

Summarization Graph Indexing: Beyond Frequent Structure-based Approach

Lei Zou¹ *, Lei Chen², Huaming Zhang³, Yansheng Lu¹, and Qiang Lou⁴

¹ Huazhong University of Science and Technology, Wuhan, China,
{zoulei, lys}@mail.hust.edu.cn

² Hong Kong of Science and Technology, Hong Kong, China,
leichen@cse.ust.hk

³ The University of Alabama in Huntsville, Huntsville, AL, 35899, USA
hzhang@cs.uah.edu

⁴ The Temple University, USA
qianglou@temple.edu

Abstract. Graph is an important data structure to model complex structural data, such as chemical compounds, proteins, and XML documents. Among many graph data-based applications, sub-graph search is a key problem, which is defined as *given a query Q , retrieving all graphs containing Q as a sub-graph in the graph database*. Most existing sub-graph search methods try to filter out false positives (graphs that are not possible in the results) as many as possible by indexing some frequent sub-structures in graph database, such as [20, 22, 4, 23]. However, due to ignoring the relationships between sub-structures, these methods still admit a high percentage of false positives. In this paper, we propose a novel concept, *Summarization Graph*, which is a complete graph and captures most topology information of the original graph, such as sub-structures and their relationships. Based on Summarization Graphs, we convert the filtering problem into retrieving objects with set-valued attributes. Moreover, we build an efficient signature file-based index to improve the filtering process. We prove theoretically that the pruning power of our method is larger than existing structure-based approaches. Finally, we show by extensive experimental study on real and synthetic data sets that the size of candidate set generated by Summarization Graph-based approach is only about 50% of that left by existing graph indexing methods, and the total response time of our method is reduced 2-10 times.

1 Introduction

As a popular data structure, graphs have been used to model many complex data objects in real world, such as, chemical compounds [18] [11], proteins [2], entities in images [15], XML documents[21] and social networks [3]. Due to the wide usage of graphs, it is quite important to provide users to organize, access and analyze graph data efficiently. Therefore, graph database has attracted a lot of attentions from database and data mining community, such as sub-graph search [16, 20, 9, 22, 8, 6, 4, 23], frequent sub-graph

* This work was done when the first author visited Hong Kong University of Science and Technology as a visiting scholar. The first author was partially supported by National Natural Science Foundation of China under Grant 70771043.

mining [10, 13, 19], correlation sub-graph query [12] and so on. Among many graph-based applications, it is very important to retrieve related graphs containing a query graph from a large graph database efficiently. For example, given a large chemical compound database, a chemist wants to find all chemical compounds having a particular sub-structure.

In this paper, we focus on *sub-graph search*, which is defined as follows: *given a query graph Q , we need to find all data graphs G_i , where query graph Q is sub-graph isomorphism to data graph G_i* . Because sub-graph isomorphism is NP-complete [7], we have to employ filtering-and-verification framework to speed up the search efficiency. Specifically, we use an effective and efficient pruning strategy to filter out the false positives (graphs that are not possible in the results) as many as possible first (that is *filtering process*), then validate the remaining candidates by subgraph isomorphism checking (that is *verification process*). Therefore, the overall response time consists of filtering time and verification time. The state art of pruning strategy in the literature is frequent structures-based pruning. These methods apply data mining algorithms offline first to find frequent sub-structures (sub-graphs or sub-trees, we also call them *features*) [20, 22, 4, 23] in graph database. Then some discriminate sub-structures are chosen as indexed features. After that, we build the inverted index for each feature, namely, the data graphs containing these features are linked to these features. The relationship between indexed features and data graphs is similar to the one between indexed words and documents in IR (Information Retrieve). To answer a sub-graph query Q , Q is represented as a set of features and all the graphs that may contain Q are retrieved by examining the inverted index. The rational behind this type of filtering strategy is that if some features of graph Q do not exist in a data graph G , G cannot contain Q as its sub-graph.

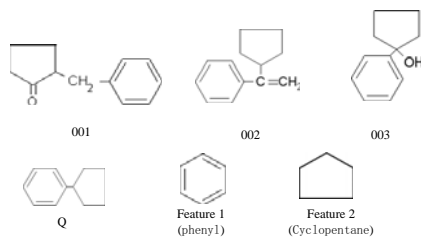


Fig. 1. Motivation Example

However, existing sub-structure based approaches treat each data graph as a bag of indexed features. They fail to capture the relationship among the features. As a consequence, many false positives still exist after filtering. Given a graph database with 3 data graphs 001–003 and a query Q in Fig. 1, assume that we choose feature 1 and 2 as indexed features. Since the query Q and data graphs 001–003 all contain both feature 1 and feature 2 respectively, after filtering, the candidates are still 001, 002 and 003, which is exactly the whole database. *Can we reduce the candidate size based on the same fea-*

ture 1 and 2? Our answer is yes. Observed from Fig. 1, the length of the *shortest path* between feature 1 and feature 2 is 1 in query Q , but 2 in 001 and 002, and 1 in 003. It means that only graph 003 may be a correct answer, and we can filter out 001 and 002 safely. Considering both feature and pairwise feature relationship is primary motivation of this paper. However, simply checking feature first, then feature relationship is neither effective nor efficient, which will be discussed in Section 3.2.

In feature-based approaches, more indexed features means higher pruning power, but also means longer filtering time and larger space cost. On the other hand, less features lead to poor pruning power in the filtering process, which leads to bad performance in overall response time. Thus, there is a trade off between number of features and pruning power. In this paper, we propose a novel filtering method, which requires less features and provides higher pruning power, compared to the previous approaches, such as gIndex [20], FG-index [4]. Specifically, we encode both features and feature relationships of graph Q into Summarization Graph (i.e. $Sum(Q)$) and use Summarization Graph to conduct filtering. The contributions that we made in this paper are listed as follows:

1. We propose a novel concept, Summarization Graph, which is a complete graph and captures most topology information of the original graph, such as sub-structures and their relationships.
2. We encode summarization graph into a set, and we convert a filtering process into retrieving of objects with set-valued attributes to get candidate set, which can be solved efficiently by building an effective signature-based index.
3. Last but not least, through extensive experiments on a real dataset AIDS and synthetic datasets, we show that the candidate size of our methods is only about 50% of that achieved by existing approaches on average and the total searching time is reduced 2-10 times in our method.

The remaining of the paper is organized as follows. The related work is discussed in Section 2. Summarization Graph framework is proposed in Section 3. The performance study is reported in Section 4. Section 5 concludes the paper.

2 Related Work

Usually, to speed up the sub-graph search, we use the filtering-and-verification framework. First, we remove false positives (graphs that are not possible in the results) by *pruning strategy*; Then, we perform sub-graph isomorphism algorithm on each candidate to obtain the final results. Obviously, less candidates mean better search performance.

So far, many pruning strategies have been proposed [16, 20, 9, 22, 8, 6, 4, 23]. The most popular approaches are *feature-based*. The approaches apply data mining techniques to extract some discriminate sub-structures as indexed features, then, build inverted index for each feature. Query graph Q is denoted as a set of features, and the pruning power always depends on selected features. By the inverted indexes, we can fix the complete set of candidates. Many algorithms have been proposed to improve the effectiveness of selected features, such as gIndex [20], TreePi [22], FG-Index [4] and

Tree+ δ [23]. In gIndex, Yan et al. propose a “discriminative ratio” for features. Only frequent and discriminative subgraphs are chosen as indexed features. In TreePi, due to manipulation efficiency of trees, Zhang et al. propose to use frequent and discriminate subtrees as indexed feature. In FG-Index [4], James et al. propose to use frequent sub-graphs as indexed features. In [23], Zhao et al. use “subtree” and a small number of discriminative sub-graphs as indexed features instead of sub-graphs in gIndex and FG-Index.

Different from the above approaches, in Closure-tree [9], He and Singh propose pseudo subgraph isomorphism by checking the existence of a semi-perfect matching from vertices in query graph to vertices a data graph (or graph closure). The major limitation of Closure-tree is the high cost in filtering phase, since Closure-tree needs to perform expensive structure comparison and maximal matching algorithm in filtering phase. There are some other interesting recent work in graph search problem, such as [6] [8]. In [6], Williams et al. propose to enumerate all connected induced subgraphs in the graph database, and organize them in a Graph Decomposition Index (GDI). This method cannot work well in a graph database with large-size graphs, due to combination explosion of enumerating all connected induced subgraphs. Jiang et al. propose gString [8] for compound database. It is not trivial to extend gString to graph database in other applications.

3 Summarization Graph

As mentioned in Section 1, feature-based indexing methods are not effective due to ignoring the relationship between features. However, it is not trivial to include the relationship into the indexes, checking feature first, then relationship is neither effective nor efficient, which will be discussed in Section 3.2.

In this section, we first give some preliminary background related to sub-graph search in Section 3.1. Then, we present a novel concept, *Summarization Graph*, which encodes the features as well as their relationships in Section 3.2. In Section 3.3, we present how Summarization Graph is used for filtering. Finally, in Section 3.4, we give an efficient signature file-based index structure to improve the filter efficiency.

3.1 Preliminary

Definition 1. Graph Isomorphism. Assume that we have two graphs $G_1 < V_1, E_1, L_{1v}, L_{1e}, F_{1v}, F_{1e} >$ and $G_2 < V_2, E_2, L_{2v}, L_{2e}, F_{2v}, F_{2e} >$. G_1 is graph isomorphism to G_2 , if and only if there exists at least one bijective function $f : V_1 \rightarrow V_2$ such that: 1) for any edge $uv \in E_1$, there is an edge $f(u)f(v) \in E_2$; 2) $F_{1v}(u) = F_{2v}(f(u))$ and $F_{1v}(v) = F_{2v}(f(v))$; 3) $F_{1e}(uv) = F_{2e}(f(u)f(v))$.

Definition 2. Sub-graph Isomorphism. Assume that we have two graphs G' and G , if G' is graph isomorphism to at least one sub-graph of G under bijective function f , G' is sub-graph isomorphism to G under injective function f .

Definition 3. (Problem Definition) Given a query graph Q , we need to find all data graphs G_i , where Q is sub-graph isomorphism to G_i in a large graph database.

As discussed in Section 1, due to hardness of sub-graph isomorphism, sub-graph search should adopt *filtering-and-verification* framework. In feature-based approaches, given a query Q , we can fix the complete set of candidates by scanning the inverted index. In the filtering process, in order to find all features that are contained in the query Q , we need to perform sub-graph isomorphism. After filtering, for each candidate, we also need to perform sub-graph isomorphism to check whether it is a correct answer. Therefore, the cost of sub-graph search can be modeled below:

$$Cost = |N_f| * C_f + |N_c| * C_c \quad (1)$$

Here, $|N_f|$ is the number of sub-graph isomorphism test in the filtering process. $|N_c|$ is the size of candidate size. C_f and C_c are average cost of sub-graph isomorphism in filtering and verification process respectively. Intuitively, if we choose more features, $|N_c|$ will be smaller. However, more indexed features also lead to more sub-graph isomorphism tests in the filtering process (namely, $|N_f|$ is large). Therefore, to improve sub-graph search performance, $|N_f|$ and $|N_c|$ should be both small. In this paper, we will show that our method can use less feature to obtain greater pruning power.

3.2 Summarization Graph

In this subsection, we first give some important definitions as the base of Summarization Graph. Then we illustrate, through an example, the reason not combining feature-index and feature relationship information directly. Finally, our Summarization Graph framework is discussed in details.

Definition 4. Occurrence. Given two graphs G and S , if there are m different sub-graphs in G that are all isomorphic to the graph S , we call these sub-graphs as **Occurrences** of S in G , which is denoted as $O_i(G, S)$, where $i = 1...m$.

For example, in Fig.1, there is only one occurrence of phenyl in 001, 002, and 003. Similarly, in Fig.2, there are two occurrences of triangle in 001 and 002 respectively.

Definition 5. Distance between occurrences. Given a graph G and two occurrences $O_i(G, S_1)$ and $O_j(G, S_2)$, where S_1 and S_2 are sub-graphs of G , $Dist(O_i(G, S_1), O_j(G, S_2))$ is the distance between two occurrences, which is defined as follows:

1. If $O_i(G, S_1)$ does not overlap with $O_j(G, S_2)$, then $Dist(O_i(G, S_1), O_j(G, S_2)) = Min(SDist(V_n, V_m))$, where $SDist(V_n, V_m)$ is the shortest path length between vertices V_n and V_m , and V_n, V_m are two vertices in $O_i(G, S_1)$ and $O_j(G, S_2)$ respectively.
2. If $O_i(G, S_1)$ overlaps with $O_j(G, S_2)$, then $Dist(O_i(G, S_1), O_j(G, S_2)) = 0 - ComVer$, where $ComVer$ is the number of common vertices in $O_i(G, S_1)$ and $O_j(G, S_2)$.

For example, in Fig.1, the distance between two occurrences, feature 1 (phenyl) and feature 2 (cyclopentane), is 2 in 001 and 002, and 1 in 003. In graph 001 of Fig.2, the distance between two occurrences of triangle and square denoted by shade area is -1 , since they share one vertex.

In Fig.2, given a query Q , 001 should be result, which contains Q . Obviously, it is not efficient to include the distance information into feature index by simply checking feature first then distances (more processing time in filter phase). Furthermore, the straightforward approach is not effective either (there are still many false positives left). For example, in Fig.2, F_1 (triangle) and F_2 (square) are chosen as indexed features. The distances between all occurrences of F_1 and F_2 are also recorded in Fig.2. There are two occurrences of F_1 and F_2 in 001 respectively, thus, there are four distances among these occurrences, and the same applies to 002. Unfortunately, as shown in Fig. 2, all distance information is the same between two graphs. Thus, simply plug in the distance information to feature-based approach may not help to improve the filtering power. In order to address the above issue, in this paper, we introduce a novel concept, *Summarization Graph*, which encodes both the features and their distance relationships together.

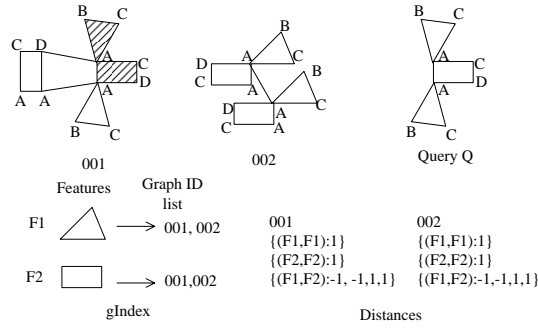


Fig. 2. Running Example

Fig.3 shows the steps to construct summarization graph. Given a graph 001, for each feature occurrence O_i , it is denoted as a vertex in the corresponding summarization graph, that is $Sum(001)$. The pair-wise distances among occurrences are recorded as edge weights in $Sum(001)$ in Fig.3. Then, we encode the edge weights into corresponding vertices in $Sum(001)$. Since $Sum(001)$ is a complete graph, we can denote it by the set of vertices in $Sum(001)$. The formal definition about summarization graph is given as follows:

Definition 6. Summarization Graph. Given a graph G and a set of its sub-graphs F_i , for every occurrence of F_i in G , it is denoted as a vertex in the Summarization Graph of G , i.e. $Sum(G)$. Each vertex label in $Sum(G)$ is a set of Pairs. Each pair is denoted as $\langle \text{label}, \text{length} \rangle$, which denotes the topology and distance information about this occurrence in G .

To transform original graphs into Summarization Graph, 1) we should find some sub-graphs as features; 2) with these features, we transform all data graphs and the query graph into Summarization Graphs. Since feature selection approaches in the pre-

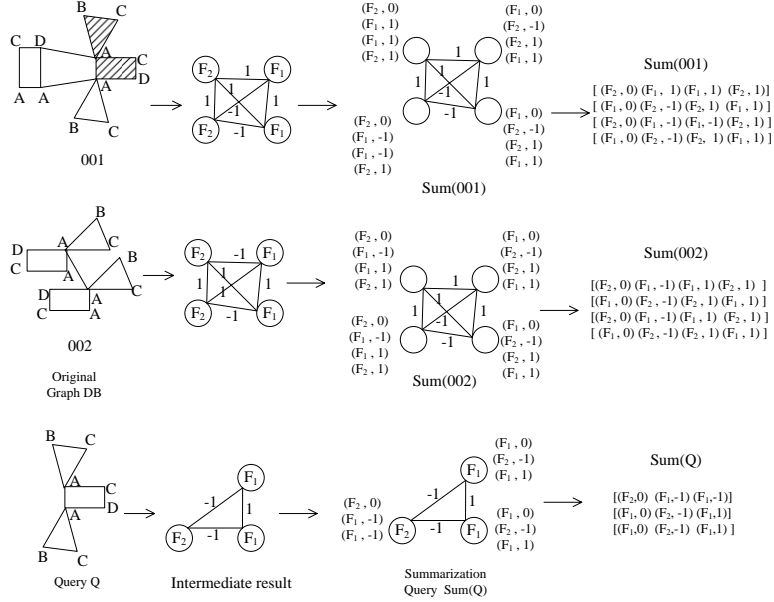


Fig. 3. Generating Summarization Graphs

vious work are orthogonal to our methods, we omit the discussions about feature selection. Interested readers can refer to [20, 22, 4, 23] for details.

Here, we discuss how to build a Summarization Graph for an original graph as follows. There are three sub-steps to generate Summarization Graph for an original graph: 1) find all occurrences of features. All occurrences form the vertexes of the Summarization Graph; 2) compute pair-wise distances between occurrences; 3) insert the distance information into vertexes of Summarization Graphs. The whole process of generating Summarization Graph is illustrated with an example in Fig.3. Notice that a Summarization Graph is a complete graph with unlabeled edges, which facilitate the latter process. In Fig.3, after sub-step 2, we get a complete graph with edge label (that is intermediate result). Then, we insert edge label, i.e. distances, into vertexes. Each vertex in the Summarization Graph is a set of Pair, and each pair is denoted as $\langle Label, Length \rangle$. Take the up-right vertex in summarization graph 001 in Fig.3, i.e. Sum(001), for example. The pair set is $[(F_1, 0)(F_2, -1)(F_2, 1), (F_1, 1)]$. Since the corresponding vertex label in intermediate result is F_1 , there is a Pair $(F_1, 0)$ in the vertex of Sum(001). In the intermediate result, there is an edge with distance 1 from the right-up vertex to another vertex with label F_2 . So there is a Pair $(F_2, 1)$. Based on the similar process, we get another two Pairs, $(F_2, -1)$ and $(F_1, 1)$. Now, all together the four Pairs form the vertex label in the summarization graph.

With summarization graph, Fig.4 illustrates the overall framework of sub-graph search. In our framework, first, for all data graphs in the original graph database, we transform them into their corresponding Summarization Graphs. Then, Inverted Sum-

marization Graph DB together with a signature file-based index is created for Summarization Graph DB.

In the online phase, when the query graph comes, 1) it is first transformed into its corresponding summarization graph, 2) and then the query summarization graph is searched in the signature file-based indexes and Inverted Summarization Graph DB to generate candidate set. 3) Finally, the sub-graph isomorphism checking algorithm is applied to the candidate set to get true answers. The filtering and indexing methods will be presented in next two subsections.

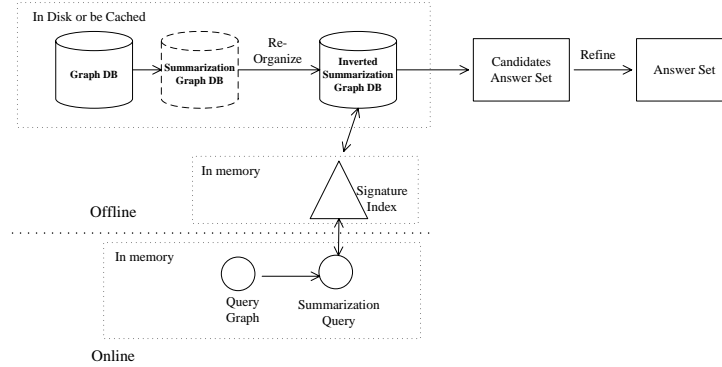


Fig. 4. The Summarization Graph Framework

3.3 Summarization Graph-based Filtering

Definition 7. Corresponding Vertices. Given two Summarization Graphs, $Sum(Q)$ and $Sum(G)$, for a vertex V_1 in $Sum(Q)$ and a vertex V_2 in $Sum(G)$, if they satisfy the following conditions, we say that V_2 corresponds to V_1 , where V_1 is a set of pairs $\langle F_i, L_i \rangle$, and V_2 is also a set of pairs $\langle F_j, L_j \rangle$: We can find an injective function from Pairs $\langle F_i, L_i \rangle$ in V_1 to Pairs $\langle F_j, L_j \rangle$ in V_2 , where

- 1) if $L_i \leq 0$, then $F_i = F_j$ and $L_i = L_j$
- 2) if $L_i > 0$, then $F_i = F_j$ and $0 < L_j \leq L_i$.

Lemma 1. Given two Graphs Q and G , and their Summarization Graphs, $Sum(Q)$ and $Sum(G)$, respectively, if Q is a sub-graph of G , for each vertex in $Sum(Q)$, there must exist a corresponding vertex in $Sum(G)$.

Proof. (Sketch): 1) If Q is a sub-graph of G , according to Definition 2, we can find at least an injective function from Q to G . Each vertex and edge in Q has a unique corresponding image in G . Obviously, for each feature occurrence in Q , there is also a corresponding occurrence in G . According to the definition about summarization graph, for each vertex I in $Sum(Q)$, there must be a corresponding vertex J in $Sum(G)$. For any Pair $p \langle F_1, L_1 \rangle$ in I :

- a) if $L_1 = 0$: Because J is corresponding to I , so their feature labels should be same. So there must be a Pair q in $J < F_2, L_2 >$, where $L_2 = 0$ and $F_2 = F_1$.
- b) if $L_1 < 0$: It means the occurrence I overlaps another occurrence of F_1 with $|L_1|$ vertexes in Q . Since for each occurrence of features in Q , there also exists a corresponding occurrence in G . If two occurrences overlap $|L_1|$ vertexes in Q , the corresponding occurrences also overlap $|L_1|$ vertexes in G . Therefore, there must be a Pair q in $J < F_2, L_2 >$, where $L_2 = L_1, F_2 = F_1$.
- c) if $L_1 > 0$: It means that the minimum distance between the occurrence I and another occurrence of F_1 is L_1 in Q . According to graph theory, obviously, the minimum distance between these two corresponding occurrences in G cannot be larger than L_1 , if Q is a sub-graph of G . Because, at least, the minimum path between these two occurrences in Q must be preserved in G .

In conclusion, Lemma 1 is correct. \square

Filtering Rule 1. Given two graphs Q and G , their corresponding Summarization Graphs are $Sum(Q)$ and $Sum(G)$ respectively. If there exists some vertex in $Sum(Q)$, it has no corresponding vertex in $Sum(G)$, Q cannot be a sub-graph of G . \square

According to Lemma 1, it is straightforward to know that the following theorem holds.

Theorem 1. For sub-graph search, **Filtering Rule 1** satisfies no-negative requirement (no dismissals in query results).

Since we consider both feature and pair-wise feature relationship together, it has greater filtering power than feature-based approaches. Thus, we have the following remark.

Remark 1. Based on the same selected feature set, Filtering Rule 1 has greater filtering power than feature-based methods.

001	002	Query Q
$[(F_2, 0) (F_1, 1) (F_1, 1) (F_2, 1)]$	$[(F_2, 0) (F_1, -1) (F_1, 1) (F_2, 1)]$	$[(F_2, 0) (F_1, -1) (F_1, -1)]$
$[(F_1, 0) (F_2, -1) (F_2, 1) (F_1, 1)]$	$[(F_1, 0) (F_2, -1) (F_2, 1) (F_1, 1)]$	$[(F_1, 0) (F_2, -1) (F_1, 1)]$
$[(F_2, 0) (F_1, -1) (F_1, -1) (F_2, 1)]$	$[(F_2, 0) (F_1, -1) (F_1, 1) (F_2, 1)]$	$[(F_1, 0) (F_2, -1) (F_1, 1)]$
$[(F_1, 0) (F_2, -1) (F_2, 1) (F_1, 1)]$	$[(F_1, 0) (F_2, -1) (F_2, 1) (F_1, 1)]$	$[(F_1, 0) (F_2, -1) (F_1, 1)]$

Fig. 5. Summarization Graph Database

Now, all data graphs in the graph database have been converted into their corresponding Summarization Graphs, which form the Summarization Graph Database, as shown in Fig.5. The query $Sum(Q)$ is also shown in Fig.5. We can sequentially scan the Summarization Graph Database to prune false alarms. In fact, pruning according to Filter Rule 1 is not time-consuming since it is a subset checking process. However, in order to avoid sequential scan in the Summarization Graph Database, we want to store

the Summarization Graph Database like an inverted index, as shown in Fig.6. We call it *Inverted Summarization Graph Database*. In Fig.6, given a query Q , we convert it into its corresponding Summarization Graph, i.e $Sum(Q)$, which is a set of vertexes. For each vertex in $Sum(Q)$, according to the Lemma 1, scanning the vertexes list on the left to find the corresponding vertexes in the Inverted Summarization Graph Database, we will get a graph list. The intersection of these graph lists will be candidate results for sub-graph search. In the running example, for a vertex $[(F_2, 0), (F_1, -1), (F_1, -1)]$ in $Sum(Q)$, we will get a graph list “001”. For a vertex $[(F_1, 0), (F_2, -1), (F_1, 1)]$ in $Sum(Q)$, the graph list is “001,002”. The intersection of graph lists is “001”, which is the candidate.

Vertexes	Graph ID list
$[(F_2, 0) (F_1, 1) (F_1, 1) (F_2, 1)]$	→ 001, ...
$[(F_1, 0) (F_2, -1) (F_2, 1) (F_1, 1)]$	→ 001, 002...
$[(F_2, 0) (F_1, -1) (F_1, -1) (F_2, 1)]$	→ 001, ...
$[(F_2, 0) (F_1, -1) (F_1, 1) (F_2, 1)]$	→ 002, ...
...

Fig. 6. Inverted Summarization Graph Database

3.4 Signature-based Index

As discussed in the above subsection, using the Inverted Summarization Graph avoids the sequential scan in the whole database. However, we have to sequentially scan the vertexes list on the left of Fig.6. In order to avoid this, we build a signature file-based index for the Inverted Summarization Graph Database. In fact, the vertex list in the Inverted Summarization Graph Database is the objects with set-value attributes. Each vertex is an object and it has a set of attributes, i.e. Pairs. According to the methods in [17], we propose a hash function and build an efficient index for the Inverted Summarization Graph Database. For each pair $\langle F_i, L_i \rangle$ of a vertex in summarization graph, if $L_i \leq 0$, we assign a distinct bit-string to the Pair. In Fig.7, on the left, we assign a distinct bit-string to each Pair. For example, for the vertex $[(F_2,0)(F_1,-1)(F_1,1)(F_2, 1)]$, its hash value is $(0000\ 0001) \mid (0100\ 0000) = (0100\ 0001)$, where $(0000\ 0001)$ is the bit-string of $(F_2,0)$ and $(0100\ 0000)$ is the bit-string of $(F_1,-1)$. Using the hash function, each vertex in the Inverted Summarization Graph Database has a hash value, thus, we can index all vertexes by a hash table.

Given a query sub-graph Q , its Summarization Graph is $Sum(Q)$. Using the same hash function, for each vertex V_i in $Sum(Q)$, it has a hash value, denoted by $Hash(V_i)$. Scanning the hash table, for each hash value H_i in the hash table, if $(H_i \& Hash(V_i)) \neq Hash(V_i)$, the vertexes in the bucket cannot correspond to V_i in $Sum(Q)$.

Furthermore, we can also built a S-tree index [17] for the hash table, as shown in Fig.8. S-tree is a balanced tree, where each node has at least m children ($m \leq 2$), and

at most M children ($(M+1)/2 \geq m$). Assume that the intermedin node (directory node) I in S-Tree has n child nodes C_i ($i=1\dots n$), we set $I[j] = (C_1[j] \vee \dots \vee C_i[j] \vee \dots \vee C_n[j])$, where $I[j]$ is the j -th bit in I and $C_i[j]$ is the j -th bit in C_i . Given a vertex V_i in $Sum(Q)$, it has a hash value, denoted by $Hash(V_i)$. For an intermedin node I in S-tree, if $(I \& Hash(V_i) \neq Hash(V_i))$, all descendant nodes can be pruned. For example, given a vertex $V [(F_2,0) (F_1,-1) (F_1,-1)]$ in $Sum(Q)$ in Fig. 5, its hash code is $Hash(V)=(0100\ 0001)$. Obviously, for the intermedin node I_1 in S-tree in Fig.8, $I_1 \& Hash(V) \neq Hash(V)$. Therefore, all descendant nodes of I_1 cannot be corresponding vertex to V in $Sum(Q)$, and they can be pruned safely. Because of S-tree, we reduce the search space. Due to limited space, the more details about S-tree are omitted, and interested readers can refer to [17].

Pair	Vertex	
	$[(F_2, 0) (F_1, 1) (F_1, 1) (F_2, 1)]$	$\longrightarrow (0000\ 0001)$
$(F_2, 0) \longrightarrow 0000\ 0001$	$[(F_1, 0) (F_2, -1) (F_2, 1) (F_1, 1)]$	$\longrightarrow (0000\ 0100) \mid (0001\ 0000) = (0001\ 0100)$
$(F_1, 0) \longrightarrow 0000\ 0100$	$[(F_2, 0) (F_1, -1) (F_1, -1) (F_2, 1)]$	$\longrightarrow (0000\ 0001) \mid (0100\ 0000) \mid (0100\ 0000) = (0100\ 0001)$
$(F_2, -1) \longrightarrow 0001\ 0000$	$[(F_2, 0) (F_1, -1) (F_1, 1) (F_2, 1)]$	$\longrightarrow (0000\ 0001) \mid (0100\ 0000) = (0100\ 0001)$
$(F_1, -1) \longrightarrow 0100\ 0000$		

Fig. 7. Hash Function

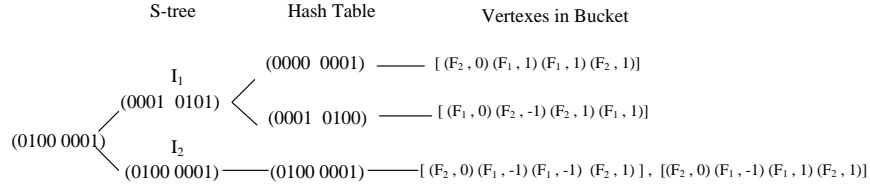


Fig. 8. S-Tree

4 Experiments

In this section, we report our experiment results to evaluate our methods with existing approaches. In our experiment, some of state of art feature-based approaches, such as gIndex, Closure-Tree and FGIndex, are chosen to compare with our method. All experiments are done on a 1.7GHz Pentium 4 PC with 1 GB main memory, running RedHat Fedora Core 3. All algorithms are implemented in standard C++ with STL library support and compiled by g++ 3.4.2 compiler.

4.1 Datasets

1) **AIDS Antiviral Screen Dataset** This dataset is available publicly on the website of the Developmental Therapeutics Program ⁵. We generate 10,000 connected and labeled graphs from the molecule structures and omit Hydrogen atoms. The graphs have an average number of 24.80 vertices and 26.80 edges, and a maximum number of 214 vertices and 217 edges. A major portion of the vertices are C, O and N. The total number of distinct vertex labels is 62, and the total number of distinct edge labels is 3. We refer to this dataset as AIDS dataset. Each query set Q_m has 1000 connected query graphs and query graphs in Q_m are connected size- m graphs (the edge number in each query is m), which are extracted from some data graphs randomly, such as Q_4 , Q_8 , Q_{12} , Q_{16} , Q_{20} and Q_{24} .

2) **Synthetic Dataset** The synthetic dataset is generated by a synthetic graph generator provided by authors of [14]. The synthetic graph dataset is generated as follows: First, a set of S seed fragments are generated randomly, whose size is determined by a Poisson distribution with mean I . The size of each graph is a Poisson random variable with mean T . Seed fragments are then randomly selected and inserted into a graph one by one until the graph reaches its size. Parameter V and E denote the number of distinct vertex labels and edge labels respectively. The cardinality of the graph dataset is denoted by D . We generate the graph database using the same parameters with gIndex in [20]: $D=10,000$, $S=200$, $I=10$, $T=50$, $V=4$, $E=1$.

In gIndex, Closure-tree and FG-Index algorithms, we choose the default or the suggested values for parameters according to [20, 9, 4].

4.2 Methods

Off-line process: (Database Transformation and Index Building)

For the graph database, 1) we use the gIndex algorithm to find all indexed features. The maximal size of frequent sub-graph is set to 10 in all experiments; 2) we use C++ VFLib library [1] to find all feature occurrences in each data graph; 3) for each data graph in the original database, we change it into Summarization Graph; 4) we re-organize the Summarization Graph Database to get Inverted Summarization Graph Database, and build a hash table and S-tree on the vertexes list. When computing pair-wise occurrence distances, we first get all vertex-pair shortest paths in each data graph using Johnson algorithm [5], then get occurrence distances according to Definition 5.

On-line process: (Query Executing)

Given a query graph Q , we transform it into Summarization Graph, i.e. Sum(Q). For each vertex in Sum(Q), according to the S-tree and hash table, we find the corresponding vertexes in Inverted Summarization Graph Database. Then we get a graph list for each corresponding vertex. The intersection of these graph lists is the Candidate Answer Set. The above process is called getting candidate process. For each data graph in candidates, we have to do sub-graph isomorphic checking, which we call refining process.

⁵ <http://dtp.nci.nih.gov/>

4.3 Performance Studying

We evaluate offline and online performance of our method respectively, and compare them with other existing approaches. Notice that, the online performance is crucial for sub-graph search, such as pruning power and total response time.

1) Offline Performance.

Fig.9 shows the offline performance of Summarization Graph method. Compared with feature-based methods, such as gIndex and FGIndex, our method needs to compute the feature distances. Therefore, our offline processing time is larger than other approaches. We encode both features and feature distances of graph G into $Sum(G)$. Even though we utilize signature file to store $Sum(G)$, the space cost of our method is also a little larger than gIndex and FGIndex, and smaller than Closure-Tree.

Observed from Fig.9, the offline performance of our method is not as good as other approaches. However, in the online performance, our method outperform other methods orders of magnitude (see Fig. 10). Therefore, it is really worth to paying more time on off-line processing.

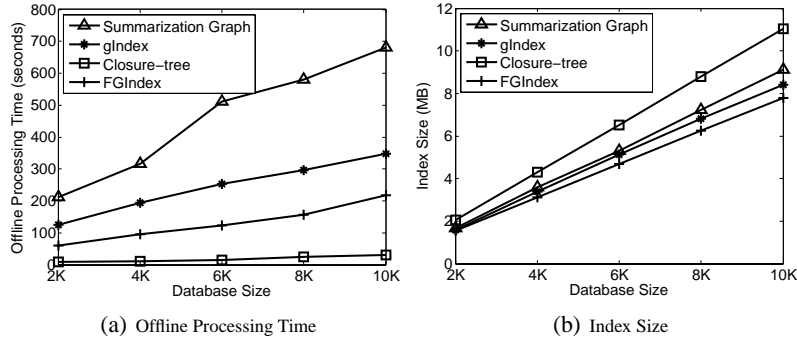


Fig. 9. Evaluate the Offline Performance in AIDS dataset

2) Online Performance.

In Fig. 10, we evaluate the online performance our method in both AIDS and synthetic datasets, and compare it with other methods.

As we all know, since sub-graph isomorphism is NP-complete problem, all sub-graph methods adopt *filter-and-refine* framework. Obviously, the less candidates after filtering, the faster query response time. Therefore, pruning power in the filter process is critical to overall online performance. Compared with existing feature-based methods, we not only consider features but also the pairwise feature relationships. Therefore, we have less candidates after filter process. Observed from Fig. 10(a), the candidate size in our method is less than 50% of that in other methods.

On the other hand, to obtain the fast overall response time, the filtering time is also critical. To avoid the sequential scan in the Inverted Summarization Graph Database (see Fig. 6), we build the S-tree (see Fig.8). Benefited from S-Tree, we reduce the

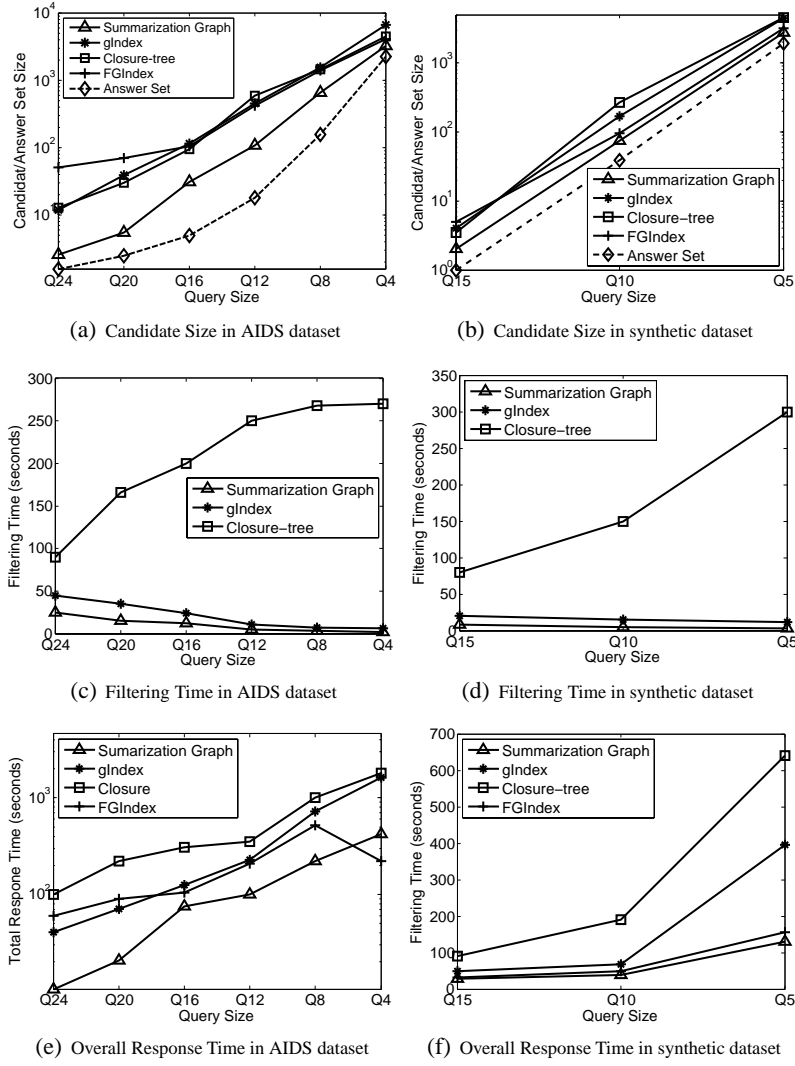


Fig. 10. Evaluate the Online Performance

search space of our method in the filter process. Therefore, our method has less filtering time than other approaches.

In the refine process, we use C++ VFLib library [1] to implement sub-graph isomorphism algorithm. Because our method has the least candidates and our refining time is faster than other methods, the overall response time is the fastest in most cases, as shown in Fig 10(e). In FGIndex, if the query Q is a frequent sub-graph, namely, Q is isomorphism to an indexed features, it can report the answer set without refine process. However, there are few queries that are frequent sub-graphs, especially when queries

are large graphs. Thus, only in Q_4 , the response time in FGIndex is smaller than that in Summarization Graph method.

5 Conclusions

In this paper, we proposed an novel sub-graph search algorithm, which is based on Summarization Graph model. For each original general graph, we transformed it into its corresponding Summarization Graph. Then we converted Summarization Graphs into complete graphs with unlabeled edges by inserting edge labels into their adjacent vertexes, which captured most structure information of the original graph. Then, in the filtering process of sub-graph search, we utilize the method of retrieving objects with set-valued attributes to get candidate answer set. We proved that our methods could get higher filtering power than existing methods from the view of graph theory. The extensive experiments also validate the performance of our methods, which outperforms existing approaches in pruning power and overall response time.

Acknowledgments. The authors would like to thank Dr. Xifeng Yan and Prof. Jiawei Han for providing gIndex, and Dr. Huahai He and Prof. Ambuj K. Singh for providing Closure-tree, and Mr. Michihiro Kuramochi and Prof. George Karypis for providing the synthetic graph data generator.

References

1. Available at <http://amalfi.dis.unina.it/graph>.
2. H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. Frequent subtree mining - an overview. *Nucleic Acids Research*, 23(10), 2000.
3. D. Cai, Z. Shao, X. He, X. Yan, and J. Han. Community mining from multi-relational networks. In *PKDD*, 2005.
4. J. Cheng, Y. Ke, W. Ng, and A. Lu. *fg-index*: Towards verification-free query processing on graph databases. In *SIGMOD*, 2007.
5. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithm (2nd Edition)*. The MIT Press, 2000.
6. J. H. D.W. Williams and W. Wang. Graph database indexing using structured graph decomposition. In *ICDE*, 2007.
7. S. Fortin. The graph isomorphism problem. *Department of Computing Science, University of Alberta*, 1996.
8. P. Y. H. Jiang, H. Wang and S. Zhou. Gstring: A novel approach for efficient search in graph databases. In *ICDE*, 2007.
9. H. He and A. K. Singh. Closure-tree: An index structure for graph queries. In *ICDE*, 2006.
10. A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *PKDD*, 2000.
11. C. A. James, D. Weininger, and J. Delany. Daylight theory manual daylight version 4.82. *Daylight Chemical Information Systems, Inc.*, 2003.
12. Y. Ke, J. Cheng, and W. Ng. Correlation search in graph databases. In *SIGKDD*, 2007.
13. M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *ICDM*, 2001.

14. M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *ICDM*, 2001.
15. E. G. M. Petrakis and C. Faloutsos. Similarity searching in medical image databases. *IEEE Transactions on Knowledge and Data Engineering*, 9(3), 1997.
16. D. Shasha, J. T.-L. Wang, and R. Giugno. Algorithmics and applications of tree and graph searching. In *PODS*, 2002.
17. E. Tousidou, P. Bozaris, and Y. Manolopoulos. Signature-based structures for objects with set-valued attributes. *Inf. Syst.*, 27(2), 2002.
18. P. Willett. Chemical similarity searching. *J. Chem. Inf. Comput. Sci.*, 38(6), 1998.
19. X. Yan and J. Han. gspan: Graph-based substructure pattern mining. *Proc. of Int. Conf. on Data Mining*, 2002.
20. X. Yan, P. S. Yu, and J. Han. Graph indexing: A frequent structure-based approach. In *SIGMOD*, 2004.
21. N. Zhang, M. T. Özsu, I. F. Ilyas, and A. Aboulnaga. Fix: Feature-based indexing technique for xml documents. In *VLDB*, 2006.
22. S. Zhang, M. Hu, and J. Yang. Treepi: A novel graph indexing method. In *ICDE*, 2007.
23. P. Zhao, J. X. Yu, and P. S. Yu. Graph indexing: Tree + delta \geq graph. In *VLDB*, 2007.