Searching Substructures with Superimposed Distance*

Xifeng Yan† Feida Zhu† Jiawei Han† Philip S. Yu‡
†Department of Computer Science
University of Illinois at Urbana-Champaign
{xyan, feidazhu, hanj}@cs.uiuc.edu
‡IBM T. J. Watson Research Center
psyu@us.ibm.com

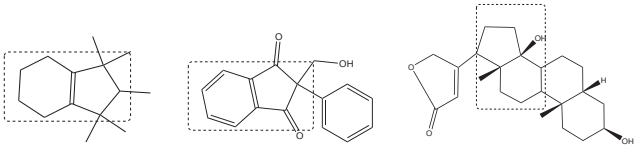
Abstract

*Efficient indexing techniques have been developed for the exact and approximate substructure search in large scale graph databases. Unfortunately, the retrieval problem of structures with categorical or geometric distance constraints is not solved yet. In this paper, we develop a method called PIS (**P**artition-based Graph **I**ndex and **S**earch) to support similarity search on substructures with superimposed distance constraints. PIS selects discriminative fragments in a query graph and uses an index to prune the graphs that violate the distance constraints. We identify a criterion to distinguish the selectivity of fragments in multiple graphs and develop a partition method to obtain a set of highly selective fragments, which is able to improve the pruning performance. Experimental results show that PIS is effective in processing real graph queries.*

1 Introduction

With the increasing volume of graph databases, there is a strong need for fast graph search systems. Unfortunately, traditional indexing mechanisms can no longer address the challenging issues raised by complex graph databases: Given an exponential number of subgraphs in a complex structure, we simply do not know what to index and how to index. Interest has been growing in using unconventional indexing techniques to tackle the search problem. Previous studies focused on two kinds of graph search tasks: (1) the exact substructure (or full structure) search, and (2) the approximate substructure (or full structure) search. The exact substructure search finds all of the graphs in a database that contain the query structure, while the approximate substructure search finds inexact matches in the database. Shasha et al. [12] proposed a path-based approach for the exact substructure search. Yan et al. [16] devised discriminative frequent structures and used them as indexing features. Holder et al. [7] adopted the minimum description length principle for the approximate search. Raymond et al. [10] developed a three-tier algorithm for structure similarity search.

The two search scenarios mentioned so far are mainly involved with the topological structure of graphs. However, there are other similarity search problems that are as important, but which we are unable to handle yet. Let us first check an example.



(a) 1H-Indene (b) Omephine (c) Digitoxigenin

Figure 1. A Chemical Database

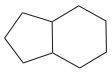


Figure 2. A Query Graph

Example 1 *Figure 1 shows a sample 2D chemical dataset consisting of three molecules. Omephine in Figure 1(b) is an anticoagulant. Digitoxigenin in Figure 1(c) is well-known for its strong cardiotonic effect. Figure 2 shows a query graph. The three sample molecules contain the same topological substructures as the query graph. However, some of their edge labels are different from those in the query graph. We define a mutation distance as the number of times one has to relabel edges in one graph in order*

* The work was supported in part by the U.S. National Science Foundation NSF IIS-02-09199/IIS-03-8215, and an IBM Faculty Award. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

to get another graph. According to this definition, the mutation distance between 1H-Indene in Figure 1(a) and the query graph is 1*: we need to mutate one edge label in 1H-Indene so that it contains exactly the query structure, with exactly the same labels. If a user wants to find graphs whose mutation distance from the query graph is less than 2, the query system should return the first and the third graphs in Figure 1.*

The example above indicates that the substructure search with superimposed distance constraints (SSSD) is a general graph search problem. We formulate the SSSD problem as follows: *Given a set of graphs $D = \{G_1, G_2, \ldots G_n\}$ and a query graph Q, find all graphs G in D such that Q is isomorphic to a subgraph Q' of G and the optimal distance between Q and Q' is less than a threshold σ*. We can also rephrase the SSSD problem as a constrained graph alignment problem: We want to find an alignment of the query graph in target graphs such that the minimum superimposed distance between Q and its image in the target graphs is less than σ.

One solution to this new substructure search problem is to enumerate all of the isomorphic images of Q in the target graphs and check their distance. This brute-force approach may not work well since it is time-consuming to check each graph in a large scale database. In this paper, we develop an algorithm, called PIS (**P**artition-based Graph **I**ndex and **S**earch), to tackle the SSSD problem. Our strategy is to first build a fragment-based index on the graph database, then partition each query graph into highly selective fragments, use the index to efficiently identify the set of candidate graphs, and verify each candidate to find all eligible answers. Our approach has two advantages over the brute-force method: (1) All operations except the candidate verification are only involved with the index structure, thus avoiding one-by-one subgraph isomorphism computation for graphs in the database. The isomorphism computation is performed on the candidate graph set, which is of a significantly smaller size. (2) The candidate set itself is identified efficiently by pruning most invalid graphs with the help of selective fragments and a distance lower bound introduced in this paper.

We call the index strategy of PIS *fragment-based index*. Graphs in the database are decomposed into fragments (probably overlapping) and indexed to facilitate similarity search. Fragments with the same topology can be indexed using an R-tree [4, 11] or a metric-based index [6]. We observed that, for many distance measures, the superimposed distance between a query graph and a target graph is lower-bounded by the sum of distances between their corresponding indexed non-overlapping fragments.

This lower bound leads to efficient pruning of most invalid graphs in the database. A query graph is partitioned into fragments according to the index structure. Since there are multiple ways to partition a query graph, it is important to choose the optimal one that achieves the best pruning performance. We identify the criterion of an optimal partition that should give a set of non-overlapping fragments with the highest selectivity. This optimization problem is, as we will later prove, equivalent in computational complexity to a well-known NP-hard problem: maximum weighted independent set (MWIS). Although theoretical results show that MWIS does not have any polynomial approximation solution, the heuristic greedy algorithm we developed works well for real chemical datasets. We call the overall search strategy *partition-based search*.

Our contribution in this study is an examination of a new search problem in graph databases and the proposal of a partition-based index and search algorithm. The development of our method exposes new database management challenges in complicated graph databases.

2 Preliminaries

Graphs with attributes are called *labeled graphs*. A graph G is a subgraph of G' if there exists a subgraph isomorphism from G to G', denoted by $G \subseteq G'$. G' is called a supergraph of G. The skeleton (without labels) of a graph is called its *structure* or *topology*. The definition of subgraph isomorphism in this paper only considers the structure of a graph.

If G is a subgraph of G' and vice versa, we say G is isomorphic to G', written $G \cong G'$. If G is a subgraph of G' and also has the same label information with G', we say G is a subgraph of G' with reserved label information, written $G \sqsubseteq G'$.

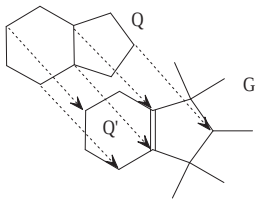


Figure 3. Superposition

Example 2 *Figure 3 shows a superposition between the query graph in Figure 2 (Q) and the first graph in Figure 1 (G). Q' is the image of Q in G. As one can see, $Q \not\sqsubseteq G$ although $Q \subseteq G$.*

Subgraph isomorphism only gives the structural comparison between two graphs. The label information is also critical in determining the characteristics of graphs. Thus, we need a distance measure to differentiate labeled graphs with

the same structure. This kind of distance is termed *superimposed distance*, a distance measure applied to two superimposed graphs. Here we introduce two commonly used measures: Mutation Distance (MD) and Linear Mutation Distance (LD).

Suppose we have two isomorphic labeled graphs, G and G'. We can build a superposition from G to G', which maps each vertex of G to a unique vertex in G'. The mutation distance between G and G' is defined as follows,

$$\mathbf{MD} = \sum_{v'=f(v)} \mathbf{D}(l(v), l'(v')) + \sum_{e'=f(e)} \mathbf{D}(l(e), l'(e'))$$

where $\mathbf{D}$ is a mutation score matrix, l is a label function, and f is an isomorphic function, $f : V(G) \rightarrow V(G')$. The mutation score matrix includes the distance score between a mutation from one label to another label. If the labels are numeric, a linear distance function may be appropriate for distance measure, e.g.,

$$\mathbf{LD} = \sum_{v'=f(v)} |w(v) - w'(v')| + \sum_{e'=f(e)} |w(e) - w'(e')|$$

where w and w' are the weight functions of G and G'.

Since multiple superpositions may exist for two isomorphic graphs, we usually select the best superposition that has the smallest distance.

Definition 1 (Minimum Superimposed Distance) *Given two graphs, Q and G, let M be the set of subgraphs in G that are isomorphic to Q, $M = \{Q' | Q' \sqsubseteq G \wedge Q' \cong Q\}$. The minimum superimposed distance between Q and G is the minimum distance between Q and Q' in M,*

$$d(Q, G) = \min_{Q' \in M} d(Q, Q'), \tag{1}$$

where $d(Q, Q')$ is a distance function of two isomorphic graphs Q and Q'.

Definition 2 (Substructure Search with Superimposed Distance (SSSD)) *Given a set of graphs $D = \{G_1, G_2, \ldots G_n\}$ and a query graph Q, SSSD is to find all $G_i \in D$ such that $d(Q, G_i) \leq \sigma$.*

A naive solution is to scan the whole database and check whether a target graph has a superposition with a distance less than the threshold. This solution is not scalable. A better solution, which we call *topoPrune*, gets rid of graphs that do not contain the query structure first, and then checks the remaining candidates to find the qualified graphs. topoPrune is more efficient than the naive approach. However, it still suffers huge computational costs since it has to enumerate the superpositions of a query graph in a large set of candidate graphs. If most of the candidate graphs are not qualified, topoPrune could be very inefficient.

3 Framework of PIS

Besides structure pruning, we can also utilize the superimposed distance constraint to prune candidates. In PIS, we partition a query graph Q into non-overlapping fragments g_1, g_2, ..., and g_n, and use them to do pruning. If a distance function satisfies the following inequality,

$$\sum_{i=1}^{n} d(g_i, G) \leq d(Q, G), \tag{2}$$

we can set the lower bound of the superimposed distance between Q and G by the superimposed distance between g_i and G. Whenever $\sum_{i=1}^{n} d(g_i, G) > \sigma$, we can safely remove G from the answer set. For this kind of pruning, we only need two operations: (1) enumerate fragments in the query graph and (2) search the index to calculate the superimposed distance $d(g_i, G)$. We have

$$d(g_i, G) = \min_{g' \sqsubseteq G \wedge g' \cong g_i} d(g_i, g'). \tag{3}$$

Therefore, if we index all of the fragments in G that have the same topology with g_i, we can calculate $d(g_i, G)$ through the index directly. This kind of pruning needs to check the index only, not the original database.

In summary, we are able to use the lower bound given in Eq. (2) to prune more unqualified graphs by indexing fragments in graph databases. This method consists of two components: *fragment-based index* and *partition-based search*. We first formalize the definition of graph partition.

Definition 3 (Graph Partition) *Given a graph $Q = (V, E)$, a partition of G is a set of subgraphs $\{g_1, g_2, \ldots, g_n\}$ such that $V(g_i) \subseteq V$ and $V(g_i) \cap V(g_j) = \varnothing$ for any $i \neq j$.*

Interestingly, many distance functions hold the inequality in Eq. (2) for a given partition. Both distances we mentioned, mutation distance and linear mutation distance, have this inequality. We leave the proof to readers.

In Eq. (3), if a fragment g is indexed, then all of the fragments having the same topology as g should be indexed, since the right side of Eq. (3) has to access all of the superpositions of g in G.

Definition 4 (Structural Equivalence Class) *Labeled graphs G and G' belong to the same equivalence class if and only if $G \cong G'$. The structural equivalence class of G is written $[G]$.*

We formulate the framework of PIS (partition-based graph index and search) in the following three steps.

1. **Fragment-based Index**: We select a set of structures as indexing features according to the criteria proposed

in GraphGrep [12] or gIndex [16]. For each structure f (f is a bare structure without any label), we enumerate all of the fragments in the database that belong to $[f]$ and build an index in which a range query $d(g, g') \leq \sigma$ can be evaluated efficiently, where g and g' are labeled graphs and their skeleton is f.

2. **Partition-based Search**: For a given query graph Q, we partition it into a set of indexed non-overlapping fragments, $g_1, g_2, \ldots, g_n$. For each fragment g_i, we find its equivalence class in the index and submit a range query $d(g_i, g') \leq \sigma$ to find all of the fragments g' in the database that meet the superimposed distance threshold. We then sum up their distance to obtain the lower bound of $d(Q, G)$ for each graph G in the database,

$$\sum_{i=1}^{n} d(g_i, G) = \sum_{i=1}^{n} \min_{g' \sqsubseteq G \wedge g' \cong g_i} d(g_i, g'). \quad (4)$$

   If G does not have any subgraph g' such that $g' \cong g_i$, we drop G from the answer set (structure violation). If the lower bound in Eq. (4) is greater than σ, we also drop G from the answer set (superimposed distance violation). The resulting *candidate answer set*, C_Q, will include all of the graphs that pass the filtering: $C_Q = \{G | G \in D \wedge \sum_{i=1}^{n} d(g_i, G) \leq \sigma\}$.

3. **Candidate Verification**: We calculate the real superimposed distance between Q and the candidate graphs returned in the second step, and then remove graphs that do not satisfy the distance threshold.

4 Fragment-based Index

In this section, we present the details of constructing a fragment-based index, the first step in the framework of PIS. The index construction has two steps. In the first step, we select structures as features. These structures do not include label information. In the second step, any fragment in the database that has the selected structure is identified and indexed. That is, for each selected structure f, we enumerate all of the fragments in the graph database that belong to $[f]$.

Figure 4 illustrates the procedure of inserting a selected fragment g into the index. The structure of g is first transformed into a sequence $s(g)$, which is indexed in a hash table. We use a canonical representation of g that can translate a graph into a unique sequence. If two graphs belong to the same class, they will share the same canonical representation. When the hashing is performed on g, we only consider the canonical representation of its structure, not its labels. By doing so, we can group different fragments according to their structural equivalence class. There are several forms

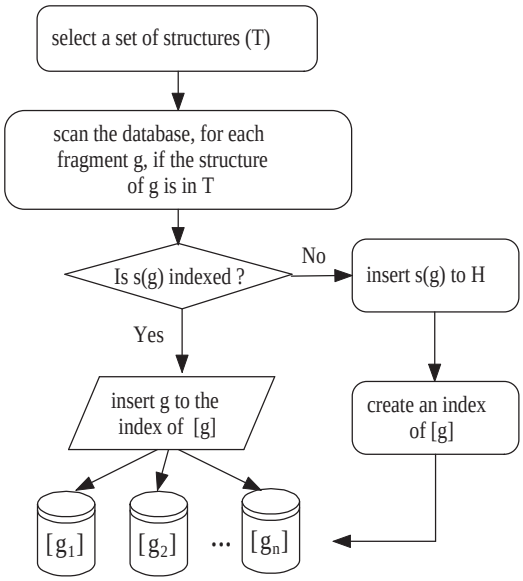


Figure 4. Index Construction

of canonical representation available. A naive one is to concatenate rows or columns of the adjacency matrix of a graph into an integer sequence and use the minimum sequence as the canonical representation for this graph. There are more sophisticated canonical representations such as DFS coding [15]. Overall, we can always find a representation function $s : G \mapsto S$ such that if $G \cong G'$, $s(G) = s(G')$ and if $G \ncong G'$, $s(G) \neq s(G')$, where S is a sequence space.

Using a canonical representation system, we can quickly identify the class of a graph by checking its canonical representation. The canonical representations are indexed in a hash table H, as shown in Figure 5.

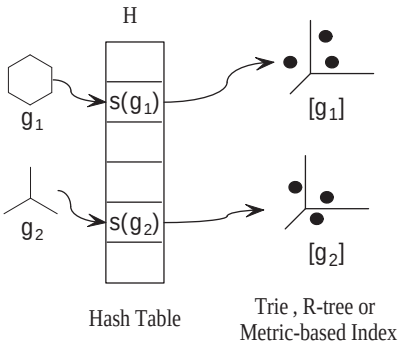


Figure 5. The Index Components of PIS

For each equivalence class (every hash table entry), we build an index structure to facilitate range queries $d(g, g') \leq \sigma$. There are various kinds of indexing structures available

for this task. The selection of index structure is determined by the type of distance function. For the mutation distance, we can use a trie to accommodate the sequential representations of the labeled graphs. For linear mutation distance, we can use an R-tree to do the range query.

In summary, for a fragment g in the database, when hashing is performed, the label information of g is ignored, i.e, only the skeleton structure is considered. When g is inserted into the index of $[g]$, its label information is included.

Example 3 *Let D be a graph database where graphs have weighted edges. A user applies a linear mutation distance, $LD(G, G') = \sum_{e'=f(e')} |w(e) - w'(e')|$, to measure the superimposed distance in D. Assume we index all of the fragments having the same structure with g_2 shown in Figure 5. For any fragment g' in D, if $g' \cong g_2$, we can transform g' into a feature vector in a three dimensional space, where each dimension records the weight of one of its edges. We construct an R-tree to index g'. If a query fragment g is isomorphic to g_2, we submit a range query to that R-tree to find all of the vectors g' such that $LD(g, g') \leq \sigma$.*

5 Partition-based Search

Using the fragment-based index, we develop a search strategy to prune candidates for a given query graph. In order to apply the lower bound in Eq. (2), we need to partition the query graph into several non-overlapping indexed fragments. Since the index is built beforehand, a query graph may be partitioned in more than one way. Thus, we have to select an optimal partition that can achieve the best pruning performance. Let us first check an example.

Example 4 *Suppose we index all of the edges in the sample database (Figure 1) and want to find the graphs whose mutation distance with the query graph (Figure 2) is less than 2. If we partition the query graph into single edges, we will not be able to filter any graph since $\sum_{i=1}^{10} d(g_i, G) = 0$, where g_i is an edge in the query graph (the query graph has 10 edges). In contrast, if we select a six-carbon ring fragment, we may successfully prune the graph in Figure 1(b) since its mutation distance with this fragment is 3, greater than the threshold.*

As shown in the above example, different partitions may have different pruning power. The question is how to find an optimal partition. Intuitively, a partition is optimal if it generates the highest lower bound for $d(Q, G)$ such that, if the lower bound is greater than the threshold σ, G can be immediately discarded from the candidate set. The optimal partition of a query graph Q for SSSD on a single graph G is given by:

$$P_{opt(Q,G)} = \arg \max_{P} \sum_{i=1}^{n} d(g_i, G) \tag{5}$$

where $P = \{g_1, g_2, \ldots, g_n\}$ is a partition of Q.

However, when we are given a large graph database, it is simply unaffordable to find an optimal partition between the query graph and each graph in the database. As a tradeoff, we need to find a partition in the query graph that is generally good for all of the graphs in the database, in the sense that it can simultaneously prune away most invalid graphs and quickly give us a small candidate set for further verification. In other words, we need a partition whose fragments have the greatest pruning power, which we measure by the notion of *selectivity* defined as follows.

Definition 5 (Selectivity) *Given a graph database $D = \{G_1, G_2, \ldots, G_n\}$ and a fragment g, if $[g]$ is indexed, the selectivity of g is defined by its average minimum distance between g and the graphs in the database, written as $w(g) = \frac{\sum_{i=1}^{n} d(g,G_i)}{n}$.*

The selectivity can roughly measure the distance between a fragment and an average graph in the database. When $g \not\subseteq G$, $d(g, G) = \infty$. In order to avoid the singularity of $w(g)$, we set the cutoff value of $d(g, G)$ to the maximum distance threshold σ. The closer $w(g)$ to σ, the more selective the fragment g. Using the selectivity as a weight function, we are able to define an optimal partition of a query graph Q for a large graph database with a fragment-based index I,

$$P_{opt(Q,I)} = \arg \max_{P} \sum_{i=1}^{n} w(g_i) \tag{6}$$

where $P = \{g_1, g_2, \ldots, g_n\}$ is a partition of Q. We call this optimization problem *the index-based partition problem*.

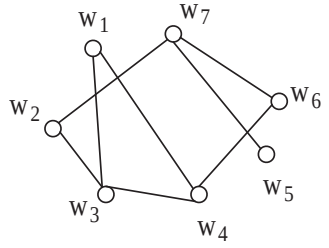


Figure 6. Overlapping-Relation Graph

The Index-based partition problem has a connection to the Maximum Weighted Independent Set problem (MWIS [1]) . Let $g_1, g_2, \ldots, g_m$ be the indexed fragments in Q. We construct an *overlapping-relation graph* $\widetilde{Q}$ to model the overlapping relation among $\{g_i\}$: each fragment g_i is represented as a node v_i in $\widetilde{Q}$; and if g_i and g_j overlap, we connect v_i and v_j. Each vertex v_i is associated with a weight $w_i = d(g_i, G)$ equal to the selectivity of g_i. Figure 6 depicts

an overlapping-relation graph that has seven vertices, corresponding to seven fragments in a query graph. The Index-based Partition is equivalent to finding an independent set with maximum weights in $\widetilde{Q}$.

Definition 6 (Maximum Weighted Independent Set) *A finite Graph G=(V, E) and a function w: $V \mapsto R^+$. A maximum weighted independent set is a subset $S_{opt} \subseteq V$ such that*

$$S_{opt} = \arg \max_{S} \sum_{v \in S} w(v), \tag{7}$$

where S is an independent set of G, i.e. $\forall v, w, \in S$, $(v, w) \notin E$.

A general MWIS problem is NP-hard, as can be shown by an immediate reduction from MIS (Maximum Independent Set), which is a well-known NP-hard problem [3]. Unfortunately, the Index-based Partition problem has the same hardness.

Theorem 1 *Index-based Partition is NP-hard.*

Proof. We prove the theorem by showing that Index-based Partition is at least as hard as MWIS. We give polynomial-time reduction from an instance of MWIS to an instance of Index-based Partition. Let an instance, (I, Q), of index-based partition be an index structure I and a query graph Q. Let an instance, (G, w), of MWIS be a graph $G = (V, E)$ with a weight function $w : V \mapsto R^+$.

Given an instance (G, w) of MWIS, we construct an instance (I, Q) of Index-based Partition as follows:(assuming G contains no self-loops, and it's easy to extend the argument to cases containing self-loops) For each vertex $v_i \in V(G), 1 \leq i \leq |V(G)|$, let all the neighbors of v_i be $\{v_i^1, v_i^2, \ldots, v_i^{n_i}\}$. Replace v_i with a ring of n_i vertices $Ring(v_i) = \{u_1, u_2, \ldots, u_{n_i}\}$, add i self-loops to each vertex on this ring, and replace each edge $v_i v_i^j$ with a new edge $u_j v_i^j$, $1 \leq j \leq n_i$. Do this to all vertices of G and we thus obtain our query graph Q. Each ring, $Ring(v_i)$, together with all its adjacent edges now forms a subgraph $sub(v_i)$ of unique topology in Q. We then construct the index I with each $sub(v_i)$ as a key $[sub(v_i)]$ and set $w(sub(v_i))$, the selectivity of $sub(v_i)$, equal to $w(v_i)$, the weight of vertex v_i, in the original MWIS instance. Run an algorithm for Index-based Partition on this constructed instance (I, Q) and let the solution be P. Observe that, constrained by the index I, P must be a set of subgraphs as described, i.e. each is a ring whose vertices all have the same number of self-loops and each vertex has one "dangling" adjacent edge. Given P, we obtain a solution S to the original MWIS problem as follows: S is initially empty. For each subgraph in P, if each vertex on the ring has i self-loops, add v_i to S. It's easy to verify that this is by construction a bijection between the set of solutions to MWIS and the set of solutions to Index-based Partition, because every maximum weight independent set induces a unique partition of maximum weight and every partition of maximum weight uniquely corresponds to a maximum weight independent set.

Since MWIS is NP-hard and Index-based Partition is at least as hard as MWIS, Index-based Partition is also NP-hard. ■

Figure 6 illustrates the connection between an optimal partition and MWIS. In our problem setting, we often have knowledge about the size of a partition, i.e., the maximum independent set size in $\widetilde{Q}$.

Lemma 1 *Given a query graph Q, let $\widetilde{Q}$ be the corresponding overlapping-relation graph. Let S_{opt} be the maximum weighted independent set of $\widetilde{Q}$, then $|S_{opt}| \leq |Q|/l$, where l is the minimum indexed fragment size.*

Assume the weighted graph $\widetilde{Q} = (\widetilde{V}, \widetilde{E})$ is given in a standard adjacency list representation and let L_v be the linked list of $\widetilde{V}$. Algorithm 1 shows a greedy algorithm to solve MWIS. At each iteration, $Greedy()$ selects a vertex with the maximum weight in L_v and removes all of its adjacent vertices from L_v. This process is repeated until L_v becomes empty.

Algorithm 1 Greedy

Input: A graph $\widetilde{Q} = (\widetilde{V}, \widetilde{E})$ and a function $w : \widetilde{V} \mapsto R$.
Output: An independent set S.

1: let $S \leftarrow \varnothing$;
2: **while** $L_v \neq \varnothing$ **do**
3: scan L_v and find v with maximum $w(v)$;
4: $S \leftarrow S \cup \{v\}$;
5: remove v and all neighbors of v from L_v;
6: **return** S;

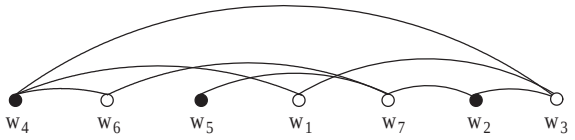


Figure 7. Greedy Selection

Example 5 *Figure 7 shows a running example of $Greedy()$. Suppose the weights of vertices have the following order, $w_4 \geq w_6 \geq w_5 \geq w_1 \geq w_7 \geq w_2 \geq w_3$. $Greedy()$ choose w_4, w_5, and w_2 as a solution.*

The result returned by $Greedy()$ may not be optimal. We use the optimality ratio, defined by $\frac{w(S)}{w(S_{opt})}$, to measure the

quality of a returned independent set in comparison with an optimal solution.

Theorem 2 *Given a graph $\widetilde{Q} = (\widetilde{V}, \widetilde{E})$, $Greedy()$ runs in $O(cn)$ time and has an optimality ratio of $1/c$, where $n = |\widetilde{V}|$ and $c = \arg\ \max_S |S|$, S is an independent set of $\widetilde{Q}$,*

In Theorem 2, c is the maximum independent set size of $\widetilde{Q}$, which is also the maximum partition size of Q. According to Lemma 1, $c \leq |Q|/l$, where $|Q|$ is the query graph size and l is the minimum indexed fragment size. In practice, we always find c to be a small constant.

We can further improve $Greedy()$ so that a $\lceil c/k \rceil$ optimality ratio can be guaranteed. Instead of selecting a vertex with the maximum weight, we select a *maximum independent k-set*, a set of k vertices that are not adjacent and whose sum of weights is maximum among all independent k-sets. The maximum independent k-set is allowed to have less than k vertices. In each iteration, we select a maximum independent k-set and remove all the neighbors of its vertices in $\widetilde{Q}$. Since we have to enumerate all independent k-sets in n vertices, the new algorithm, called $EnhancedGreedy(k)$, runs in $O(ckn^k)$.

Theorem 3 *Given a graph $\widetilde{Q} = (\widetilde{V}, \widetilde{E})$, $Enhanced\ Greedy(k)$ achieves a guaranteed optimality ratio of $\lceil c/k \rceil$ in $O(ckn^k)$ time, where $n = |\widetilde{V}|$, $c = \text{argmax}_S|S|$, S is an independent set of $\widetilde{Q}$, and $1 \leq k \leq |\widetilde{V}|$.*

Theoretically, $EnhancedGreedy(k)$ has a better optimality ratio than $Greedy()$ in the worst case, though it is very slow when k is large. However, we found that $EnhancedGreedy(k)$ (k is set at 2) has comparable performance with $Greedy()$ in real datasets, indicating $Greedy()$ actually works well on average. Theorems 2 and 3 also indicate that if we can increase the size of the smallest indexed fragments, we can improve the optimality ratio in the worst case. Therefore, we prefer indexing larger fragments. Furthermore, larger fragments are usually more selective than small ones. Unfortunately, the number of fragments increases exponentially with their size. In practice, we have to make a tradeoff.

6 Implementation

In this section, we outline our partition-based graph search method in Algorithm 2.

We denote the candidate graph set by C_Q for a given query graph Q and the set of indexed fragments in Q by F. F may contain many overlapping fragments in Q. In the first step, it enumerates the indexed fragments in a query

Algorithm 2 Partition-based Graph Search

Input: Graph database $D = \{G_1, \ldots, G_n\}$,
Query graph Q,
Maximum distance threshold σ.
Output: Candidate answer set C_Q.

1: $C_Q \leftarrow D$;
2: $F \leftarrow \varnothing$;
3: **for each** fragment $g \sqsubseteq Q$ and $[g]$ is indexed **do**
4: $\quad F \leftarrow F \cup \{g\}$;
5: remove fragments g from F if $w(g) \leq \epsilon$;
6: **for each** fragment $g \in F$ **do**
7: $\quad$ calculate g's canonical label, $s(g)$;
8: $\quad$ locate the index structure I pointed by $s(g)$;
9: $\quad$ submit a range query $d(g, g') \leq \sigma$ to I;
10: $\quad T \leftarrow \varnothing$;
11: $\quad$ **for each** pair $\langle g', G \rangle$ s.t. $d(g, g') \leq \sigma$ **do**
12: $\quad\quad$ **if** $G \in T$ **then**
13: $\quad\quad\quad d(g, G) \leftarrow \min(d(g, G), d(g, g'))$;
14: $\quad\quad$ **else**
15: $\quad\quad\quad d(g, G) \leftarrow d(g, g')$;
16: $\quad\quad\quad T \leftarrow T \cup \{G\}$;
17: $\quad C_Q \leftarrow C_Q \cap T$;
18: $\quad w(g) \leftarrow \frac{\sum_{G \in T} d(g,G)}{n} + \frac{n - |T|}{n} \times \sigma$;
19: construct an overlapping relation graph for Q;
20: select a partition P according to $Greedy()$;
21: **for each** $G \in C_Q$ **do**
22: $\quad$ **if** $\sum_{g \in P} d(g, G) > \sigma$ **then**
23: $\quad\quad C_Q \leftarrow C_Q \setminus \{G\}$;
24: **return**;

graph Q (Lines 3–4). On Line 5, we drop all of the fragments whose selectivity is less than ϵ. Since they are contained nearly by all graphs in the database, these fragments do not have pruning capability. We may tune the value of ϵ to maximize the performance.

For each fragment in F, we submit a range query to find all of the graphs whose distance with that fragment is less than or equal to the maximum distance threshold (Lines 7–17). The range query will be answered by the corresponding index structure such as trie, R-tree, or metric-based index. Line 17 eliminates the graphs that do not contain a fragment in Q or the graphs whose superimposed distance with that fragment is greater than σ. The intersection operation in Line 17 will retain those qualified graphs.

Line 18 computes the selectivity of each fragment. We note that there are $(n - |T|)$ graphs that do not contain the structure of g (or whose superimposed distance with g is greater than σ), and each of them will contribute σ/n to $w(g)$ according to Definition 5. Lines 19–20 construct an overlapping relation graph and find a partition through

the $Greedy()$ algorithm. The resulting partition is used to prune graphs that do not satisfy the minimum distance threshold (Lines 21–23).

In our implementation, we do not store real graphs in the index. Instead, we assign a unique graph identifier (an integer) to each graph in the database. Thus, $\langle g', G \rangle$ (Line 11) actually is a pair of a fragment identifier and a graph identifier. Algorithm 2 will return an identifier list. Overall, Algorithm 2 does not directly access the original graphs in the database.

7 Experimental Results

In this section, we perform an empirical study to evaluate the efficiency of PIS. The performance of PIS is compared with topoPrune, the structure pruning algorithm introduced in Section 2. We demonstrate that PIS can substantially improve search efficiency in real graph databases.

The real dataset is from an AIDS antiviral screen database containing the structures of chemical compounds. This dataset is available on the website of the Developmental Therapeutics Program (NCI/NIH)[1]. In this dataset, thousands of compounds have been checked for evidence of anti-HIV activity. The dataset has around 44,000 structures.

We build topoPrune and PIS based on the gIndex algorithm [16]. gIndex first mines frequent structures and then retains discriminative ones as indexing features. Other kinds of features can also be used in PIS. For example, PIS can take paths [12] as features to build the index. topoPrune and PIS are implemented in C++ with standard template library. All of the experiments are done on a 2.5GHZ, 1GB-memory, Intel Xeon PC running Fedora 2.0.

The test dataset consists of $10,000$ graphs that are randomly drawn from the AIDS screen database. These graphs have 25 nodes and 27 edges on average. The maximum one has 214 nodes and 217 edges in total. Note that in this dataset most of the atoms are carbons and most of the edges are carbon-carbon bonds. This characteristic makes the substructure search very challenging. We use the edge mutation distance to define the superimposed distance between two isomorphic graphs. The distance is the number of edges whose labels are mismatched when we superimpose the query graph to a target graph. We select around $2,000$ fragments in this dataset as indexing features, which are grouped together according to their structural equivalence class. Fragments belonging to the same class are put in a trie after they are sequentialized.

The query graphs are directly sampled from the database and are grouped together according to their size. We denote a query set by Q_m, where m is the query graph size. For example, if the graphs in a query set have 20 edges each, the query set is written Q_{20}. Different from the experimental setting in [16], the edges in our dataset are assigned with edge labels, such as single bond, double bond, and so on. We ignore vertex labels in this test in order to make the problem hard. The queries under examination are "finding graphs in the database that contain the query structure and have at most σ mismatched edge labels".

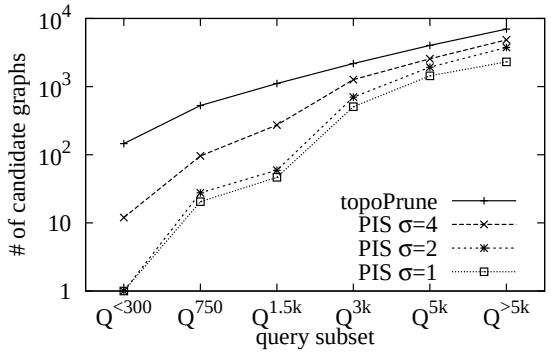


Figure 8. Structure Query with 16 edges

Figure 8 depicts the performance of topoPrune and PIS for the query set Q_{16}. For a given query, we write the number of candidate graphs returned by topoPrune as Y_t and that returned by PIS as Y_p. We divide the query graphs into 6 groups based on the value of Y_t: $0 \leq Y_t < 300$, $300 \leq Y_t < 750$, $750 \leq Y_t < 1,500$, $1,500 \leq Y_t < 3,000$, $3,000 \leq Y_t < 5,000$, and $5,000 \leq Y_t \leq 10,000$. These six groups are written as $Q^{<300}, Q^{750}, Q^{1.5k}, Q^{3k}, Q^{5k}$, and $Q^{>5k}$. In each group, we average Y_t and its counterpart Y_p. The X axis shows the six groups in an order. The Y axis shows the average number of candidate graphs in each group. A better algorithm should filter as many graphs as possible before performing real superimposed distance computation. We plot the performance of PIS with different superimposed distance thresholds (σ). The performance of topoPrune will not change with the distance threshold since it only applies structure pruning.

Figure 8 demonstrates that PIS outperforms topoPrune up to 100 times. We depict the candidate graph reduction ratio $\frac{Y_t}{Y_p}$ in Figure 9. We can see that there is a huge reduction in the number of candidate graphs returned by PIS when topoPrune returns less than $1,000$ candidates. The reduction ratio gradually decreases when more graphs contain the query structure. In the query set $Q^{>5k}$, the reduction ratio is down to 300% when $\sigma = 1$ and 150% when $\sigma = 4$.

Figure 10 depicts the candidate graph reduction ratio of PIS for the query set Q_{24}. Similar performance patterns show in this query set. The pruning process in PIS takes less than 1 second per query, which is negligible compared to the result verification cost.

[1] http://dtpsearch.ncifcrf.gov/FTP/AIDO99SD.BIN

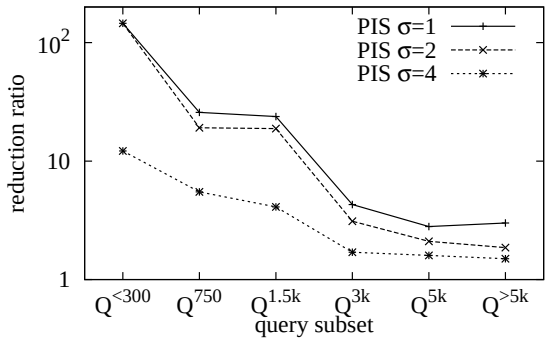


Figure 9. Reduction: PIS over topoPrune

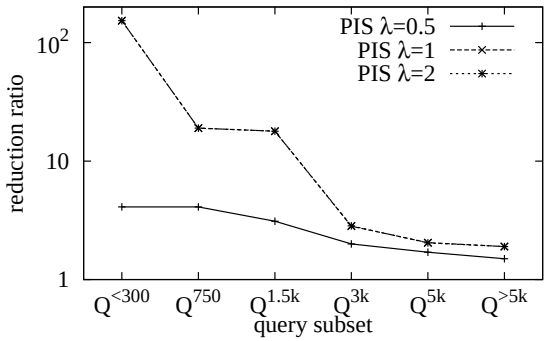


Figure 11. Cutoff Value Sensitivity

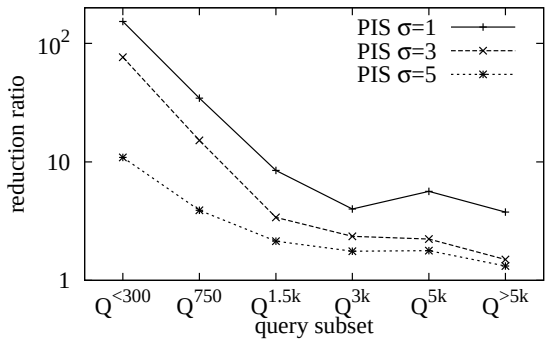


Figure 10. Structure Query with 24 edges

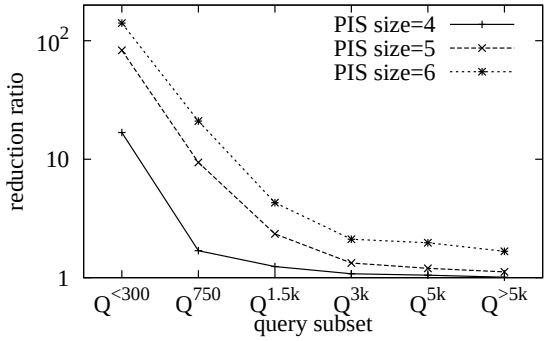


Figure 12. Performance vs. Fragment Size

Next, we check the sensitivity of the cutoff setting in the selectivity computation. In the previous experiments, we set $d(g, G) = \sigma$, when $g \not\subseteq G$ or $d(g, G) > \sigma$. This setting seems to be ad hoc. However, it can be justified through the following experiments. Suppose the cutoff value of $d(g, G)$ is set to $\lambda\sigma$ ($0 \leq \lambda$). We vary the value of λ. If $\lambda \gg 1$, the selectivity of g turns out to be proportional to the number of graphs that do not contain g; Figure 11 shows the pruning performance for the query set Q_{16} with the distance constraint, $\sigma = 2$.

According to Figure 11, we find that the pruning performance descends when $\lambda < 1$. In contrast, there is no performance change when $\lambda > 1$. The two curves of $\lambda = 1$ and $\lambda = 2$ are completely overlapping, indicating that the pruning is not sensitive to the setting of λ when it is greater than 1.

We then test the pruning performance with varying sizes of maximum indexed fragments, from 4 edges to 6 edges. The results are depicted in Figure 12. As discussed in Section 5, the pruning performance will improve if we index larger fragments, since larger fragments are not only more selective, but also result in smaller partition sizes. In this case, the greedy partition algorithm has a better bound in comparison with the optimal one.

8 Related Work

Structure search including similarity search has been studied in several fields. Shasha et al. [12] proposed a path-based approach for the exact substructure search. Yan et al. [16] devised discriminative frequent structures and used them as indexing features to improve search performance. Holder et al. [7] adopted the minimum description length principle for the approximate search. Raymond et al. [10] developed a three-tier algorithm for full structure similarity search, which became a commercial tool in Pfizer. Funk et al. studied how to build a 3D model search engine using spherical harmonics [2]. For 3D structure comparison and protein structure superposition, efficient algorithms such as geometric hash [14], DALI [8], and LOCK [13] were developed. However, these methods mainly focus on aligning 3D points along a sequential skeleton (protein primary structure), not the general SSSD problem that we examined in this paper.

The initial work on substructure similarity search was done by Hagadone [5]. He applied vertex and edge labels to screening. Messmer and Bunke [9] studied the reverse substructure similarity search problem in pattern recognition. Our recent work [17] accessed the substructure similarity

problem based on the number of allowable missing edges, instead of the SSSD problem studied in this paper.

9 Conclusions

In this paper, we proposed a new graph search problem that has additional similarity requirements for the categorical or geometric attributes associated with graphs. Existing algorithms are unable to process this new search request efficiently. Thus, we proposed a novel strategy that selects "discriminative" fragments in a query graph and uses an index to find graphs that contain isomorphic subgraphs to these fragments while the overall distance is retained within a given threshold. We developed two components, fragment-based index and partition-based search, to implement this strategy. We also identified a criterion to distinguish the selectivity of different fragments and demonstrated that a good partition should have a set of highly selective non-overlapping fragments. Surprisingly, we can transform this partition selection problem to the well-known maximum weighted independent set problem (MWIS). Although MWIS does not have a polynomial-time solution, we showed that a greedy solution works well for improving search efficiency in real datasets.

References

[1] C. Bron and J. Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Comm. of the ACM*, 16:575–577, 1973.

[2] T. Funkhouser, P. Min, M. Kazhdan, J. Chen, A. Halderman, D. Dobkin, and D. Jacobs. A search engine for 3d models. *ACM Trans. on Graphics*, 22:83–105, 2003.

[3] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman & Co., New York, 1979.

[4] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proc. 1984 ACM Int. Conf. on Management of Data (SIGMOD'84)*, pages 47 – 57, 1984.

[5] T. Hagadone. Molecular substructure similarity searching: efficient retrieval in two-dimensional structure databases. *J. Chem. Inf. Comput. Sci.*, 32:515–521, 1992.

[6] G. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. In *ACM Trans. on Database Systems (TODS)*, pages 517 – 580, 2003.

[7] L. Holder, D. Cook, and S. Djoko. Substructure discovery in the subdue system. In *Proc. AAAI94 Workshop on Knowledge Discovery in Databases (KDD94)*, page 169 180, 1994.

[8] L. Holm and C. Sander. Protein structure comparison by alignment of distance matrices. *J. of Molecular Biology*, 233:123–138, 1993.

[9] B. Messmer and H. Bunke. A new algorithm for error-tolerant subgraph isomorphism detection. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 20:493 – 504, 1998.

[10] J. Raymond, E. Gardiner, and P. Willett. Rascal: Calculation of graph similarity using maximum common edge subgraphs. *The Computer Journal*, 45:631–644, 2002.

[11] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-Tree: A dynamic index for multi-dimensional objects. In *Proc. 1987 Int. Conf. on Very Large Data Bases (VLDB'87)*, pages 3–11, 1987.

[12] D. Shasha, J. Wang, and R. Giugno. Algorithmics and applications of tree and graph searching. In *Proc. 21th ACM Symp. on Principles of Database Systems (PODS'02)*, pages 39–52, 2002.

[13] A. Singh and D. Brutlag. Hierarchical protein structure superposition using both secondary structure and atomic representations. In *Proc. 5th Int. Conf. on Intelligent Systems for Molecular Biology (ISMB'97)*, pages 284 – 293, 1997.

[14] H. Wolfson and I. Rigoutsos. Geometric hashing: An introduction. *IEEE Computational Science and Engineering*, 4:10–21, 1997.

[15] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *Proc. 2002 Int. Conf. on Data Mining (ICDM'02)*, pages 721–724, 2002.

[16] X. Yan, P. Yu, and J. Han. Graph indexing: A frequent structure-based approach. In *Proc. 2004 ACM Int. Conf. on Management of Data (SIGMOD'04)*, pages 335 – 346, 2004.

[17] X. Yan, P. Yu, and J. Han. Substructure similarity search in graph databases. In *Proc. 2005 ACM Int. Conf. on Management of Data (SIGMOD'05)*, pages 766 – 777, 2005.