Authenticated Subgraph Similarity Search in Outsourced Graph Databases

Yun Peng, Zhe Fan, Byron Choi, Jianliang Xu, Sourav S Bhowmick

Abstract—*Subgraph similarity search* is used in graph databases to retrieve graphs whose subgraphs are similar to a given query graph. It has been proven successful in a wide range of applications including bioinformatics and chem-informatics, etc. Due to the cost of providing efficient similarity search services on ever-increasing graph data, database outsourcing is apparently an appealing solution to database owners. Unfortunately, query service providers may be untrusted or compromised by attacks. To our knowledge, no studies have been carried out on the *authentication* of the search. In this paper, we propose authentication techniques that follow the popular *filtering-and-verification* framework. We propose an authentication-friendly metric index called `GMTree`. Specifically, we transform the similarity search into a search in a graph metric space and derive small verification objects ($\mathcal{VO}$s) to-be-transmitted to query clients. To further optimize `GMTree`, we propose a sampling-based pivot selection method and an authenticated version of `MCS` computation. Our comprehensive experiments verified the effectiveness and efficiency of our proposed techniques.

Index Terms—Subgraph similarity search, query authentication, outsourced database.

1 Introduction

Graphs have been used widely to model complex data in many emerging applications, including proteins in biology, compounds in chemistry, attributed graphs in computer vision, ecology and web topology. In these real applications, subgraph similarity search (or simply *similarity search*) is a query frequently used as there may not be exact match for a user-specified search (*e.g.*, [1]–[9]). Similarity search can be formally described as follows. *Given a query graph q, a graph database $\mathcal{D}$ and a threshold (radius) t, retrieve graphs in $\mathcal{D}$ whose similarity distances from q are not greater than t.* For example, in chemistry, it is well-known that chemical structures discovered by the popular virtual screening method may contain laboratory errors. A compound being searched for may not match any compounds in the database. Hence, practical databases (*e.g.*, PubChem [10]) often return graphs that are similar to the query.

Similarity search is known to be an NP-hard problem. The owners of graph databases may lack the IT resources and expertises to provide efficient searches of their databases. For example, we issued a small query for a benzene structure to the prototype of a recent chemical database [11] and the query took 7.8 minutes. Such a performance may not be ideal for many applications. Further, graph data is growing explosively in volume. For instance, recent reports [10] indicate that from 2006 to 2012, PubChem's compound data increased from 57G bytes to 141G bytes. It would be inefficient to process such a large amount of data with a commodity machine.

For the reasons mentioned above, graph database outsourcing is appealing to database owners. Specifically, voluminous data is delegated to a powerful third-party service provider ($\mathcal{SP}$). The client may submit queries to the $\mathcal{SP}$ as if he/she is accessing a utility and the $\mathcal{SP}$ provides query processing services on the data owner's behalf. Data outsourcing has been adopted in many industry sectors. For instance, in drug engineering, many commercial $\mathcal{SP}$s [12]–[15] support outsourcing of pharma databases. Drug laboratories may then focus on the curation of their data.

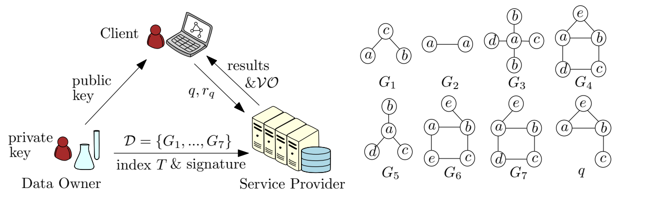


Fig. 1. An example of an outsourced graph database

Unfortunately, the service provider may be untrusted and/or compromised by attacks and clients may consequently receive tampered results. For instance, Fig. 1 shows an outsourced molecular graph database $\mathcal{D}$, a query molecule q and a distance threshold t 0.25. (The edge labels are omitted for brevity of presentation.) Assume that G_4, G_6 and G_7 are answers which should be returned as the query result. The service provider may deliberately return incorrect results (*e.g.*, G_3), distort t to 0.1, or return partial results (*e.g.*, only G_4). These significantly limit the practicality of graph database outsourcing. An *authentication mechanism* is thus necessary.

The majority of works on subgraph similarity search adopts a *filtering-and-verification* framework (*e.g.*, [1]–[6], [8], [9]), which consists of two key phases. First, in the *filtering* phase, indexes are proposed to prune (or filter) the data graphs that are certainly not the answer. The remaining graphs form a *candidate set* (*a superset of answers*). Second, in the *verification* phase, each candidate is checked by computing its distance from the query to *verify* if it is an answer. Despite the popularity of the framework, to the best of our knowledge, its authentication has not yet been studied and *this paper takes the first step toward an authenticated framework for the search.*

To facilitate the technical discussions, we briefly list the main steps of query authentication [16]: the data owner publishes its database, index and signature to an $\mathcal{SP}$. The

$\mathcal{SP}$ processes queries from a client and returns to the client both the query result and a verification object ($\mathcal{VO}$) which often encodes query processing traces such as index traversals. Using the query result and the $\mathcal{VO}$, the client constructs the digest of the database/index and compares it with the signature of the data owner to authenticate the query result.

As the filtering-and-verification framework is not specially designed for query authentication, we note that a naïve application of existing query authentication techniques leads to at least three problems. First, no previous index specifically considered whether the candidate graphs were located together in the graph database, which directly affects the $\mathcal{VO}$ needed. For instance, candidate and non-candidate graphs may be alternately stored in the database; and in this scenario, each candidate graph needs an item in the $\mathcal{VO}$ to authenticate that no candidate has been missed. As the number of candidate graphs for similarity search can be large, the $\mathcal{VO}$ for authenticating them can also be large. Second, one performance bottleneck at the client side is the distance computation on large candidate graphs, since the distance computation time is exponential to the graph size. Unfortunately, most existing approaches index similarity search by features or subgraphs (*e.g.*, [1], [2], [4], [5], [8], [9]). The larger the graph is, the more features/subgraphs are there for indexing. Thus, large graphs are often included in candidate graphs. Third, clients are required to perform the costly subgraph similarity computation numerous times in order to authenticate the processing traces at the $\mathcal{SP}$. Since such computation has already been done once at the $\mathcal{SP}$, it is inefficient for the client to redo it from scratch.

In this paper, we propose an authentication-friendly metric-based index, called *Graph Metric Tree* (GMTree), to address the aforementioned technical challenges. Its novelties mainly rely on the authentication techniques associated with GMTree. Specifically, for the first problem, we transform the subgraph similarity search into a search of a graph metric space and exploit the triangle inequality. Traditional metric indexes (such as [17]–[19]) can then be adapted to index graphs. GMTree is designed based on vp-tree [19], while other metric indexes can be adopted with minor modifications. As candidate graphs are often located together, our GMTree needs a notably smaller $\mathcal{VO}$ than a baseline derived from a previous work Grafil (denoted as Grafil*) as verified by our experiments. Moreover, subgraphs of data graphs can be pivots of GMTree. In contrast, the pivots of previous metric indexes were atomic data. We propose a pivot selection technique that exploits this property. Our experiments show that our pivot selection method reduces the query time and $\mathcal{VO}$ size by a factor of about 6.5 and 2, respectively. For the second problem, we derive an upper bound for pruning large non-answer graphs. We exploit the triangular inequality of a metric function and there is often a large graph distance between large non-answers and small queries. In particular, our experiments show that the largest 5% of GMTree's candidate graphs are 1.5 times smaller than those of Grafil* and the client's time spent on authenticating candidate graphs is thus reduced by up to 35%. For the third problem, we propose an authenticated version of the state-of-the-art MCS computation technique [20]. When the $\mathcal{SP}$ determines the MCSs between the query and indexed graphs, it records some hints in the $\mathcal{VO}$. The correctness of the hints can be authenticated by a scan on the hints. This significantly reduces the authentication time at the client. Our experiment shows that the $\mathcal{VO}$ overhead of our authenticated MCS computation is about 10K bytes but it reduces the authentication time at the client by about 50%.

The *main contributions* can be summarized as follows.

- We cast the subgraph similarity search into a similarity search in a graph metric space.
- We adopt a metric index to form GMTree to support efficient authentication. We propose the $\mathcal{VO}$ definition on GMTree, the $\mathcal{VO}$ construction and its authentication.
- We propose a sampling-based pivot selection method to optimize the GMTree construction.
- We develop an authenticated subgraph similarity computation method that significantly reduces the clients' authentication time.
- We conduct an extensive experimental evaluation of proposed techniques. Our experiments on a real dataset AIDS show that the authentication time of GMTree makes up less than 20% of its query time. The $\mathcal{VO}$ generated from GMTree is well controlled under 30K bytes, less than 3% of an authentication method extended from Grafil with basic authentication techniques.

Organization. The rest of the paper is organized as follows: Sec. 2 discusses the related work. Sec. 3 presents the background and problem statement. The metric based pruning is studied in Sec. 4. We present the GMTree index in Sec. 5 and its authentication techniques in Sec. 6. Sec. 7 details authenticated MCS computation and pivot selection techniques. Our experiments are presented in Sec. 8. Sec. 9 concludes the paper. All proofs are presented in the appendix.

2 Related Work

There have been many studies on authentication of various kinds of queries [21]–[26] but these queries are significantly different from subgraph similarity search. Despite recent interest in graph databases, the related work on its authentication is very limited. Yiu et al. [27] propose to authenticate shortest path queries on road networks. Goodrich et al. [28] study the authentication of path and connectivity queries on general graphs. Subgraph similarity search is different and more computationally costly than these queries. Moreover, the ordering of data objects in road networks can be precomputed offline by network-based distance, for example. Such an ordering is absent in graph data. Kundu and Bertino [29], [30] verify whether a given subgraph/subtree is in fact a subgraph/subtree of a given large data graph/tree without leakage of structural information. However, we study a large graph database instead of one large graph/tree. Martel et al. [31] propose a Search DAG (Directed Acyclic Graph) which is a generic model for authenticating a broad class of data structures. However, subgraph similarity search is more than a DAG search.

Another related topic is subgraph similarity search. He et al. [6] propose a CTree to index the hierarchical graph closure but CTree's heuristic method only supports approximate similarity search. Williams et al. [5] and Tian et al. [8]

propose graph decomposition methods to enumerate all unique and k-size subgraphs of data graphs, respectively; if adopted, they may lead to large $\mathcal{VO}$s. Yan et al. [1] propose a MCS-based similarity search method, which prunes non-candidate graphs by the count of structural features. Mongiovi et al. [9] extends it by incorporating with the identity of features for more filtering power. Jiang et al. [32] and Shang et al. [2] address the practical interests of connected subgraphs. Recently, Yuan et al. [4] study the MCS-based similarity search on probability graphs. Their similarity measures are based on maximum common *edge* subgraph (MCES). However, they are not a metric function. Zhu et al. [3] propose a MCES-based similarity metric measure. Their similarity value is dominated by the answer graph size *when the answer graph is much larger than the query graph*. In contrast, our similarity measure is defined to be relative to the query graph size. Further, we note that recent works adopt the maximum common *induced* subgraph based similarly, especially for biological and chemical applications [33]. There are some existing works on large graphs. Tian et al. [34] propose a neighborhood-based index, but they only support approximate search. Khan et al. [35] propose a novel neighborhood-based similarity measure to reduce isomorphism testing. However, they require all nodes of a query graph matched to a data graph. The similarity measure is not a metric either. There is another stream of works (*e.g.*, [6], [36]) on exact subgraph queries following the filtering-and-verification framework but there has been no work on its authentication. Moreover, similarity search is more flexible than exact queries.

The MCS computation problem is mainly studied for the maximum common induced subgraph (MCIS), as reported in [37]–[39]. Induced subgraphs are also widely used in recent subgraph similarity search works (*e.g.*, [2], [5], [34], [40]). We adopt MCIS for its metric properties [41].

For similarity search in metric spaces, several search trees are proposed to index the metric space (*e.g.*, [19] and [17]). However, these indexes are designed for neither indexing graphs nor their authentication. As noted in Sec. 1, several technical challenges must be addressed when they are adopted. Other works (*e.g.*, [42]) propose to cast metric spaces into low dimensional vector spaces and conduct similarity search there. However, it is not clear how graphs can be represented in vectors for similarity search and authentication.

3 Background and Problem Statement

This section first provides some background and problem statement and then presents the overview of our approach.

3.1 Backgrounds

3.1.1 Background for Querying Graphs

In this paper, we study *undirected labeled data graphs*, or simply *graphs* in the subsequent discussions. A graph is denoted as $G = (V, E, \Sigma, \lambda)$, where V is a set of nodes, $E : V \times V$ is a set of edges, Σ is a set of labels and λ is a function mapping a vertex or edge to a label. The size of G is defined as the number of vertices in G, denoted as $|G|$. Following a popular stream of works, we consider a *a graph database as a large collection of graphs having hundreds of nodes, which are common in chemical and biological databases.*

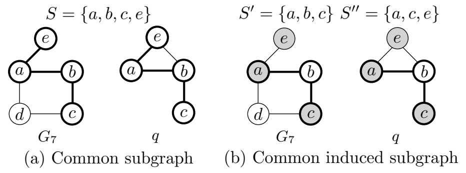

$S = \{a, b, c, e\}$
$S' = \{a, b, c\}$ $S'' = \{a, c, e\}$
e
a
b
d
c
G_7
q
(a) Common subgraph
(b) Common induced subgraph

Fig. 2. Common subgraph vs common induced subgraph

Isomorphism and common induced subgraph. Two graphs G and G' are isomorphic if there is a bijection f between V and V' such that for every vertex $u \in V$, $f(u) \in V'$ and $\lambda(u) = \lambda'(f(u))$, and for every edge $(u_1, u_2) \in E$, $(f(u_1), f(u_2)) \in E'$ and $\lambda(u_1, u_2) = \lambda'(f(u_1), f(u_2))$. A *common (induced) subgraph* between two graphs $G(V, E)$ and $G'(V', E')$ is an (induced) subgraph S of G that is isomorphic to an (induced) subgraph S' of G'. In general, the common subgraph and the common induced subgraph could be different. In this paper, the *maximum common induced subgraph* is referred to as the MCS between them, denoted as $mcs(G, G')$. Note that $mcs(G, G')$ is not necessarily unique for two given graphs G and G'. Moreover, $mcs(G, G')$ can be connected or disconnected.

Example 3.1: Consider the graph G_7 and the query q in our running example Fig. 1. Fig. 2(a) presents one common subgraph between G_7 and q, S=$\{a, b, c, e\}$ (highlighted by bold circles and lines). However, S is not an induced common subgraph because S is not an induced subgraph of q as $(e, b) \in q$. In comparison, two common induced subgraphs, connected S'=$\{a, b, c\}$ and disconnected S''=$\{a, c, e\}$ are presented in Fig. 2(b) (highlighted by bold circles and lines, and gray filling, respectively). Since G_7 and q have no common induced subgraph larger than 3, S' and S'' are two MCSs of them.

Graph similarity. There have been various similarity definitions of graphs in the literature. Among the existing definitions, only MCS-based graph similarity measure is a metric distance function. Specifically, the graph distance between a query graph q and a data graph G is defined as follows:

Definition 3.1: Given a query graph q and a data graph G, the *graph distance* between q and G is

$$d(q, G) = 1 - \frac{|mcs(q, G)|}{\max\{|q|, |G|\}}. \qquad (1)$$

This graph distance is a metric function [41], which satisfies the following properties, for any two graphs G and G',

- $d(G, G') \geq 0$, *(positiveness)*
- $d(G, G') = 0$, iff G and G' are isomorphic
- $d(G, G') = d(G', G)$ *(symmetry)*
- $d(G, G') + d(G', G'') \geq d(G, G'')$ *(triangle inequality)*

Subgraph similarity. Graph distance is related to the ratio between $|mcs(q, G)|$ and the larger one of $|q|$ and $|G|$. However, $|G|$ is often much larger than $|q|$. In this case, $d(q, G)$ produces a number close to 1, even when q itself is already a subgraph of G. Hence, users may often be more interested in the relative size of $mcs(q, G)$ with respect to q, *e.g.*, [1], [2]. Computing the relative size of $mcs(q, G)$ with respect to q is equivalent to comparing the similarity between q and G's subgraphs whose sizes are not larger than q. This is often referred to as *subgraph distance*, presented in Definition 3.2 .

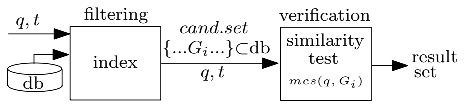


Fig. 3. Sketch of the filtering-and-verification framework

Definition 3.2: The *subgraph distance* between a query graph q and a data graph G is

$$d_s(q, G) = 1 - \frac{|mcs(q, G)|}{|q|}. \quad (2)$$

The number of missing vertices of q, $\sigma = |q| - |mcs(q, G)|$, can be readily derived from subgraph distance of q and G. Unfortunately, subgraph distance is *not* a metric function.

Example 3.2: We use Fig. 2(b) to show the difference between the graph distance and the subgraph distance. In Fig. 2(b), because $|mcs(q, G_7)| = 3$, so $d(q, G_7) = 1 - 3/5 = 2/5$ and $d_s(q, G_7) = 1 - 3/4 = 1/4$.

Definition 3.3: (Subgraph Similarity Search Problem). Given a graph database $\mathcal{D} = \{G_1, G_2, ..., G_n\}$, a query graph q and a threshold t, the *subgraph similarity search problem* is to retrieve all the graphs $G_i \in \mathcal{D}$, such that $d_s(q, G_i) \leq t$.

3.1.2 Background for Query Authentication

One-way hash function. A one-way hash function, denoted as $h(\cdot)$, is easy to determine a hash value $h(m)$ from a given pre-image m. It is hard to invert a given hash value of a random pre-image. Examples of such functions are `MD5` and `SHA`. In this paper, we often *use the term digest to refer to hash value.*

Public-key digital signature scheme. The scheme involves a public key and a private key. Only the data signer has a private key and can generate digital signatures of messages. The public key is known to everyone and the public may use it to verify the integrity of the signatures. For instance, `RSA` is a popular public-key digital signature scheme.

Merkle Hash Tree. The seminal work of Merkle Hash Tree (`MHT`) [43] has been adopted in many authentication works. `MHT` is a binary search tree and hash values are associated to its nodes. A leaf node ℓ contains data value dv and its hash $h(dv)$ is associated with ℓ. For an internal node, `MHT` associates the hash, denoted as $h(h_1||h_2)$, of the hash of its children h_1 and h_2, where "$||$" denotes concatenation. A data owner signs the hash of the root of `MHT`. In a nutshell, given a range query $[a, b]$, the query service provider transmits to clients (i) the answers in $[a, b]$ and (ii) the hash values of the index nodes at the "boundary" of the search of $[a, b]$ of `MHT`. The answers are complete only if the hash of the root computed by the client agrees with the signature provided by the data owner.

3.2 Problem Formulation

System Model. We assume the current state-of-the-art system model of database outsourcing [16]. The system model comprises three parties: the *data owner* $\mathcal{DO}$, the *service provider* $\mathcal{SP}$ and the *client*.

The $\mathcal{DO}$ maintains a database $\mathcal{D}$. To support subgraph similarity search on $\mathcal{D}$, the $\mathcal{DO}$ or the $\mathcal{SP}$ builds an *index* T of $\mathcal{D}$. The $\mathcal{DO}$ signs the root of T, and outsources $\mathcal{D}$, T and the signature to the $\mathcal{SP}$.

The $\mathcal{SP}$ answers queries of a client on $\mathcal{DO}$'s behalf and returns the *result set* $\mathcal{RS}$ back to the client. Since the $\mathcal{SP}$ may not be trusted, the $\mathcal{SP}$ is required to return the results $\mathcal{RS}$ together with their verification objects $\mathcal{VO}$s.

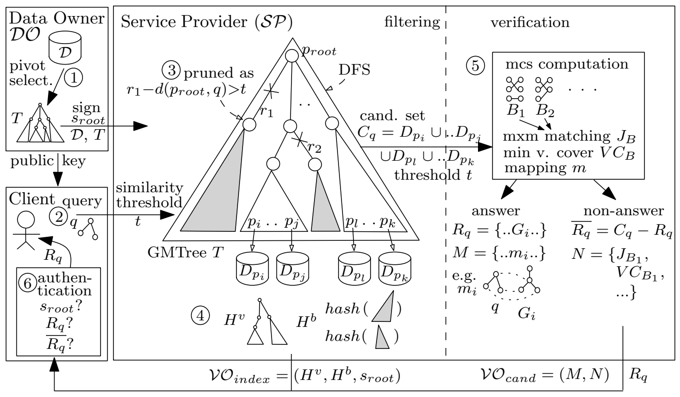


Fig. 4. An overview of our authentication method

The client uses $\mathcal{RS}$ and $\mathcal{VO}$ to synthesize the digest of the root of T. By using the public key of $\mathcal{DO}$, the client verifies whether the digest agrees with the signature provided by $\mathcal{DO}$. In addition, the clients are required to verify the following.

(i) Soundness: for $\forall G \in \mathcal{RS}$, G is an answer and $G \in \mathcal{D}$;

(ii) Completeness: for $\forall G \notin \mathcal{RS}$, G is not an answer.

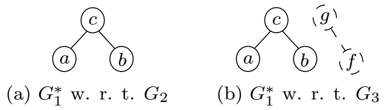
Threat Model. In our model, the $\mathcal{SP}$ is not always trustable. It may be a potential adversary or subverted by attackers. In either case, the $\mathcal{SP}$ may alter the data or the index, tamper with the similarity threshold, return partial answers, or abort the computation. We consider an authentication framework is *secure* if attacking such a framework is as hard as inverting a one-way hash function or breaking the public-key digital signature scheme.

Problem Statement. *Given the above system and thread models, we seek an efficient authentication mechanism where the client can issue a subgraph similarity search and verify the soundness and completeness of the results returned by a query service provider.*

3.3 Query Paradigm and Overview of Our Method

A popular subgraph similarity search paradigm is the *filtering-and-verification* framework [1]–[6], [8], [9]. As an example, Fig. 3 shows an overview of query processing of Grafil [1]. The query q and distance threshold t are first processed with the database in the filtering phase. Grafil filters non-answer graphs by its index and obtains a *candidate set* (*i.e.*, a superset of answers). In the verification phase, it computes the subgraph distance to q for each candidate graph. The candidates whose distances to q do not exceed t are query answers.

We design our authenticated subgraph similarity search technique by following the well-received filtering-and-verification framework. Fig. 4 gives an overview of our authenticated subgraph similarity search techniques. ① The data owner $\mathcal{DO}$ indexes the graph database $\mathcal{D}$ with the `GMTree` T (to be detailed in Sec. 5). The $\mathcal{DO}$ signs T and passes the database $\mathcal{D}$ and the index T to the service provider $\mathcal{SP}$ together with the signature s_{root}. ② The client issues a query graph q with a distance threshold t to the $\mathcal{SP}$. ③ In the filtering phase, the $\mathcal{SP}$ performs a traversal on the `GMTree`. Subtrees that do not contain any answers are pruned by using the condition derived from graph distance. The data graphs that are indexed by the remaining subtrees form the candidate set C_q. ④ For the sake of authentication, the $\mathcal{SP}$ introduces the



(a) G_1^* w. r. t. G_2 (b) G_1^* w. r. t. G_3

Fig. 5. Examples of augmented graphs

visited nodes into H^v and puts the digests for pruned subtrees in H^b where H^v, H^b and T's signature s_{root} form the $\mathcal{VO}$ *for the index* denoted as $\mathcal{VO}_{index}$. ⑤ In the verification phase, for each $G \in C_q$, the $\mathcal{SP}$ applies the authenticated MCS method to compute $d_s(q, G)$. If $d_s(q, G) \leq t$, G is an answer and the $\mathcal{SP}$ stores the mapping of $mcs(q, G)$ in a list M; Otherwise, G is not an answer and the $\mathcal{SP}$ adds the structures for computing MCS at the client into a list N. M and N comprise the $\mathcal{VO}$ *for the candidates* denoted as $\mathcal{VO}_{cand}$. ⑥ The query result $\mathcal{RS}$ and $\mathcal{VO}$ are returned and the client performs the authentication using the $\mathcal{VO}_{index}$ and the $\mathcal{VO}_{cand}$.

4 Metric Based Filtering

Among the studies that follow the filtering-and-verification framework, in this paper, we adopt the metric-based filtering approach as the metric properties can be exploited to address the authentication of subgraph similarity search.

Lemma 4.1: *Let q, G_1 and G_2 denote the query graph and two data graphs, respectively. Given a similarity threshold t, if $1 - \frac{|mcs(G_1,G_2)|}{|G_1|} - d(q, G_1) > t$, then $d_s(q, G_2) > t$.*

Lemma 4.1 states that subgraph similarity between a graph and a query can be expressed in terms of graph similarity and $1 - \frac{|mcs(G_1,G_2)|}{|G_1|}$. However, $1 - \frac{|mcs(G_1,G_2)|}{|G_1|}$ is not a metric function either. We address this by introducing a notion of *augmented graphs*.

Definition 4.1: An *augmented graph* G_1^* with respect to G_2 is defined as follows:

- $G_1^* = G_1$ if $|G_1| \geq |G_2|$;
- Otherwise, $G_1^* = G_1 \cup A$, where A is an augmented subgraph having nodes and edges with labels never occurred in G_2 and possible queries, until $|G_1^*| = |G_2|$.

Example 4.1: Consider the graphs G_1, G_2 and G_3 in Fig. 1. Fig. 5(a) shows the augmented graph G_1^* of G_1 w. r. t. G_2, where G_1^*=G_1 as $|G_1|$>$|G_2|$. Fig. 5(b) shows the augmented graph G_1^* of G_1 w. r. t. G_3, where G_1^*=$G_1 \cup A$ and A is an augmented subgraph of two nodes, denoted by the dashed circles and lines.

We can establish our pruning condition Theorem 4.2 by applying Lemma 4.1.

Theorem 4.2: *Given a query q, an augmented graph G_1^* and a graph G_2, if $d(G_1^*, G_2) - d(q, G_1^*) > t$, then $d_s(q, G_2) > t$.*

We remark that the augmented graph G_1^* is virtual, which does not need to be materialized in indexing, *i.e.*, original graph G_1 can still be used. To support the pruning condition of $d_s(q, G_2)$ by Theorem 4.2, $d(q, G_1^*)$ and $d(G_1^*, G_2)$ are needed. Regarding $d(q, G_1^*)$, we store $|G_1^*|$ in index, which can be easily computed by Definition 4.1 without materializing G_1^*. Since $mcs(G_1, q) = mcs(G_1^*, q)$, we can compute $d(q, G_1^*) = 1 - \frac{|mcs(G_1,q)|}{\max(|q|,|G_1^*|)}$ by computing $mcs(G_1, q)$ on-the-fly. Regarding $d(G_1^*, G_2)$, it could be indexed in our index.

Since we have transformed the similarity search into a search in graph metric space $(\mathcal{U}, d)$, where $\mathcal{U}$ denotes the graph database and d is the graph distance defined in Definition 3.1, one may attempt to directly adopt the traditional metric index to index the graph metric space. For example, vp-tree [19] partitions the metric space by using *pivots* with certain radii (akin to the MBRs of the RTree). An internal node of vp-tree contains *a pivot p with a radius r_p*. The pivot divides $(\mathcal{U}, d)$ into two subspaces: (i) the subspace *covered* by the pivot $\mathcal{U}_p = \{O \in \mathcal{U} \mid d(O, p) \leq r_p\}$; and (ii) the remaining subspace $\mathcal{U}_{\overline{p}} = \{O \in \mathcal{U} \mid d(O, p) > r_p\}$. The subspaces are recursively partitioned and indexed with subtrees. Pivots can be simply selected from data graphs. This is regarded as the baseline method and will be compared in our experiments.

5 Graph Metric Tree

As motivated in Sec. 1, existing techniques on similarity search are not designed for authentication and may encounter several problems when adopted. In this section, we propose an index called Graph Metric Tree (GMTree), which forms the basis of our authentication algorithm. This section focuses on similarity search and the details for authentication are presented in Sec 6.

5.1 GMTree Structure

GMTree is designed based on a variant of vp-tree [44], where a metric space is partitioned into a collection of non-overlapping "circular" subspaces with the same center.

Definition 5.1: Given a graph metric space $(\mathcal{U}, d)$, where $\mathcal{U} = \{G_1, ..., G_n\}$ and d is the graph distance (Definition 3.1), a *pivot* p is a graph, where $p \in \mathcal{D} \cup \mathcal{S}$ and $\mathcal{S} = \{S | S \subset G_i, G_i \in \mathcal{D}\}$. Given p, $\mathcal{U}$ is partitioned into c circular non-overlapping subspaces with radius $r_p^0, ..., r_p^{c-1}$ as follows:

- $\mathcal{U}^0$: $\{G | r_p^0 \leq d(p^*, G) < r_p^1, G \in \mathcal{U}\}$, where $r_p^0 = 0$;
- $\mathcal{U}^i$: $\{G | r_p^i \leq d(p^*, G) < r_p^{i+1}, G \in \mathcal{U}\}$, for $1 \leq i < c$-1; and
- $\mathcal{U}^{c-1}$: $\{G | d(p^*, G) \geq r_p^{c-1}, G \in \mathcal{U}\}$.

Definition 5.2: A GMTree T is a 4-ary tuple (V, E, r, c), where V, E, r and c are the nodes, the edges, the root and a user-specified fanout, respectively. The leaf nodes and the internal nodes are defined as below.

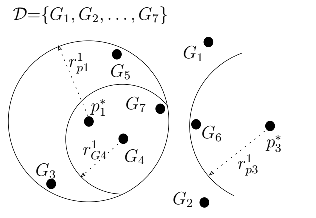
(i) A leaf node v_ℓ covering a metric space $\mathcal{U}_\ell$ is a tuple (g_1,...,g_n) of pointers, where $n \leq c$, g_i points to G_i and $G_i \in \mathcal{U}_\ell$.

(ii) An internal node v covering $\mathcal{U}$ is a 4-ary tuple (p, $|p^*|$, $\mathcal{T}_p$, $\mathcal{R}_p$), where p is the pivot graph and it is a (proper) subgraph of one of the graphs in $\mathcal{U}$, $|p^*|$ is the size of the largest graph in $\mathcal{U}$, $\mathcal{T}_p$ is a collection of sub-GMTrees, and $\mathcal{R}_p$ is a collection of radii that splits $\mathcal{U}$ into c roughly equally sized subspaces [44]. The sub-GMTrees index a set of circular subspaces defined as follows:

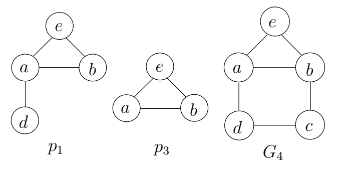
- T_p^0 covers the space $\mathcal{U}^0$: $\{G \mid r_p^0 \leq d(p^*, G) < r_p^1, G \in \mathcal{U}\}$, where $r_p^0 = 0$;
- $T_p^i \in \mathcal{T}_p$ covers the space $\mathcal{U}^i$: $\{G \mid r_p^i \leq d(p^*, G) < r_p^{i+1}, G \in \mathcal{U}\}$, for $1 \leq i < c - 1$; and
- T_p^{c-1} simply covers the remaining space: $\mathcal{U}^{c-1}$: $\{G | d(p^*, G) \geq r_p^{c-1}, G \in \mathcal{U}\}$

The root r is a special internal node, with no parent, which covers the entire graph metric space. For each leaf v_ℓ, we store a list $1 - \frac{|mcs(p,G_i)|}{|p|}$, where G_i's are the data graphs covered by v_ℓ and p is their pivot.

The fanout c in Definition 5.2 is used by data owners to balance the trade-off between authentication time and $\mathcal{VO}$ size.



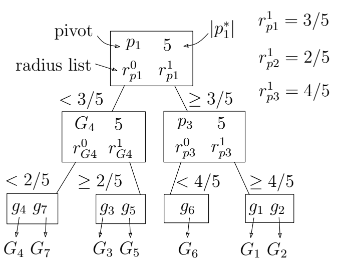
(a) Data graphs



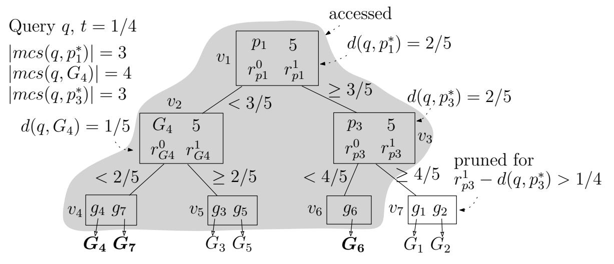
(b) Pivots

$\lvert mcs(p_1, G_1)\rvert = 1$	$\lvert mcs(G_4, G_3)\rvert = 3$
$\lvert mcs(p_1, G_2)\rvert = 1$	$\lvert mcs(G_4, G_5)\rvert = 3$
$\lvert mcs(p_1, G_3)\rvert = 3$	$\lvert mcs(G_4, G_4)\rvert = 5$
$\lvert mcs(p_1, G_4)\rvert = 4$	$\lvert mcs(G_4, G_7)\rvert = 4$
$\lvert mcs(p_1, G_5)\rvert = 3$	$\lvert mcs(p_3, G_1)\rvert = 1$
$\lvert mcs(p_1, G_6)\rvert = 2$	$\lvert mcs(p_3, G_2)\rvert = 1$
$\lvert mcs(p_1, G_7)\rvert = 3$	$\lvert mcs(p_3, G_6)\rvert = 2$

(c) Size of MCSs between pivots and data graphs



(d) GMTree



(e) Query on GMTree

$$h(v_1) = h(h(p_1)||h(5)||h(v_2)||h(r^0_{p1})||h(v_3)||h(r^1_{p1}))$$
$$h(v_2) = h(h(G_4)||h(5)||h(v_4)||h(r^0_{G4})||h(v_5)||h(r^1_{G4}))$$
$$h(v_3) = h(h(p_3)||h(5)||h(v_6)||h(r^0_{p3})||h(v_7)||h(r^1_{p3}))$$
$$h(v_4) = h(h(g_4)||h(g_7)||h(G_4)||h(|mcs(G_4, G_4)|)||h(G_7)||h(|mcs(G_4, G_7)|))$$
$$h(v_5) = h(h(g_3)||h(g_5)||h(G_3)||h(|mcs(G_4, G_3)|)||h(G_5)||h(|mcs(G_4, G_5)|))$$
$$h(v_6) = h(h(g_6)||h(G_6)||h(|mcs(p_3, G_6)|))$$
$$h(v_7) = h(h(g_1)||h(g_2)||h(G_1)||h(|mcs(p_3, G_1)|)||h(G_2)||h(|mcs(p_3, G_2)|))$$
$$s_{root} = sign(h(v_1))$$

(f) Digests and structure signature

Fig. 6. Illustration of `GMTree` construction, query evaluation and digests

Previous authentication works (*e.g.*, [25], [45]) have already reported that the authentication time generally decreases as the index fanout increases and meanwhile, the $\mathcal{VO}$ size increases with the fanout. Most importantly, in query processing at an internal node v, determining its subtrees that may contain answers requires only *one* similarity comparison (between the pivot of v and the query) as we use one pivot and multiple radii to split the space covered by v.

An alternative way to support large fanouts is to incorporate multiple pivots in a node v as in `mvp-tree` [44]. Our method can be easily extended to `mvp-tree` as each `mvp-tree` node essentially collapses several `vp-tree` nodes.

5.1.1 GMTree Construction

A `GMTree` is constructed by a simple recursive algorithm. The algorithm takes a graph database $\mathcal{D}$:{G_1,..., G_n}, the graph distance function d and a user-specified fanout c as input. At each recursive step, a pivot graph p and a collection of radii $\mathcal{R}_p$ are decided to cover the subspaces of the metric space, as defined in Definition 5.1(ii). For presentation clarity, we postpone the technical details of pivot selection to Sec. 7.2. The recursive step is to construct a sub-`GMTree` for the graphs covered in each subspace. The algorithm terminates when the graph number in the subspace is no more than fanout c.

Example 5.1: `GMTree` can be illustrated with an example. Fig. 6(a) shows a graph metric space formed by our example database $\mathcal{D}$:{$G_1, G_2, \ldots, G_7$} given in Fig. 1. Three pivots p_1, G_4 and p_3 are used to index the space, whose structures are shown in Fig. 6(b). Fig. 6(c) lists the size of `MCS`'s between the pivots and the data graphs. Its `GMTree` constructed is presented in Fig. 6(d), whose fanout c is 2. The first pivot p_1 and two radii r^0_{p1}, r^1_{p1} divide the metric space into two subspaces $\mathcal{U}^0$ = {G_3, G_4, G_5, G_7} and $\mathcal{U}^1$ = {G_1, G_2, G_6}. Since $\mathcal{U}^0$ and $\mathcal{U}^1$ contain more than c graphs, the construction proceeds recursively. For example, the subspace $\mathcal{U}^0$ is divided by G_4 with radii $r^0_{G_4}$ and $r^1_{G_4}$. Since there are two graphs in the subspaces of $\mathcal{U}^0$, the construction terminates.

The query evaluation algorithm is incorporating the pruning of Theorem 4.2 into a depth-first traversal on `GMTree`. At the leaf levels, we apply Lemma 4.1 to perform filtering. This is illustrated with Example 5.2. Due to space limitation, we present the detailed algorithm in Appendix B.

Example 5.2: Consider the q and $\mathcal{D}$ in our running example Fig. 1. Suppose t is 1/4. Fig. 6(e) shows the query evaluation on the `GMTree` of $\mathcal{D}$ constructed in Fig. 6. In Fig. 6(e), the visited nodes are marked with the gray shadow. The nodes are traversed in the following order: v_1, v_2, v_4, v_5, v_3, and v_6. The subtree with the pivot G_4 (rooted at v_2) is accessed because $r^0_{p1} - d(q, p^*_1) \leq t$. Similarly, the children of v_2 are accessed because they are not pruned. v_4 and v_5 are leaf nodes and they point to data graphs. G_3, G_4, G_5 and G_7 cannot be filtered by Lemma 4.1 and they are included in the candidates. Only G_4 and G_7 among them are included in the answer. Similarly, the remaining answer G_6 can be retrieved from the subtree of pivot p_3. Finally, the subtree of v_7 is pruned since $r^1_{p3} - d(q, p^*_3) > t$.

6 Authentication with GMTree

This section details the authentication with `GMTree`. Specifically, we present the signing of `GMTree`, the $\mathcal{VO}$ definition, the $\mathcal{VO}$ construction and the authentication algorithm.

6.1 Signing GMTree

Similar to other indexes for authentication, `GMTree` associates hash values to its data and nodes hierarchically and the data owner signs the hash of the root of `GMTree`. The hash values (*a.k.a* digests) of the data graphs, leaf nodes and internal nodes are defined as follows.

Definition 6.1: The *digest of a data graph G* in $\mathcal{D}$ is defined as $h(G) = h(V(G)||E(G))$, where $V(G)$ and $E(G)$ are canonical representations of vertices and edges of G, respectively.

We remark that any canonical form (*e.g.*, [30]) of $V(G)$ and $E(G)$ can be used in Definition 6.1. The only requirement is that it must be known to both the $\mathcal{SP}$ and clients.

Definition 6.2: The *digest of a leaf node* v_ℓ: (g_1,. . .,g_n) is defined as $h(v_\ell) = h(h(g_1)||...||h(g_n)||h(G_1)||h(|mcs(p, G_1)|)||...||h(G_n)||h(|mcs(p, G_n)|))$,
where g_i is a pointer pointing to the graph $G_i \in \mathcal{D}$, for i = 1...n and p is their pivot.

The *digest of an internal node* v: (p, $|p^*|$, $\mathcal{T}_p$, $\mathcal{R}_p$) is $h(v)$ = $h(h(p)||h(|p^*|)||h(T^0_p)||h(r^0_p)||...||h(T^{c-1}_p)||h(r^{c-1}_p))$,
where $\mathcal{T}_p$ = (T^0_p,. . .,T^{c-1}_p) and $\mathcal{R}_p$ = (r^0_p,. . .,r^{c-1}_p).

Suppose the GMTree's root is v_r. Its digest $h(v_r)$ is constructed recursively by Definitions 6.1 and 6.2. The $\mathcal{DO}$ signs $h(v_r)$ using a public-key digital signature scheme (*e.g.*, RSA).

Example 6.1: Fig. 6(f) shows the digests of nodes and the signature of the GMTree of Example 5.2.

6.2 Definition of Verification Objects

Verification objects $\mathcal{VO}$s contain five parts: (i) the query answers and their MCS mappings with q for checking correctness; (ii) the non-answers in the candidate set C_q; (iii) the visited nodes of the GMTree traversal; (iv) the boundary nodes of the traversal; and (v) the GMTree's signature. Specifically, $\mathcal{VO}$s are given below.

Definition 6.3: A $\mathcal{VO}$ is a 5-ary tuple $(M, N, H^v, H^b, s_{root})$, where

- M:$[G_1,h(G_1),m_1,\ldots,G_n,h(G_n),m_n]$ is a list of answer graphs in $\mathcal{RS}$:$[G_1,\ldots,G_n]$, their digests and their MCS mappings with q;
- N:$[G_1, h(G_1), |mcs(p_1, G_1)|, \ldots, G_j, h(G_j), |mcs(p_j, G_j|]$ is a list of the non-answers in the candidates C_q and the filtered data graphs by Lemma 4.1 with their digests and the mcs sizes with their pivots;
- H^v stores the traversal during the query evaluation as follows:
  - for each internal node v, we append to H^v
    - the *relevant content* $v(p, |p^*|, mcs(q,p), x_v)$, where x_v is some hint to verify $mcs(q,p)$ (construction of x_v is detailed in Sec. 7.1);
    - the radius list: r_p^1,...,r_p^{c-1} (and r_p^0 is not needed since it is always zero); and
    - the subtrees that are not pruned T_p^0,...,$T_p^{t_v-1}$, where t_v is the number of subtrees that are not pruned;
  - for each leaf node v_ℓ, we append to H^v:
    - the digest $h(v_\ell)$ of v_ℓ;
    - the list of pointers $g_1,\ldots,g_n$ in v_ℓ; and
    - the digests of pointers $h(g_1),\ldots,h(g_n)$.
- H^b stores the traversal boundary. For each internal node v visited in traversal, we append to H^b
  - the digests of subtrees $h(T_p^{t_v}),\ldots,h(T_p^{c-1})$ that are pruned by Theorem 4.2; and
  - the digests of their radii $h(r_p^{t_v}),\ldots,h(r_p^{c-1})$.
- s_{root} is the signature of the GMTree provided by the $\mathcal{DO}$.

We make some remarks on the $\mathcal{VO}$'s definition. (i) Clients rely on H^v to authenticate the traversal of the GMTree by a scan of records stored in H^v (detailed in Sec. 6.5). (ii) N must be introduced as clients need to verify that non-answers in C_q are in fact non-answers. (iii) For each visited node v in H^v, its sub-GMTrees pruned are indexing the subspaces further away from v's pivot than the bound determined by Theorem 4.2. Since these sub-GMTrees are always *consecutive* in v's children, the candidates are indexed near to each other.

The implementation of H^b can be further optimized by a straightforward adoption of authenticating search trees, *e.g.*, embedding MHT into $(\mathcal{T}_p, \mathcal{R}_p)$ of each v. To avoid confusions, we omit such implementation details in the presentation of the $\mathcal{VO}$ definition, the construction and authentication algorithms.

```
Procedure auth_similarity
Input: db $\mathcal{D}$, GMTree $T$ rooted at $v$ and query $Q = (q,t)$
Output: $\mathcal{VO}$ including query result $\mathcal{RS}$
01 if $v$ is a leaf of $T$
      /* query answers $\mathcal{RS}$ is included in $M$ */
02    $M_v = [G \mid d_s(q, G) \leq t, G \in v]$
03    $\mathcal{VO}.M = \mathcal{VO}.M \oplus [(G, h(G), m(G, q)) \mid G \in M_v]$,
      where $\oplus$ denotes concatenation
      /* non-answer candidates and filtered graphs in $v$ */
04    $M_v{}^f = [G \mid G \in v, 1 - \frac{|mcs(p,G)|}{|p|} - d(q,p) > t]$ //p is pivot of $G$
      $M_v{}' = [G \mid G \in v] - M_v - M_v^f$
05    $\mathcal{VO}.N = \mathcal{VO}.N \oplus [(G, h(G), |mcs(G,p)|) \mid G \in M_v{}' \cup M_v^f]$
06    $H^v = H^v \oplus h(v) \oplus [(g, h(g)) | g \in v]$
07 return $\mathcal{VO}$
08 denote the pivot of $v$ as $p$
09 $d = d(q, p^*) = 1 - \frac{|mcs(q,p)|}{\max(|p^*|,|q|)}$  //$mcs(q,p) = mcs(q,p^*)$
   /* $\mathcal{VO}$ construction */        /* $v$ is actually visited */
10 $\mathcal{VO}.H^v = \mathcal{VO}.H^v \oplus v(p, |p^*|, mcs(q,p), x_v) \oplus r_p^1$
        $\oplus \ldots \oplus r_p^{c-1}$ //$r_p^0$ is always 0, not add to $\mathcal{VO}$ for saving
   /* examining the sub-GMTree of $v$ */
11 for each $i$ in $[0,\ldots, c-1]$
12    if $r_p^i - d \leq t$
         /* visited sub-GMTree – put its content to $H^v$ */
13       $\mathcal{VO}.H^v = \mathcal{VO}.H^v \oplus T_p^i$
14    else
         /* pruned sub-GMTrees – put their digests to $H^b$ */
15       $\mathcal{VO}.H^b = \mathcal{VO}.H^b \oplus h(T_p^i) \oplus h(r_p^i) \ldots \oplus$
                                 $h(T_p^{c-1}) \oplus h(r_p^{c-1})$
16       break
   /* traversal of query evaluation */
17 for each $i$ in $[0,\ldots, c-1]$
18    if $r_p^i - d \leq t$
19       $\mathcal{VO} \oplus$ auth_similarity($\mathcal{D}$,$T_p^i$,$Q$)
20    else    break
21 add $s_{root}$ of $T$ of $\mathcal{DO}$ to $\mathcal{VO}$, if $s_{root}$ is not in $\mathcal{VO}$ yet
22 return $\mathcal{VO}$
```

Fig. 7. **Procedure** auth_similarity

6.3 $\mathcal{VO}$ Construction

To facilitate authenticated query processing, the $\mathcal{SP}$ not only evaluates queries but also simultaneously constructs $\mathcal{VO}$ in Procedure auth_similarity (shown in Fig. 7). For presentation clarity, we omit the verbose pseudo-code that introduces delimiters to $\mathcal{VO}$. The inputs of Procedure auth_similarity are a database $\mathcal{D}$, GMTree T and query $Q = (q,t)$. The output is the $\mathcal{VO}$ including the query result $\mathcal{RS}$. The main ideas of the $\mathcal{VO}$ construction can be described as follows.

(i) In Lines 01-07, the traversal on the GMTree reaches a leaf node v. The algorithm determines the answer graphs in Line 02, and stores them together with their digests and their MCS mappings with q in M in Line 03. Similarly, in Lines 04-05, the algorithm decides the non-answers, and stores them with their digests in N. $h(v)$, pointers in v and digests of the pointers are also added to H^v in Line 06.

(ii) Lines 08-20 are the traversal at internal nodes of GMTree. As the traversal proceeds, auth_similarity recursively constructs H^v (Lines 10-13) and H^b (Lines 14-15). Specifically, when an internal node v is visited, its relevant content and its radius list (Line 10) and sub-GMTrees not pruned are added to H^v (Lines 12-13). On the other hand, if T_p^i is pruned, T_p^i and all its subsequent sub-GMTrees are not visited by the traversal and therefore their digests are added to H^b (Lines 14-15).

(iii) Lines 17-20 simply recursively traverse the GMTree.

(iv) Finally, if the signature of the GMTree's root is not yet present in the $\mathcal{VO}$, s_{root} is added to $\mathcal{VO}$ in Line 21.

$$\begin{aligned}
M = [&G_4, h(G_4), m(q, G_4)\\
&G_6, h(G_6), m(q, G_6),\\
&G_7, h(G_7), m(q, G_7)];\\
N = [&G_3, h(G_3), |mcs(G_4, G_3)|\\
&G_5, h(G_5), |mcs(G_4, G_5)|];\\
H^b = [&h(v_7), h(r^1_{p3})];\\
&s_{root}, \text{ provided by the } \mathcal{DO}
\end{aligned}
\qquad
\begin{aligned}
H^v = [&v_1(p_1, 5, mcs(q, p_1), x_1), r^1_{p1}, v_2, v_3,\\
&v_2(G_4, 5, mcs(q, G_4), x_2), r^1_{G4}, v_4, v_5,\\
&h(v_4), g_4, h(g_4), g_7, h(g_7),\\
&h(v_5), g_3, h(g_3), g_5, h(g_5),\\
&v_3(p_3, 5, mcs(q, p_3), x_3), r^1_{p3}, v_6,\\
&h(v_6), g_6, h(g_6)];
\end{aligned}$$

Fig. 8. VO of Example 6.2

Example 6.2: In this example, we show the $\mathcal{VO}$ construction for the query result of Example 5.2. Recall from Fig 6(a) that the traversal of the `GMTree` is v_1, v_2, v_4, v_5, v_3 and v_6. `auth_similarity` performs this traversal (Lines 17-20) and constructs H^v in this order. For the visited leaf nodes, v_4, v_5 and v_6, their digests, pointers and digests of pointers are added to H^v (Line 06). For the visited internal nodes v_1, v_2 and v_3, `auth_similarity` adds their relevant contents, radius lists, and subtrees not pruned into H^v (Lines 10, 12-13). Procedure `auth_similarity` also adds the digests for pruned subtrees to H^b (Lines 14-16). In the example, G_4, G_6 and G_7 are answers, and G_3 and G_5 are non-answers. When the traversal of `auth_similarity` reaches the leaf nodes v_4, v_5 and v_6, it adds G_4, G_6 and G_7 together with their digests and mappings into M (Lines 02-03), and adds G_3 and G_5 together with their digests into N (Lines 04-05), respectively. For the pruned node v_7, `auth_similarity` adds its digest and the digest of its radius to boundary H^b (Line 15). Finally, the $\mathcal{SP}$ has s_{root} and adds it to the $\mathcal{VO}$ in Line 21. The $\mathcal{VO}$ constructed by `auth_similarity` is shown in Fig. 8

Discussion. Candidate graphs obtained from `GMTree` often form groups (in the form of lists of *consecutive* graph ids). The reason is that if a leaf node v is visited during query processing, all graphs covered by v (which are stored together) are candidates and form a group. Hence, the group size is at least the number of graphs of a leaf node. Moreover, the leaf nodes visited are often consecutive. For example, G_4, G_7, G_3, G_5, G_6 in Fig. 6(e) can form a group. It is evident that deriving $\mathcal{VO}$ for consecutive candidate graphs is efficient.

6.4 Cost Model of VO Size

This subsection models the overall $\mathcal{VO}$ size of `GMTree`. Since the candidate graphs obtained from traversing `GMTree` are grouped, the boundary nodes are few and a small number of digests in H^b are needed to authenticate the traversal. Due to the same reason, the number of visited sub-`GMTrees` is relatively small. These two factors lead to smaller $\mathcal{VO}$ sizes. In contrast, if the candidate and non-candidate graphs are stored alternately in the worst case, each individual candidate needs a digest in H^b which results in large $\mathcal{VO}$s.

For example, given a `GMTree` of fanout 8, if 3/4 sub-`GMTrees` are visited and $|\mathcal{D}| = 10000$, the `GMTree` only needs 5% digests of the worst case. (The full arithmetic calculations are presented in Appendix A.3.) It is verified by experiments (see the first experiment of Sec. 8.1) that a baseline approach is close to the worst case.

Let s_{sig}, s_{rev}, s_{rad}, s_{ptr}, s_h , s_i and s_G denote the sizes of a signature, a relevant content of a node, a radius value, a pointer, a digest, an integer and a data graph, respectively. Let $V_{visited}$ denote the set of visited nodes of `GMTree`. The overall $\mathcal{VO}$ size of a query (q, t) on a `GMTree` T of a fanout c is shown in Fig. 9. Fig. 9 shows that the $\mathcal{VO}$ size depends on the number of visited subtrees. In particular, for H^b, $\sum_{\substack{v \text{ is visited} \\ v \text{ is internal}}} c$ is the number of subtrees of visited internal nodes and $|V_{visited}| - 1$ is the number of visited ones except the root. Since the visited subtrees of an internal node are consecutive, their difference is the number of boundary nodes. It is multiplied by $2s_h$ as we need two digests for each boundary node. Due to space limitation, the arithmetic derivation is presented in Appendix A.4.

$$\begin{aligned}
|\mathcal{VO}| = &\sum_{\substack{v \text{ is visited} \\ v \text{ is internal}}} (s_{rev} + (c-1)s_{rad}) + \sum_{v \text{ is visited}} s_{ptr} - s_{ptr} && //H^v \text{ for internal}\\
&+2s_h\Big(\sum_{\substack{v \text{ is visited} \\ v \text{ is internal}}} c - (|V_{visited}| - 1)\Big) && //H^b\\
&+\sum_{\substack{v \text{ is visited} \\ v \text{ is leaf}}} ((c+1)s_h + c \times s_{ptr}) && //H^v \text{ for leaf}\\
&+\sum_{\substack{v \text{ is visited} \\ v \text{ is leaf}}} c(s_h + s_G + s_i) && //M \text{ and } N\\
&+s_{sig} && //s_{root}
\end{aligned}$$

Fig. 9. Cost Model of VO Size

6.5 Authentication Algorithm

To authenticate the query results, clients are required to rerun the traversal on `GMTree` and synthesize the digest of `GMTree`'s root from the $\mathcal{VO}$. Specifically, clients decide if (i) the graphs in $\mathcal{VO}.M$ and $\mathcal{VO}.N$ are similar and dissimilar, respectively, to the query; (ii) no boundary index node overlaps with the query; and (iii) the synthesized digest h_{root} of `GMTrees` root agrees with the signature s_{root} provided by the $\mathcal{DO}$.

Procedure `auth` summarizes the authentication algorithm, shown in Fig. 10. `auth` takes a query Q and $\mathcal{VO}$ that contains query result $\mathcal{RS}$ as input and outputs `true` only if $\mathcal{RS}$ is sound and complete. Line 01 recomputes the root digest recursively. If the synthesized digest h_{root} agrees with the $\mathcal{DO}$'s signature s_{root}, `auth` returns `true`; otherwise, `false` (Lines 02-03). Here, we focus on the recursion logic of `auth_aux` as presented in Fig. 11.

Procedure `auth_aux` reruns the traversal recorded in H^v. Each visited node v can be either a leaf node (Lines 02-16) or an internal node (Lines 17-27). In the former case, `auth_aux` fetches $h(v)$, the pointers in v, the digests of pointers from H^v (Line 03). It also fetches the answers M_v and non-answers N_v of v from M and N, respectively (Lines 04-05). `auth_aux` then computes an intermediate string str_v to concatenate the digests of the pointers, answers and non-answers (Line 06), where `sort` is the publicly known sorting function that reorders the pointers and graphs, used by the $\mathcal{SP}$. `auth_aux` exits, if (i) $G \in M_v$ is not similar to q; or (ii) $C \in N_v$ is similar to q; or (iii) the recomputed digests do not match with the ones stored in $\mathcal{VO}$ (Lines 07-15). Otherwise, the recomputed digest of v is returned (Line 16).

In the latter case, `auth_aux` recursively visits the subtrees visited in H^v and scans the pruned subtrees recorded in H^b to recompute the digest of v. Specifically, `auth_aux` first retrieves the relevant content, radius list and subtrees that are visited from H^v (Line 18). `auth_aux` exits, if $mcs(q, p)$ is detected incorrect using the hint x_v (Line 19). (We will dedicate Sec. 7.1 for this.) Otherwise, `auth_aux` computes the distance between the query and the augmented pivot $d(q, p^*)$ in Line 20 and Lines 21-27 reconstruct the digest of v. By Definition 6.2, the construction of the digest of internal node v needs the digests of its subtrees. Hence, `auth_aux` recursively constructs the digests of the subtrees that are not pruned (Lines 23-24) and fetches the digests of pruned subtrees from H^b (Lines 25-27). The digest of v is returned (Line 28).

Due to space restrictions, the proofs of the soundness and completeness of authentication are provided in Appendix A.6.

```
Procedure auth
Input: $Q = (q,t)$, $\mathcal{VO} = (M, N, H^v, H^b, s_{root})$,
    , where $s_{root}$ is the signature of the digest of GMTree's root
Output: true iff $\mathcal{RS}$ is correct: sound and complete
01 $h_{root}$ = auth_aux($Q$, $\mathcal{VO}$)
02 if $h_{root}$ agrees with $s_{root}$    return true
03 else    return false
```

Fig. 10. **Procedure** `auth`

Example 6.3: Continuing with Example 6.2, `auth` reconstructs the digest $h(v_1)$ of the root of `GMTree` by rerunning the traversal recorded in H^v (Line 01). `auth` then compares the reconstructed $h(v_1)$ with the signature s_{root}. If $h(v_1)$ agrees with s_{root}, the query result is correct (Line 02).

Procedure `auth_aux` proceeds the traversal as follows. The traversal starts with the root v_1. `auth_aux` first detects the correctness of the $mcs(q, p_1)$ in $\mathcal{VO}$ (Lines 18-19). If $mcs(q, p_1)$ is correct, `auth_aux` reconstructs the digest of v_1 as $h(v_1)$=$h(h(p_1)||h(|p_1^*|)||h(v_2)||h(r_{p_1}^0)||h(v_3)||h(r_{p_1}^1))$ (Lines 21-27). Here, $h(p_1)$, $h(|p_1^*|)$, $h(r_{p_1}^0)$ and $h(r_{p_1}^1)$ can be readily computed from $\mathcal{VO}$ (Lines 18). However, `auth_aux` needs to visit subtrees of v_2 and v_3 to reconstruct $h(v_2)$ and $h(v_3)$ recursively (Lines 23-24).

When the traversal reaches the internal node v_2, `auth_aux` computes $h(v_2)$ by the same logic with $h(v_1)$. However, for v_3, `auth_aux` can detect v_7 is pruned (Line 22) and fetch $h(v_7)$ and $h(r_{p3}^1)$ directly from H^b (Lines 25-27). `auth_aux` only needs to visit the subtree of v_6 to compute $h(v_6)$ (Line 24).

When the traversal reaches the leaves v_4, v_5 and v_6, `auth_aux` fetches their pointers, answers (G_4, G_6 and G_7), non-answers (G_3 and G_5) and their digests from $\mathcal{VO}$, respectively (Lines 03-05). `auth_aux` verifies if (i) the answers are similar to q (Lines 09-11), (ii) the non-answers are dissimilar (Lines 12-14) and (iii) recomputed digests match with the ones stored in $\mathcal{VO}$ (Lines 08,10,13 and 15). `auth_aux` then returns the recomputed digests $h(v_4)$, $h(v_5)$ and $h(v_6)$ (Lines 06, 16).

```
Procedure auth_aux
Input: $Q = (q,t)$, $\mathcal{VO}$=$(M, N, H^v, H^b, s_{root})$
Output: recomputed digest of $v$
01 tok = $H^v$.getNext()
02 if tok represents a leaf node    //check ans. & non-ans.
03   denote tok as $[h(v), g_1, h(g_1), \ldots, g_n, h(g_n)]$
04   $M_v = [(G, h(G), m) | (G, h(G), m) \in M \wedge$ $G$ pointed by some $g_i]$
05   $N_v$=$[(C, h(C), |mcs|) | (C, h(C), |mcs|) \in N \wedge C$ pointed by some $g_i]$
  /* $str_v$ is used to reconstruct the digest of $v$ */
06    $str_v$ = hash(sort($M_v \oplus N_v \oplus$ $[g_1,...,g_n]$))
  /* if hash values do not match, program exits */
07    for each $i \in [1..n]$
08       if $h(g_i)$ $\neq$hash($g_i$) raise exception
09    for each $G \in M_v$
10       if !is_similar($G,q,m,t$) or $h(G)$ $\neq$hash($G$)
11          raise exception
12    for each $C \in N_v$
13       if is_similar($C,q,t$) or $h(C)$ $\neq$hash($C$)
14          raise exception
15    if $h(v)$ $\neq$hash($str_v$) raise exception
  /* leaf $v$ is authenticated, return its digest */
16    return hash($str_v$)

17 else //tok represents an internal node
18    denote tok as
            $[v(p, |p^*|, mcs(q,p), x_v), r_p^1, ..., r_p^{c-1}, T_p^0, ..., T_p^{t_v - 1}]$
            //$r_p^0 = 0$ by def. $T_p^i$s are visited subtrees
19    if $mcs(q,p)$ is detected incorrect using $x_v$ raise exception
     //Sec. 7.1 will detail the algo. to verify $mcs(q,p)$ from $x_v$
20    $d = d(q,p^*) = 1 - \frac{|mcs(q,p)|}{\max(|p|,|p^*|)}$ //$mcs(q,p)$=$mcs(q,p^*)$
  /* reconstruct digest of $v$ */
21    $str_v$ = hash($p$)||hash($|p^*|$)
22    if !($r_p^{t_v-1} \leq d + t < r_p^{t_v}$) raise exception
23    for each $j$ in [0,...,$t_v - 1$]
24       $str_v$ = $str_v$ || auth_aux($Q$,$\mathcal{VO}$) || $h(r_p^j)$
25    for each $j$ in [$t_v$,...,$c - 1$]
26       $h(T_p^j)$ = $H^b$.getNext() and $h(r_p^j)$ = $H^b$.getNext()
27       $str_v$ = $str_v$ || $h(T_p^j)$ || $h(r_p^j)$
28    return hash($str_v$)
```

Fig. 11. **Procedure** `auth_aux`

7 Optimization Problems

This section presents two optimizations for `GMTree`, namely authenticated `MCS` computation and pivot selection problem.

7.1 Authenticated MCS Computation

As the `MCS` computation is extensively involved in client-side authentication, we present an authenticated version of the state-of-the-art, vertex-cover-based `MCS` computation method (denoted as `VC-mcs`) [20] to save client's authentication time.

7.1.1 Overview of `VC-mcs`

Given two graphs G and G', to determine $mcs(G, G')$, the `VC-mcs` performs the following tasks:

1) Determine a vertex cover VC_G of G;
2) Compute all common subgraphs $\mathcal{S}$ between VC_G and G';
3) For all $S \in \mathcal{S}$, determine G - VC_G and G' - $f(S)$, where f is the subgraph isomorphic mapping of S;
4) Determine all maximal independent sets $\mathcal{M}_S$ of G' - $f(S)$;
5) For each $M \in \mathcal{M}_S$, determine the maximum matching J_M in the bipartite graph $B_{(G,G',M)}$, where $B_{(G,G',M)}$ = (U, V, E), U and V are the vertices of G - VC_G and M, respectively, and for $\forall u \in U$, $\forall v \in V$, (u, v) $\in E$ if
   - u and v have the same label;
   - The neighbors of u in S should be mapped to the neighbors of v in $f(S)$, and vice versa; and
   - The edge of u and u's neighbor in S should have identical label of v and v's neighbor in $f(S)$.
6) $S \cup J_M$ forms a larger common subgraph than S, in which $\max\{S \cup J_M\}$, for $M \in \mathcal{M}_S$ is called the extension of S, denoted as $ext(S)$; and
7) $mcs(G, G')$ is the maximum one of all $ext(S)$, $S \in \mathcal{S}$.

As reported in [20], the time complexity of `VC-mcs` is $O(3^{|G'|/3}|G'|^{2.5}(|G'| + 1)^{|VC_G|})$, where $O(3^{|G'|/3})$ is for Step 4), $O(|G'|^{2.5})$ is for Step 5) and $O((|G'| + 1)^{|VC_G|})$ is the number of iterations of Steps 3)-6).

7.1.2 Authenticated `VC-mcs`

A naive method for client's authentication is that after receiving the $mcs(q, G)$ from the $\mathcal{SP}$, the client recomputes it from scratch. However, it is inefficient, as `VC-mcs` has already been done once by the $\mathcal{SP}$. Therefore, our authenticated `VC-mcs` (`auth-VC-mcs`) requires the $\mathcal{SP}$ to record some intermediate results as hints in $\mathcal{VO}$ during its `VC-mcs`. Importantly, the correctness of the hints can be easily determined by a scan by the client. `MCS`s are not recomputed from scratch.

Definition 7.1: Given a query q and a graph G, the *hint* of $mcs(q, G)$ is a 4-ary tuple ($mis(G)$, s_{mis}, $\mathcal{J}$, $\mathcal{VC}$), where

- $mis(G)$ is all the MISs of G precomputed by the $\mathcal{DO}$;
- s_{mis} is the $\mathcal{DO}$'s signature of $mis(G)$; and
- $\mathcal{J}$ and $\mathcal{VC}$ are the maximum matchings and the minimum vertex covers of the bigraphs in Step 5) of `VC-mcs`, respectively, computed by the $\mathcal{SP}$ on-the-fly.

Protocol. The protocol of `auth-VC-mcs` with respect to the $\mathcal{DO}$, the $\mathcal{SP}$ and the client is as follows, where ① is offline and others are done on-the-fly.

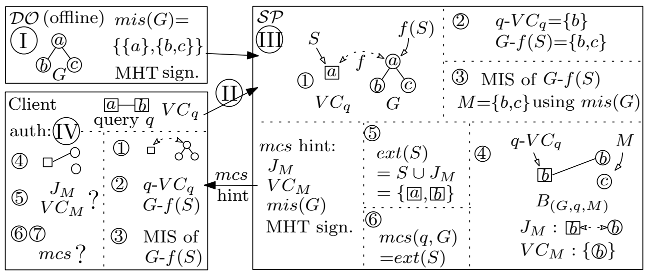

$\mathcal{DO}$ (offline) $mis(G)$= {{a},{b,c}}
MHT sign.
Ⅰ
Client
auth: Ⅳ
query q
VC_q
Ⅱ
④
⑤ J_M VC_M ?
⑥⑦ mcs ?
① ② q-VC_q G-$f(S)$
③ MIS of G-$f(S)$
mcs
hint
$\mathcal{SP}$
Ⅲ
S $f(S)$ f
① VC_q G
② q-VC_q={b} G-$f(S)$={b,c}
③ MIS of G-$f(S)$ M={b,c} using $mis(G)$
mcs hint: J_M VC_M $mis(G)$ MHT sign.
⑤ $ext(S)$ $= S \cup J_M$ = {a,b}
⑥ $mcs(q,G)$ $=ext(S)$
④ q-VC_q M
$B_{(G,q,M)}$
J_M : b ↔ b
VC_M : {b}

Fig. 12. The process of `auth-VC-mcs`

Ⅰ The $\mathcal{DO}$ precomputes all MISs of a graph G, and sends the MISs with their signature to the $\mathcal{SP}$ offline. MISs are sorted (by vertex ids) and compressed using classical compression algorithms. Since MISs often contain numerous frequent patterns (ids), they can be efficiently compressed.

Ⅱ The client computes a vertex cover VC_q of q and transmits both VC_q and q to the $\mathcal{SP}$.

Ⅲ The $\mathcal{SP}$ receives the q and VC_q from the client. It performs Steps 2)-7) of `VC-mcs` and constructs the hints of $mcs(q,G)$ for G as follows:

- Add into the hints all the compressed maximal independent sets of G and their signatures; and
- For all $S{\in}\mathcal{S}$ and $M{\in}\mathcal{M}_S$, add to the hints the maximum matching J_M and the minimum vertex cover VC_M of $B_{(G,q,M)}$ (using an associative hashtable of $B_{(G,q,M)}$).

Ⅳ Upon receiving the $\mathcal{VO}$ for G, the client retrieves $mcs(q,G)$ and its hint in $\mathcal{VO}$, and performs the *simplified* Steps 2)-7) of `VC-mcs` to verify the $mcs(q,G)$. The simplifications are the following.

- In Step 4), compute the MISs of $G - f(S)$ by excluding $f(S)$ from the MISs of G; and
- In Step 5), reconstruct the $B_{(G,q,M)}$ and retrieve its J_M and VC_M from the hints. Then, if J_M and VC_M are not a matching and a vertex cover of $B_{(G,q,M)}$, respectively, or if $|J_M| \neq |VC_M|$, $mcs(q,G)$ is subverted.

`auth-VC-mcs` is more efficient than `VC-mcs` for the client. Specifically, the time complexity of the Step 4) is reduced to $O(|G||mis(G)|)$ and that of the Step 5) is reduced to $O(|G|^2)$ and hence the client's time complexity is reduced to $aud_c = O(|G||mis(G)||G|^2(|G|+1)^{|VC_q|})$. The total authentication time for the client is $O((\frac{|\mathcal{D}|}{c} + |C_q| - |R_q|) \times aud_c + |R_q| \times |G|)$, as authenticating an answer just needs $O(|G|)$. The time complexity for the $\mathcal{SP}$ is $aud_{sp} = O(|G||mis(G)||G|^4(|G|+1)^{|VC_q|})$, as it is $O(|G|^2)$ to compute the minimum vertex cover of a bigraph. The total time for the $\mathcal{SP}$ is $O((\frac{|\mathcal{D}|}{c} + |C_q|) \times aud_{sp})$. We provide the proof of the correctness of `auth-VC-mcs` in Appendix A.7 for presentation brevity.

Example 7.1: Fig. 12 shows the process of `auth-VC-mcs`.

Ⅰ The $\mathcal{DO}$ precomputes all the MISs $mis(G)$ of G, and sends $mis(G)$ and its signature to the $\mathcal{SP}$ offline.

Ⅱ The client submits to the $\mathcal{SP}$ the query q with a vertex cover VC_q=$\{a\}$.

Ⅲ Once receiving the query q and a vertex cover VC_q of q, the $\mathcal{SP}$ enumerates all the common subgraphs between VC_q and G in Step ①. In this example, VC_q only has one common subgraph S=$\{a\}$ with G as indicated by the mapping f in Fig. 12. In Step ②, the $\mathcal{SP}$ computes the q-VC_q and G-$f(S)$. In Step ③, the $\mathcal{SP}$ decides all the MISs of G-$f(S)$ using $mis(G)$. In Step ④, for each MIS M of G-$f(S)$, the $\mathcal{SP}$ constructs a bigraph $B_{(G,q,M)}$ between q-VC_q and M. We only have one bigraph here as G-$f(S)$ has only one MIS M=$\{b,c\}$ in this example. The $\mathcal{SP}$ computes the maximum matching J_M and the minimum vertex cover VC_M of $B_{(G,q,M)}$. After that, in Steps ⑤ and ⑥, the $\mathcal{SP}$ obtains $mcs(q,G) = S \cup J_M$ as this example only has one S and one J_M. Finally, the $\mathcal{SP}$ adds J_M, VC_M, $mis(G)$ and its signature and G into the hint of $mcs(q,G)$, and sends them in the $\mathcal{VO}$ to the client.

Ⅳ Upon receiving the $\mathcal{VO}$, in Steps ① to ④, the client recomputes the common subgraph S=$\{a\}$ between VC_q and G, and reconstructs the bigraph $B_{(G,q,M)}$ using $mis(G)$ and q in the hint. Then, in Step ⑤, the client checks if J_M and VC_M in the hint are a matching and a vertex cover of $B_{(G,q,M)}$, respectively, and if $|J_M|$=$|VC_M|$. If they are `true`, and the recomputed $mcs(q,G)$ equals to the one in the hint as shown in Steps ⑥ and ⑦, the $mcs(q,G)$ is authenticated.

Discussion. We remark that when G is large and $|mcs(q,G)|$ is close to $|q|$, `VC-mcs` may be slower than direct backtracking [20]. To address this, we complement `VC-mcs` with a simple enumeration method (denoted as `Enum`), which has the same time complexity as the backtracking in such scenarios [46]. The hint of `Enum` is simply $|mcs(q,G)|$. Specifically, if $|mcs(q,G)|{>}\beta{\times}|q|$, where $\beta \in [0,1]$ is a user-defined parameter whose value is close to 1, the client simply enumerates the subgraphs of q of size $|mcs(q,G)|$+1. If (i) no such subgraph is contained in G; and (ii) the $mcs(q,G)$ is a common subgraph of q and G, then the $mcs(q,G)$ is authenticated. Otherwise, we adopt `auth-VC-mcs` for other cases, as `Enum` is generally not efficient to handle $O(C_{|q|}^{|q|/2})$ subgraphs. Moreover, β can also be tuned between the authentication time and $\mathcal{VO}$ size, as `Enum`'s hint is smaller than `auth-VC-mcs`.

7.2 Pivot Selection

To optimize the performances of `GMTree`, we propose a sampling approach to select pivots for a `GMTree` that produces a small candidate set. Existing works on pivot selection often adopt to minimize some *cost* function in a heuristic manner. In this paper, we adopt the min max covering radius heuristic as it produces the fewest candidates as reported in [17].

A unique property of our graph data is that they can be decomposed into subgraphs, whereas the data objects in the conventional metric space are assumed atomic. In our preliminary experiments, we observe that using subgraphs as pivots could produce smaller costs than using the data graphs. Therefore, in this paper, we are going to consider the subgraphs in our pivot selection. Our min max covering radius heuristic is formally presented as follows.

Definition 7.2: Given a set of graphs $\mathcal{D} = \{G_1, G_2, \ldots, G_m\}$ and the set of subgraphs $\mathcal{S} = \{S | S \subset G, G \in \mathcal{D}\}$, select one graph p as pivot from $\mathcal{D} \cup \mathcal{S}$ to minimize the cost function

$$X = \max_{i=1\ldots m}\{d(p^*, G_i)\}, \tag{3}$$

where p^* is the augmented pivot constructed from p as discussed in Sec. 4.

The pivot selection problem is NP-hard. Due to space limitations, we present the proof in Appendix A.8. For practical considerations, we propose a sampling method to select pivots.

Sampling approach. We randomly sample s graphs from the pruned search space. This can be established by first randomly choosing data graphs from $\mathcal{D}$ and then randomly deleting some vertices from the chosen data graphs. We try all sample graphs and compute the costs. The minimum cost found in this way is the sample minimum, denoted as $\hat{a}$. Note that $\hat{a}$ is a random variable. We use $\hat{a}$ to approximate the population minimum, denoted as a. Suppose b is the population maximum. Then, $\hat{a} \in [a, b]$. Since $\mathcal{D} \cup \mathcal{S}$ is finite, a and b exist and are finite. By the definition of graph distance, $0 \leq a \leq \hat{a} \leq b \leq 1$. Suppose the graph distance between any two graphs is uniformly distributed in $[a, b]$. Let ϵ bound the error of $\hat{a}$ from a. Then, for any $0 < \epsilon < b - a$, we have

$$Pr(\hat{a} - a \leq \epsilon) \geq 1 - (1 - \epsilon)^s \tag{4}$$

Formula (4) states that the sample minimum $\hat{a}$ is very close to the population minimum a, *i.e.*, $\hat{a} - a$ is bounded by an arbitrarily small number ϵ, with a high probability. Specifically, $1 - (1 - \epsilon)^s$ grows exponentially with respect to the sample size s. For example, when $\epsilon = 0.1$, just 30 random samples can guarantee $\hat{a} - a < 0.1$ with probability larger than 95%.

Determining pivot size. The sampling may select large pivots to minimize the number of candidates, but the large pivots will decelerate the GMTree traversal. We propose a simple cost model to quantify these. We denote the cost at a certain pivot size as $Cost$. $Cost$ consists of two costs: (i) the number of the maximum matchings $|\mathcal{J}|$ and the minimum vertex covers $|\mathcal{VC}|$ in the MCS computation of pivots, as the auth-VC-mcs needs a scan on these structures; and (ii) the number of candidates.

$$Cost = |\mathcal{J}| + |\mathcal{VC}| + k \times |C_q|. \tag{5}$$

k denotes the relationship between $|\mathcal{J}|+|\mathcal{VC}|$ and $|C_q|$, which is determined by issuing many random queries.

To choose an optimal pivot size using our model, we can simply increase the pivot size and stop once $\Delta Cost$ is observed larger than zero. To obtain the $Cost$ for each pivot size, we apply the sampling method on the graphs of that size in $\mathcal{D} \cup \mathcal{S}$ to determine the pivots.

The time complexity of GMTree construction with the sampling-based pivot selection is $O((\log_c |\mathcal{D}| - 1) \times s \times |G| \times dist \times |\mathcal{D}|)$, where $dist$ is the time complexity of VC-mcs in Sec. 7.1.1. For presentation brevity, we provide the detailed analysis in Appendix A.9.

8 Experimental Evaluation

In this section, we present an extensive experimental evaluation that verifies the efficiency of our proposed techniques and the effectiveness of our optimizations. We performed an experimental comparison with the baseline method in Sec. 4 and Grafil*. Grafil* is extended from the seminal subgraph similarity search method Grafil [1] with a basic authentication. In particular, the edge-based similarity definition of Grafil has been modified and has been replaced by Definition 3.2. Regarding the baseline method, we used data graphs as pivots and turned off all the proposed optimizations. Regarding Grafil*, we built MHTs on Grafil's matrix index for authentication. Specifically, we built the MHT on the column vectors of the matrix, as Grafil uses its matrix in columns.

Experimental settings. We ran our experiments on a server with a Dual 6-core 2.66GHz CPU running CentOS 5.6. Our implementation was written in Java with JDK 1.6. The maximum memory for our Java Virtual Machine was set to 4G bytes. We used an external graph isomorphism library VFLib [47] and the state-of-the-art MCS computation library CCP4 [48]. SHA and RSA were used as our crypto signing schemes.

Benchmark datasets. We used a real dataset AIDS as in [1]–[3], [6] and a synthetic dataset SYN [49] in our evaluation. Since PubChem [10] is used in our motivating examples, we tested our algorithms on a PUBCHEM dataset in Appendix D. Our AIDS dataset was provided by X. Yan et al. [1], which contains 10,000 molecular graphs. Our synthetic data generator was provided by J. Cheng et al. [49]. The statistics of the datasets are presented in Tbl. 2 in Appendix C.

Query workload. Similar to [1], [2], we tested query graphs of different sizes: $Q4$, $Q8$ and $Q12$. $Qm, m = 4, 8, 12$ means that the query graph has m vertices. Qm is also a mixture of queries of different densities. Each Qm contains 1,000 query graphs. Following [2], the possible numbers of missing vertices σ of q are 1, 2 and 3. Due to space restrictions, we present the complete description in Appendix C.

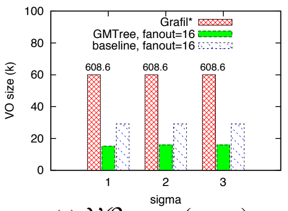
Our experiments present the performances of our techniques on the $\mathcal{SP}$'s query time, the client's authentication time and the $\mathcal{VO}$ size, respectively. *The reported performances were averaged performances of our techniques on 1,000 queries in each query set.* $\mathcal{VO}$s were simply compressed by the gzip package in JDK 1.6. To study the GMTree's characteristics under a wide range of fanouts, our experiments were conducted with fanouts of 2^2, 2^4 and 2^6, unless otherwise specified.

8.1 Comparison with the Baseline and Grafil*

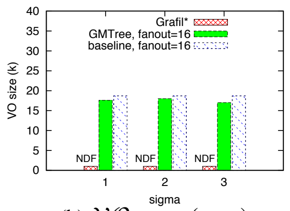
This experiment compares the performances of GMTree with the baseline and Grafil* in terms of the $\mathcal{VO}$ size and the time on AIDS and SYN, respectively.

Comparison of $\mathcal{VO}_{index}$ size. Figs. 13(a)-(b) compare the $\mathcal{VO}_{index}$ size on AIDS and SYN, respectively. Fig. 13(a) presents that our $\mathcal{VO}_{index}$ is only 3% of Grafil*. The reason is that the column vectors of the matrix of Grafil* are high dimensional as the matrix often has thousands of rows and MHTs on high dimensional data usually produces large $\mathcal{VO}$s. In addition, since the candidate graphs of our GMTree are located near to each other, we only needs about 6% digests of Grafil* to authenticate the traversal boundary. We do not show Grafil* on SYN because that the feature extraction on SYN can not finish due to its high density. Figs. 13(a)-(b) also present that our $\mathcal{VO}_{index}$ is clearly smaller than that of the baseline as the filtering of the baseline is poor.

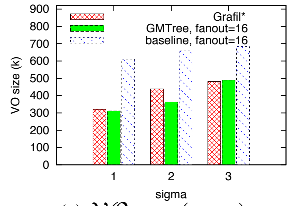
Comparison of $\mathcal{VO}_{cand}$ size. Figs. 13(c)-(d) compare the $\mathcal{VO}_{cand}$ size on AIDS and SYN, respectively. From Fig. 13(c), we note that our $\mathcal{VO}_{cand}$ is in the same order of magnitude as Grafil*, since we have comparative numbers of candidate graphs. Specifically, for $Q8$ on AIDS, the precision of GMTree (*i.e.*, $|R_q|/|C_q|$) is 0.7K/4.7K, 3.4K/4.9K and 4.6K/6.6K for σ=1, 2, 3, respectively. In comparison, the precision of Grafil* is 0.7K/5.8K, 3.4K/7.7K and 4.6K/8.5K for σ=1, 2, 3, respectively. We highlight that the similarity search by nature has



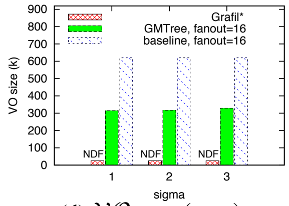
(a) $\mathcal{VO}_{index}$ (AIDS)



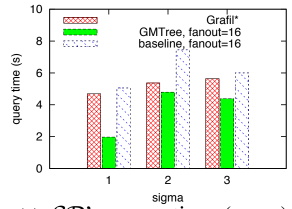
(b) $\mathcal{VO}_{index}$ (SYN)



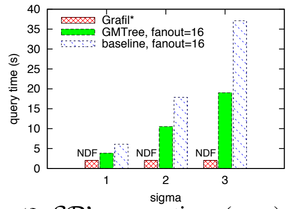
(c) $\mathcal{VO}_{cand}$ (AIDS)



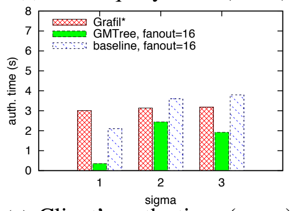
(d) $\mathcal{VO}_{cand}$ (SYN)



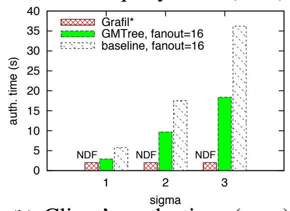
(e) $\mathcal{SP}$'s query time (AIDS)



(f) $\mathcal{SP}$'s query time (SYN)



(g) Client's auth. time (AIDS)



(h) Client's auth. time (SYN)

Fig. 13. Comparison of GMTree with Grafil* and the baseline in terms of $\mathcal{VO}$ size and time on AIDS and SYN

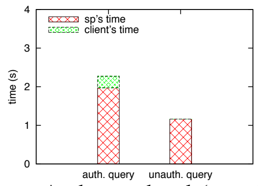
large candidate sets and that is why we optimize the MCS computation. While the size of GMTree is in the same order of magnitude as Grafil* (on AIDS, GMTree is 4.9M bytes and Grafil* is 1.6M bytes), the candidates of GMTree are smaller than those of Grafil*, by using Lemma 4.1 on GMTree's leaf nodes. In particular, the average size of the top 5% largest candidates of GMTree is 1.5 times smaller than that of Grafil*. Figs. 13(c)-(d) also show that our $\mathcal{VO}_{cand}$ is about 40% smaller than that of the baseline.

Comparison of query time. Figs. 13(e)-(f) compare the $\mathcal{SP}$'s query time on AIDS and SYN, respectively. Fig. 13(e) shows that our GMTree outperforms Grafil*. In particular, our GMTree is faster than Grafil* by a factor of 2 when $\sigma = 1$. Figs. 13(e)-(f) also presents that the GMTree is about 40% faster than the baseline on average.

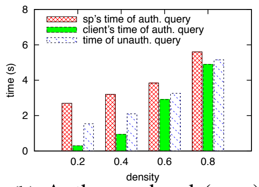
Comparison of authentication time. Figs. 13(g)-(h) compare the authentication time on AIDS and SYN. Fig. 13(g) presents that the GMTree is at least 23% and 28% faster than Grafil* and the baseline, respectively, on AIDS. Fig. 13(h) shows that GMTree is at least 44% faster than the baseline on SYN.

8.2 Authenticated Query Overhead

In this experiment, we show the performance breakdown of authenticated and unauthenticated query in Fig. 14. Figs. 14(a)-(b) show that the overhead needed to support authenticated queries is about 100% of the time of unauthenticated query. However, Fig. 14(a) shows that the client's time is much smaller than the unauthenticated one. Fig. 14(b) shows further the comparison on data graphs with various densities.



(a) Auth. overhead (AIDS)



(b) Auth. overhead (SYN)

Fig. 14. Authentication overheads on AIDS and SYN

We observe that GMTree is more efficient and effective on sparse graphs as the distance computation requested increases significantly as the density increases. (It is also observed by [2], [8].) This is consistent with Fig. 14(a) that GMTree is efficient on real datasets since chemical molecular graphs are often sparse (as reported in *e.g.*, [1], [46]). GMTree is more efficient than the state-of-the-art technique Grafil* on the real dataset (Fig. 13). Finally, we remark that the $\mathcal{SP}$ is often equipped with powerful machines, which is not the case here.

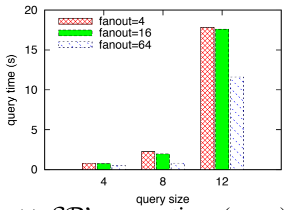
8.3 Experiments on query size.

In this experiment, we use the AIDS and SYN datasets and their $Q4$, $Q8$ and $Q12$ query sets. To study the effect of query size on our techniques, we fix σ to 1. We vary σ in later experiments. Fig. 15 shows the $\mathcal{SP}$'s query time, the client's authentication time and the overall $\mathcal{VO}$ size, respectively. In the following, we simply use the terms query time and authentication time for short, if it is clear from the context.

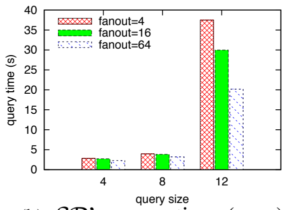
Query time. From Figs. 15(a)-(b), we note that the query time increases significantly with the growth of query size. This is consistent to the complexity of MCS computation. We also note that the query time reduces as the fanout increases. For example, on AIDS, the query time of $Q8$ at fanout 64 is 41% of that at fanout 16. This is because that the larger fanout, the less MCS computation.

Authentication time. Figs. 15(c)-(d) present that the authentication time is much smaller than the query time. In particular, on AIDS, the authentication time of $Q4$ at fanout 16 is about 1/2 of its query time, 1/6 for $Q8$ and 1/8 for $Q12$, respectively. This is mainly for three reasons. (i) Regarding our Enum method, if the size of the MCS between q and pivot p is declared to be α, the client only needs to decide whether q has a subgraph of size $\alpha + 1$ of p, while in query evaluation the $\mathcal{SP}$ may need to determine all subgraphs of sizes from $\alpha + 1$ to $|q|$. (ii) Regarding our vertex-cover based method, the client needs not to compute the maximum matchings and the minimum vertex covers on-the-fly. (iii) Checking mapping for answer graphs is efficient.

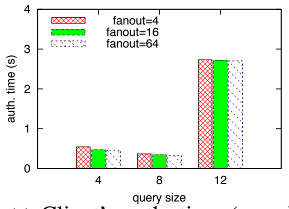
Figs. 15(c)-(d) also present that the authentication time first reduces slightly and then increases significantly with the growth of query size. This is because that when the query size increases, the number of candidate graphs decreases, but each candidate's authentication time increases. The slight reduction from $Q4$ to $Q8$ is because that the reduction of candidates has slightly more impact than the increase of each candidate's authentication time. In particular, at fanout 16 on AIDS, the candidate number reduces about 4K from $Q4$'s 9K to $Q8$'s 5K, but each candidate's authentication time only increases 2.4 μs from 4.8 μs to 7.2 μs. The time is averaged from 1,000 queries. The following notable increase from $Q8$ to $Q12$ is because that each candidate's authentication time increases much faster than the reduction of the number of candidates.



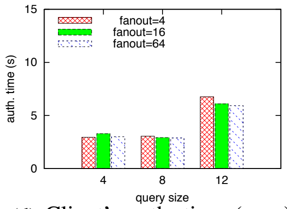
(a) $\mathcal{SP}$'s query time (AIDS)



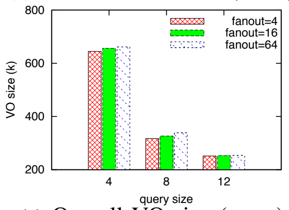
(b) $\mathcal{SP}$'s query time (SYN)



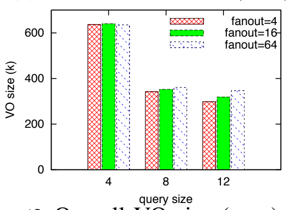
(c) Client's auth. time (AIDS)



(d) Client's auth. time (SYN)

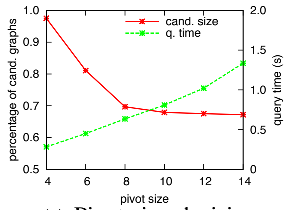


(e) Overall $\mathcal{VO}$ size (AIDS)

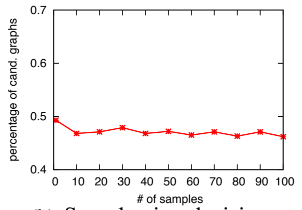


(f) Overall $\mathcal{VO}$ size (SYN)

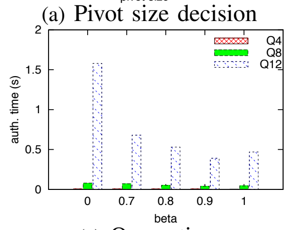
Fig. 15. Experiments on query size on AIDS and SYN



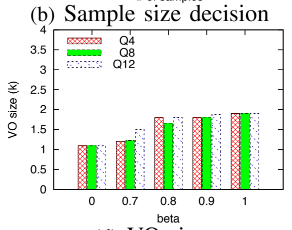
(a) Pivot size decision



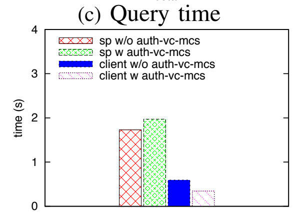
(b) Sample size decision



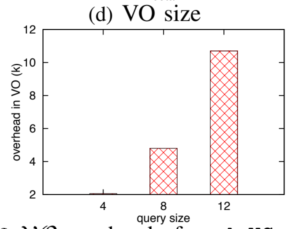
(c) Query time



(d) VO size



(e) Time benefit of `auth-VC-mcs`



(f) $\mathcal{VO}$ overhead of `auth-VC-mcs`

Fig. 16. Performance results of optimizations

In particular, at fanout 16 on AIDS, the number of candidates decreases only about 20% from $Q8$'s 5K to $Q12$'s 4K, but each candidate's authentication time significantly increases about 10 times from 7.2 μs to 67 μs.

VO size. Figs. 15(e)-(f) show the total $\mathcal{VO}$ size. We observe that $\mathcal{VO}$ size decreases with the growth of query size. It is because that $\mathcal{VO}$ size is dominated by the candidate graphs and larger queries result in fewer candidate graphs. We also observe that the $\mathcal{VO}$ size increases as the fanout increases, because that the larger the fanout, the more candidates produced. It is consistent to the observations of MRTree and MBTree.

8.4 Effectiveness of Optimizations on GMTree

In this experiment, we focus on the AIDS datasets, as other datasets exhibit similar performance characteristics.

Pivot selection. Figs. 16(a)-(b) present the pivot size decision and the sample number decision in our pivot selection, respectively. The fanout of `GMTree` is 16 and σ is 1, in this experiment. The x-axis of Fig. 16(a) is pivot sizes; the left y-axis is the percentage of candidates out of the whole database and the right y-axis is the query time. Fig. 16(a) shows the balance between the number of candidates and the query time on diverse pivot sizes. Note that the query time here is linear to $|\mathcal{J}|$+$|\mathcal{VC}|$. The reducing line shows that the number of candidates reduces with the growth of pivot size, whereas the increasing line indicates that the query time increases. Fig. 16(a) shows the optimal pivot size is ∼10 that is consistent to our pivot size decision model in Sec. 7.2. In Fig. 16(b), the x-axis is sample sizes; and y-axis is the percentage of candidates out of the whole database. The line in Fig. 16(b) converges after 10 samples. This means that we only take a small number of samples to obtain stable performance, which is consistent to our analysis in Sec. 7.2.

Authenticated MCS computation. To clearly illustrate the performance of our authenticated `MCS` computation between q and pivots, the query time in this experiment does not include the time to handle candidate graphs, and the $\mathcal{VO}$ is just that caused by pivots. The fanout of `GMTree` is 16 and σ is 1, in this experiment. Figs. 16(c)-(d) show the query time and the $\mathcal{VO}$ size of our `auth-VC-mcs`, respectively. Recall that if the declared `MCS` size is larger than $\beta \times |q|$, we use the `Enum` method; otherwise, the `VC-mcs` method. Fig. 16(c) presents that the query time first reduces and then increases with the growth of β. This shows the balance between the `Enum` and the `VC-mcs`: (i) the smaller β, the more `Enum` is used; and (ii) the `Enum` is more efficient for large `MCS`s, whereas `VC-mcs` is better for small `MCS`s. Fig. 16(d) shows that the $\mathcal{VO}$ size increases with the growth of β. This is because that the $\mathcal{VO}$ needed by the `Enum` is smaller than that of the `VC-mcs`.

Figs. 16(e)-(f) show the effectiveness of our `auth-VC-mcs`. Fig. 16(e) shows that while taking 12% more query time of $\mathcal{SP}$, the `auth-VC-mcs` saves 40% client's authentication time. It is desirable in practice as the $\mathcal{SP}$ often has an advanced computation ability, whereas the client does not. Fig. 16(f) shows that the $\mathcal{VO}$ overhead of the `auth-VC-mcs` is 10K bytes, which is just 2% of the overall $\mathcal{VO}$.

Remarks. Our experiments reveal the characteristics of `GMTree` with respect to its parameters. The $\mathcal{SP}$ can easily offer performance options such as "*query time optimized*", "*balanced*", and "*network optimized*" for data owners/clients to choose from and build the corresponding indexes offline.

9 Conclusion

This paper studies the authenticated subgraph similarity search in outsourced graph databases. We transform the subgraph similarity search into a search in a graph metric space and propose `GMTree`. Our novelties and technicalities reside in the authentication techniques. First, candidate graphs determined by `GMTree` are often localized. Secondly, we propose a pivot selection which allows using data subgraphs as pivots. Thirdly, we propose an authenticated `MCS` computation to reduce the computation at clients. Our experiments show that `GMTree` is efficient and the $\mathcal{VO}$s are small. In future work, we will investigate supergraph similarity search.

Acknowledgement. This work is partly supported by GRF210510.

References

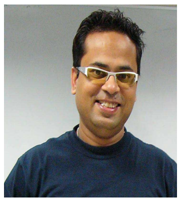
[1] X. Yan, P. S. Yu, and J. Han, "Substructure similarity search in graph databases," in *SIGMOD*, 2005, pp. 766–777.
[2] H. Shang, X. Lin, Y. Zhang, J. X. Yu, and W. Wang, "Connected substructure similarity search," in *SIGMOD*, 2010, pp. 903–914.
[3] Y. Zhu, L. Qin, J. X. Yu, and H. Cheng, "Finding top-k similar graphs in graph databases," in *EDBT*, 2012, pp. 456–467.
[4] Y. Yuan, G. Wang, L. Chen, and H. Wang, "Efficient subgraph similarity search on large probabilistic graph databases," *Proc. VLDB Endow.*, vol. 5, no. 9, pp. 800–811, 2012.
[5] D. W. Williams, J. Huan, and W. Wang, "Graph database indexing using structured graph decomposition," in *ICDE*, 2007, pp. 976–985.
[6] H. He and A. K. Singh, "Closure-tree: An index structure for graph queries," in *ICDE*, 2006, pp. 38–38.
[7] S. Ranu and A. K. Singh, "Indexing and mining topological patterns for drug discovery," in *EDBT*, 2012, pp. 562–565.
[8] T. Yuanyuan, M. R. C., S. Carlos, S. D. J., and P. J. M., "Saga: a subgraph matching tool for biological graphs," *Bioinformatics*, vol. 23, no. 2, pp. 232–239, 2007.
[9] M. Mongiovi, R. D. Natale, R. Giugno, A. Pulvirenti, A. Ferro, and R. Sharan, "Sigma: a set-cover-based inexact graph matching algorithm." *J. Bioinformatics and Computational Biology*, no. 2, pp. 199–218, 2010.
[10] NCBI, "PubChem," *http://pubchem.ncbi.nlm.nih.gov*, 2012.
[11] B. A. Grning, C. Senger, A. Erxleben, S. Flemming, and S. Gnther, "Compounds in literature (cil): screening for compounds and relatives in pubmed," *Bioinformatics*, pp. 1341–1342, 2011.
[12] O2I, *http://www.outsource2india.com*, 2013.
[13] I. Outsourcing, *http://www.informaticsoutsourcing.com*, 2013.
[14] Silico., *http://wbbiotech.nic.in/wbbiotech/writereaddata/silicogene.htm*, 2013.
[15] Dataoutsourcingindia, *http://www.dataoutsourcingindia.com*, 2013.
[16] H. Pang and K. L. Tan, "Query answer authentication," in *Morgan & Claypool Publishers*, 2012.
[17] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces," in *VLDB*, 1997, pp. 426–435.
[18] J. K. Uhlmann, "Satisfying general proximity/similarity queries with metric trees," *Inf. Process. Lett.*, vol. 40, no. 4, pp. 175–179, 1991.
[19] P. N. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *SODA*, 1993, pp. 311–321.
[20] F. N. Abu-Khzam, N. F. Samatova, M. A. Rizk, and M. A. Langston, "The maximum common subgraph problem: Faster solutions via vertex cover," in *AICCSA*, 2007, pp. 367–373.
[21] H. Pang, A. Jain, K. Ramamritham, and K. lee Tan, "Verifying completeness of relational query results in data publishing," in *SIGMOD*, 2005, pp. 407–418.
[22] Y. Yang, S. Papadopoulos, D. Papadias, and G. Kollios, "Spatial outsourcing for location-based services," in *ICDE*, 2008, pp. 1082–1091.
[23] W. Cheng and K.-L. Tan, "Query assurance verification for outsourced multi-dimensional databases," *J. Comput. Secur.*, vol. 17, no. 1, pp. 101–126, 2009.
[24] Y. Yang, D. Papadias, S. Papadopoulos, and P. Kalnis, "Authenticated join processing in outsourced databases," in *SIGMOD*, 2009, pp. 5–18.
[25] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin, "Dynamic authenticated index structures for outsourced databases," in *SIGMOD*, 2006.
[26] H. Pang and K. Mouratidis, "Authenticating the query results of text search engines," *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 126–137, 2008.
[27] M. L. Yiu, Y. Lin, and K. Mouratidis, "Efficient verification of shortest path search via authenticated hints." in *ICDE*, 2010, pp. 237–248.
[28] M. Goodrich, R. Tamassia, N. Triandopoulos, and R. Cohen, "Authenticated data structures for graph and geometric searching," in *Topics in Cryptology CT-RSA 2003*, 2003, vol. 2612, pp. 295–313.
[29] A. Kundu and E. Bertino, "How to authenticate graphs without leaking," in *EDBT*, 2010, pp. 609–620.
[30] ——, "Structural signatures for tree data structures," *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 138–150, 2008.
[31] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine, "A general model for authenticated data structures," *Algorithmica*, vol. 39, no. 1, pp. 21–41, Jan. 2004.
[32] H. Jiang, H. Wang, P. Yu, and S. Zhou, "Gstring: A novel approach for efficient search in graph databases," in *ICDE*, 2007, pp. 566–575.
[33] M. Mann, F. Nahar, H. Ekker, R. Backofen, P. Stadler, and C. Flamm, "Atom mapping with constraint programming," in *Principles and Practice of Constraint Programming*, 2013, vol. 8124, pp. 805–822.
[34] Y. Tian and J. M. Patel, "Tale: A tool for approximate large graph matching," in *ICDE*, 2008, pp. 963–972.
[35] A. Khan, N. Li, X. Yan, Z. Guan, S. Chakraborty, and S. Tao, "Neighborhood based fast graph search in large networks," in *SIGMOD*, 2011, pp. 901–912.
[36] X. Yan, P. S. Yu, and J. Han, "Graph indexing: a frequent structure-based approach," in *SIGMOD*, 2004, pp. 335–346.
[37] H.-C. Ehrlich and M. Rarey, "Maximum common subgraph isomorphism algorithms and their applications in molecular science: a review," *WIREs Comp. Mol. Sci.*, vol. 1, no. 1, pp. 68–79, 2011.
[38] P. Vismara and B. Valery, "Finding maximum common connected subgraphs using clique detection or constraint satisfaction algorithms," in *Comm. in Comp. and Info. Sci.*, 2008, vol. 14, pp. 358–368.
[39] J. W. Raymond and P. Willett, "Maximum common subgraph isomorphism algorithms for the matching of chemical structures," *Journal of Computer-Aided Molecular Design*, vol. 16, pp. 521–533, 2002.
[40] J. Lauri, "Subgraphs as a measure of similarity," in *Structural Analysis of Complex Networks*, 2011, pp. 319–334.
[41] H. Bunke and K. Shearer, "A graph distance metric based on the maximal common subgraph," *Patt. Rec. Lett.*, vol. 19, no. 3-4, pp. 255–259, 1998.
[42] J. T.-L. Wang, X. Wang, K.-I. Lin, D. Shasha, B. A. Shapiro, and K. Zhang, "Evaluating a class of distance-mapping algorithms for data mining and clustering," in *SIGKDD*, 1999, pp. 307–311.
[43] R. C. Merkle, "A certified digital signature," in *CRYPTO*, 1989.
[44] G. R. Hjaltason and H. Samet, "Index-driven similarity search in metric spaces (survey article)," *TODS*, vol. 28, no. 4, pp. 517–580, 2003.
[45] K. Mouratidis, D. Sacharidis, and H. Pang, "Partially materialized digest scheme: an efficient verification method for outsourced databases," *The VLDB Journal*, vol. 18, no. 1, pp. 363–381, 2009.
[46] E. B. Krissinel and K. Henrick, "Common subgraph isomorphism detection by backtracking search," *Software: Practice and Experience*, vol. 34, no. 6, pp. 591–607, 2004.
[47] SIVALab, "VFLib," *http://www.cs.sunysb.edu/ algorith/implement/vflib/implement.shtml*, 2013.
[48] RCaH, "CCP4," *http://www.ccp4.ac.uk/index.php*, 2013.
[49] J. Cheng, Y. Ke, W. Ng, and A. Lu, "Fg-index: towards verification-free query processing on graph databases," in *SIGMOD*, 2007, pp. 857–872.
[50] W.-S. Han, J. Lee, M.-D. Pham, and J. X. Yu, "igraph: a framework for comparisons of disk-based graph indexing techniques," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 449–459, Sep. 2010.

Yun Peng is a PhD student in the Department of Computer Science, Hong Kong Baptist University. He received his BSci and MPhil degrees in Computer Science from Shandong University in 2006 and Harbin Institute of Technology (HIT) in 2008, respectively. His research interests include graph-structured databases. He is a member of the Database Group at Hong Kong Baptist University (http://www.comp.hkbu.edu.hk/∼db/).

Zhe Fan is a PhD student in the Department of Computer Science, Hong Kong Baptist University. He received his BEng degree in Computer Science from Sourth China University of Technology in 2011. His research interests include graph-structured databases. He is a member of the Database Group at Hong Kong Baptist University. (http://www.comp.hkbu.edu.hk/∼db/).

Byron Choi received the bachelor of engineering degree in computer engineering from the Hong Kong University of Science and Technology (HKUST) in 1999 and the MSE and PhD degrees in computer and information science from the University of Pennsylvania in 2002 and 2006, respectively. He is now an assistant professor in the Department of Computer Science at the Hong Kong Baptist University.

Jianliang Xu is an associate professor in the Department of Computer Science, Hong Kong Baptist University. He received his BEng degree in computer science and engineering from Zhejiang University, Hangzhou, China, in 1998 and his PhD degree in computer science from Hong Kong University of Science and Technology in 2002. He held visiting positions at Pennsylvania State University and Fudan University.



Sourav S Bhowmick is an Associate Professor in the School of Computer Engineering, Nanyang Technological University. He is a Visiting Associate Professor at the Biological Engineering Division, Massachusetts Institute of Technology. He held the position of Singapore-MIT Alliance Fellow in Computation and Systems Biology program (05'-12'). He received his Ph.D. in computer engineering in 2001.

APPENDIX A PROOFS

In this appendix, we present all the proofs skipped in the main body of this paper.

A.1 Proof of Lemma 4.1

Lemma 4.1: *Let q, G_1 and G_2 denote the query graph and two data graphs, respectively. Given a similarity threshold t, if $1 - \frac{|mcs(G_1,G_2)|}{|G_1|} - d(q, G_1) > t$, then $d_s(q, G_2) > t$.* ■

Proof: We conduct the proof through a detailed case analysis.

Case A: $|q| \geq |G_1|$. It follows that

$$1 - \frac{|mcs(G_1,G_2)|}{|q|} \geq 1 - \frac{|mcs(G_1,G_2)|}{|G_1|} \tag{6}$$

Case A.1: $|G_2| \geq |q|$.

In this case, G_2 has at least one subgraph of size $|q|$. Let $S = \{S_1, S_2, \ldots, S_n\}$ denote all the subgraphs of G_2 of size $|q|$. There must exist at least one S^m in S, such that (i) $|mcs(q, G_2)| = |mcs(q, S^m)|$, and (ii) $|mcs(q, S^m)| \geq |mcs(q, S_i)|$ for $S_i \in S$.

By (i), $d_s(q, G_2) = d(q, S^m)$ and by (ii), $d(q, S_i) \geq d(q, S^m)$ for $S_i \in S$. We then have the following necessary and sufficient condition

$$d_s(q, G_2) > t \Leftrightarrow d(q, S_i) > t, \text{ for all } S_i \in S \tag{7}$$

Now our task is to prove $d(q, S_i) > t$ for all $S_i \in S$.

Since S_i is a subgraph of G_2, $|mcs(G_1, G_2)| \geq |mcs(G_1, S_i)|$ for all $S_i \in S$. Hence, for $\forall S_i \in S$,

$$1 - \frac{|mcs(G_1,S_i)|}{|S_i|} \geq 1 - \frac{|mcs(G_1,G_2)|}{|S_i|} \tag{8}$$

Because of $|q| = |S_i|$, (8) becomes

$$1 - \frac{|mcs(G_1,S_i)|}{|S_i|} \geq 1 - \frac{|mcs(G_1,G_2)|}{|q|} \tag{9}$$

Combine (6) and (9) together, we have

$$1 - \frac{|mcs(G_1,S_i)|}{|S_i|} \geq 1 - \frac{|mcs(G_1,G_2)|}{|G_1|} \tag{10}$$

Since $|G_1| \leq |S_i|$, $d(G_1, S_i) = 1 - \frac{|mcs(G_1,S_i)|}{|S_i|}$, (10) becomes

$$d(G_1, S_i) > 1 - \frac{|mcs(G_1,G_2)|}{|G_1|}$$

Since $1 - \frac{|mcs(G_1,G_2)|}{|G_1|} - d(q, G_1) > t$ is given in Lemma 4.1, $d(G_1, S_i) - d(q, G_1) > t$. Finally, considering $d(q, S_i) \geq d(G_1, S_i) - d(q, G_1)$ by the triangle inequality, we obtain $d(q, S_i) > t$, for all $S_i \in S$.

Case A.2: $|G_2| < |q|$.

In this case, we have $d_s(q, G_2) = d(q, G_2)$, as $\max(|q|, |G_2|) = |q|$. Our task is then to prove $d(q, G_2) > t$.

Case A.2.1: $|G_1| \leq |G_2| < |q|$.

Since $d(G_1, G_2) = 1 - \frac{|mcs(G_1,G_2)|}{|G_2|} > 1 - \frac{|mcs(G_1,G_2)|}{|G_1|}$, then $d(G_1, G_2) - d(q, G_1) > t$ by the condition given in Lemma 4.1. By the triangle inequality, $d(q, G_2) > t$.

TABLE 1
Table of frequently used notations

$\mathcal{D}$	graph database	$\mathcal{U}$	graph metric space
G and g	data graph and its pointer	G^*	augmented graph of graph G
$\mathcal{T}_p$	set of sub-GMTrees of node with pivot p	$\mathcal{R}_p$	set of radii of node with pivot p
s_{root}	signature of GMTree	$h()$	hash function
d	graph distance	d_s	subgraph distance
s_{sig}	size of a signature	s_{rev}	size of a relevant content of a node
s_{rad}	size of a radius value	s_{ptr}	size of a pointer
s_h	size of a digest	s_G	size of a data graph
q	query graph	t	threshold of query
C_q	candidate set of query (q, t)	R_q	answer set of query (q, t)
c	fanout of GMTree		

Case A.2.2: $|G_2| < |G_1| < |q|$.

In this case, $d(G_1, G_2) = 1 - \frac{mcs(G_1,G_2)}{|G_1|}$. Hence, $d(G_1, G_2) - d(q, G_1) > t$ by the condition given in Lemma 4.1. By the triangle inequality, $d(q, G_2) > t$.

Case B: $|q| < |G_1|$.

Case B.1: $|G_2| \geq |q|$.

Let $S = \{S_1, S_2, \ldots, S_n\}$ denote the set of subgraphs of G_2 of the size q. The necessary and sufficient condition (7) still applies. Our task is to prove $d(q, S_i) > t$ for all $S_i \in S$.

Since S_i is a subgraph of G_2, $|mcs(G_1, G_2)| \geq |mcs(G_1, S_i)|$. Then, for all $S_i \in S$,

$$1 - \frac{|mcs(G_1,G_2)|}{|G_1|} \leq 1 - \frac{|mcs(G_1,S_i)|}{|G_1|} = d(G_1, S_i)$$

Since $1 - \frac{|mcs(G_1,G_2)|}{|G_1|} - d(q, G_1) > t$ is given in Lemma 4.1, $d(G_1, S_i) - d(q, G_1) > t$ for all $S_i \in S$. Thus, by the triangle inequality, $d(q, S_i) > t$ for all $S_i \in S$.

Case B.2: $|G_2| < |q|$.

In this case, $d_s(q, G_2) = d(q, G_2)$. Since $|G_1| > |G_2|$, $d(G_1, G_2) = 1 - \frac{|mcs(G_1,G_2|)}{|G_1|}$. Thus, $d(G_1, G_2) - d(q, G_1) > t$ by the condition given in Lemma 4.1. By the triangle inequality, $d(q, G_2) > t$.

□

A.2 Proof of Theorem 4.2

Theorem 4.2: *Given a query q, an augmented graph G_1^* and a graph G_2, if $d(G_1^*, G_2) - d(q, G_1^*) > t$, then $d_s(q, G_2) > t$.* ■

Proof: We recall the definition of augmented graph Definition 4.1 as follows.

An *augmented graph* G_1^* with respect to G_2 is defined as follows:

- $G_1^* = G_1$ if $|G_1| \geq |G_2|$;
- Otherwise, $G_1^* = G_1 \cup A$, where A is an augmented subgraph having nodes and edges with labels never occurred in G_2 and possible queries, until $|G_1^*| = |G_2|$.

We conduct the proof by applying Lemma 4.1 and Definition 3.1 as follows.

$$
\begin{aligned}
& d(G_1^*, G_2) - d(q, G_1^*) > t \\
\Rightarrow\quad & 1 - \frac{|mcs(G_1^*, G_2)|}{\max\{|G_1^*|, |G_2|\}} - d(q, G_1^*) > t \\
& \qquad \text{by Definition 3.1} \\
\Rightarrow\quad & 1 - \frac{|mcs(G_1^*, G_2)|}{|G_1^*|} - d(q, G_1^*) > t \\
& \qquad \text{as } |G_1^*| \geq |G_2| \\
\Rightarrow\quad & d_s(q, G_2) > t \\
& \qquad \text{by Lemma 4.1}
\end{aligned}
$$

□

A.3 Calculation of H^b in Sec. 6.4

We first present the formula for the size of H^b and the number of candidate graphs of GMTree. Then, we plug in some numbers to construct an example presented in Sec. 6.4.

Proposition A.3: *Given a* GMTree *T with a fanout c indexing a graph database $\mathcal{D}$, if each node in T has a fraction $k \in (0,1)$ of sub-*GMTree*s visited, the number of candidate graphs is*

$$|\mathcal{D}| \times k^{\log_c |\mathcal{D}| - 1}$$

and the number of digests needed in H^b is

$$\frac{(1-k)c \times [1 - (kc)^{\log_c |\mathcal{D}| - 1}]}{1 - kc}.$$

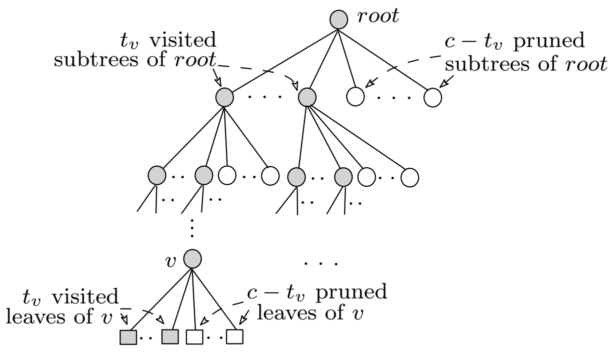
■

Proof: Since GMTree T is a complete tree, T has $|\mathcal{D}|/c$ leaves and T's height $h = \log_c |\mathcal{D}|$.
Regarding the number of candidate graphs: Since each node of T has a fraction k of subtrees visited, the total number of leaf nodes visited is $(kc)^{h-1}$. Since each leaf node contains c data graphs, the number of candidate graphs is $c^h \times k^{h-1} = |\mathcal{D}| \times k^{\log_c |\mathcal{D}| - 1}$.
Regarding the number of digests in H^b: Since each node of T has a fraction k of subtrees visited, each node has $(1-k)c$ subtrees pruned. Therefore, T's i-th level has $(1-k)c \times (kc)^{i-1}$ subtrees pruned for $1 \leq i \leq h-1$. Since we need a digests for each pruned subtree in H^b, the total number of digests in H^b is $\times \sum_{i=1}^{h-1} (1-k)c \times (kc)^{i-1} = \frac{(1-k)c \times [1-(kc)^{\log_c |\mathcal{D}|-1}]}{1-kc}$
□

Now we are ready to plug in some numbers to construct the example presented in Sec. 6.4. If $k = 3/4, c = 8, |\mathcal{D}| = 10000$, the number of candidate graphs is about 3700 and GMTree needs 186 digests in H^b to authenticate the presence of all the candidate graphs. In comparison, if the candidate graphs and non-candidate graphs are stored alternately, each candidate graph needs a digest and hence 3700 digests are needed in the worst case. Therefore, GMTree just needs 5% digests of the worst case.

A.4 Proof of Cost Model of $\mathcal{VO}$ Size in Sec. 6.4

Let s_{sig}, s_{rev}, s_{rad}, s_{ptr}, s_h and s_G denote the sizes of a signature, a relevant content of a node, a radius value, a pointer, a digest and a data graph, respectively. Let $V_{visited}$ denote the set of visited nodes of GMTree. Please refer to Tbl. 1 for the frequently used notations.



root
t_v visited subtrees of root
$c - t_v$ pruned subtrees of root
v
t_v visited leaves of v
$c - t_v$ pruned leaves of v

Fig. 17. Illustration for $\mathcal{VO}$ Size Analysis

Proposition A.4: *The overall $\mathcal{VO}$ size of a query (q,t) on a* GMTree *T of a fanout c is:*

$$
\begin{aligned}
|\mathcal{VO}| = & \sum_{\substack{v \text{ is visited} \\ v \text{ is internal}}} \left(s_{rev} + (c-1)s_{rad}\right) + \sum_{v \text{ is visited}} s_{ptr} - s_{ptr} && //H^v \textit{ for internal} \\
& + 2s_h\Big(\sum_{\substack{v \text{ is visited} \\ v \text{ is internal}}} c - (|V_{visited}| - 1)\Big) && //H^b \\
& + \sum_{\substack{v \text{ is visited} \\ v \text{ is leaf}}} \left((c+1)s_h + c\, s_{ptr}\right) && //H^v \textit{ for leaf} \\
& + \sum_{\substack{v \text{ is visited} \\ v \text{ is leaf}}} c(s_h + s_G + s_i) && //M \textit{ and } N \\
& + s_{sig} && //s_{root}
\end{aligned}
$$

■

Proof: Procedure auth_similarity performs a traversal on T and constructs $\mathcal{VO}$ simultaneously by Definition 6.3. During traversal, when auth_similarity visits a node v (*e.g.*, grey circles and rectangles in Fig. 17), auth_similarity adds several terms to verify v in $\mathcal{VO}$. Specifically,

(i) if v is an internal node (*e.g.*, the grey circles in Fig. 17),
  - auth_similarity adds s_{rev} bytes in H^v for the relevant content of v (For simplicity of analysis, we do not consider the hints that optimize MCS verification, which affects the analysis in a non-trivial manner);
  - Suppose t_v subtrees of v are not pruned in Lines 11-13 of Fig. 7 (*e.g.*, the grey children of $root$ in Fig. 17). auth_similarity adds $t_v \times s_{ptr}$ bytes for the subtree not pruned to H^v;
  - auth_similarity adds $(c-1) \times s_{rad}$ bytes for the radii to H^v. (Since the first radius is always zero, we do not add it to $\mathcal{VO}$ for saving);
  - auth_similarity also adds $2(c - t_v) \times s_h$ bytes for the pruned subtrees (*e.g.*, the white children of $root$ in Fig. 17), one s_h for the digest of each pruned subtrees and one s_h for the digests of each radius to H^b.

(ii) Otherwise, v is a leaf (*e.g.*, the grey rectangles in Fig. 17),
  - auth_similarity adds $(1+c) \times s_h + c \times s_{ptr}$ bytes to H^v, including $h(v)$, c pointers and c digest of pointers;
  - auth_similarity also adds $c \times s_G + c \times s_h + c \times s_i$ bytes to M and N for the answers, non-answers and their digests. (Note that the MCS mappings m in M is not included for analysis simplicity as m's size is at most s_G bytes.)

At the end of the traversal, s_{sig} bytes for s_{root} are added to $\mathcal{VO}$.

In all, the size of $\mathcal{VO}$ is

$$\begin{aligned}
|\mathcal{VO}| \;=\; & \sum_{\substack{v \textit{ is visited} \\ v \textit{ is internal}}} \big[s_{rev} + t_v s_{ptr} + (c-1)s_{rad}\big] \quad //H^v \text{ for internal} \\
+ & \sum_{\substack{v \textit{ is visited} \\ v \textit{ is internal}}} \big[2(c-t_v)s_h\big] \quad //H^b \\
+ & \sum_{\substack{v \textit{ is visited} \\ v \textit{ is leaf}}} \big[(c+1)s_h + c s_{ptr}\big] \quad //H^v \text{ for leaf} \\
+ & \sum_{\substack{v \textit{ is visited} \\ v \textit{ is leaf}}} \big[c s_G + c s_h + c s_i\big] \quad //M \text{ and } N \\
+ & \; s_{sig} \quad //s_{root}
\end{aligned} \tag{11}$$

Since t_v is the number of visited children of v, hence $\sum_{\substack{v \textit{ is visited} \\ v \textit{ is internal}}} t_v$ counts the number of nodes visited in traversal excluding the root. That is

$$\sum_{\substack{v \textit{ is visited} \\ v \textit{ is internal}}} t_v = |V_{visited}| - 1$$

Therefore, Formula (11) becomes

$$\begin{aligned}
|\mathcal{VO}| = & \sum_{\substack{v \textit{ is visited} \\ v \textit{ is internal}}} \big(s_{rev} + (c-1)s_{rad}\big) + \sum_{v \textit{ is visited}} s_{ptr} - s_{ptr} \quad //H^v \text{ for internal} \\
& + 2s_h\Big(\sum_{\substack{v \textit{ is visited} \\ v \textit{ is internal}}} c - (|V_{visited}|-1)\Big) \quad //H^b \\
& + \sum_{\substack{v \textit{ is visited} \\ v \textit{ is leaf}}} \big((c+1)s_h + c\, s_{ptr}\big) \quad //H^v \text{ for leaf} \\
& + \sum_{\substack{v \textit{ is visited} \\ v \textit{ is leaf}}} c(s_h + s_G + s_i) \quad //M \text{ and } N \\
& + \; s_{sig} \quad //s_{root}
\end{aligned}$$

□

A.5 Proof of Formula (4) in Sec. 7.2

Proposition A.5: *Suppose $\hat{a}$, a and b are the sample minimum, population minimum and population maximum, respectively. Suppose further the graph distance between any two graphs is uniformly distributed in $[a,b]$. Let s denote the sample size and ϵ bound the error of $\hat{a}$ from a. Then, for any $0 < \epsilon < b-a$, we have*

$$Pr(\hat{a} - a \le \epsilon) \ge 1 - (1-\epsilon)^s.$$ ■

Proof: Since the graph distance d between any two graphs is uniformly distributed in $[a,b]$, let $p(d)$ denote the probability density function of d, we have

$$p(d) = \begin{cases} \frac{1}{b-a} & \text{if } x \in (a,b) \\ 0 & \text{otherwise} \end{cases}$$

Because we randomly choose s samples, the sample minimum $\hat{a}$ is also a random variable. Let $Pr(\hat{a} = \omega)$ denote the probability of ω is the sample minimum. The probability of $\hat{a} = \omega$ equals to

$$Pr(\hat{a} = \omega) = s \times \big[1 - Pr(d < \omega)\big]^{s-1} \times p(\omega), \tag{12}$$

where $\big[1 - Pr(d<\omega)\big]^{s-1}$ means $s-1$ samples are larger than ω and the coefficient s means ω can occur at any sample.

Since $\hat{a}$ is the sample minimum, $a \le \hat{a} \le b$ and random variable $\hat{a} - a$ measures the error of $\hat{a}$ from a. Since $0 \le a \le b \le 1$, the error $\hat{a} - a \le b - a \le 1$.

Given $\epsilon \in (0, b-a)$ as a bound of error, the probability of $\hat{a} - a$ is bounded by ϵ equals to

$$\begin{aligned}
Pr(\hat{a} < a + \epsilon) &= \int_0^{a+\epsilon} Pr(\hat{a} = z)dz \\
&= \int_a^{a+\epsilon} Pr(\hat{a} = z)dz
\end{aligned} \tag{13}$$

Integrating Equation (12), Equation (13) becomes

$$\begin{aligned}
& \int_a^{a+\epsilon} s\big[1 - Pr(d<z)\big]^{s-1} p(z)dz \\
= & \int_a^{a+\epsilon} s\big[1 - \frac{z-a}{b-a}\big]^{s-1} \frac{1}{b-a} dz \\
= & \int_a^{a+\epsilon} s(\frac{1}{b-a})^s (b-z)^{s-1} dz \\
= & (\frac{1}{b-a})^s \int_a^{a+\epsilon} s(b-z)^{s-1} dz \\
= & -(\frac{1}{b-a})^s \int_a^{a+\epsilon} s(b-z)^{s-1} d(b-z) \\
= & -(\frac{1}{b-a})^s (b-z)^s \big|_a^{a+\epsilon} \\
= & -(\frac{1}{b-a})^s \big[(b-a-\epsilon)^s - (b-a)^s\big] \\
= & 1 - (1 - \frac{\epsilon}{b-a})^s \\
& \text{because } 0 < b-a < 1, \epsilon > 0 \\
> & 1 - (1-\epsilon)^s
\end{aligned}$$

□

A.6 Proof of Soundness and Completeness of Procedure `auth`

Theorem A.6: *Procedure* `auth` *is sound and complete.* ■

Proof: (*Proof of soundness*) Assume that a graph in the query result in $\mathcal{VO}$ is modified or bogus. Since the hash function is assumed to be one-way, the correct hash of the leaf node (Lines 06,15 of `auth_aux`) cannot be synthesized. Hence, h_{root} will not agree with the signature s_{root} provided by the $\mathcal{DO}$, and the malicious modification is detected by the client (Line 02 of `auth`).

Given the graphs returned by the $\mathcal{SP}$ are not modified, Lines 09-14 of `auth_aux` guarantee the soundness of the answers.

(*Proof of completeness*) Let G is an answer of Q but not included M. Suppose v_G is the leaf node containing G. Either the content of v_G or the hash containing v_G is stored in $\mathcal{VO}$. In the former case, G will be extracted from N in Line 05 of `auth_aux` and detected in Lines 12-14 of `auth_aux`. In the latter case, since v_G contains G, v_G must overlap with the query q. By applying the similar argument, one of the ancestors of v_G or v_G itself overlaps with q, but it is wrongly included in the $\mathcal{VO}.H^b$ by the $\mathcal{SP}$. Then, there are only two possible cases. Firstly, the number of pruned subtrees (*i.e.*, t_v) determined by the pruning condition in Line 22 of `auth_aux` will not match the number of digests in H^b (Lines 25-27 of `auth_aux`). Secondly, the radius list has been forged, which makes Line 22 of `auth_aux` passes. In either case, the digest cannot be synthesized. The missing result will be detected by Line 02 of `auth`. □

A.7 Proof of Correctness of `auth-VC-mcs`

Theorem A.7: `auth-VC-mcs` *is correct.* ■

Proof: Foremost, we assume the correctness of `VC-mcs` [20]. From the procedure of the client's authentication, we note that the client reads the following from $\mathcal{VO}$: the graph p, the maximal independent sets $mis(p)$ of p to construct the bigraphs, the maximum matchings J_M's and minimum vertex covers VC_M's of the bigraphs and the signatures. Other structures used in the authentication process are constructed by the client himself. Since the signatures cannot be tampered with, we can prove the correctness of `auth-VC-mcs` by proving the correctness of p, $mis(p)$ and J_M and VC_M, respectively.

Case A. If p is tampered with. The synthesized digest of the root of `GMTree` will not agree with the signature s_{root} in $\mathcal{VO}$ as detected in Procedure `auth` in Sec. 6.

Case B. If the $mis(p)$ of p is tampered with. The client can easily detect it by using $mis(p)$'s signature recorded in $\mathcal{VO}$ and $\mathcal{DO}$'s public key.

Case C. If the maximum matching J_M or the minimum vertex cover of a bigraph $B_{(p,q,M)}$ is tampered with. The client can easily detect it by checking whether $|J_M| = |VC_M|$ since we have a theorem in graph theories that the size of the maximum matching of a bigraph B equals to the size of the minimum vertex cover of B.

We have proved the correctness of q, $mis(q)$ and J_Ms and VC_Ms in $\mathcal{VO}$, and hence the correctness of `auth-VC-mcs` is proved.

A.8 Proof of NP-hardness of the Pivot Selection Problem

Theorem A.8: *The Pivot Selection Problem is NP-hard.* ■

Proof: Let us recall the definition of the pivot selection problem as follows.

Given a set of graphs $\mathcal{D} = \{G_1, G_2, \ldots, G_m\}$ and the set of subgraphs $\mathcal{S} = \{S | S \subset G, G \in \mathcal{D}\}$, select one graph p as pivot from $\mathcal{D} \cup \mathcal{S}$ to minimize the cost function

$$X = \max_{i=1 \ldots m} \{d(p^*, G_i)\},$$

where p^* is the augmented pivot constructed from p as discussed in Sec. 4.

We first prove that the simplest case of the pivot selection problem (*i.e.*, when $m = 2$) is NP-hard. Then, the general pivot selection problem is readily NP-hard.

When $m = 2$, $\mathcal{D} = \{G_1, G_2\}$ and $\mathcal{S} = \{S | S \subset G_1 \vee S \subset G_2\}$. We have (i) the graph $p = mcs(G_1, G_2)$ is an optimum pivot and (ii) all other optimum pivots are supergraphs of $mcs(G_1, G_2)$. We prove (i) and (ii) separately as follows.

(i) $p = mcs(G_1, G_2)$ is an optimum pivot

We prove it by contradiction. Suppose there is another graph $p' \in \mathcal{D} \cup \mathcal{S}$ which is better than p, *i.e.*,

$$\begin{aligned} & \max\{1 - \tfrac{|mcs(p,G_1)|}{\max\{|p^*|,|G_1|\}}, 1 - \tfrac{|mcs(p,G_2)|}{\max\{|p^*|,|G_2|\}}\} \\ > \ & \max\{1 - \tfrac{|mcs(p',G_1)|}{\max\{|p'^*|,|G_1|\}}, 1 - \tfrac{|mcs(p',G_2)|}{\max\{|p'^*|,|G_2|\}}\} \end{aligned} \quad (14)$$

Since $|p^*| = |p'^*| = \max\{|G_1|, |G_2|\}$ by Definition 4.1, (14) becomes

$$\begin{aligned} & \max\{1 - |mcs(p, G_1)|, 1 - |mcs(p, G_2)|\} \\ > \ & \max\{1 - |mcs(p', G_1)|, 1 - |mcs(p', G_2)|\} \\ \Leftrightarrow \ & \min\{|mcs(p, G_1)|, |mcs(p, G_2)|\} \\ < \ & \min\{|mcs(p', G_1)|, |mcs(p', G_2)|\} \end{aligned} \quad (15)$$

Since $p = mcs(G_1, G_2)$, (15) becomes

$$|mcs(G_1, G_2)| < \min\{|mcs(p', G_1)|, |mcs(p', G_2)|\} \quad (16)$$

However, (16) is impossible. The reason is that p' can only be G_1, G_2 or their subgraphs, which are analyzed one-by-one as follows.

Case A: If $p' = G_1$, $\min\{|mcs(p', G_1)|, |mcs(p', G_2)|\} = |mcs(G_1, G_2)|$

Case B: If $p' = G_2$, similar to Case A.

Case C: If p' is a subgraph of G_1, $|mcs(p', G_1)| \geq |mcs(p', G_2)|$ and $|mcs(p', G_2)| \leq |mcs(G_1, G_2)|$ Hence, $|mcs(G_1, G_2)| \geq \min\{|mcs(p', G_1)|, |mcs(p', G_2)|\}$.

Case D: If p' is a subgraph of G_2, similar to Case C.

We have found a contradiction and hence $p = mcs(G_1, G_2)$ is an optimum pivot.

(ii) all other optimum pivots are supergraphs of $mcs(G_1, G_2)$

We also prove it by contradiction. Suppose there is an optimum pivot $p'' \in \mathcal{D} \cup \mathcal{S}$ that does not contain $p = mcs(G_1, G_2)$. Then, we have the following.

$$\begin{aligned} & |mcs(G_1, G_2)| > \min\{|mcs(p'', G_1)|, |mcs(p'', G_2)|\} \\ \Leftrightarrow \ & \max\{1 - |mcs(p, G_1)|, 1 - |mcs(p, G_2)|\} \\ < \ & \max\{1 - |mcs(p'', G_1)|, 1 - |mcs(p'', G_2)|\} \\ & \qquad \text{by Definition 4.1 } |p^*| = |p''^*| = \max\{|G_1|, |G_2|\} \\ \Leftrightarrow \ & \max\{1 - \tfrac{|mcs(p,G_1)|}{\max\{|p^*|,|G_1|\}}, 1 - \tfrac{|mcs(p,G_2)|}{\max\{|p^*|,|G_2|\}}\} \\ < \ & \max\{1 - \tfrac{|mcs(p'',G_1)|}{\max\{|p''^*|,|G_1|\}}, 1 - \tfrac{|mcs(p'',G_2)|}{\max\{|p''^*|,|G_2|\}}\} \\ & \qquad \text{by Definition 3.1 and } mcs(p, G_i) = mcs(p^*, G_i) \\ \Leftrightarrow \ & \max\{d(p^*, G_1), d(p^*, G_2)\} < \max\{d(p''^*, G_1), d(p''^*, G_2)\} \end{aligned}$$

This means p'' is not an optimum pivot as it is worse than p. It is a contradiction and all optimum pivots contain $mcs(G_1, G_2)$.

Therefore, finding the optimum pivot for $m = 2$ is equivalent to compute the `MCS` between G_1 and G_2. Since the `MCS` computation is NP-hard, the pivot selection problem when $m = 2$ is NP-hard. Consequently, the general pivot selection problem is NP-hard. □

A.9 Time Complexity Analysis of `GMTree` construction

Proposition A.9: *The time complexity of* `GMTree` *construction with the sampling-base pivot selection is*

$$O((\log_c |\mathcal{D}| - 1) \times s \times |G| \times dist \times |\mathcal{D}|),$$

```
Procedure similarity
Input: db $\mathcal{D}$, GMtree $T$ rooted at $v$ and query $Q = (q, t)$
Output: Query result $\mathcal{RS}$
01 denote the pivot of $v$ as $p$
02 if $v$ is a leaf
03     for each $G \in v$
04         if $1 - \frac{|mcs(p,G)|}{|p|} - d(q,p) \leq t$        //Lemma 4.1
05             if $d_s(q, G) \leq t$ then $\mathcal{RS}$ = $\mathcal{RS} \cup G$
06     return $\mathcal{RS}$
07 $d = d(q, p^*) = 1 - \frac{|mcs(q,p)|}{\max(|p^*|,|q|)}$ //$mcs(q,p) = mcs(q,p^*)$
08 for each $i$ in [0,..., $c-1$]
09     if $r_p^i - d \leq t$        //Theorem 4.2
10         $\mathcal{RS}$ = $\mathcal{RS} \cup$ similarity($\mathcal{D}$,$T_p^i$,$Q$)
11     else    break
12 return $\mathcal{RS}$
```

Fig. 18. **Procedure** similarity

where $dist = O(3^{|G'|/3}|G'|^{2.5}(|G'|+1)^{|VC_G|})$ *is the time complexity of* VC-mcs *in Sec. 7.1.1.* ∎

Proof: Consider the i-th level of GMTree, for any internal node v, we need to perform the pivot selection. We need $O(|G|)$ time to sample a pivot and $O(dist \times |\mathcal{D}_v|)$ time to compute the distance between the pivot with all data graphs covered by v, denoted as $\mathcal{D}_v$. We need s samples and hence $O(s \times |G| \times dist \times \mathcal{D}_v)$ time for node v.

Since the internal nodes of the i-th level of GMTree together covers all data graphs in $\mathcal{D}$, *i.e.*, $\sum_v |\mathcal{D}_v| = |\mathcal{D}|$, all internal nodes of the i-th level need $O(s \times |G| \times dist \times \sum_v |\mathcal{D}|) = O(s \times |G| \times dist \times |\mathcal{D}|)$;

Since the height of GMTree is $O(\log_c |\mathcal{D}|)$, the total time complexity is $O((\log_c |\mathcal{D}| - 1) \times s \times |G| \times dist \times |\mathcal{D}|)$ as the leaf nodes do not need pivot selection. □

Appendix B Subgraph Similarity Search on GMTree

In this appendix, we present the details of the algorithm of subgraph similarity search on GMTree.

In a nutshell, given a query $Q = (q, t)$, where q is the query graph and t is its radius, the search algorithm is incorporating the pruning of Theorem 4.2 into a depth-first traversal on GMTree.

Procedure similarity presents the details of search algorithm as shown in Fig. 18. The inputs are a database $\mathcal{D}$, the GMTree index T and query $Q = (q, t)$. The output is the query result $\mathcal{RS} = \{G \mid d_s(q, G) \leq t, G \in \mathcal{D}\}$. Procedure similarity traverses the GMTree index and filters out subspaces that certainly do not contain answers (Lines 08-11 by using the pruning condition of Theorem 4.2). The subtrees indexing the remaining subspaces may contain answers and they are traversed (Line 10). When a leaf node is reached, the graphs that cannot been filtered by Lemma 4.1 are candidates (Line 04) and checked individually whether they are in fact answers (Line 05).

Appendix C Query Workload in Experiments

Regarding AIDS, our query workload was modified from the benchmark queries $Q4$, $Q8$ and $Q12$ used in [1]. Our modifications were two. (1) Qm, $m = 4, 8, 12$ means that the

TABLE 2
Characteristics of datasets

	# graphs	# avg. nodes	# avg. edges	avg. dens.	# v. labels	# e. labels
AIDS	10K	25.4	27.4	0.104	51	4
SYN	10K	12.9	50.4	0.65	5	5
PUBCHEM	10K	18.8	19.6	0.165	23	3

query graph has m vertices (versus m edges in [1]), since our subgraph distance is defined based on the number of graph vertices. (2) Qm is a mixture of queries of diverse densities. To obtain a query of a certain density, we randomly selected a query in Qm and added or removed vertices randomly until it satisfied the requirement. Each Qm contains 1,000 query graphs. Regarding SYN, we used the method of [50] to obtain a set of queries. For example, for $Q12$, we first randomly selected 1,000 graphs whose sizes were larger than 12 from SYN. Then, for each graph, we randomly removed vertices from it until its size was 12 and applied the same modifications as AIDS. Following [2], the possible numbers of missing vertices σ of q are 1, 2 and 3.

Appendix D Additional Experiments

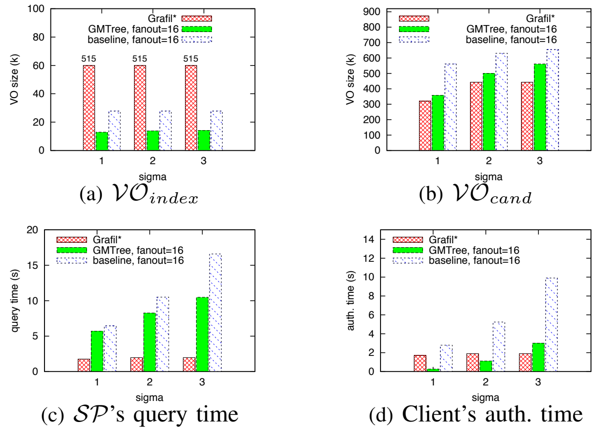
In this appendix, we first present the experimental results on the PUBCHEM dataset. The results on PUBCHEM are similar to those on AIDS and SYN. We then present the performances of GMTree using data graphs as pivots on AIDS. We finally present the experimental results on σ on AIDS and SYN.

D.1 Experiments on PUBCHEM

Experimental settings. The experiments on PUBCHEM were conducted with the same settings of the experiments on AIDS and SYN in Sec. 8.

PUBCHEM **dataset.** We followed the method of [50] to sample 10,000 graphs as our test dataset PUBCHEM from the PubChem chemical structure database [10].

Query workload. The same method to obtain the queries on SYN was applied on PUBCHEM to obtain 1,000 queries. The possible numbers of missing vertices σ of q was also set to 1, 2 and 3.



(a) $\mathcal{VO}_{index}$ (b) $\mathcal{VO}_{cand}$ (c) $\mathcal{SP}$'s query time (d) Client's auth. time

Fig. 19. Comparison of GMTree with Grafil* and the baseline method in terms of $\mathcal{VO}$ size and time on PUBCHEM

Similar to the experiments on AIDS and SYN, we present the results of our experiments on PUBCHEM in terms of the comparison with the baseline and Grafil*, the experiments on query size and the experiments on σ.

D.1.1 Comparison with the baseline and Grafil*

Comparison of $\mathcal{VO}$ size. Figs. 19(a)-(b) present the sizes of the $\mathcal{VO}_{index}$ and the $\mathcal{VO}_{cand}$, respectively. From Fig. 19(a), we note that the $\mathcal{VO}_{index}$ of our `GMTree` is only 30% of Grafil* and smaller than 50% of the baseline. From Fig. 19(b), we note that `GMTree` produces slightly more $\mathcal{VO}_{cand}$ than Grafil*. However, `GMTree` produces smaller number of large candidate graphs than Grafil*, which would affect the authentication efficiency significantly. Fig. 19(b) also presents that our `GMTree` produces smaller $\mathcal{VO}_{cand}$ than the baseline.

Comparison of time. Figs. 19(c)-(d) present the $\mathcal{SP}$'s query time and the client's authentication time, respectively. From Fig. 19(c), we note that the `GMTree`'s query at the $\mathcal{SP}$ is 3 times slower on average than Grafil*. However, the client's authentication time is usually the bottleneck and the `GMTree`'s authentication is 6 times faster (on average) than Grafil* as shown in Fig. 19(d). From Figs. 19(c)-(d), we also note that the $\mathcal{SP}$'s query time and the client's authentication time of the `GMTree` are 30% and 60% smaller than those of the baseline, respectively.

D.1.2 Experiments on query size

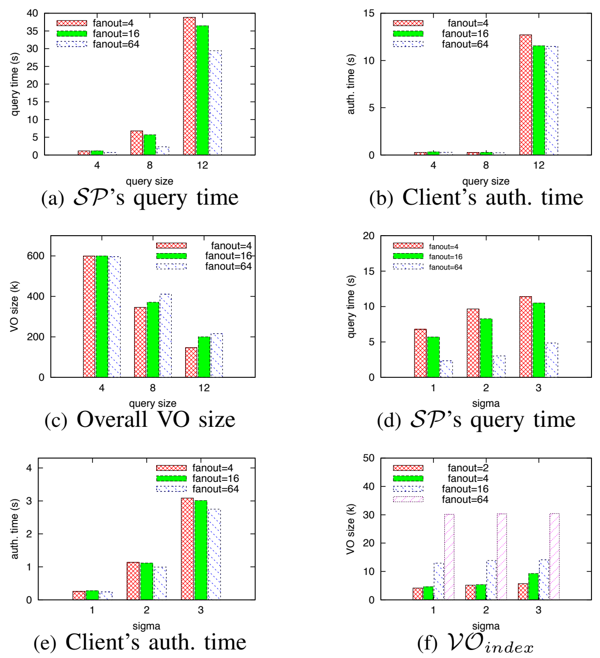
Query time. Fig. 20(a) presents the $\mathcal{SP}$'s query time versus the query size. From Fig. 20(a), we note that the $\mathcal{SP}$'s query time increases significantly with the growth of the query size. It is consistent to the complexity of computing `MCS`. From Fig. 20(a), we also note that the query time reduces as the fanout increases. It is because that the larger the fanout, the less the distance computation involved in query processing.

Authentication time. Fig. 20(b) presents the client's authentication time. Fig. 20(b) presents that the client's authentication time is much smaller than the query time at the $\mathcal{SP}$. In particular, the authentication of $Q12$ at fanout 16 is 3 times faster than query. From Fig. 20(b), we also note that the client's authentication time increases as the query size increases. It is because that the `MCS` computation in the client's authentication for larger queries takes longer. Since the number of distance computation reduces with the growth of the fanout (as discussed in the query time above), the client's authentication time reduces with the growth of the fanout as shown in Fig. 20(b).

Overall $\mathcal{VO}$ size. Fig. 20(c) presents the overall $\mathcal{VO}$ size. We observe that the overall $\mathcal{VO}$ size reduces with the growth of the query size. It is because that larger queries produced fewer candidate graphs, which dominate the $\mathcal{VO}$. We also note that the $\mathcal{VO}$ size increases with the growth of fanout. It is consistent with the observations of MRTree and MBTree.

D.1.3 Experiments on σ

Query time. Fig. 20(d) presents the $\mathcal{SP}$'s query time. From Fig. 20(d), we note that the $\mathcal{SP}$'s query time increases as σ increased. It is because that the larger query range, the more data graphs needed to be tested. We also note that the query time reduces as the fanout increases. It is because that the



(a) $\mathcal{SP}$'s query time (b) Client's auth. time (c) Overall VO size (d) $\mathcal{SP}$'s query time (e) Client's auth. time (f) $\mathcal{VO}_{index}$

Fig. 20. Experiments on query size and σ on PUBCHEM

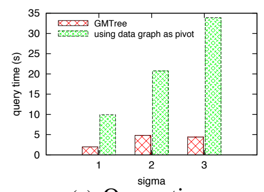
lager fanout, the fewer pivots needed to be accessed in `GMTree` traversal.

Authentication time. Fig. 20(e) presents the client's authentication time. From Fig. 20(e), we observe that the client's authentication time increases with the growth of σ. It is because that the larger query range, the more candidate graphs needed to be authenticated. We also note that the client's authentication time reduces as σ increases. It is because that the number of pivots involved in the client's authentication reduces with the growth of fanout.

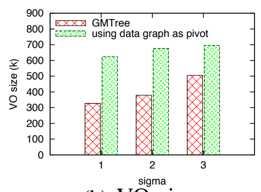
$\mathcal{VO}_{index}$ size. Fig. 20(f) presents the $\mathcal{VO}_{index}$ size of `GMTree`. Fig. 20(f) shows that the $\mathcal{VO}_{index}$ increases with the growth of σ. It is because that the number of accessed internal nodes increases as σ increases. Fig. 20(f) also presents that the $\mathcal{VO}_{index}$ size increases as the fanout increases. It is because that the number of pruned subtrees increases as the fanout increases. However, the $\mathcal{VO}_{index}$ is well controlled within 30K bytes.

D.2 Additional Experiments on Using Data Graph as Pivots on AIDS

This experiment used the same settings on AIDS as presented in Sec. 8. `GMTree` by default allows to use subgraphs as pivots as detailed in Sec. 7.2. In this experiment, we construct a specific `GMTree` that uses the data graph as pivots. The fanout of the `GMTree` is 16. Figs. 21(a)-(b) present the comparison of the query time and the total $\mathcal{VO}$ size, respectively. From Fig. 21(a), we observe that the query time of our method is smaller than 20% of using data graph as pivots. Moreover, the $\mathcal{VO}$ sizes of our method are 52%, 56% and 72% of using data graph as pivots at $\sigma = 1$, 2 and 3, respectively, as shown in Fig. 21(b). This experiment verifies the effectiveness of using subgraphs as pivots.

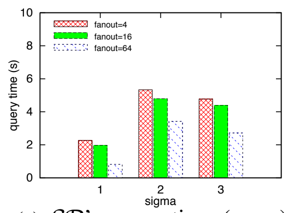


(a) Query time

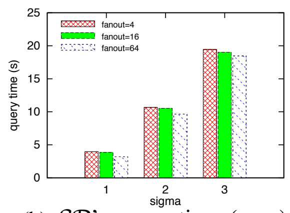


(b) VO size

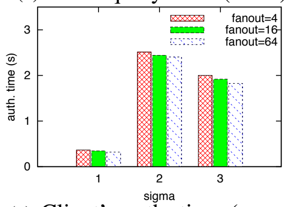
Fig. 21. Performance of using data graph as pivots on AIDS



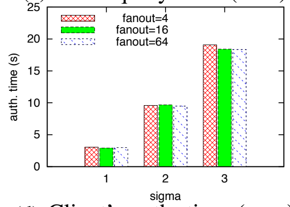
(a) $\mathcal{SP}$'s query time (AIDS)



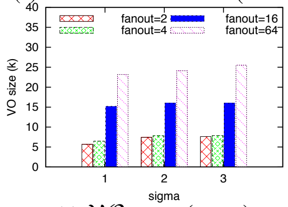
(b) $\mathcal{SP}$'s query time (SYN)



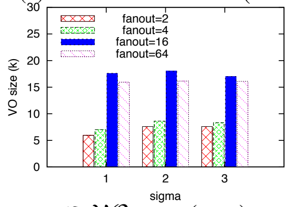
(c) Client's auth. time (AIDS)



(d) Client's auth. time (SYN)



(e) $\mathcal{VO}_{index}$ (AIDS)



(f) $\mathcal{VO}_{index}$ (SYN)

Fig. 22. Effects of σ on AIDS and SYN

D.3 Additional Experiments on σ on AIDS and SYN

This experiment used the same settings as presented in Sec. 8. Similar to the experiments in Sec. 8, this experiment presents the performances of `GMTree` in terms of the query time, the authentication time and the $\mathcal{VO}$ size, respectively. We use AIDS and SYN and we focus on $Q8$ in this experiment, since other two query sets exhibit similar performance characteristics.

Query time. Figs. 22(a)-(b) present the query time on AIDS and SYN, respectively. From Figs. 22(a)-(b), we observe that the query time increases as σ increases. It is because that the larger σ, the more candidate graphs produced. We note from Fig. 22(a) that the query time of $\sigma = 3$ is slightly smaller than that of $\sigma = 2$. It is because that (i) for $\sigma = 2$ and 3, AIDS's answer graphs account for most of candidate graphs, *i.e.*, 74% and 69% for $\sigma = 2$ and 3 at the fanout 16, respectively. (ii) Answer graphs of $\sigma = 2$ are all answers of $\sigma = 3$. (iii) If a graph is an answer of $\sigma = 2$ or 3, our `Enum` method can detect it for $\sigma = 3$ with a lower time cost than $\sigma = 2$, because that a smaller number of subgraphs need to be determined. Figs. 22(a)-(b) also present that the query time reduces as the fanout increases. It is because that the larger fanout, the fewer pivots accessed in traversal. SYN's time reduction is slight as shown in Fig. 22(b), because that graphs in SYN are dense and their `MCS` computation dominates the query time.

Authentication time. Figs. 22(c)-(d) present the authentication time on AIDS and SYN, respectively. From Figs. 22(c)-(d), we observe that the authentication time increases with the growth of σ. It is because that the larger σ, the more candidate graphs needed to be verified. Because of the large percentage of answer graphs in the candidate set (see discussion in query time above), $\sigma = 3$ of AIDS takes shorter authentication time than $\sigma = 2$ as shown in Fig. 22(c). Figs. 22(c)-(d) also show that the authentication time reduces with the growth of fanout. It is because that the number of pivots involved in authentication reduces as the fanout increases.

$\mathcal{VO}_{index}$ **size.** Figs. 22(e)-(f) present the size of $\mathcal{VO}_{index}$ on AIDS and SYN, respectively. Recall that the $\mathcal{VO}_{index}$ mainly contains two parts: (i) the relevant content, `MCS` and hints of accessed internal nodes and (ii) the digests of pruned subtrees. Figs. 22(e)-(f) present that the $\mathcal{VO}_{index}$ increases with the growth of σ. It is because that the number of accessed internal nodes increases as σ increases. Figs. 22(e)-(f) also present that the $\mathcal{VO}_{index}$ increases as the fanout increases. It is because that the number of pruned subtrees increases as the fanout increases. However, the $\mathcal{VO}_{index}$ is well controlled within 30K bytes. Fig. 22(f) presents that the $\mathcal{VO}_{index}$ reduces slightly from $\sigma = 2$ to 3 on SYN. It is because that the number of pruned subtrees reduces more significantly than the increase of accessed node number. In particular, the number of pruned subtrees reduces 16% from $\sigma = 2$ to 3, but the number of accessed node increases only 1% at fanout 16.