) and its graph profile (Line 9), and associate them with p_l^i in the hash structure $\mathcal{H}_i$ (Line 10). We remark that different partitioning methods may result in identical graph partitions (in terms of half-edge graph isomorphism) at different layers of ML-Index, we maintain only one copy of the inverted index and the graph profile to save the index space. Figure 4 illustrates the schematic structure of ML-Index.

The time complexity of Algorithm 2 is $O(|\mathcal{G}| \times L \times (O(\mathcal{P}) + O(|V_g| + |E_g|)))$, where $O(\mathcal{P})$ is the average-time complexity of graph partitioning, and $O(|V_g| + |E_g|)$ is the time complexity of graph profile construction for all partitions of $g \in \mathcal{G}$. If the selectivity-aware graph partitioning (Section IV-B) is adopted, the time complexity of Algorithm 2 turns out to be $O(|\mathcal{G}| \times L \times (|V| + |E|))$, where $(|V| + |E|)$ is the average size of graphs in $\mathcal{G}$. The space complexity of ML-Index is $O(L \times |\mathcal{F}| \times (|\mathcal{G}| + |\Sigma|))$, where $|\mathcal{F}|$ is the average number of index partitions at each layer of ML-Index, and Σ is the label set of vertices and edges of $\mathcal{G}$.

Theorem 3. Consider a graph $g \in \mathcal{G}$, which is a false positive *w.r.t.* the query graph q, i.e., $\text{GED}(q, g) > \tau$. The probability of g being identified as a false positive by ML-Index gets exponentially large (approaching 1) *w.r.t.* the number L of independent GED lower bounds in ML-Index. □

Proof: Please refer to Appendix A. ■

Theorem 3 states that ML-Index is guaranteed to identify false-positive graphs from $\mathcal{G}$ *w.h.p.* by crosschecking multiple independent GED lower bounds. In order to secure the independency of the partition-based GED lower bounds in ML-Index, we consider the following strategies in index construction. First, we randomly select initial seeds from $g \in \mathcal{G}$ when applying the selectivity-aware partitioning method at different layers of ML-Index. Second, we choose different values of the parameter k_i at different layer i of ML-Index. This way, the index partitions and the GED lower-bound constraints will vary significantly across different layers of ML-Index. As a result, ML-Index has the theoretical tightness guarantee for the GED lower bound, which was not promised in the existing, state-of-the-art similarity search solutions.

[1]Note that we can apply the same selectivity-aware partitioning algorithm with different initial seeds (by random selection), and different values of the parameter k_i, thus resulting in L different sets of partitioned index features.

Algorithm 3: Similarity Search Algorithm

Input: Graph database $\mathcal{G}$, query graph q, GED threshold τ, ML-Index $\{\mathcal{H}_1, \ldots, \mathcal{H}_L\}$
Output: $\mathcal{O} = \{g | \text{GED}(g, q) \leq \tau, g \in \mathcal{G}\}$

```
1  begin
       /* Candidate Generation */
2      Create an array A that maintains for each graph
         g ∈ G the number of matching partitions w.r.t. q;
3      for i ← 1 to L do
4          foreach g ∈ G do
5              A[g] ← 0;
6          for p ∈ H_i do
7              if R(p) ⪯ R(q) and (p ⊆ q) then
8                  foreach g ∈ I(p) do
9                      A[g] ← A[g] + 1;
10         C_i ← ∅;
11         foreach g ∈ G do
12             if A[g] ≥ k_i then
13                 C_i ← C_i ∪ {g};
14     C ← ⋂_{i=1}^{L} C_i;
       /* GED Verification */
15     O ← ∅;
16     foreach g ∈ C do
17         if GED(g, q) ≤ τ then
18             O ← O ∪ {g};
19     return The result set O;
```

We also remark that ML-Index is a generalized graph indexing framework. The GED lower bounds in ML-Index are not confined to partitioned-based lower bounds, so any existing GED lower bound can be synergistically incorporated in ML-Index to enable a powerful, collective filtering strategy towards bringing a significant false-positive reduction for similarity search with theoretical performance guarantees.

V. Similarity Search Algorithm

After ML-Index is built from $\mathcal{G}$, we can use it to answer similarity search queries, as detailed in Algorithm 3. We first create an array $\mathcal{A}[\cdot]$ to maintain for each data graph $g \in \mathcal{G}$, the number of matching partitions *w.r.t.* the query q (Line 2), and it is initialized to 0 (Lines $4-5$). At the ith layer of ML-Index $(1 \leq i \leq L)$, we evaluate for each partitioned index feature p if $p \subseteq q$ is satisfied (the graph profile checking, $\mathcal{R}(p) \preceq \mathcal{R}(q)$, is performed first to short-circuit the costly half-edge subgraph isomorphism testing, $p \subseteq q$). If $p \subseteq q$ is true, p is a matching partition *w.r.t.* q, and it is also a half-edge subgraph of all the data graphs $g \in \mathcal{G}$ in p's inverted index, $\mathcal{I}(p)$. We thus increment $\mathcal{A}[g]$, accordingly (Lines $6-9$). Based on Theorem 1, the candidate set at the ith layer of ML-Index, $\mathcal{C}_i$, includes all the data graphs with no less than k_i matching partitions, where k_i is the instantiated GED lower-

bound parameter at the ith layer of ML-Index (Lines $10-13$). After this layer-by-layer evaluation, the final candidate set, $\mathcal{C}$, contains the data graphs satisfying all L disparate GED lower-bounds specified in ML-Index (Line 14). Finally, we adopt some exact GED computational method to achieve the final results, $\mathcal{O}$, from $\mathcal{C}$ (Lines $15-19$).

To examine the time complexity of Algorithm 3, we consider the following critical factors: (1) T_{iso} : the average time complexity of half-edge subgraph isomorphism from index feature p to the query q, $p \subseteq q$; (2) T_{ged} : the average time complexity of exact GED computation, GED(q, g), where $g \in \mathcal{G}$; and (3) T_o : all the other time consumed for initialization and set-based operations. As a result, the overall runtime cost of Algorithm 3 can be formulated as

$$\mathcal{T} = |\bigcup_{i=1}^{L} \mathcal{H}_i| * T_{iso} + |\bigcap_{i=1}^{L} \mathcal{C}_i| * T_{ged} + T_o. \quad (6)$$

Specifically, the first component, $|\bigcup_{i=1}^{L} \mathcal{H}_i| * T_{iso}$, represents the overall time to generate the candidate sets at different layers of ML-Index, while the second component, $|\bigcap_{i=1}^{L} \mathcal{C}_i| * T_{ged}$, is the overall time for GED verification. Former empirical studies have demonstrated that T_{iso} is typically three orders of magnitude less than T_{ged} in real-world graphs [34]. Therefore, the main computational bottleneck lies in the GED verification component, $|\bigcap_{i=1}^{L} \mathcal{C}_i| * T_{ged}$, and the key to enhancing similarity search performance is to reduce the candidate set size, $|\mathcal{C}| = |\bigcap_{i=1}^{L} \mathcal{C}_i|$, which is also the goal of ML-Index. We also note that by introducing the multi-layered index structure for ML-Index, we have to spend extra space for new index features and extra time for half-edge subgraph isomorphism testings. However, the significant gain in false-positive graph reductions has far outweighed such marginal cost, as reported in Section VI.

VI. Experiments

A. Graph Databases

We consider three publicly available graph databases to benchmark different similarity search methods. The details of graph databases are summarized as follows,

1) **AIDS:** this is an antivirus screen chemical compound database from the Developmental Therapeutics Program at NCI/NIH[2]. There are $42,687$ graph-structured chemical compounds with 25.6 vertices and 27.6 edges by average, and 62 vertex labels (like elements **C**, **O**, **N**, and **P**) and 3 edge labels in total;
2) **PROTEIN:** this is a protein graph database from the Protein Data Bank[3] containing 600 graphs with 32.6 vertices and 62.1 edges by average. There are 3 vertex labels and 5 edge labels in this database. The graphs are denser and less label-informative than those in the AIDS database;
3) **GRAPHGEN:** this is a synthetic graph generator that creates large collections of labeled graphs[4]. The generator is regulated by a series of parameters: the number of graphs, $|\mathcal{G}|$, the average size of each graph in terms of the number of edges, $|E|$, the number of unique vertex/edge labels, $|\Sigma|$, and the average density of graphs, defined as $d = 2|E|/|V|(|V|-1)$. If not specified explicitly, the default parameters are set as: $|\mathcal{G}| = 10K$, $|E| = 40$, $d = 0.1$, $|\Sigma| = 4/4$ denoting there are 4 distinct vertex labels and 4 distinct edge labels, respectively.

The query set Q is generated by randomly sampling 100 graphs from each graph database, respectively. That is, query graphs in Q have similar structure/label characteristics as the data graphs in the corresponding graph databases.

B. Experimental Setup

We carry out experimental studies for ML-Index in comparison with Pars [34], which outperformed other existing similarity search methods [37], [28], [35], [27], and has been by far the most efficient method. In particular, we consider the following algorithms in the experimental studies,

1) Pars[34]**:** the state-of-the-art graph indexing method adopting a single, degraded GED lower bound ($k = 1$) and random partitioning for index generation and similarity search;
2) Selectivity**:** a *single-layer* graph indexing method using the *generalized* GED lower bound with an instantiated parameter $k > 1$ (Theorem 1), and the selectivity-aware graph partitioning (Algorithm 1) for index generation and similarity search;
3) ML-Index-L**:** a *multi-layered* graph indexing method comprising L distinct layers, each of which represents a distinct partition-based GED lower bound parameterized by k_i, and a selectivity-aware partitioning method, $\mathcal{P}_i$, for index generation. We set $L = 4$ in real-world graph databases, and $L = 3$ in synthetic graph databases (When L is set with other values, we have witnessed similar experimental findings, which are omitted for brevity).

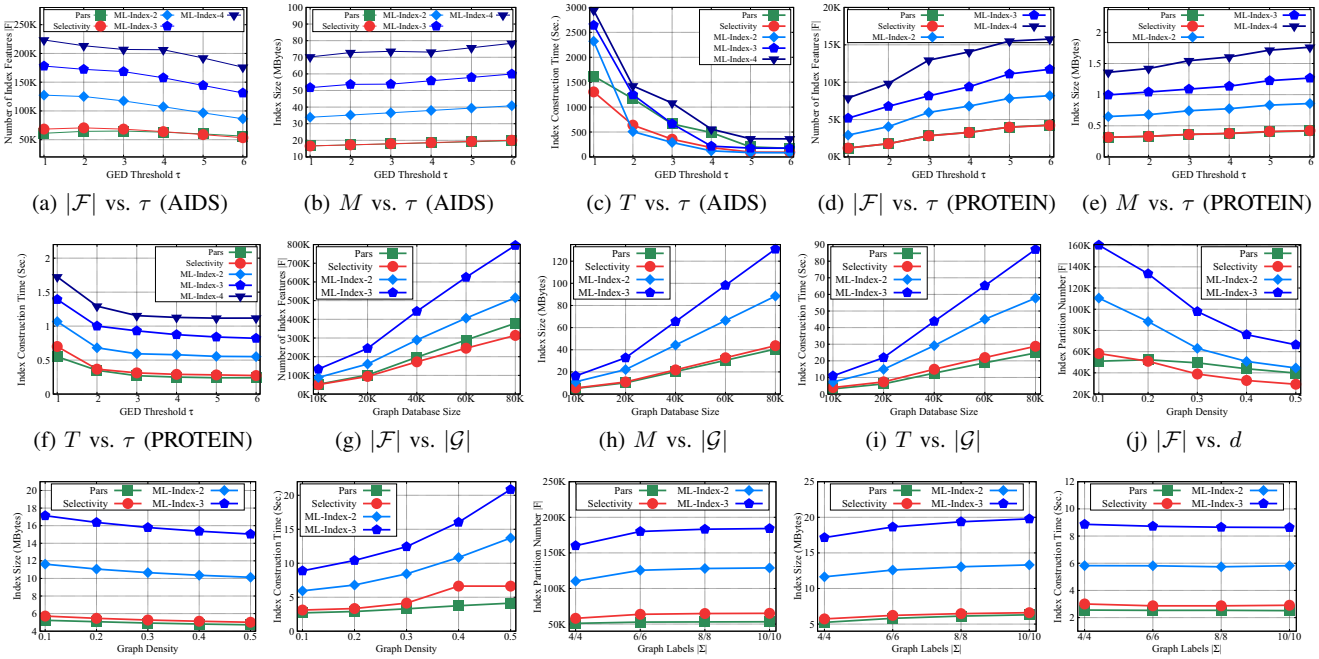
We consider the following performance evaluation metrics in the experimental studies: (1) **Index construction cost**, including the number $|\mathcal{F}|$ of indexed partitions, the in-memory index size M, and the index construction time T; (2) **Candidate set size**, $|\mathcal{C}|$, which is the most critical indicator of the similarity search performance. The results reported here are the *average* candidate set size for 100 queries in the query set Q; (3) **Query execution time**, consisting of the time for candidate generation and the time for exact GED verification, is the real response time for similarity search. Again, the time reported here is the *average* response time of 100 given queries in Q. In our experiments, we use the state-of-the-art method [22] for exact GED verification.

All our experiments were carried out on an Intel i7 3.20GHz quad-core PC with 8GB memory running Windows 7 operating system. The algorithms are implemented in C++ and compiled in Microsoft Visual Studio 2015.

C. Experiments for Index Construction

First of all, we evaluate the index construction cost of different methods in different graph databases. Note that all

[2] dtp.nci.hih.gov/docs/aids/aids_data.html
[3] www.iam.unibe.ch/fki/databases/iam-graph-database
[4] www.cse.ust.hk/graphgen



(a) $|\mathcal{F}|$ vs. τ (AIDS) (b) M vs. τ (AIDS) (c) T vs. τ (AIDS) (d) $|\mathcal{F}|$ vs. τ (PROTEIN) (e) M vs. τ (PROTEIN)

(f) T vs. τ (PROTEIN) (g) $|\mathcal{F}|$ vs. $|\mathcal{G}|$ (h) M vs. $|\mathcal{G}|$ (i) T vs. $|\mathcal{G}|$ (j) $|\mathcal{F}|$ vs. d

(k) M vs. d (l) T vs. d (m) $|\mathcal{F}|$ vs. $|\Sigma|$ (n) M vs. $|\Sigma|$ (o) T vs. $|\Sigma|$

Fig. 5: Index construction cost in terms of the number of index features, $|\mathcal{F}|$, the index size M (in megabytes), and the index construction time T (in seconds) for different similarity search methods in different graph databases.

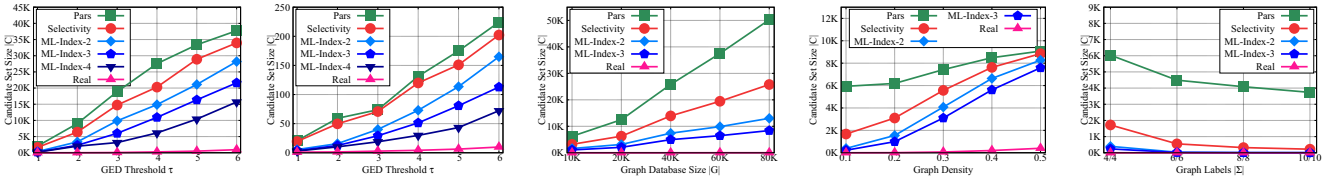
graph indexes are pre-built offline, given a graph database, $\mathcal{G}$, and we consider the GED threshold, τ, with practically small values ($\tau \leq 6$), as users are typically more inclined to search for similar graphs from graph databases.

We first examine the index construction cost in the AIDS graph database. The number $|\mathcal{F}|$ of index features, the index size M, and the index construction time T of different methods are illustrated in Figure 5(a), (b), and (c), respectively. In Figure 5(a), by varying the values of τ from 1 up to 6, we recognize that the numbers of index features, $|\mathcal{F}|$, of the one-layer indexing methods, Pars and Selectivity, are fairly stable and close with each other, while $|\mathcal{F}|$ of the multi-layered indexing approaches, ML-Index-2, ML-Index-3, and ML-Index-4, decrease steadily. This is because a growing number of selectivity-aware partitions in the multi-layered indexes turn out to be identical (in terms of half-edge subgraph isomorphism), and their inverted indexes and graph profiles can be reused across different layers of ML-Index, thus resulting in succinct index structures. This is further verified in terms of the index size, M (in megabytes), as shown in Figure 5(b). Even for the largest index, ML-Index-4, with four layers of partitioned index features, it only consumes less than 80 megabytes, which is very cost-effective, and can safely reside in memory. As to the index construction time T (in seconds) in Figure 5(c), we find that the multi-layered index ML-Index can be built efficiently. In particular, the index construction time T for multi-layered indexes is slightly greater than the construction time for single-layer indexing methods (T for ML-Index-4 is within 2.5x of T for Selectivity, given different values of τ). With the increase of τ, this gap of index construction time becomes marginal. This is mainly because when τ gets larger, each data graph $g \in \mathcal{G}$ is accordingly partitioned into a larger number $(\tau+k)$ of smaller-size partitions, thus leading to a speedup in half-edge subgraph isomorphism computation, which typcailly takes more than 90% of the total index construction time.

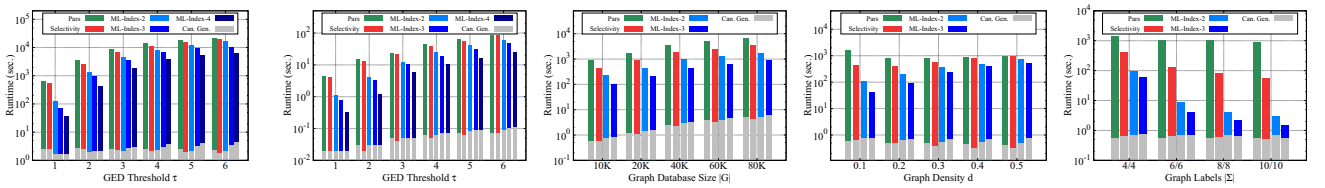
We then examine the index construction cost in the PROTEIN graph database, the graphs of which are dense and with few vertex/edge labels. The experimental results are illustrated in Figure 5(d)-(f). Here we witness similar trends and findings as the ones from the AIDS database, with one exception that there are much fewer numbers of identical index partitions shared by different layers of ML-Index, thus leading to a steady growth of the number of index features, $|\mathcal{F}|$, *w.r.t.* τ, as shown in Figure 5(d). However, the memory M consumed by different graph indexes is still very small (less than 2 megabytes even for ML-Index-4), as shown in Figure 5(e), and all indexes can be successfully constructed within 2 seconds, as shown in Figure 5(f).

We further evaluate the index construction cost on a series of synthetic graph databases generated by GRAPHGEN, and set $\tau = 5$ by default in the following experiments. Figure 5(g)-(i) present the scalability results of index construction for different methods. By varying the number of data graphs, $|\mathcal{G}|$, from 10K up to 80K, we recognize that the number of index features, $|\mathcal{F}|$, the index size M, and the index construction time T all grow linearly, exhibiting excellent scalability for all different graph indexing methods. For the largest graph database with 80K graphs, the size of ML-Index-3 is 131 megabytes, and it can be efficiently built within 90 seconds.

We then tune the graph density parameter, d, to generate graph databases with varied graph densities, and the experimental results for index construction are illustrated in Figure 5(j)-(l). When d increases from 0.1 to 0.5, the number of index features, $|\mathcal{F}|$, decreases steadily for Selectivity, ML-



(a) $|\mathcal{C}|$ vs. τ (AIDS) (b) $|\mathcal{C}|$ vs. τ (PROTEIN) (c) $|\mathcal{C}|$ vs. $|\mathcal{G}|$ (d) $|\mathcal{C}|$ vs. d (e) $|\mathcal{C}|$ vs. $|\Sigma|$



(f) Runtime (AIDS) (g) Runtime (PROTEIN) (h) Runtime vs. $|\mathcal{G}|$ (i) Runtime vs. d (j) Runtime vs. $|\Sigma|$

Fig. 6: The similarity search performance in terms of the candidate set size, $|\mathcal{C}|$, and the overall runtime, T.

Index-2, and ML-Index-3, because a growing number of selectivity-aware partitions turn out to be identical, which are graph partitions shared by dense graphs in the graph database. As a result, the index size M decreases steadily as well. However, the index construction time T increases as it typically takes more time to partition denser graphs.

Finally, we examine the index construction cost *w.r.t.* the label set size, $|\Sigma|$. We generate a series of graph databases with a varied number of vertex/edge labels, and construct graph indexes in these graph databases. The experimental results are presented in Figure 5(m)-(o). We note that when there are more distinct vertex/edge labels in the graph database, both $|\mathcal{F}|$ and M grows slightly larger, due primarily to a more diverse set of index features generated from the graph database. However, the index construction time T is very stable for all different graph indexing methods.

D. Experiments for Similarity Search

We then report our experimental studies for the similarity search performance of different methods in different graph databases. We consider the candidate set size, $|\mathcal{C}|$, as the principal indicator for similarity search performance. Furthermore, we also report the overall runtime concerning both candidate generation and exact GED verification, which is also the real response time for similarity search in graph databases.

Figure 6(a) illustrates the candidate set size, $|\mathcal{C}|$, *w.r.t.* the GED threshold, τ, for different methods in the AIDS database. We also include the size, $|\mathcal{O}|$, of the similarity search results to demonstrate the ultimate goal we strive to attain (colored in pink triangles). We have the following important findings in the experimental studies. First of all, the values of $|\mathcal{C}|$ for Selectivity (colored in red circle) are consistently smaller than those for Pars (colored in green square), with an average reduction of 30% false-positive graphs in the candidate set. The reason is twofold. First, the generalized GED lower bound with parameter $k > 1$ is tighter than the degraded one ($k = 1$). Second, the selectivity-aware partitioning method in Selectivity is more effective than random partitioning in Pars toward generating high-selectivity index partitions, which further help filter false-positive graphs during candidate generation. More importantly, we recognize that the multi-layered indexing approaches, including ML-Index-2, ML-Index-3, and ML-Index-4, have achieved significant false-positive graph reductions during candidate generation. With the increase of the number L of layers in ML-Index, we find $|\mathcal{C}|$ is guaranteed to reduce consistently, indicating that the multi-layered indexing method, ML-Index, has excellent filtering capabilities, and can bring guaranteed improvement for false-positive graph reduction in comparison to the single-layer graph indexing methods such as Pars and Selectivity. In particular, ML-Index-4 is the most effective method whose candidate sets, $\mathcal{C}$, are 2.5x to 10.7x smaller than the ones returned by the state-of-the-art method, Pars.

The same experiments are carried out in the PROTEIN database, and the results are presented in Figure 6(b). It can be witnessed that, given different values of τ, the multi-layered indexing methods, ML-Index-2, ML-Index-3, and ML-Index-4, achieve significant and consistent false-positive graph reductions, compared with the single-layer indexing methods. Specifically, the candidate sets, $\mathcal{C}$, of ML-Index-4 are 3.4x up to 17.6x smaller than the ones returned by Pars, leading to significant performance gains for similarity search given different values of τ.

We further evaluate the candidate set sizes, $|\mathcal{C}|$, in a series of synthetic graph databases. First of all, we examine the scalability results for similarity search by varying the number of graphs ranging from 10K up to 80K in the graph database $\mathcal{G}$, and report the candidate set sizes, $|\mathcal{C}|$, for different methods, as shown in Figure 6(c). We note that, with a growth of the number of graphs in $\mathcal{G}$, the number of candidate graphs, $|\mathcal{C}|$, returned by Pars increases significantly, which will incur a large amount of costly GED computation. However, when we introduce the multi-layered index structure in ML-Index, a significant portion of false-positive graphs are identified and filtered even in very large graph databases. In particular, ML-Index-4 achieves at least an order of magnitude improvement for false-positive graph reductions, compared with Pars, in graph databases of different sizes.

We then generate a series of graph databases ($|\mathcal{G}| = 10K$) with varied graph densities d ranging from 0.1 up to 0.5, and examine the similarity search performance in these graph databases. The experimental results are reported in Figure 6(d). We recognize that when graphs become dense, the numbers

$|\mathcal{C}|$ of candidate graphs returned by different similarity search methods turn out to increase significantly. The reason is that, given a dense graph $g \in \mathcal{G}$, some of its $(\tau+k)$ graph partitions become dense accordingly. As a result, the probability of more than one graph edit operations arising from a dense partition, p_i, increases as well. However, once identified as a mismatching partition, p_i is only counted as *one* mismatching partition in the GED lower bound evaluation, though there might be multiple graph edit operations co-occur within p_i. This phenomenon indicates that the detection of false positive graphs becomes difficult for dense graphs. However, even in this hard case, ML-Index-4 still outperforms Pars, meaning that it is beneficial to crosscheck multiple GED lower bounds to ensure effective false-positive reductions especially in the graph databases with many dense graphs.

We also examine how the number of vertex/edge labels, $|\Sigma|$, of a graph database is related to $|\mathcal{C}|$, and the results are presented in Figure 6(e). When $|\Sigma|$ increases, a significant reduction of $|\mathcal{C}|$ has been recognized for all different methods. Interestingly enough, when $|\Sigma| = 6/6$ (and larger), ML-Index-2 and ML-Index-3 can successfully identify and filter all false-positive graphs from $\mathcal{C}$. Namely, $\mathcal{C} = \mathcal{O}$. This indicates that the consideration of vertex/edge labels in the modeling and design of the selectivity-aware graph partitioning method turns out to be effective, thus leading to high-selectivity index partitions from graph databases, and significant reductions of false-positive graphs during similarity search.

We finally evaluate the overall runtime cost of all similarity search methods in different graph databases, and the results are illustrated in Figure 6(f)-(j) (Note that the experimental settings are the same as in the corresponding experimental studies for $|\mathcal{C}|$, respectively). The similarity search response time reported here comprise both the time for candidate generation (colored in grey), and the time for exact GED verification. We note that the time spent for candidate generation is at most several seconds, which is marginal in comparison with the time spent for exact GED verification. In the two real-world graph databases, AIDS (Figure 6(f)) and PROTEIN (Figure 6(g)), the multi-layered indexing method, ML-Index-4, achieves an order of magnitude improvement in the similarity search performance, compared with the state-of-the-art method, Pars, especially when the GED threshold, τ, is small ($\tau \leq 3$). Meanwhile, in a series of synthetic graph databases characterized by the graph database size $|\mathcal{G}|$ (Figure 6(h)), graph density d (Figure 6(i)), and the number of vertex/edge labels $|\Sigma|$ (Figure 6(j)), the search performance gap between ML-Index-3 and Pars can be as large as two orders of magnitude. This further verifies that our multi-layered graph indexing approach, ML-Index, is an efficient and high-performance similarity search method in real-world and synthetic graph databases under different experimental settings.

VII. Conclusion

The similarity search problem plays a fundamental and critical role in managing and querying graph-structured data, and has found widely varying applications in real-world large-scale graph databases. In this paper, we considered the similarity search problem that is defined on the graph edit distance (GED) constraint, and further proposed a new, multi-layered graph indexing solution, ML-Index, to address this challenging problem in graph databases. We employed a parameterized, partition-based GED lower bound for false-positive graph identification and filtering, and designed a selectivity-aware graph partitioning algorithm for high-quality index feature generation. We further incorporated and crosschecked multiple instantiated GED lower bounds in ML-Index such that false-positive graphs can be filtered with theoretical performance guarantees. The experimental studies on both real and synthetic graph databases have demonstrated that ML-Index is an efficient and cost-effective indexing method, which has significantly outperformed the state-of-the-art method, Pars, for similarity search in large-scale graph databases.

References

[1] C. C. Aggarwal and H. Wang. *Managing and Mining Graph Data*. Springer Inc., 2010.

[2] P. Barceló Baeza. Querying graph databases. In *Proceedings of the 32nd Symposium on Principles of Database Systems (PODS'13)*, pages 175–188, 2013.

[3] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The protein data bank. *Nucleic Acids Res*, 28:235–242, 2000.

[4] S. Berretti, A. Del Bimbo, and E. Vicario. Efficient matching and indexing of graph models in content-based retrieval. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(10):1089–1105, 2001.

[5] C.-E. Bichot and P. Siarry. *Graph Partitioning*. Wiley, 2011.

[6] A. Buluc, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz. Recent advances in graph partitioning. *ArXiv e-prints*, 2013.

[7] H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recogn. Lett.*, 18(9):689–694, 1997.

[8] H. Bunke. Error correcting graph matching: On the influence of the underlying cost function. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(9):917–922, 1999.

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

[10] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):265–298, 2004.

[11] D. J. Cook and L. B. Holder. *Mining Graph Data*. John Wiley & Sons, 2006.

[12] S. Fankhauser, K. Riesen, and H. Bunke. Speeding up graph edit distance computation through fast bipartite matching. In *Proceedings of the 8th International Conference on Graph-based Representations in Pattern Recognition (GBRPR'11)*, pages 102–111, 2011.

[13] X. Gao, B. Xiao, D. Tao, and X. Li. A survey of graph edit distance. *Pattern Anal. Appl.*, 13(1):113–129, 2010.

[14] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[15] K. Gouda and M. Arafa. An improved global lower bound for graph edit similarity search. *Pattern Recogn. Lett.*, 58:8–14, 2015.

[16] H. He and A. K. Singh. Graphs-at-a-time: query language and access methods for graph databases. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data (SIGMOD'08)*, pages 405–418, 2008.

[17] H. W. Kuhn and B. Yaw. The hungarian method for the assignment problem. *Naval Res. Logist. Quart*, pages 83–97, 1955.

[18] L. Libkin, W. Martens, and D. Vrgoč. Querying graphs with data. *J. ACM*, 63(2):14:1–14:53, 2016.

[19] M. Neuhaus and H. Bunke. *Bridging the Gap Between Graph Edit Distance and Kernel Machines*. World Scientific Publishing, 2007.

[20] H. Ogata, S. Goto, K. Sato, W. Fujibuchi, H. Bono, and M. Kanehisa. KEGG: kyoto encyclopedia of genes and genomes. *Nucleic Acids Research*, 27(1):29–34, 1999.

[21] S. Ranu, M. Hoang, and A. Singh. Answering top-k representative queries on graph databases. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD'14)*, pages 1163–1174, 2014.

[22] K. Riesen, S. Emmenegger, and H. Bunke. A novel software toolkit for graph edit distance computation. In *9th International Workshop on Graph-Based Representations in Pattern Recognition*, pages 142–151, 2013.

[23] H. Shang, X. Lin, Y. Zhang, J. X. Yu, and W. Wang. Connected substructure similarity search. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD'10)*, pages 903–914, 2010.

[24] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li. Efficient subgraph matching on billion node graphs. *Proc. VLDB Endow.*, 5(9):788–799, 2012.

[25] Y. Tian, R. C. Mceachin, C. Santos, D. J. States, and J. M. Patel. SAGA: A subgraph matching tool for biological graphs. *Bioinformatics*, 23(2):232–239, 2007.

[26] E. Ukkonen. Approximate string-matching with q-grams and maximal matches. *Theor. Comput. Sci.*, 92(1):191–211, 1992.

[27] G. Wang, B. Wang, X. Yang, and G. Yu. Efficiently indexing large sparse graphs for similarity search. *IEEE Trans. on Knowl. and Data Eng.*, 24(3):440–451, 2012.

[28] X. Wang, X. Ding, A. K. H. Tung, S. Ying, and H. Jin. An efficient graph indexing method. In *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering (ICDE'12)*, pages 210–221, 2012.

[29] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02)*, pages 721–724, 2002.

[30] X. Yan, P. S. Yu, and J. Han. Substructure similarity search in graph databases. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data (SIGMOD'05)*, pages 766–777, 2005.

[31] Y. Yuan, G. Wang, J. Y. Xu, and L. Chen. Efficient distributed subgraph similarity matching. *The VLDB Journal*, 24(3):369–394, 2015.

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

[33] S. Zhang, J. Yang, and W. Jin. SAPPER: Subgraph indexing and approximate matching in large graphs. *Proc. VLDB Endow.*, 3(1-2):1185–1194, 2010.

[34] 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.

[35] X. Zhao, C. Xiao, X. Lin, and W. Wang. Efficient graph similarity joins with edit distance constraints. In *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering (ICDE'12)*, pages 834–845, 2012.

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

[37] W. Zheng, L. Zou, X. Lian, D. Wang, and D. Zhao. Graph similarity search with edit distance constraint in large graph databases. In *Proceedings of the 22nd ACM International Conference on Conference on Information & Knowledge Management (CIKM'13)*, pages 1595–1600, 2013.

[38] G. Zhu, X. Lin, K. Zhu, W. Zhang, and J. X. Yu. TreeSpan: Efficiently computing similarity all-matching. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD'12)*, pages 529–540, 2012.

[39] Y. Zhu, L. Qin, J. X. Yu, and H. Cheng. Finding top-k similar graphs in graph databases. In *Proceedings of the 15th International Conference on Extending Database Technology (EDBT'12)*, pages 456–467, 2012.

APPENDIX

Proof of Theorem 1. We assume, by contradiction, that there are less than k partitions of g that are half-edge subgraph isomorphic to q. Namely, there are more than $(\tau+k)-k=\tau$ partitions that are not half-edge subgraph isomorphic to q. For each p_i of these τ partitions where $p_i \subsetneq q$, we need *at least* one graph edit operation to modify p_i in order to modify g to q as a consequence, thus amounting up to more than τ graph edit operations, which is in contradiction to the known fact that $\mathrm{GED}(g,q) \leq \tau$. □

Proof of Theorem 2. Based on the assumption that all graph edit operations occur irrespectively of g, we cast the GED problem into the classic *balls-and-bins* model, where graph edit operations correspond to $m=\mathrm{GED}(q,g)$ balls, and $(\tau+k)$ partitions correspond to $n=\tau+k$ bins. Such m balls will be placed into n bins, and a matching partition $p_i \subseteq q$ $(1 \leq i \leq \tau+k)$ corresponds to an empty bin because there is no graph edit operation occurring in p_i. Since any ball is placed into a bin with an equal probability of $1/n$, the probability of the first bin out of $(\tau+1)$ bins being empty is

$$(1-\frac{1}{n})^m = (\frac{\tau}{\tau+1})^{\mathrm{GED}(q,g)}$$

while the probability of the first k bins out of $(\tau+k)$ bins being empty is

$$(1-\frac{k}{n})^m = (\frac{\tau}{\tau+k})^{\mathrm{GED}(q,g)}$$

Because $\mathrm{GED}(q,g) > \tau \geq 0$, we remark that when $k>1$

$$(\frac{\tau}{\tau+k})^{\mathrm{GED}(q,g)} < (\frac{\tau}{\tau+1})^{\mathrm{GED}(q,g)}$$

□

Proof of Theorem 3. Given the false-positive graph g, we denote the probability of g being detected as a false positive by the i-th GED lower bound at the i-th layer $(1 \leq i \leq L)$ of ML-Index is $\Pr_i(g)$. So, the probability of g that fails to be detected and filtered at the i-th layer of ML-Index is $1-\Pr_i(g)$. Analogously, the probability of g satisfying all L GED lower-bounds at L different layers of ML-Index while still not being detected as a false positive is $\prod_{i=1}^{L}(1-\Pr_i(g))$. Therefore, the probability of g being identified as a false positive by ML-Index is

$$1-\prod_{i=1}^{L}(1-\Pr_i(g)). \tag{7}$$

Assume that ML-Index adopts the naive random partitioning method at all L different layers for graph partitioning and index generation, and we further assume $\Pr_i(g)=1/2$ for all $1 \leq i \leq L$. That is, the probability of whether g can be successfully identified as a false-positive graph, or not, at any layer of ML-Index is half-half. (Noth that the values of GED lower-bound parameters, k_i, can be different, thus resulting in different index partitions with varied filtering capabilities at different layers of ML-Index, and if we choose the selectivity-aware graph partitioning method, as opposed to the random partitioning, the probability $\Pr_i(g)$ can be significantly higher). Then, the probability of g being identified as a false positive graph by ML-Index is

$$1-\prod_{i=1}^{L}(1-\frac{1}{2}) = 1-\frac{1}{2^L},$$

which can be exponentially high and approaches 1, when the number L of index layers in ML-Index is set large. □