

# Top-K Subgraph Matching Query in A Large Graph

Lei Zou<sup>1</sup> \*      Lei Chen<sup>2</sup>      Yansheng Lu<sup>1</sup>

<sup>1</sup>Huazhong Univ.of Sci. & Tech., Wuhan, China, {zoulei,lys}@mail.hust.edu.cn

<sup>2</sup>Hong Kong Univ.of Sci. & Tech., Hong Kong, China, leichen@cse.ust.hk

## Abstract

Recently, due to its wide applications, subgraph search has attracted a lot of attention from database and data mining community. Sub-graph search is defined as follows: given a query graph  $Q$ , we report all data graphs containing  $Q$  in the database. However, there is little work about sub-graph search in a single large graph, which has been used in many applications, such as biological network and social network.

In this paper, we address top-k sub-graph matching query problem, which is defined as follows: given a query graph  $Q$ , we locate top-k matchings of  $Q$  in a large data graph  $G$  according to a score function. The score function is defined as the sum of the pairwise similarity between a vertex in  $Q$  and its matching vertex in  $G$ . Specifically, we first design a balanced tree (that is  $G$ -Tree) to index the large data graph. Then, based on  $G$ -Tree, we propose an efficient query algorithm (that is **Ranked Matching algorithm**). Our extensive experiment results show that, due to efficiency of pruning strategy, given a query with up to 20 vertices, we can locate the top-100 matchings in less than 10 seconds in a large data graph with 100K vertices. Furthermore, our approach outperforms the alternative method by orders of magnitude.

## 1. Introduction

Graph is an important data structure in computer science, which can model objects as *vertices* and their pairwise relationships as *edges*. Graph database finds many applications in chem-informatics [22], or pattern recognition [4] and complex network [24]. Basically, graph databases can be divided into two categories [13]:

First, *graph-transaction setting*. There is a large set of relatively small graphs (<300 vertices)(called *transactions*) in the database, such as compound structures and graph patterns. Recently, a lot of important work about sub-graph search has been conducted in the graph-transaction database [19, 23, 11, 25, 9, 6,

\*The work was done when the first author visited Hong Kong University of Science and Technology as a visiting scholar.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

3].

Second, *single-graph setting*. Different from the first category, there is only one graph in the database, however, the size of graph is very large (>10K vertices), such as biological network [14], social network [21] and WWW [1]. There are many interesting problems in this category, such as community detection [2] and frequent sub-graph mining [13] and connectivity [20].

Among of these graph-based problems, (sub)graph matching [8] is one of fundamental problems with many practical applications, such as compound search [10] [22], pattern recognition [4]. On the other hand, as we know, sub-graph isomorphism is NP-complete [8]. In fact, it is exact the intrinsic hardness and popular applications that motivate people to design practical algorithms for graph matching problems, such as [19, 23, 11, 25, 9, 6, 3]

### 1.1 Motivation

In this paper, we propose a novel sub-graph matching query in single-graph database, called *Top-K Subgraph Matching Query*. Given a query graph  $Q$ , we report the top-k matching positions in the large graph  $G$  with the k largest “scores”. Before the formal definition in Section 1.2, let us consider some motivating examples as follows.

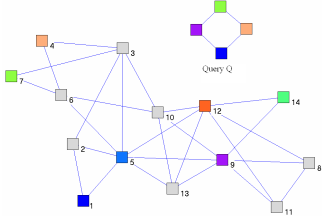
Recently, in life science, due to rapid advances in experimental designs, biologists can obtain a large amount of data describing biological interactions at a genome scale. We can model these interaction networks as large graphs, such as protein-protein interaction networks (PPI) [17], metabolic networks [14] and so on. Vertices and edges are used to represent biological entities and interactions between them. As a motivation example for our problem, let us consider the following scenario: A scientist working on an unwell-studied species has constructed a small portion of a protein pathway complex  $Q$  based on analysis of various experimental data. The scientist is interested in predicting more biological activity about the unwell-studied species. This task can be expressed as locating some “similar” matchings of  $Q$  in a large PPI network  $G$  about a well-studied species. For each matching, we can compute its scores. We only report top-k matchings. The score of a matching is defined as the sum of the pairwise similarity between a vertex (this is a protein) in  $Q$  and its matching vertex in  $G$ . We can evaluate the similarity between two vertices in terms of the protein sequence similarity [18]. Most existing approaches always assume that the query pathway complex is a path or tree, or relax matching definitions (not sub-graph isomorphism, only considering pairwise vertex distances)[24]. Furthermore, it is difficult for these methods to work on a very large data graph.

Actually, top-k sub-graph matching query problem has many other applications, such as social network. Assume that an official

**Table 1.** Meanings of Symbols Used

$Dia(Q)$	The Diameter of Query $Q$
$Sim(u, v)$	The similarity between $u$ and $v$
$Score(X)$	The matching score of $X$
$S(N)$	the node area of $N$
$ES(N, d)$	the $d$ -extension node area of $N$

security department has built a large social network. Each vertex corresponds to a dangerous individual, and each edge denotes the interaction between two corresponding individuals, as shown in Figure 1. Some personal characters about a danger individual are recorded, such as appearance, accent and so on. In order to find a terrorist group, through a long time detection, polices have obtained some personal characters about 4 terrorists (such as appearance and so on), and the interactions among them. Then, they draw a query graph  $Q$  in Figure 1.  $Q$  is not always a complete graph, since the interactions among members in the terrorist group are always well pre-defined. Some members have no interaction with others, even though they are in the same terrorist group. In Figure 1, the vertex color is used to denote the personal character. Similarity between personal characters is denoted by the similarity between the vertex colors. In order to locate the terrorist group in the network, we want to find 4 matching vertices (individuals) in  $G$ , which are similar with vertices in  $Q$  respectively. Furthermore, the interaction among them are the same with that in  $Q$ . Though vertex 1 in Figure 1 is similar with one query vertex, it is eliminated due to no interactions with other suspects. The vertices (5, 9, 12, 14) is a matching of  $Q$  in  $G$ . There may exist many matchings of  $Q$  in the network. Polices should pay more attention to top-k “good” matchings.

**Figure 1. Terrorist Network**

## 1.2 Problem Definition

First, we briefly review the terminologies that we will use in this paper. Table 1 lists commonly used symbols in this paper.

**DEFINITION 1.1. Graph.** A graph  $G$  is defined as  $G = (V, E)$ , where  $V$  is the set of vertices, and  $E$  is a set of vertex pairs.

In this paper, we use graphs to model a large network, such as protein-protein interaction network (PPI), social network and WWW. Notice that, different from graph-transaction databases, vertices have no category labels. They correspond to different objects. For example, in PPI, each vertex corresponds to a protein and the edge corresponds to the interaction between two proteins. In social network, we always use a vertex to denote an individual and an edge to denote the interaction between each other.

**DEFINITION 1.2. Graph Isomorphism.** Assume that we have two graphs  $Q < V_1, E_1 >$  and  $G < V_2, E_2 >$ . Graph  $Q$  is isomorphism to graph  $G$ , iff<sup>1</sup> there exists at least one bijective

<sup>1</sup>iff: if and only if

function  $f: V_1 \rightarrow V_2$  such that for any edge  $uv \in E_1$ , there is an edge  $f(u)f(v) \in E_2$ .

**DEFINITION 1.3. Sub-Graph Isomorphism.** Assume that we have two graphs  $Q < V_1, E_1 >$  and  $G < V_2, E_2 >$ . If there exists at least one sub-graph  $X$  in graph  $G$ , graph  $Q$  is isomorphism to  $X$  under the bijective function  $f$ , graph  $Q$  is sub-graph isomorphism to graph  $G$ .

As mentioned above, a vertex in a graph corresponds to an object. For a vertex  $u$  in query  $Q$  and a vertex  $v$  in data graph  $G$ , we use  $Sim(u, v)$  to evaluate the similarity between two objects associated with vertices  $u$  and  $v$ .  $Sim(u, v)$  depends on different applications. In the PPI example of Section 1.1,  $Sim(u, v)$  is defined as the similarity between two corresponding proteins. In terrorist network example,  $Sim(u, v)$  is defined as the similarity between two corresponding individual characters.

How to evaluate  $Sim(u, v)$  is not focus of this paper, which depends on different applications. In our problem,  $Sim(u, v)$  for any pair  $(u, v)$  is also the input of the algorithm, where  $u$  and  $v$  are two vertices in query  $Q$  and data graph  $G$  respectively. Given a vertex  $u$  in query  $Q$ , a vertex  $v$  in the graph  $G$  can be a matching vertex to  $u$  iff  $Sim(u, v) \geq \gamma$ , where  $\gamma$  is user specified parameter. Therefore, we have the following definition.

**DEFINITION 1.4. Sub-Graph Matching.** Assume that we have two graphs  $Q < V_1, E_1 >$  and  $G < V_2, E_2 >$ .  $Q$  is isomorphism to  $X$  under some bijective function  $f$ , where  $X$  is a sub-graph of  $G$ . We call  $X$  a **sub-graph matching** (matching for short) of  $Q$  in graph  $G$ , iff, for any vertex  $u$  in  $Q$ ,  $Sim(u, f(u)) \geq \gamma$ . Furthermore, the vertex  $f(u)$  in  $G$  is called the **matching vertex** w.r.t<sup>2</sup> vertex  $u$  in query  $Q$ .

It is clear that, given a query graph  $Q$ , there may exist many matchings of  $Q$  in the large data graph. For each matching  $X$ , we define the matching score  $Score(X)$  as follows:

**DEFINITION 1.5. Matching Score.** Given a query graph  $Q$  and a large data graph  $G$ ,  $X$  is a matching of  $Q$  in graph  $G$ . The score of  $X$  is defined as

$$Score(X) = \sum_{i=1}^{|V(Q)|} Sim(v_i, f(v_i)) \quad (1)$$

, where  $|V(Q)|$  is the number of vertices in  $Q$ .

We would like to point out that matching score function also depends on different applications. In fact, our algorithm proposed in this paper holds for any *monotone aggregate* matching score function. For the clear presentation, we use sum function as  $Score(X)$  in this paper. Our *top-k subgraph matching* query is defined as follows:

**DEFINITION 1.6. (Problem Definition) Top-K Subgraph Matching Query** (top-k matching for short). Given a query graph  $Q$ , a large data graph  $G$  and  $Sim(u, v)$  for any vertex pair  $(u, v)$  ( $u$  and  $v$  are two vertices in  $Q$  and  $G$  respectively), we want to locate  $k$  matchings  $X_i$  in  $G$ ,  $i=1\dots k$ , whose matching scores are the first  $k$  largest.

Furthermore, in our problem,  $|V(Q)| < \log|V(G)|$ , namely, query  $Q$  is a small graph compared with the data graph  $G$ .

**Example 1** (Running Example). Given a query graph  $Q$  and a large data graph  $G$  in Figure 2, we want to locate Top-2 matchings. The vertex similarity threshold is  $\gamma=0.1$ . In Figure 2, we only report all vertex pairs whose similarities are no less than  $\gamma$ .

<sup>2</sup>w.r.t: with regard to

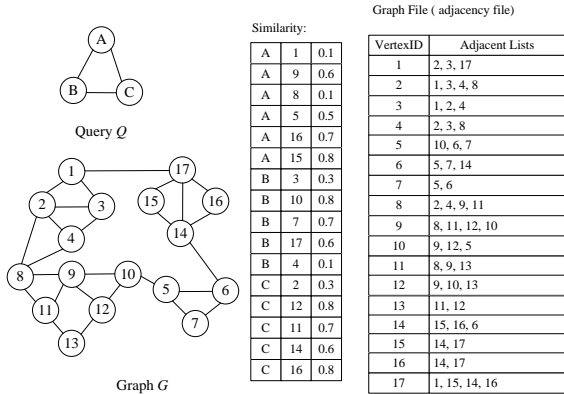


Figure 2. Running Example

Notice that, each vertex in query  $Q$  and data graph  $G$  corresponds to one object. ‘A’ and ‘1’ in Figure 2 are vertex ID, not vertex labels.

### 1.3 Our Approaches

In this problem, we have to face sub-graph isomorphism during query processing, known to be NP-complete. Obviously, for any matching  $X$  of  $Q$  in the data graph  $G$ ,  $X$  is located into a local area with diameter  $\leq Dia(Q)$  in  $G$ , where  $Dia(Q)$  is the diameter of query  $Q$ . We can check the local area around each vertex in  $G$  by sub-graph isomorphism until we find all matchings of  $Q$  in  $G$ . In order to reduce the number of sub-graph isomorphism checking, we can apply feature-based pruning strategy. *If a feature (that is sub-structure) of a query  $Q$  does not exist in some local area of  $G$ , the local area cannot contain any matching of  $Q$ , namely, the local area can be pruned safely.* Feature-based pruning is a popular method in sub-graph search problem [19, 23, 25, 3]. However, in our problem, given a query  $Q$ , the matchings of  $Q$  exist in most local areas of the data graph  $G$ , especially when  $|Q|$  is small. It means that we can prune few local areas in  $G$ . The underlying reason is that all vertices in our problem are non-labeled. Therefore, feature-based pruning cannot work well in the problem.

In this paper, we adopt another kind of pruning strategy, that is *score-upper bound pruning*: *for a local area in the data graph  $G$ , if there exist no matching whose score can be in top- $k$  matching scores, the local area can be filtered out safely.* The efficiency of score-upper bound pruning is derived from clusters in the large graph. In many graph clusters, matching scores are small. Since, in these clusters, there exist few vertices having similar characters to that in query  $Q$ . In summary, we made the following contributions in this paper:

1. We propose a novel sub-graph search problem in a single large graph, which locates and reports top- $k$  matchings.
2. We design a height-balanced tree, *G-Tree*, to index a large graph.
3. Based on *G-Tree* index, to answer *top- $k$  matching* query, we propose **Ranked Matching** algorithm.
4. Extensive experiments on two real data sets evaluate the efficiency of our methods.

The remainder of the paper is organized as follows: Related work is discussed in Section 2. We design *G-tree* index and propose novel **Ranked Matching** algorithms in Section 3. We evaluate

our method in the extensive experiments in Section 4. Finally, We conclude the paper in Section 5.

## 2. Related Work

Recently, there is increasing research interest in graph databases. Here, we make a brief review of the related work, which can be categorized into three groups: 1) sub-graph search in graph-transaction database; 2) pathway search in the biological network; 3) graph cluster and community detection.

**Sub-graph search in graph-transaction database.** Given a query graph  $Q$ , we need to report all data graphs containing  $Q$  in the database [19, 23, 11, 25, 9, 6, 3]. The popular pruning strategy in sub-graph search problem is feature-based pruning: If a feature (that is sub-structure) of query  $Q$  does not exist in some data graph  $G_i$ ,  $G_i$  can be filtered out safely. The straightforward method of extending the strategy to our problem is that, if a local area of the large graph  $G$  does not consist a feature of query, the local area can be pruned. However, it has no good pruning power. The underlying reason is that all vertices in our problem has no categorical labels.

**Pathway search in the biological network.** Given a pathway complex  $Q$ , we want to find a sub-structure  $X$  in the biological network that is most similar to  $Q$ . In the recent work [24], authors propose *GraphMatch* algorithm. The matching definition in [24] replaces sub-graph isomorphism by pair-wise vertex distance, which is different from ours. In experiments, we choose *GraphMatch* as an alternative method for comparison. *GraphMatch* can work well in biological network (about 5K vertices), but it can not work on a very large graph (100K vertices).

**Graph cluster and community detection** The goal of community detection is to cluster the vertices in the large graph into groups that share common characteristics. Many graph partition and clustering algorithms have been proposed in the literature, such as spectral clustering [15], METIS [12] and so on. In fact, the pruning effect of our method benefits from “community” and “graph clusters”, which is discussed in Section 3.2.

## 3. Ranked Matching Algorithm

In this section, we design *G-Tree* to index a large data graph in Section 3.1. Then, based on *G-Tree*, we propose *RM* algorithm for top- $k$  matching query problem in Section 3.2. *Advanced RM* algorithm is proposed in Section 3.3.

### 3.1 G-Tree Structure

Many real large graphs always display the *hierarchical topology* [16], such as PPI, social network and WWW. In these large graphs, there exist many clusters. The vertices in a cluster are highly connected, and they have only a few or no neighbors outside this cluster. In PPI, these clusters always correspond to different functional groups. In friendship network, they denote the communities with shared interests. Iteratively, small clusters are also densely interconnected, which form larger clusters. Inspired by the *hierarchical topology* in the real large graphs, we propose a *G-Tree* index as follows.

**DEFINITION 3.1. G-Tree.** Given a large data graph  $G$ , *G-Tree* index is a height balanced tree, where:

1. Each node  $N$  in *G-Tree* corresponds to a set of vertices in graph  $G$ . The set of vertices is called **Node Area** w.r.t node  $N$ , denoted as  $S(N)$ . In the same level of *G-Tree*, all node areas form a graph  $G$ ’s partition.

2. Given an intermediate node  $N$  and one of its children  $M$  in  $G$ -Tree,  $S(M)$  is a subset of  $S(N)$ .
3. Each node has at least  $m$  children unless it is root,  $m > 2$ . Each node has at most  $M$  children and  $\frac{(M+1)}{2} \geq m$ .

In this paper, for presentation convenience, we preserve *node* for  $G$ -Tree and *vertex* for the data graph and query graph. The  $G$ -tree index for the running example is illustrated in Figure 3.

**Tree Construction** The straightforward approach to build  $G$ -Tree is to perform insertion sequentially. Here, we discuss  $G$ -Tree construction in up-bottom fashion. We can use graph partition algorithm, such as METIS [12], on the data graph to obtain nodes in the first level. Iteratively, for each node, we obtain its child nodes by graph partition. We terminate the above process when each leaf node area can be stored in a disk page.

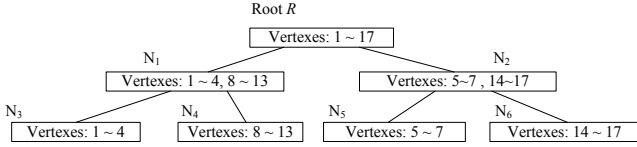


Figure 3.  $G$ -Tree

**Insertion and Split** To insert a vertex  $v$ , an insertion operation begins at the root of  $G$ -Tree and iteratively chooses a child node until it reaches a leaf node. The main challenge of insertion is the criterion for choosing a child node. In order to handle the problem, we propose the *Cluster Coefficient* in Definition 3.2. We always find a child node  $N_i$  that maximizes the cluster coefficient  $Cluster(v \cup N_i)$  among all  $N_i$ .

DEFINITION 3.2. **Cluster Coefficient.**

$$Cluster(N) = \frac{|AllEdges(N)| - |OutEdges(N)|}{|AllEdges(N)|}$$

, where  $|OutEdges(N)|$  is the number of out edges linking vertices in  $S(N)$  to vertices out of  $S(N)$ .  $|AllEdges(N)|$  is the number of all edges adjacent to vertices in  $S(N)$ .

Analogous to other balanced tree, such as  $R$ -tree, for a node  $N$ , we need to split  $N$  into two nodes if the number of children is larger than  $M$ , where  $M$  is maximal fanout. We perform graph partition on  $N$  in order to obtain new nodes  $N_1$  and  $N_2$ . Splitting may cause the parent node to split as well and this procedure may repeat all the way up to the root.

**Deletion and Merge** To delete a vertex  $v$  from the graph  $G$ , we find the leaf node in  $G$ -Tree where  $v$  is stored. For any node  $N$  along the path, we delete  $v$  from node area  $S(N)$ . After deletion, if node  $N$  has less than  $m$  children, where  $m$  is minimal fanout of  $G$ -Tree, we merge  $N$  into another  $N_i$  in the same level. The criterion for choosing  $N_i$  is to maximize  $Cluster(N_i \cup N)$  among all  $N_i$ . The procedure may propagate up to the root.

### 3.2 RM Algorithm

The large search space is the challenge to the top- $k$  matching query problem. According to analysis in Section 1.3, we adopt the *score upper-bound* pruning strategy in this paper. First, we discuss the topological relationships between a matching and a *node area* (defined in Definition 3.1) in  $G$ -Tree as follows.

DEFINITION 3.3. **Contain, Overlap and Outside.** Given a query  $Q$  and a large graph  $G$ , a subgraph  $X$  of  $G$  is a matching of  $Q$ .  $N$  is a node of  $G$ -Tree. We use  $S(X)$  to denote the set of vertices in  $X$ . The topological relationship between the matching  $X$  and the node area  $S(N)$  are:

- 1) **Contain.**  $S(X) \subseteq S(N)$ ;
- 2) **Overlap.**  $S(X) \cap S(N) \neq \phi$  AND  $S(X) \not\subseteq S(N)$ ;
- 2) **Outside.**  $S(X) \cap S(N) = \phi$ .

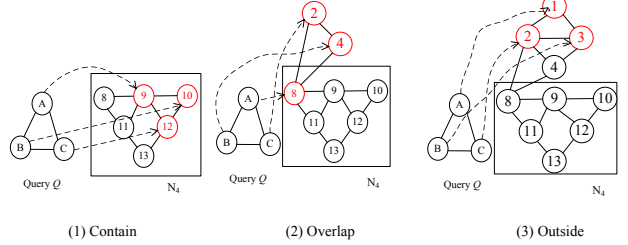


Figure 4. The Relationship Between a Node Area and a Matching

Figure 4 shows the examples about *contain*, *overlap* and *outside*.

DEFINITION 3.4. **Boundary Vertex.** Given a large graph  $G$  and a node  $N$  in  $G$ -tree for graph  $G$ , for a vertex  $v$  in node area  $S(N)$ , if there exists at least one neighbor vertex that is not in  $S(N)$ ,  $v$  is a Boundary Vertex w.r.t  $S(N)$ .

DEFINITION 3.5.  **$d$ -Extension Node Area.** Given a node  $N$  in  $G$ -Tree, a vertex  $v$  is a boundary vertex w.r.t  $S(N)$ .  $N(v, d) = \cup\{v_j | \text{the distance between } v_j \text{ and } v \text{ is no larger than } d\}$ . We assume that there are  $m$  boundary vertices  $v_i$  in  $S(N)$ . The  $d$ -Extension Node Area is denoted as  $ES(N, d) = (\cup_{i=1}^{i=m} N(v_i, d)) \cup S(N)$ .

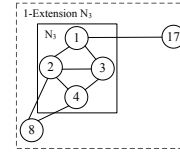


Figure 5.  $d$ -Extension Node Area

Figure 5 shows 1-Extension  $N_3$  (that is  $ES(N_3, 1)$ ) in the running example.

THEOREM 3.1. Given a query graph  $Q$  and a large graph  $G$ ,  $X$  is a matching of  $Q$  in  $G$ . For a node area  $S(N)$ , if  $X$  is contained in  $S(N)$  or overlapped with  $S(N)$ ,  $X$  must be contained in  $ES(N, Dia(Q))$ , where  $Dia(Q)$  is the  $Q$ 's diameter.

Given a query graph  $Q$  with  $n$  vertices and Extension node area  $ES(N, Dia(Q))$ ,  $X$  is a matching contained in  $ES(N, Dia(Q))$ . To evaluate the upper bound of  $Score(X)$ , we work as follows. For each vertex  $u_i$  in query graph  $Q$ , we find all candidate matching vertices  $v_{ij}$  in  $ES(N)$ , where  $j = 1..m$ . Then we define  $Sim(u_i, ES(N, Dia(Q))) = Max(Sim(u_i, v_{ij}))$ . According to Definition 1.4, if  $Sim(u_i, ES(N, Dia(Q))) < \gamma$ , it is clear to know that there exists no matching  $X$  contained in  $ES(N, Dia(Q))$ . It means that the node  $N$  in  $G$ -Tree can be pruned safely. Otherwise, we use the following theorem to evaluate the upper bound of  $Score(X)$ .

**THEOREM 3.2.** Given a query graph  $Q$  with  $n$  vertices  $u_i$ ,  $X$  is a matching of  $Q$  contained in or overlapped with node area  $S(N)$ . We define  $UpScore(Q, N) = \sum_{i=1}^{i=n} Sim(u_i, ES(N, |Dia(Q)|))$ .  $UpScore(Q, N)$  is the upper bound of  $Score(X)$ .

Given a query  $Q$  in Figure 2 and a node  $N_3$  of G-Tree in Figure 3,  $UpScore(Q, N_3) = Sim(A, ES(N_3, Dia(Q))) + Sim(B, ES(N_3, Dia(Q))) + Sim(C, ES(N_3, Dia(Q))) = 0.1 + 0.6 + 0.3 = 1.0$ .

The pseudo-code for  $RM$  algorithm is given in Algorithm 1. All child nodes  $N_i$  of root are inserted into the max-heap  $H$  in non-ascending order of  $UpScore(Q, N_i)$  (Line 1). Initially, we set  $\beta = -\infty$  (Line 2). If the heap head  $N$  is a leaf node, we remove  $N$  from the heap and perform sub-graph isomorphism to find all matchings of  $Q$  in  $ES(N, Dia(Q))$ . They are inserted into the max-heap  $R$  in non-ascending of their matching scores. We update  $\beta$  to be the  $k$ -th largest matching score in  $R$  (Lines 7-10). If the heap head  $N$  is non-leaf node, we remove  $N$  from the heap and insert all its child nodes  $C_i$  into the heap in non-ascending order of  $UpScore(Q, C_i)$  (Lines 12-13).  $RM$  algorithm terminates when  $\beta$  is not larger than  $UpScore(Q, N)$ , where  $N$  is the heap  $H$ 's head. (Line 5)

Action	Heap contents	$R$
Access root	$\langle N_2, 2.4 \rangle$ $\langle N_1, 2.2 \rangle$	$\emptyset$
Expand $N_2$	$\langle N_6, 2.4 \rangle$ $\langle N_1, 2.2 \rangle$ $\langle N_5, 1.9 \rangle$	$\emptyset$
Check $N_6$	$\langle N_1, 2.2 \rangle$ $\langle N_5, 1.9 \rangle$	$\langle \text{matching 1}, 2.0 \rangle$
Expand $N_4$	$\langle N_4, 2.2 \rangle$ $\langle N_5, 1.9 \rangle$ $\langle N_3, 1.0 \rangle$	$\langle \text{matching 1}, 2.0 \rangle$
Check $N_4$	$\langle N_5, 1.9 \rangle$ $\langle N_3, 1.0 \rangle$	$\langle \text{matching 2}, 2.2 \rangle$ $\langle \text{matching 1}, 2.0 \rangle$ $\langle \text{matching 3}, 0.5 \rangle$

Matching ID	Vertex ID lists
matching 1	$\langle 15, 17, 14 \rangle$
matching 2	$\langle 9, 10, 12 \rangle$
matching 3	$\langle 8, 4, 2 \rangle$

**Figure 6. Heap Content**

In the running example, we first access the root and insert all its child nodes into the max-heap  $H$ , as shown in Figure 6. Since  $UpScore(Q, N_2) = 2.4 > UpScore(Q, N_1) = 2.2$ ,  $N_2$  is before  $N_1$  in the heap.  $N_2$  is heap head, therefore, we “expand”  $N_2$ .  $N_5$  and  $N_6$  are children of  $N_2$ , which are inserted into the heap.  $N_6$  is the next expanded node. Since  $N_6$  is a leaf node, we remove  $N_6$  from the heap and perform sub-graph isomorphism to find all matchings of  $Q$  in  $ES(N_6, Dia(Q))$ . We find *matching 1* in  $ES(N_6, Dia(Q))$  with score 2.0, which is inserted into the result set  $R$ . Now,  $N_1$  is heap node.  $N_1$ 's children, that are  $N_3$  and  $N_4$ , are inserted into the heap. Next, since  $N_4$  is a leaf node, we remove  $N_4$  from the heap and perform sub-graph isomorphism to find all matchings in  $ES(N_4, Dia(Q))$ , that are *matching 2* and *matching 3*. We keep the threshold  $\beta$  to be the  $k$ -th largest matching score by now ( $k=2$  in the running example). Now,  $\beta = 2.0 > UpScore(Q, N_5) = 1.9$ , where  $N_5$  is the heap head. It indicates that all left nodes can be pruned safely. We report top-2 matchings in the heap  $R$  (that are *matching 2* and *1*) and the algorithm terminates.

**THEOREM 3.3. (Algorithm Correctness)**  $RM$  algorithm can report the correct top- $K$  matchings.

---

### Algorithm 1 Ranked Matching ( $RM$ for short)

---

**Require:** Input: query  $Q$ , a large graph  $G$  and G-Tree index  $T$ .

Output: Top- $k$  matchings of  $Q$  in  $G$

- 1: Insert all child nodes  $N_i$  of  $T$ 's root into the max-heap  $H$  according to  $UpScore(Q, ES(N_i, Dia(Q)))$ .
  - 2: Set  $\beta = -\infty$ .
  - 3: **while**  $|H| > 0$  **do**
  - 4:   the head of heap  $H$  is node  $N$ .
  - 5:   **if**  $\beta > UpScore(Q, N)$  **then**
  - 6:     Break
  - 7:   **if**  $N$  is a leaf node **then**
  - 8:     Remove  $N$  from the heap  $H$  and find all matchings of  $Q$  in  $ES(N, Dia(Q))$  by sub-graph isomorphism.
  - 9:     The matchings are inserted into the max-heap  $R$  in descending order of their matching scores.
  - 10:     Update  $\beta$  to be the  $k$ -th largest matching score.
  - 11:   **else**
  - 12:     Remove  $N$  from the heap  $H$
  - 13:     For all child nodes  $C_i$  of  $N$ , they are inserted into max-heap  $H$  in descending order of  $UpScore(Q, C_i)$
  - 14: Report the top- $k$  matchings in  $R$ .
- 

**Proof.**(sketch) It is straightforward to know that, if we assess every leaf node by sub-graph isomorphism, we can find the correct top- $k$  matchings. If the algorithm terminates when  $|H| = 0$  (Line 3), it means that each leaf node has been checked. Therefore, the algorithm can report the correct top- $k$  answers. If the algorithm terminates when  $\beta > UpScore(Q, N)$  (Line 6), it means that, for un-assessed leaf nodes, there exists no matching whose score is larger than  $\beta$  ( $\beta$  is the  $k$ -th largest matching score by far). It indicates that we have found the correct top- $k$  matchings. Therefore we prove the correctness of  $RM$  algorithm.  $\square$

**THEOREM 3.4.** Given a top- $k$  query  $Q$ , for any leaf node  $L$ , the sufficient and necessary condition for performing sub-graph isomorphism between  $Q$  and  $ES(L, Dia(Q))$  is that  $UpScore(Q, L) \geq \beta^*$ , where  $\beta^*$  is the  $k$ -th largest matching score in the final results.

**Proof.**(sketch) 1) *sufficient condition.* If there exists a leaf node  $L$  that has not been checked by sub-graph isomorphism so far, where  $UpScore(Q, L) \geq \beta^*$ , there must exist  $L$  or its ancestor in the heap  $H$  according to  $RM$  algorithm. It is clear to know  $\beta^* \geq \beta$  ( $\beta$  is the  $k$ -th largest matching score by far). Since  $UpScore(Q, L) \geq \beta^*$ ,  $UpScore(Q, L) \geq \beta$ . It means that the algorithm cannot terminate, according to Line 5. Therefore, for any leaf node  $L$ , if  $UpScore(Q, L) \geq \beta^*$ , it must be checked by sub-graph isomorphism before algorithm termination.

2) *necessary condition.* Assume that there exists a leaf node  $L'$ , where  $UpScore(Q, L') < \beta^*$ , and  $L'$  needs to be checked by sub-graph isomorphism before algorithm termination. Since  $L'$  needs to be checked, it means that  $L'$  is the recent heap head and  $UpScore(Q, L') > \beta$ . Since  $UpScore(Q, L') < \beta^*$  and  $L'$  is the recent heap head, it means that we cannot find any matching with score  $\geq \beta^*$  in un-accessed leaf nodes. It also indicates that the final  $k$ -th largest matching score  $< \beta^*$ , which leads to contradiction ( $\beta^*$  is the final  $k$ -th largest matching score). Therefore, there exists no leaf node  $L'$ , where  $UpScore(Q, L') < \beta^*$ , and  $L'$  needs to be checked by sub-graph isomorphism.

In conclusion, we prove the correctness of Theorem 3.4.  $\square$

**Pruning Effect Discussion.** To evaluate the pruning effect of  $RM$  algorithm, we define the *assess ratio* to be the ratio of the

number of assessed leaf nodes to the total number of leaf nodes in G-Tree. Theorem 3.4 shows the underlying reason that *RM* algorithm can obtain good pruning power. In fact, each leaf node corresponds to a cluster in the large data graph. Different clusters (leaf nodes) have different characteristics. For example, in PPI network, these clusters correspond to different functional groups. Therefore, given a query  $Q$ , for most leaf nodes  $L$ ,  $UpScore(Q, L)$  is small, since  $Q$  and  $L$  have no common characteristics. According to Theorem 3.4, these leaf nodes will not be assessed due to small  $UpScore(Q, L)$ . In experiments, we will evaluate our method by two important real Datasets: *S.cerevisiae* and *DBLP* dataset. Extensive results confirm our intuition about pruning effect.

However, when  $Dia(Q)$  is large, we always obtain a large extension node area  $ES(N, Dia(Q))$  during query processing. It means that we will spend more time to perform sub-graph isomorphism. In order to handel the problem, we propose the advanced *RM* algorithm in the following section.

### 3.3 Advanced RM algorithm

The pseudo-code of Advanced *RM* algorithm is shown in Algorithm 3.3. We partition the query  $Q$  into  $m$  parts  $Q_i$  (Line 1), and  $Max(Dia(Q_i)) < \theta$  (we discuss setting  $\theta$  later). For each query part  $Q_i$ , we maintain a max-heap  $R_i$  to store all matchings of  $Q_i$  that have been founded by now. The algorithm is iterated from  $n=1$  (Line 7~12). In each iteration step, for each  $Q_i$ , we call function  $Sub\_RM(Q_i, n, H_i, R_i)$  to return the correct top- $n$  matchings of  $Q_i$  in the max-heap  $R_i$  (Line 9 ~ 10)( $Sub\_RM$  function is analogous to *RM* algorithm). Then, for the first  $n$  matchings of  $Q_i$  in each  $R_i$ , we check whether there exist some matchings that can be assembled into a matching of  $Q$ . These assembled matchings of  $Q$  are inserted into a max-heap  $R$  (Line 11). We set  $\beta$  to be the  $k$ -th largest matching score of  $Q$  in  $R$  by now, and set  $\beta_i$  to be the  $n$ -th largest matching score of  $Q_i$  in  $R_i$  by now (Line 12). If  $\beta > \sum_{i=1}^m \beta_i$ , the iteration terminates (Line 7). The algorithm report the top- $k$  matchings in  $R$  (Line 13). The algorithm correctness can derived from the correctness of *RM* and *TA* algorithm [7]. Due to space limited, we omit the proof. Notice that re-calling function  $Sub\_RM$  at the  $n$ th iteration only needs to compute the  $n$ th largest matching of  $Q_i$ , and it does not re-compute the first  $(n-1)$  largest matchings of  $Q_i$ .

---

#### Algorithm 2 Advanced Ranked Matching (*ARM* for short)

---

**Require:** Input: query  $Q$ , a large graph  $G$  and G-Tree index  $T$ .  
Output: Top- $k$  matching of  $Q$  in  $G$ .

- 1: Partition  $Q$  into  $m$  parts  $Q_1 \dots Q_m$ , and each  $Dia(Q_i) < \theta$ .
- 2: set  $n=0$
- 3: Set  $\beta$  and all  $\beta_i$  to be  $-\infty$ .
- 4: Set max-heaps  $R$  and  $R_i$  to be  $\phi$ .
- 5: **for** each  $Q_i$  **do**
- 6:   Insert all child nodes  $N_i$  of  $T$ 's root into the max-heap  $H_i$  according to  $UpScore(Q_i, ES(N_i, Dia(Q_i)))$ .
- 7: **while**  $\beta < \sum_{i=1}^m \beta_i$  **do**
- 8:    $n=n+1$
- 9:   **for** each  $Q_i$  **do**
- 10:     Call  $Sub\_RM(Q_i, n, H_i, R_i)$
- 11:   For the first  $n$  matchings in each  $R_i$ , check whether there exist some matchings that can be assembled into a matching of  $Q$ . These matchings of  $Q$  are inserted into a max-heap  $R$ .
- 12:   set  $\beta$  to be the  $k$ -th largest matching score of  $Q$  in  $R$  by now, set  $\beta_i$  to be the  $n$ -th largest matching score of  $Q_i$  in  $R_i$  by now.
- 13: Report top- $k$  matchings of  $Q$  in the max-heap  $R$ .

---

**Discussion about  $\theta$ .** Sub-graph isomorphism in Line 7 of function

$Sub\_RM$  in Algorithm 3.3 dominates the algorithm performance. It is straightforward to know  $|ES(N, Dia(Q_i))| \approx O(degree^{|Dia(Q_i)|})$ , where  $N$  is an arbitrary leaf node and  $degree$  is the average vertex degree in the data graph. In the worst case, the time complexity of sub-graph isomorphism algorithm in Line 7 in function  $Sub\_RM$  is  $O(|V(Q_i)|^2 * |ES(N, Dia(Q_i))|)$  [5]. Therefore, according to Line 7 of function  $Sub\_RM$ , we should make  $\theta = Max(|Dia(Q_i)|)$  as small as possible. However, on the other hand, less  $\theta$  means more query parts  $Q_i$ , which leads to more cost in Line 11 of Algorithm 3.3. It is difficult to quantify the trade-off about  $\theta$  from theoretical analysis. Nonetheless, we still have some guidelines about  $\theta$ . Since, for any state art of sub-graph isomorphism algorithm, it cannot work well when  $|ES(N, Dia(Q_i))| > 1000$  [5]. Our experiment study also confirms that. Therefore, we maximize  $\theta$  when  $Max(|ES(N, Dia(Q_i))|) < 1000$ .

**Assembling matchings of  $Q_i$ .** Assume that a query  $Q$  is partitioned into  $m$  parts  $Q_i$ ,  $i=1 \dots m$ . The edges connecting these  $Q_i$  are called ‘‘cut’’ edges. In Line 11 of Advanced *RM* algorithm, if 1) for each max-heap  $R_i$ , there is a matching  $X_i$  of  $Q_i$ , and 2) for each cut edge in  $Q$ , there exists an edge connecting corresponding  $X_i$  in the data graph  $G$ , and 3)  $X_i \cap X_j = \phi$ ,  $i \neq j$ , then these matching  $X_i$  can be assembled into a matching  $X$  of  $Q$ .

---

#### $Sub\_RM(Q_i, n, H_i, R_i)$

---

- 1: **while**  $|H_i| > 0$  **do**
- 2:   the head of heap  $H_i$  is node  $N_i$ .
- 3:   set  $\beta_i$  to be the  $n$ -th largest matching score of  $Q_i$  in  $R_i$  by now.
- 4:   **if**  $\beta_i > UpScore(Q_i, N_i)$  **then**
- 5:     Break
- 6:   **if**  $N_i$  is a leaf node **then**
- 7:     Remove  $N_i$  from the heap  $H_i$  and find all matchings of  $Q_i$  in  $ES(N_i, Dia(Q_i))$  by sub-graph isomorphism.
- 8:     The matchings are inserted into the max-heap  $R_i$  in descending order of their matching scores.
- 9:   **else**
- 10:     Remove  $N_i$  from the heap  $H_i$
- 11:     For all child nodes  $C_i$  of  $N_i$ , they are inserted into max-heap  $H$  in descending order of  $UpScore(C_i)$

---

## 4. Experiments

In this section, we report our experiment results to evaluate our methods in two real data sets. All experiments are implemented by standard C++ with STL library support and compiled by Visual Studio 2003 in a P4 1.7GHz machine of 1G RAM running Windows XP. To our best knowledge, there is no existing algorithm on top- $k$  matching problem proposed in this paper. The most related and recent work is *GraphMatch* [24]. In fact, there are some differences between each other. First, *GraphMatch* relaxes sub-graph isomorphism matching by only considering pair-wise vertex distance. Second, *GraphMatch* allows existing un-matching vertices in query  $Q$  and indel vertices in the matching of  $Q$  in the data graph. In experiments, to enable performance comparison, we set parameter  $indel=0$  (not allowing indel vertices) and set a big penalty value for each un-matching vertex in  $Q$  (not allowing un-matching vertices in  $Q$  in top- $k$  matchings). We download *GraphMatch* from <http://faculty.cs.tamu.edu/shsze/graphmatch>.

Furthermore, we design a *TA* variation algorithm for top- $k$  matching query problem, called *TA-matching*. Given a query  $Q$ , for each vertex  $u$  in  $Q$ , we rank the vertices  $v_j$  in data graph  $G$  in non-

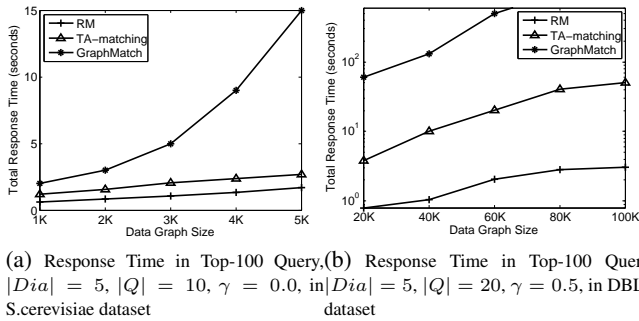
ascending order of  $Sim(u_i, v_j)$  to form the list  $L_i$ . Similar with TA-algorithm [7], we scan all lists  $L_i$  in parallel. For each encountered vertex  $v_j$ , we perform sub-graph isomorphism between  $Q$  and the local area around  $v_j$  in the data graph  $G$ . The algorithm continues until all left vertices cannot form a top-k matching (analogy to TA algorithm). TA-matching has a significant limitation: estimation about the upper bound of the left matching scores is not tight.

**Datasets** We use two real datasets, that are DBLP and protein interaction network of *S.cerevisiae*.

1) *DBLP Dataset*: We construct a co-author network  $G$ : every author is denoted as a vertex in  $G$ ; and the edge is introduced when the corresponding two authors have at least one co-authored paper. We consider about 100 important conferences in different areas to construct  $G$ . On the whole, there are about 100K vertices and about 400K edges in  $G$ . We describe each vertex (a author) by a paper title string  $title$ , combining the titles of all papers that he or she published. To generate a query  $Q$ , we randomly extract a sub-graph from  $G$  with the pre-defined vertex number. To evaluate the vertex similarity  $Sim(m, n)$  in Definition 1.5, we define it by the similarity between paper title strings of two corresponding authors. To be specific, for two vertices  $n$  and  $m$  in query  $Q$  and the co-author network  $G$  respectively, we define  $Sim(n, m) = \frac{|n.title \cap m.title|}{|n.title|}$ , where  $|n.title \cap m.title|$  is the number of common words in  $n.title$  and  $m.title$ . Obviously,  $0 \leq Sim(n, m) \leq 1$ .

2) *S.cerevisiae Dataset*: We download it from DIP<sup>3</sup> and represent it by an undirected graph  $G$  in which each vertex represents a protein and each edge represents interactions between two proteins. There are about 4934 vertices and 17346 edges in  $G$ . Similar with [24], for two vertices  $n$  and  $m$  in query  $Q$  and the data graph  $G$ , we define  $Sim(n, m)$  to be as the minus-log E-value between two corresponding protein sequence. To generate a synthetic query, we randomly extract a sub-graph from  $G$ . Then, each vertex in the sub-graph is perturbed by pre-specified amount of point mutations in its corresponding protein sequences.

All experiment results in this section are the average results of 100 queries.



**Figure 7. Scalability Test**

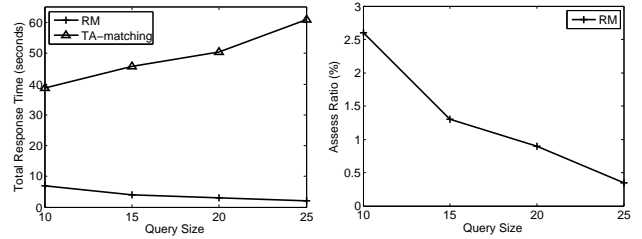
**Scalability Study** We test scalability of our methods in Figure 7 on both datasets, and compare them with *GraphMatch*. It is difficult for *GraphMatch* to work when the data graph has more than 60K vertices. Our methods have good scalability.

**Performance Study** Since there are only about 5,000 vertices in

<sup>3</sup><http://dip.doe-mbi.ucla.edu>

*S.cerevisiae* network, in order to test the performance of *RM* algorithm in a large graph, we only report the experiment results in DBLP dataset as follows. The default vertex cardinality of the data graph  $G$  is 100K in the following experiments. We only evaluate *RM* algorithm and TA-matching algorithm proposed in the paper, since it is difficult for *GraphMatch* to work on the large graph with 100K vertices.

Iteratively, we build the G-Tree by running graph partition algorithm METIS [12] on the data graph  $G$ . We set the fanout of G-tree to be 50. First, we fix the diameter of query  $Q$  to be 5 and threshold  $\gamma=0.5$  ( $\gamma$  is defined in Definition 1.5), and vary the size of  $Q$  (that is number of vertices in  $Q$ ) from 10 to 25. Total response time about top-100 queries is shown in Figure 8(a). We also evaluate the pruning effects of G-Tree in Figure 8(b) by assess ratio in *RM* algorithm. Assess ratio is defined as the ratio of the number of assessed leaf nodes in *RM* algorithm to the total number of leaf nodes in G-Tree. *RM* algorithm only performs expensive sub-graph isomorphism when it assesses the leaf nodes. Therefore, less assess ratio indicates faster response time. *RM* algorithm outperforms TA-matching algorithm in Figure 8(a). Observed from Figure 8(a), we find that the query response time in *RM* algorithm decreases with increasing query size. The underlying reason is that, when the query graph is large, G-tree provides more efficient pruning effect, as shown in Figure 8(b). Notice that, the trend of response time for *RM* algorithm in Figure 8 happens when we fix  $Dia(Q)$ . If the query diameter increases with the query size, the response time will increase. We evaluate *RM* performance with regard to the query diameter as follows.



**Figure 8. Varying Query Size**

We fix the query size to be 20 and  $\gamma=0.5$  and vary the diameter of query from 2 to 10. Figure 9(a) reports the response time and pruning effect of G-tree. The clear trend is that the response time increases with increasing query diameter. When *RM* algorithm assesses a leaf node  $N$ , it is clear to know that we obtain a large extend node area  $ES(N, Dia(Q))$  when  $Dia(Q)$  is large. It means that we will spend more time to perform sub-graph isomorphism between  $Q$  and  $ES(N, Dia(Q))$ . There is the similar problem in TA-matching algorithm.

To address large query diameter problem, we propose the Advanced *RM* algorithm in Section 3.3. We evaluate the Advanced *RM* algorithm in Figure 10. We fix the query size to be 20 and vary the query diameter from 6 to 14. Results are reported in Figure 10. Though, assess ratio of Advanced *RM* is larger than that in *RM*, Advanced *RM* algorithm improves the query performance (response time). The underlying reason is that we need less time to perform sub-graph isomorphism in Advanced *RM* algorithm, since  $|ES(N, Dia(Q_i))|$  is small. We partition the query  $Q$  into different parts and  $Max(Dia(Q_i)) < \theta = 5$ .

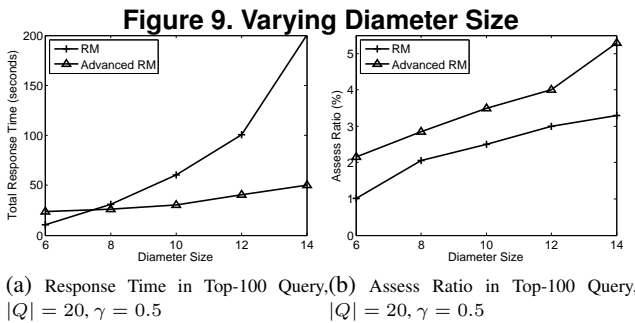
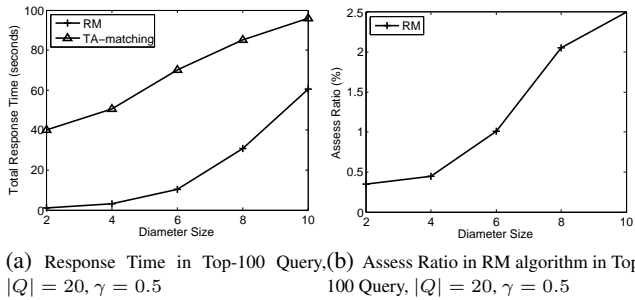


Figure 10. Evaluate Advanced RM algorithm

At last, we test RM algorithm with decreasing  $\gamma$ . We fix the query size to be 20 and  $|Dia(Q)|=5$ . When  $\gamma=0.5$ , for each vertex in query  $Q$ , there are about 1,000 candidate matching vertices in data graph  $G$ . When  $\gamma=0.1$ , for each vertex in query  $Q$ , there are more than 10,000 candidate matching vertices in data graph  $G$ . It is clear to know, the search space of sub-graph isomorphism when  $\gamma=0.1$  is larger than that in  $\gamma = 0.5$ . Therefore, Figure 11(a) shows that total response time increases with decreasing  $\gamma$ . The assess ratio of RM only depends on the final top-k answers. The sufficient and necessary condition that a leaf node needs to be performed sub-graph isomorphism is discussed in Theorem 3.4. Usually, when k is small, decreasing  $\gamma$  does not affect the final top-k answers. Therefore, assess ratio keeps invariable.

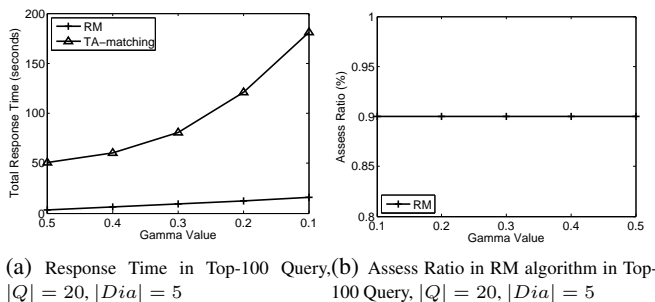


Figure 11. Vary the Threshold  $\gamma$

## 5. Conclusions

In this paper, we propose a novel sub-graph matching query problem in a very large data graph. It finds many applications in biological network and social network. To address the prob-

lem, we design a balanced tree  $G$ -Tree to index the large graph. Adopting *score-upper bound* pruning strategy, based on  $G$ -Tree, we propose the  $RM$  algorithm to locate the top-k matchings of query  $Q$  in the large data graph. Extensive experiments shows that our algorithm has good scalability and fast response time, which outperforms the alternative method by orders of magnitude.

## 6. References

- [1] A. Z. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. L. Wiener. Graph structure in the web. *Computer Networks*, 33(1-6), 2000.
- [2] D. Cai, Z. Shao, X. He, X. Yan, and J. Han. Community mining from multi-relational networks. In *PKDD*, 2005.
- [3] J. Cheng, Y. Ke, W. Ng, and A. Lu. *fg-index*: Towards verification-free query processing on graph databases. In *SIGMOD*, 2007.
- [4] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *IJPRAI*, 18(3), 2004.
- [5] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26, 2004.
- [6] J. H. D.W. Williams and W. Wang. Graph database indexing using structured graph decomposition. In *ICDE*, 2007.
- [7] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, pages 102–113, 2001.
- [8] S. Fortin. The graph isomorphism problem. *Department of Computing Science, University of Alberta*, 1996.
- [9] P. Y. H. Jiang, H. Wang and S. Zhou. Gstring: A novel approach for efficient search in graph databases. In *ICDE*, 2007.
- [10] T. R. Hagadone. Molecular substructure similarity searching: Efficient retrieval in two-dimensional structure databases. *J.Chem. Inf. Comput. Sci*, 32, 1992.
- [11] H. He and A. K. Singh. Closure-tree: An index structure for graph queries. In *ICDE*, 2006.
- [12] G. Karypis and V. Kumar. A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 1995.
- [13] M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph. *Data Mining and Knowledge Discovery*, 2005.
- [14] V. Lacroix, C. G. Fernandes, and M.-F. Sagot. Motif search in graphs: Application to metabolic networks. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 3(4), 2006.
- [15] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *NIPS*, 2001.
- [16] E. Ravasz and A.-L. Barabasi. Hierarchical organization in complex networks. *Physical Review E*, 67, 2003.
- [17] J. Scott, T. Ideker, R. M. Karp, and R. Sharan. Efficient algorithms for detecting signaling pathways in protein interaction networks. In *RECOMB*, 2005.
- [18] A. SF, G. W, M. W, M. EW, and L. DJ. Basic local alignment search tool. *J Mol Biol.*, 215(3), 1990.
- [19] D. Shasha, J. T.-L. Wang, and R. Giugno. Algorithmics and applications of tree and graph searching. In *PODS*, 2002.
- [20] S. TriBl and U. Leser. Fast and practical indexing and querying of very large graphs. In *SIGMOD Conference*, pages 845–856, 2007.
- [21] D. J. Watts. *Six Degrees: The Science of a Connected Age*. W. W. Norton & Company, 2004.
- [22] P. Willett. Chemical similarity searching. *J. Chem. Inf. Comput. Sci*, 38(6), 1998.
- [23] X. Yan, P. S. Yu, and J. Han. Graph indexing: A frequent structure-based approach. In *SIGMOD*, 2004.
- [24] Q. Yang and S. hoi Sze. Path matching and graph matching in biological networks. *J Comput Biol.*, 14(1), 2007.
- [25] S. Zhang, M. Hu, and J. Yang. Treepi: A novel graph indexing method. In *ICDE*, 2007.