Parallel top-k subgraph query in massive graphs: Computing from the perspective of single vertex

6
Author(s)

Jianliang Gao ; Bo Song ; Ping Liu ; Weimao Ke ; Jianxin Wang ; Xiaohua Hu

Abstract:
In the real world, many problems on massive graphs can be mapped to an underlying critical problem of discovering top-k subgrap graphs, subgraph queries may have enormous number of matches, and so it is inefficient to compute all matches when only top-k desired. Meanwhile, parallel algorithm is urgent for the scalability of massive graph computing. In this paper, we address the challe subgraph query in massive graph. Firstly, we present a new graph matching notion: “approximate graph simulation”. With approxir simulation, top-k subgraph query can be customized by appointing a weighted query graph, which provides good flexibility for diffe scenarios. Secondly, we propose a parallel top-k subgraph query algorithm at the level of vertex. With such algorithm, each vertex obtains its matching state separately without requiring global graph information. In the algorithm, we also design a filter mechanisr the computation and a aggregation mechanism to obtain top-k vertices for query focus. Using real-life datasets, we experimentally approach of parallel top-k subgraph query are efficient.

Published in: Big Data (Big Data), 2016 IEEE International Conference on

Date of Conference: 5-8 Dec. 2016

INSPEC Accession Number: 16652832

Date Added to IEEE *Xplore*: 06 February 2017

DOI: 10.1109/BigData.2016.7840656

ISBN Information:

Publisher: IEEE

Conference Location: Washington, DC, USA

SECTION 1.
Introduction

Research of massive graph has attracted renewed interest in recent years due to increased interests in a number of network applications such as biological networks, social networks, communication networks [14]. Network data are often represented as graphs, where nodes are labeled entities and edges represent relations among these entities [1] An important processing for graph data application is that of sub graph query.

Subgraph query is to find out the subgraphs of data graph G which match a given query graph Q. It is essential for a wide range of emerging applications such as community discovery and neighbor query in social networks, biological data analysis [13], classification of web documents [16], software plagiarism [19] and so on. A number of matching algorithms have been developed to compute the set of matches of Q in G. In real world applications, query graph is usually small, but data graph is typically large, even including billions of nodes [3] The large scale graphs such as social graphs and protein-protein interactive networks (PPI) give rise to the following problems with the subgraph query algorithms.

1. The large size of data graphs makes matching costly. For matching defined by simulation, it takes $O(|G||Q| + |G|^2)$ time to compute graph match set [4]. It can be seen that the size of data graphs could greatly affect the computational complexity. Furthermore, the matching algorithms often return an excessive number of sub graphs. It could be a difficult task for the users to inspect such a large number of matching subgraphs and select the most desirable solution.

2. The top-k subgraph query is needed to filter unwanted matching subgraphs. Exiting top-k subgraph query approaches include two phases: 1) computing all matching subgraphs which satisfy the query in a sequential way; 2) ranking such results based on given metrics. However, previous works on the matching problem could face difficulty due to the lack of scalability for the increasing scale of graphs.

3. Subgraph query should take the weighted graph into consideration. With the ever-increasing popularity of entity-centric applications, it becomes very important to study the interactions between entities, which are captured using edges in entity-relationship networks [8]. Entity-relationships usually have different importance in various applications. For example, the entity-relationship in social network might be different according to various personal influence. In protein-protein interaction networks, the probability value to present the interaction existing in practice is an important parameter. The values between entities of a network can be mapped to the weights on edges of a graph. Therefore, subgraph query according to weighted graphs becomes important in massive graph computing.

Some researches aim to solve the above problems in subgraph query. For example, diversified top-k subgraph query is proposed recently [5] [6]. To simplify the subgraph query, “query focus” is introduced as an substitution of finding the entire set of matches [6]. By appointing a focus in query graph, sub graph query returns only those vertices which match the query focus, rather than the entire matching subgraph set. Query focus based subgraph query is widely needed in e.g. expert recommendation [20] and egocentric search [21]. In fact, 15% of social queries are to find matches of specific nodes [22]. In this paper, we also take the “query focus” as the subgraph query.

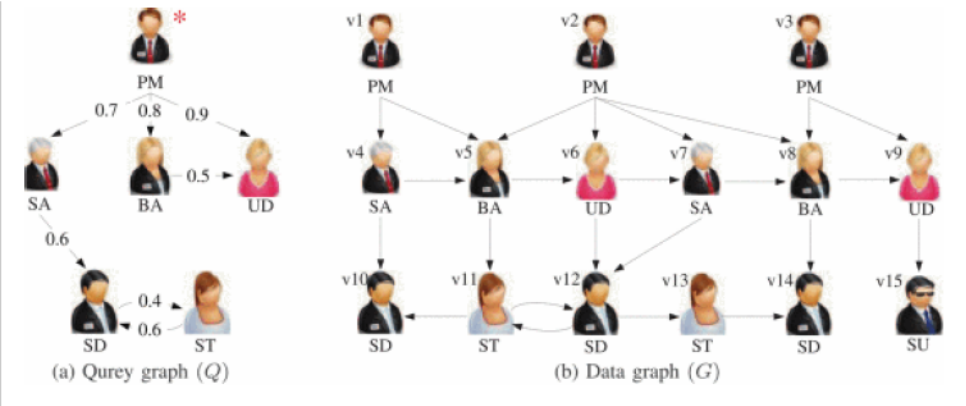


Figure 1:
Querying collaboration network

1. Example 1

Fig. 1 illustrates an example of subgraph query with query focus. A fraction of a collaboration network could be represented by a data graph G in Fig. 1b, where each vertex represents a persona, with a label for the job title of the persona such as project manager (PM), software architect (SA), business analyst (BA), user interface developer (UD), software developer (SD), software tester (ST) and software user (SU). These vertices are assigned IDs from $v1$ to $v15$. Each edge indicates a supervision relationship, e.g. edge $(v1, v4)$ indicates that $v1$ can supervise $v4$. A company may issue a requirement of finding some PMs who can supervise SA, BA and UD directly. Moreover, (1) the BA can supervise the UD; (2) the SA can supervise a SD; (3) there is a ST who works under the SD, and vice versa. Such requirements could be expressed as a graph Q shown in Fig. 1a. Here PM is the "focus" of query which is denoted with '*', i.e., only the top-k matches of PM are required to return. When graph matching is defined in terms of subgraph isomorphism [5] or subgraph simulation [6], only one subgraph with $v2$ (i.e., $v2, v5, v6, v7, v11.v12$) can be identified in G. However if two matches Of Q are required, there are no enough matches in this example in terms of graph isomorphism or graph simulation. It is too restrictive in many applications. We relax the limitation of graph simulation to permit lack of vertex or edge, named *approximate graph simulation*. Given query graph with edge weights, approximate graph simulation identifies the top- k matching subgraphs in G. For example, when $k = 2$, two top-ranked *PMS* ($v1$ and $v2$) are returned that match the query focus *PM* in Q. □

For top-k sub graph query, there are two key problems: (1) how to quantify the scores of matching subgraphs; (2) how to design parallel algorithm for massive graph. We present approximate graph simulation in this paper. Different from traditional graph isomorphism or graph simulation, approximate graph simulation processes weighted query graph. Edges of query graphs have weights which denote various meanings in real applications. For example, in team selection applications, a larger weight implies higher importance [8]. In biological networks, the weight could be defined as the probability of the interaction between entities [13]. As the weights are related to various applications, we assume in this paper that weights have been given and we will not discuss how to weight the edges. Meanwhile, the proposed approach can be applied to unweighted sub graph query when all weights are set as the same value.

In this paper, we study the top-k weighted subgraph query problem and propose a parallel algorithm for it. Our contributions are summarized as follows.

- We revise the traditional notion of graph matching by relaxing the requirements for vertices

- We revise the traditional notion of graph matching by relaxing the requirements for vertices and edges, which is named approximate graph simulation in this paper. By appointing weighted query graphs, top-k subgraph query can be customized, which increases the flexibility and usability of subgraph matching. We present the top-k subgraph query problem in terms of approximate graph simulation and then propose a method to rank the matching subgraphs.

- We propose a parallel algorithm for top-k subgraph query. The algorithm is designed at the level of single vertex and all vertices obtain their matching state separately without requiring global graph information. Therefore, it can be easily deployed in distributed systems.

- To speed up the process, we propose a filter mechanism to reduce unnecessary computing. Furthermore, we design a mechanism to implement the aggregation for top-k values in distributed systems. These mechanisms aid to obtain top-k subgraphs in massive data graphs.

- Experiments are conducted on a distributed Hadoop platform and the results from three real-world datasets show the efficiency of our approach.

The rest of the paper is organized as follows. We introduce the problem definitions in Section 2. We then present parallel top-k subgraph query algorithm in Section 3. Section 4 presents the experimental study. Related work is shown in Section 5. Finally, Section 6 concludes this paper.

SECTION 2.
Problem Definition

In this section, we present some preliminary definitions and then formulate the problem of top-k subgraph query. Subgraph isomorphism [5] is a classical graph matching which finds subgraphs from data graph which are isomorphic to the query graph. Graph simulation [6] relaxes the limitations by only requiring the matching of successive vertices. However, lack of vertices or edges are not permitted in graph isomorphism and graph simulation, which is too restrictive in many real applications. In this paper, we propose a new notion of graph matching named approximate graph simulation. The symbols are summarized in Table 1.

2.
Data Graph

A data graph is a directed graph $G = (V, E, f)$, where (1) V is a finite set of vertices; (2) $E \subseteq V \times V$, in which $(v, v\prime)$ denotes an edge from vertex v to $v\prime$ and (3) $f(\cdot)$ is a function on V such that for each vertex v in $V, f(v) = a$ where a is a constant of the attribute. In this paper, the attribute of a node carries the label of the vertex.

2.
Query Graph

A query graph is a directed graph $Q = (V_q, E_q, f_q, w, u_0)$, where (1) V_q and E_q are the sets of vertices and edges respectively; (2) f_q is a function defined on V_q for each vertex $u \in V_q$. $f_q(u)$ returns the

label of u in this paper. (3) w is a function defined on each edge $e \in E_q$, and $w(e)$ denotes the weight on edge e. (4) u_0 is the query fOCUS $u_0 \in V_q$ and for any $u \in V_q$, $u \neq u_0$, there is at least one path from u_0 to u.

The traditional graph simulation requires that all vertices and all edges of query graph G must exist in the matching results. It is often too restrictive in many applications. Furthermore, weighted query graph has not been considered. In this paper, we present a kind of new graph matching with weighted query graph: approximate graph simulation.

2. Apporoximate Matching Subgraph

Given a query graph $Q = (V_q, E_q, f_q, w, u_0)$ and a data graph $G = (V, E, f)$, a graph $G_s = (V_s, E_s, f)$ is an approximate matching subgraph, such that (1) $V_s \subset V$ and $E_s \subset E$; (2) for each v, $v \in V_s$, there exists a vertex u, $u \in V_q$ such that $f_q(u) = f(v)$, referred to as vertex matching between u and v ; (3) for each edge $e = (v, v\prime)$, $e \in E_s$, there exists $e_q = (u, u\prime)$, $e_q \in E_q$, such that $[f_q(u)] = f(v)$ and $f_q(u\prime) = f(v\prime)$, referred to as edge matching between e and e_q, denoted as $\Lambda(e) = e_q$; (4) there exists v_0 in V_s such that $f(v_0) = f(u_0)$; for any v in V_s, there exists at least one path from v_0 to v

2. Approximate Graph Simulation

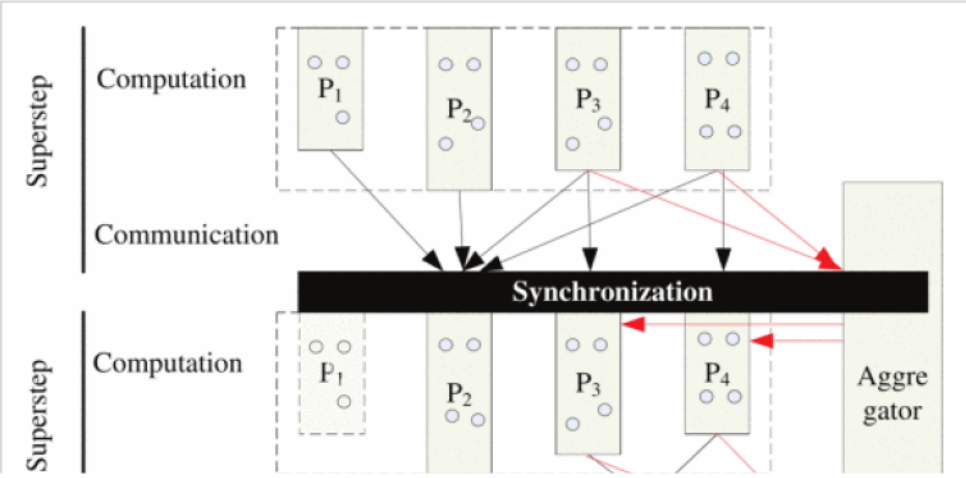
Given a query graph Q and a data graph G, approximate graph simulation aims to find out all *approximate matching subgraphs* G_s in data graph G.

In a massive data graph, the number of approximate matching sub graphs is also very large. For each approximate matching subgraph, matching degree can be quantified according to the matched edges. Given a query graph $Q = (V_q, E_q, f_q, w, u_0)$ and a matching subgraph $G_s = (V_s, E_s, f)$, the matching degree D can be calculated as:

$$D(G_s, Q) = \sum_{e \in E_q'} w(e), \tag{1}$$

View Source

where $E_q\prime$ is the set of edge matching for each edge e in E_s, i.e. $E_q\prime = \{\Lambda(e) | e \in E_s\}$.



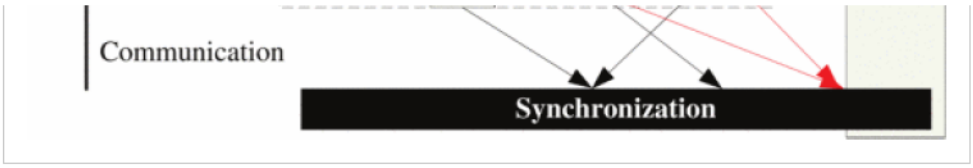


Figure 2:
The parallel computing model from the perspective of single vertex.

2. Top-k Subgraph Query

Given a query graph $Q = (V_q, E_q, f_q, w, u_0)$ and a data graph $G = (V, E, f)$, top-k subgraph query is to find k approximate matching subgraphs which have the top-k highest matching degrees. The set of top-k approximate matching subgraphs is denoted as $M(G, Q, k)$ and formulated as:

$$M(G,Q,k) = \{G_s | G_s \in \arg\max \sum_{i=1}^{k} D(G_s(i), Q)\}, \quad (2)$$

View Source

where G_s is an approximate matching sub graph and $G_s(i) \neq G_s(j)$ if $i \neq j$.

For query focus based approximate graph simulation, the top-k subgraph query can be simplified as finding the corresponding vertices of query focus in top- k approximate matching subgraphs. Matching subgraphs can then be extended from the top-k corresponding vertices of query focus. Therefore, this paper discusses the problem of finding top- k matching query focuses in the following.

SECTION 3. Parallel Top-k Subgraph Query Algorithm

The proposed approach consists of a sequence of iterations called supersteps. In each superstep, vertices take local computation and supersteps finish with synchronization. As shown in Fig. 2, vertices are assigned to four processors (3 vertices in P1, 4 in P2, 4 in P3 and 4 in P4 as an example). These processors run in parallel model. Each vertex takes local computation in the assigned processor. In the same processor, the computation can be taken in parallel or serial way according to the computation resource. The data exchange is implemented by messages scheme. There are two kinds of messages in this approach. One is messages between vertices, and the other is from vertex to aggregator. The aggregator collects global top-k matching focuses in a distributed environment. Between two consequent superstep-s, there is a synchronization to exchange information among vertices. A vertex quits the execution if it is set as inactive. For example in Fig. 2, the processing P_1 is terminated in the second superstep as the vertices become inactive. To implement top-k subgraph query, the key points are design parallel algorithm for vertex, which includes how to deal with the messages.

Table 1: Symbol definition

Name	Meaning
$Q = (V_q, E_q, f_q, w, u_0)$	Query graph
$G = (V, E, f)$	Data graph
$M(G, Q, k)$	the set of top-k matching subgraphs
u_0	Query focus
u	Vertex of Q, $u \in V_q$
v	Vertex of G, $v \in V$
$N(v)$	$v' \in N(v)$ iff $(v, v') \in E$
Γ	mapping function from e in G to e' in Q
Θ	function of node influence
Φ	function of matched score
Ψ	function of lost score
$state$	state of current vertex v, three state values: "T": $v \in M(G, Q, k)$ and $f(v) = f_q(u_0)$ "F": $v \notin M(G, Q, k)$ "H": to be decided
pList	List of parent nodes (direct predecessors)
cList	List of child nodes (direct successors)
mList	List of matched edges
lList	List of lost edges

3.1. Quantifying the Matching Degree

To quantify the matching degree of vertices in data graph, we first show the following definitions.

Reachable Edge

For vertex $vinV$, edge $(v_i, v_j) \in E$ is said to be a reachable edge of v in graph $G(V, E)$ if there exists at least one path from v to v_i.

Node Influence

Given a query graph $Q = (V_q, E_q, f_q, w, u_0)$, the node influence of a vertex $u(u \in V_q)$ is $Theta(u)$

$$\Theta(u) = \sum_{e \in E_q''} w(e) \tag{3}$$

View Source

where $E''(E'' \subset E_q)$ is the set of reachable edges of u.

We then define "matched score" and "lost score" for vertices of data graph. In large-scale applications, data graph is large and it is infeasible to compute and save all reachable edges. Therefore, it is a iteration procedure to obtain the final "matched score" and "lost score".

Matched Score

Given a query graph $Q = (V_q, E_q, f_q, w, u_0)$ and a data graph $G = (V, E, d)$, matched score of vertex $v \in V$ is defined as the sum of edges weights by mapping the reachable edges of v in a matching subgraph to Q. In a distributed model, the reachable edges need iterative computation as following:

$$\Phi(v) = \Phi_{init}(v) + \sum_{v' \in N(v)} \Phi(v') - \sum_{e \in R(v)} w(e), \tag{4}$$

View Source

where $\Phi_{init}(v)$ is the initial matched score of vertex v, $N(v)$ is the set of direct successors of v, $i.e.$

where $\Phi_{init(v)}$ is the initial matched score of vertex v; $N(v)$ is the set of direct successors of v, i.e., $N(v) = \{v\prime\}$ such that $(v, v\prime) \in E$; $\Phi_{init(v)} = \sum_{e \in E_o} w(e)$ here E_o, is the set of edges by mapping the outing edges of v to Q; $R(v)$ is the set of the edges in Q which are repeated in $\Phi_{init}(v) + \sum_{v\prime \in V(v)\Phi(v\prime)}$. Note that the maximum $\Phi(v)$ will be selected if there are more than one direct successor whose labels are the same.

Lost Score

Given query graph $Q = (V_q, E_q, f_q, w, u_0)$ and data graph $G = (V, E, f)$, for vertex $v \in V$, its lost score is defined as:

$$\Psi(v) = \sum_{u' \in (N_q(u) - N_q'(u))} w((u, u')) + \sum_{u'' \in N_q'(u)} \Psi(u'') - \sum_{e \in R(u)} w(e), \tag{5}$$

View Source

where $u \in V_q$ is the mapping vertex of v, i.e., $f(v) = f_q(u)$; $N_q(u)$ is the set of child vertices of u; $N_q\prime(u)$ is the set of vertices in Q by mapping the vertices in $N(v)$; $R(v)$ is the repeated edges set in Q which are repeated in $\sum_{u\prime \in (N_q(u) - N_q\prime(u)) w((u,u\prime))} + \sum_{u\prime\prime} \in N_q\prime(u)\psi(u\prime\prime)$.

For example, in Fig. 1, $\Phi_{init(v)} = w(\Lambda(v1, v4)) + w(\Lambda(v1, v5)) = 1.5\psi_{init(v)} = w((PM, UD)) = 0.9$. The change of matched value and lost value in iteration supersteps can be seen in Example 2.

In an iteration computing procedure, a vertex of data graph can get increasing knowledge about its matched edges and lost edges. Therefore, matched score and lost score are monotonically nondecreasing.

Lemma 1

Given a query graph $Q = (V_q, E_q, f_q, w, u_0)$ and a data graph $G = (V, E, f)$, it can be drawn that,

$$\Phi(v) + \Psi(v) \leq \Theta(u), \tag{6}$$

View Source

where $v \in V$, and $f(v) = f_q(u), (u \in V_q)$.

3.2. Filter Mechanism

In the iteration processing, it is not necessary to compute the final scores for each vertex. We draw some filter rules that can terminate the computing earlier. Firstly, we can filter the vertices whose labels are not in the set of labels of query graph vertices.

Filter Rule 1

For $\forall v \in V$ if $f(v) \notin L(Q)$ then $v \notin M(G, Q, k)$, where $L(Q)$ is the label set of query graph Q.

Secondly, as Lemma 1 shows a upper boundary for lost score, a filter rule can be found for the lost score. To draw the filter rule, we derive the following theorem.

Theorem 1

Given a query graph $Q = (V_q, E_q, f_q, w, u_0)$ a data graph $G = (V, E, f)$ and k, for $v \in V$, if $\psi(v) > \Theta(u_0) - min_{u \in V_k}(\Phi(u))$ then $v \notin M(G, Q, k)$, where V_k is the set of v_k such that

$f(v_k) = f_q(u_0)$ and $v_k \in M(G, Q, k)$.

Proof

Assuming $vinM(G, Q, k)$ there exists a path from a vertex $v_k (v_k \in V_k)$ to v according to the definition of approximate matching subgraph. Therefore

$$\Phi(v_k) \geq \Phi(v). \tag{7}$$

View Source

For $f(v_k) = f_q(u_0)$, then according to Lemma 1, there is

$$\Phi(v_k) + \Psi(v_k) \leq \Theta(u_0). \tag{8}$$

View Source

Combining the inequalities (7) and (8), it can be drawn:

$$\Psi(v) + \Phi(v_k) \leq \Theta(u_0). \tag{9}$$

View Source

Furthermore, $v_k \in V_k$, so $\Phi(v_k) \geq min_{u \in V_k}(\Phi(u))$. Put it into inequality (9), it becomes

$$\Psi(v) + \min_{u \in V_k}(\Phi(u)) \leq \Theta(u_0). \tag{10}$$

View Source

Obviously, inequality (10) contradicts the given condition $\psi(v) > \Theta(u_0) - min_{u \in V_k}(\Phi(u))$. Therefore $v \notin M(G, Q, k)$, □

From this theorem, we can obtain the second filter rule:

Filter Rule 2

For $\forall v \in V$, if if $\psi(v) > \Theta(u_0) - min_{u \in V_k}(\Phi(u))$, then $v \notin M(G, Q, k)$, where $V_k = \{v_k\}$ such that $f(v_k) = f_{q(u_0)}$ and $v_k \in M(G, Q, k)$.

Theorem 2

Given a query graph $Q = (V_q, E_q, f_q, w, u_0)$ a data graph $G = (V, E, f)$ and k, for $v \in V, (f(v) \neq f_q(u_0))$, if all $v\prime \in V$ which satisfies $(v\prime, v) \in E$ and $v\prime \notin M(G, Q, k)$, then $v \notin M(G, Q, k)$.

Proof

Assuming $v \in M(G, Q, k)$, denote the cor- responding matched focus as v_0, $(f(v_0) == f(u_0))$. According to the definition of approximate matching sub graph, there exists a path from v_0 to v since the given condition shows $f(v) \neq f_q(u_0)$. Denote parent vertex of v in this path as $v\prime\prime$ (i.e., $(v\prime\prime, v) \in E$), $v\prime\prime \in M(G, Q, k)$. It contradicts the given condition "all $v\prime \in V$ which satisfies $(v\prime, v) \in E$ and $v\prime \notin M(G, Q, k)$".□

According to Theorem 2, the third filter rule can be obtained.

Filter Rule 3

Filter Rule 3

For $\forall vinV$, if $f(v) \neq f_q(u_0)$ and $\forall(v\prime, v) \in E$ satisfies $v\prime \notin M(G,Q,k)$, then $v \notin M(G,Q,k)$.

3.3. Parallel Algorithm of Subgraph Query

From the perspective of a single vertex, we design a parallel algorithm and all vertices execute these supersteps separately. When one superstep finishes, all vertices must stop and wait for a synchronization before next superstep as shown in Fig. 2.

The detailed algorithm is shown in Algorithm 1. It consists of three basic supersteps and the following repetitive supersteps. For each vertex, it keeps four lists: *pList, cList, mList* and *lList*. *pList* and *cList* keep the parent nodes and child nodes respectively, which are important information for communication. *mList* and *1List* are the lists of the matched edges and the lost edges, which will be used to compute matched scores and lost scores. In the initial stage, the four lists (*pList, cList, mList* and *lList)* are set as empty and the state of v is set as "F". In Superstep 0, filter rule 1 is adopted to exclude some vertices according to their labels. If there is the same label in query graph, the current vertex sends its id and label ($[v, f(v)]$) to its child vertices (which are the input parameters). At the same time, it sets the state as "H" to denote further processing. Otherwise, the *match* state is kept as "F". "F" indicates that the vertex will not participate the following calculation any more. In Superstep 1, if a vertex receives messages, it saves the source vertex of the message to *pList*, as shown in Line 13 of Algorithm 1. To simplify the expression, "*pList* U *msg*" is used to show that "save the source vertex of the message to *pList* ". Then it replies id and label to the corresponding parent vertex. After the first two supersteps, every "H" vertex has the information of its matched parent vertices.

Algorithm 1 Parallel Algorithm for a Vertex

```
   Input: Current vertex v, direct successors of v, query
          graph Q
   Output: The matched state of v_0
   Initial:
1  pList=cList=∅ ; mList=lList=∅ ; state = "F";
   Super Step 0:
2  foreach u ∈ Q do
3  |  if f(v)==f_q(u) then
4  |  |  state = "H";
5  |  |  sendMessageToChildren([v, f(v)]);
6  |  end
7  end
8  if state ≠ "H" then
9  |  inactive=true;
10 |  return state;
11 end
   Super Step 1:
12 foreach msg ∈ receivedMessages do
13 |  pList = pList ∪ msg;
14 |  sendMessage(msg.getFirstItem, [v, f(v)]);
15 end
   Super Step 2:
16 foreach msg ∈ receivedMessages do
17 |  cList = cList ∪ msg;
18 end
19 [mList, lList]=initValues(cList);
20 foreach v' ∈ pList do
21 |  sendMessage(v, [v', mList, lList]);
22 end
23 if f(v) == f_q(u_0) then
24 |  aggregate(v,Φ(v));
25 end
   Super Step 3 and after:
26 if state=="F" then
27 |  inactive=true;
28 |  return state;
29 end
30 if Received messages then
31 |  Processing(messages);
32 |  if (f(v) ≠ f_q(u_0)) then
33 |  |  inactive=true;
34 |  end
35 else if (f(v) == f_q(u_0)) then
36 |  if v ∈ getAggregatedTopk then
37 |  |  state="T";
38 |  |  sendMessage(movedVertex, [v]);
39 |  |  inactive=true;
40 |  end
41 end
42 return state;
```

In Superstep 2, if a vertex receives messages, it updates *cList* according to the source vertex of the message (Line 17). With *cList*, current vertex v obtains its initial matched list (*mList)* and lost list (*lList)* as the following steps: (1) mapping edges $(v, v\prime), v\prime \in cList$ and $(v, v\prime) \in E$, to query graph; (2) obtaining the mapped edges set $\{(u, u\prime)\}, (u, u\prime) \in E_q$. Then the initial *mList* is the set of the edges $\{(u, u\prime)\}$; and the *lList* are the edges $(u, u\prime\prime), (u, u\prime\prime) \in E_q$ and $u\prime\prime \notin \{u\prime\}$. After that, The current vertex v sends its *mList* and *lList* to the parent vertices. If its label is the same with query focus, i.e. $f(v) == f_q(u_0)$, its matched value $\Phi(v)$ is send to global top-k aggregator.

In the following supersteps, vertex v calculates matched score $\Phi(v)$ and lost score $\psi(v)$ according to the updated *mList* and *lList*. The calculation is triggered once it receives messages. If there is no message, vertex v checks whether it belongs to the top- k list. If belongs to, the state of v is set as "T" and it sends message to the vertex which are moved out the top-k list (Line 38).

Algorithm 2 describes the detail of processing messages in repetitive supersteps. Once receiving message from child vertex, current vertex v updates *cList* and *pList*. If the message comes from parent vertex, v updates the state of the corresponding parent vertex in *pList*. It is a preparing work of filter rule 3. After updating the *pList* and *cList*, rule 2 and rule 3 are used to filter current vertex. If it satisfies the conditions of rule 2 or rule 3, the state of v is set as "F" and it becomes inactive. Especially, v has been moved out from the top-k list if the message came from the other top- k vertex. Because it is possible for v to be included in top- k list again, the moved vertex state is set as "H". In a superstep, if $\Phi(v)$ or $\psi(v)$ changes, the vertex sends the updated matched list *mList* and lost list *lList* to the vertices in *pList*. At last, if it is the mapping vertex of focus node u_0, its matched score is sent to the aggregator to update the top-k list.

In these supersteps, vertex obtains information by messages between vertices. Taking vertex v as example, Fig. 3 shows main messages between v and its parent vertex p and child vertex c • In Superstep 0, v sends it id and label to vertex c while receiving from vertex p. In Superstep 1, the direction of message transport becomes converse compared to Superstep 0. In Superstep 3, v sends its *mList* and *lList* to vertex p and receives them from child vertex. This kind of message is repeated in the following supersteps. Note that, if v is the mapping of query focus, it should send matched score $\Phi(v)$ to the aggregator.

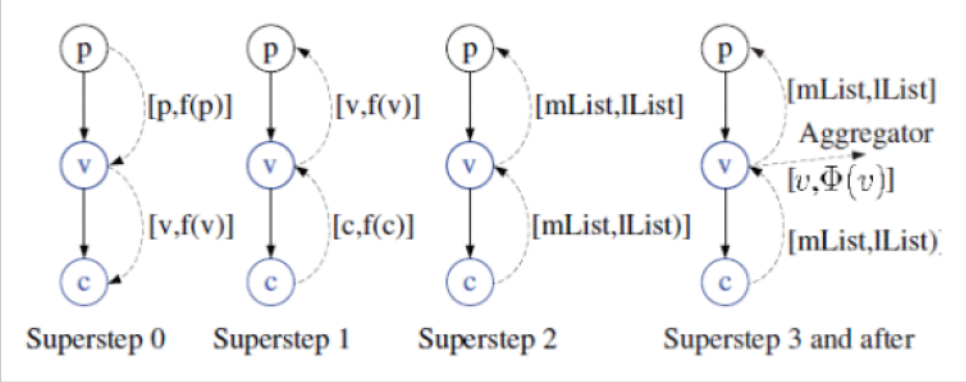


Figure 3:
The message illustration.

The parallel program terminates when all vertices become inactive. The vertices in top-k list (whose states are all "T") are the results of top- k subgraph query based on query focuses. With these vertices in top-k list, it can be further traced to top-k match subgraph $M = (G, Q, k)$.

Algorithm 2 Process Messages in Iteration Supersteps

```
   Input: Current vertex v, query graph Q, messages
   Output: cList, pList
 1 foreach msg ∈ messages do
 2     switch msg came from do
 3         case child vertex
 4             update cList;
 5         case parent vertex
 6             update pList;
 7         case others
 8             state = "H";
 9             inactive = true;
10             exit;
11         endsw
12     endsw
13 end
14 if satisfy(filter rule 2 or rule 3) then
15     state="F";
16     foreach v' ∈ cList do
17         sendMessage(v', [v, state]);
18     end
19     inactive=true;
20     exit;
21 end
22 if received message from child vertex then
23     calculate Φ(v) and Ψ(v);
24     if there is change of Φ(v) or Ψ(v) then
25         foreach v' ∈ pList do
26             sendMessage(v', [v, mList, lList]);
27         end
28         if f(v) == f_q(u_0) then
29             aggregate(v,Φ(v));
30         end
31     end
32 end
33 return newmatchValue, newloseValue;
```

Example 2

We use the query graph and data graph in Fig. 1 as an example to illustrate the proposed approach. Fig.1a shows a weighted query graph. In this example, query focus is set as "PM" and the parameter k is set as 2. To distinguish the vertices of data graph, we denote a sequence number for each vertex such as $v_i (i \in [1 \cdots 15])$.

Table 2: Matched scores $\Phi(v)$ and lost scores $\psi(v)$

		v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	v12	v13	v14
Superstep 2	$\Phi(v_i)$	1.5	**2.4**	**1.7**	0.6	0.5	0	0.6	0.5	0	0	0.6	0.4	0.6	0
	$\Psi(v_i)$	0.9	0	2.3	0	0	0	0	0	0	1	0	0	0	1
Superstep 3	$\Phi(v_i)$	**2.6**	**3.5**	2.2	0.6	0.5	0	1.0	0.5	0	0	1	1	0.6	0
	$\Psi(v_i)$	0.9	0	2.3	1	0	0	0	0	0	1	0	0	0	1
Superstep 4	$\Phi(v_i)$	2.6	**3.9**	2.2	0.6	0.5	0	1.6	0.5	0	0	1	1	0.6	0
	$\Psi(v_i)$	1.9	0	2.3	1	0	0	0	0	0	1	0	0	0	1
Superstep 5	$\Phi(v_i)$	2.6	**4.5**	2.2	0.6	0.5	0	1.6	0.5	0	0	1	1	0.6	0
	$\Psi(v_i)$	1.9	0	2.3	1	0	0	0	0	0	1	0	0	0	1

Algorithm 3 Preprocess node influence $\Theta(u)$

```
   Input: Q = (V_q, E_q, w), u (u ∈ V_q), flags
   Output: Θ(u)
 1 Θ(u) = 0;
 2 foreach u' ∈ {u'} where (u, u') ∈ E_q do
 3     Θ(u) = Θ(u) + w((u, u'));
 4     flags(u)=true;
 5     if not flags(u') then
```

```
6       Θ(u) = Θ(u) +recursion(Q, u', flags) ;
7     end
8   end
9   return Θ(u);
```

In query graph, node influence can be obtained by a breadth- first search algorithm shown in Algorithm 3. For each child node $u\prime$ of current node u, the weight of $(u, u\prime)$ is added to $Theta(u)$ and the flag of node u is set as true. Then, call the processing recursively (Line 6 in Algorithm 3). In this example, node influences in Fig. 1a are as follows: 8(A)=4.5, 8(B)=1.6, 8(C)=0.5, 8(D)=0, 8(E)=1, and 8(F)=1.

The first two supersteps (Superstep 0 and Superstep 1 in Algorithm 1) are used to exchange information between vertices. The initial matched scores $\Phi(v)$ and lost scores $\psi(v)$ are calculated in Superstep 2. So $\Phi(v)$ and $\psi(v)$ are shown in Table 2 since Superstep 2. According to filter rule 1, $v15$ keeps inactive with a match state of "F" because there is no "SU" label in query graph. Therefore, $v15$ has not been listed in Table 2.

In Superstep 2, the top-2 focus vertices of M(G,Q,2) are $v2$ and $v3$. But in Superstep 3, $v1$ has larger matched score than $v3$. So $v3$ is moved out from the top-2 focus list. In our approach, $v1$. sends a message to $v3$ and $v3$ will change its state from "T" to "H". When the lost score of $v3$ becomes 2.3, its state is set as "F" according to filter rule 2 (2.3>4.5-2.6). In Superstep 4, $v1$ gets the lost information of $v10$ and update its lost value from 0.9 to 1.9. In Superstep 5, $v2$ reaches the maximum matched scores (4.5). So $v2$ and $v1$ are selected as the final top-k focus vertices of M(G,Q,2). □

Table 3: Basic statistics on experimental datasets,

	# Vertices	# Edges
Twitter	81,306	1,768,149
Amazon	403,394	3,387,388
Google	875,713	5,105,039

SECTION 4.
Experiments

The goal of this experimental study is to evaluate the proposed algorithms, including the number of superstep, the number of messages and the run time.

4.
Platform

The experiments are conducted on a cloud platform, which consists of five instances. Each node has 16GB RAM, 4 VCPUs. Hadoop [12] and HAMA [15] are deployed on the platform. In our experimental cluster, one instance plays the role of master node, and the others are workers (data node).

4. Data Sets

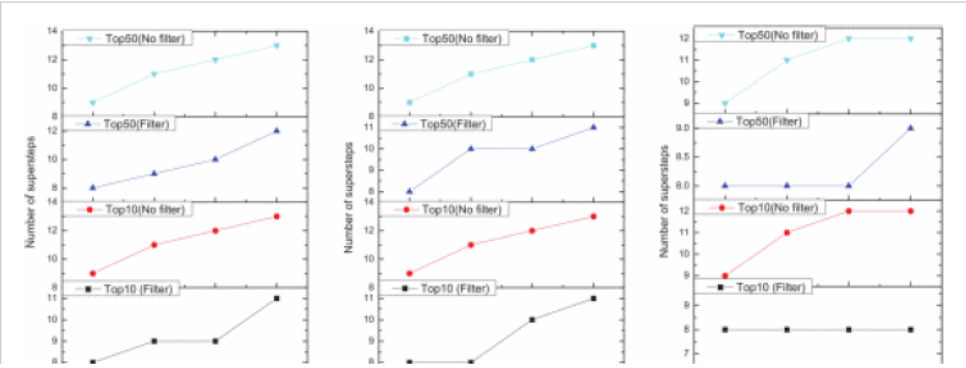
Three real world datasets are used in the experiments [18]. The information of data graph is listed in Table 3. Twitter data graph is with 81,306 vertices and 1,768,149 edges; Amazon data graph is with 403,394 vertices and 3,387,388 edges; and Google data graph is with 875,713 vertices and 5,105,039 edges. For each data graph, twelve labels are randomly assigned to the vertices of data graphs. We generate query graphs with various sizes. Denote the size of query graph as $|Q|$, then $|Q|$ include 6, 8,10 and 12 in the experiments, respectively.

4. Supersteps

Fig. 4 shows the results of supersteps. From these results, we have several observations as follows.

1. The number of supersteps is limited to an upper bound, which is related to the size of query graph. As the sizes of query graphs increase, the numbers of supersteps keep the same or increase in all situations. The more vertices in query graph, the longer path to transmit matched lists and lost lists to the top-k vertices. One interesting observation is that the numbers of supersteps sometimes keep the same with various sizes of query graphs, e.g. the top-10(filter) results in Fig 4c. It is because there is no increased path in data graphs which is caused by the increased vertices in query graphs. Another important observation is that the numbers of supersteps are bounded in all situations. For example, in Fig. 4b, the numbers of supersteps are at most 13. The reasons is that, for acyclic query graph, the number of supersteps is related to the longest path in query graph. In our algorithm, there are two preprocess supersteps and one additional superstep to update the state of top-k vertices.

2. The filter mechanism reduces the number of super-steps. The filter rules are used to terminate the computation of the vertices which are impossible to be included in the final top-k subgraph matches. Once vertices are terminated, they will not send messages in the following supersteps. Therefore, the numbers of supersteps are reduced compared to the method without filter mechanism. For example, in Fig. 4b, the superstep numbers of filter top-10 query are 8, 8, 10, 11 for four sizes of query graph, while they are 9, 11, 12, and 13 for top-10 query without filter.



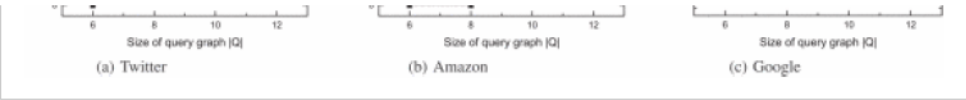


Figure 4:
Numbers of supersteps with and without filter mechanism

4. Messages

Fig. 5 shows the results of communication messages between vertices. It can be seen that messages are correlated to the sizes of query graphs. The bigger query graph generates the more temporary matched vertices, which means larger communication cost. In the experiments, we only compare the total number of messages between vertices. If a vertex sends one message, we increases the number of messages with 1.

Fig. 5 shows that the number of messages is related to not only query graph, but also datasets. For instance, in Fig. 5c, the numbers of messages with google dataset are larger than that of the other two datasets. In Fig. 5b, twitter graph is denser than amazon graph which also causes more messages.

Compared to top-k query without filter, the top-k query with filter needs less numbers of messages. It is because a filtered vertex will not send message any more. In our filter mechanism, filter rule 2 causes additional messages as a vertex informs its child vertices, which lowers the benefit of reducing messages in some degree. However the messages are still reduced in all experimental situations. Another observation is that the numbers of messages are close for various $k = 10$ and $k = 50$. The reason is that, the larger k means more match results and more messages, on the other hand, the larger k means the smallest matched score in current top-k list may become smaller, which causes less filtered vertices.

4. Running Time

The running time includes the time of local computation, global communication and synchronization time. Fig. 6 shows the results of the total running time on three datasets. It can be seen that running time increases for all datasets as query graph size $|Q|$ increases. This is consistent with the analysis of the proposed algorithms. For a given dataset, the increase size $|Q|$ means more labels in query graph during the first two supersteps. In these supersteps, local computation and global communication time increase.

Compared to the results of top-k query without filter, the runtime of top-k query with filter are always less with the same k value. It demonstrates the effectiveness of the proposed filter mechanism. However, it is possible that the run time is very close for multiple situations. For example, in Fig. 6c, the queries with filter and without filter are of the close run time when query graph size is 6. It is because the number of perfect matches is close to k, e.g. most top-50 matches have been founded when looking for top-10 matches.

SECTION 5.

Related Work

5.1. Graph Matching

Graph matching has been studied in the graph query processing literature with respect to graph isomorphism and graph simulation [31] [32].

The graph isomorphism problem consists in deciding whether two given graphs are isomorphic, i.e., whether there is a bijective mapping from the vertices of one graph to the vertices of the second graph such that the edge connections are respected. Many proposals have been presented on graph isomorphism [27] [28]. For example, a serial algorithm is proposed in [27], which tries to reduce spatial complexity and to achieve a better performance on large graphs. When applied to find isomorphic subgraphs in a massive graph, one challenge is that the time complexity is too costly. Recently, a study focuses on top-k diversified subgraph querying that asks for a set of up to k subgraphs isomorphic to a given query graph [5]. However, subgraph isomorphism problem is an NP-complete problem [29]. It becomes especially hard as the sizes of graphs increase since exact subgraph isomorphism associated with expensive time complexity has limited its efficacy.

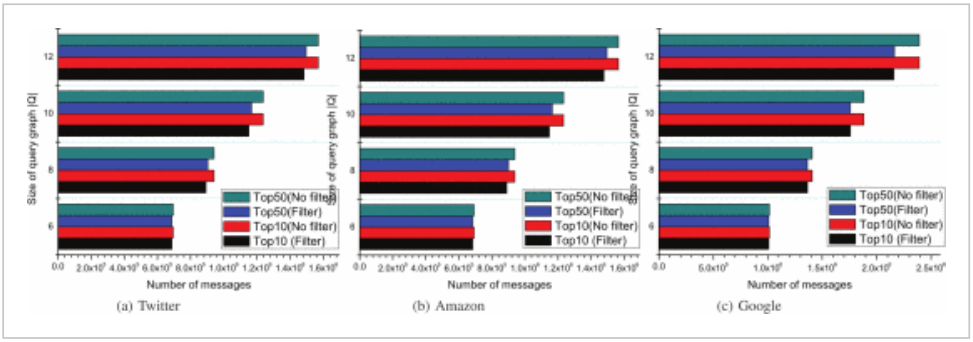


Figure 5:
Numbers of messages with and without filter mechanism

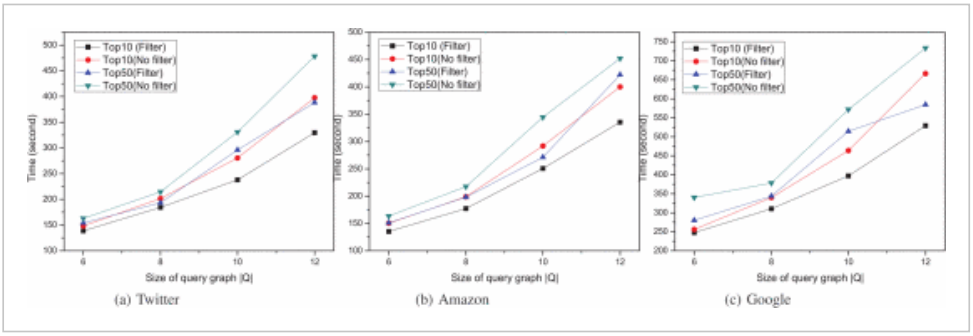


Figure 6:
Running time of the parallel algorithm with and without filter mechanism

Graph simulation provides a practical alternative to subgraph isomorphism by relaxing its stringent matching conditions. It allows matches to be found in polynomial time. Therefore, subgraph simulation techniques have received much attention due to their pragmatic applicability [13]. Several subgraph simulation models are introduced like bounded simulation[23], strong simulation[24] and relaxation simulation [30]. Bounded simulation extends graph simulation for graph pattern matching by allowing bounds on the number of hops. To extend [23], a proposal incorporated regular expressions as edge constraints is proposed [26]. These graph simulation models may modify the matching conditions in different ways. As a basic condition, all child vertices need to be matched if two vertices are matched. Recently, the simulation of permitting the

absence of one hop node is taken into consideration [30]. However, it is crucial to return approximate query sub graph for a given query graph, which permits any absence of vertices and edges. But the results include the rank information.

5.2. Top-k Query

Some prior works have proposed solutions in top-k vertex query [9], top-k path-based relevance query [10], top-k maximal clique query [11] and top-k similarity join, etc. Yan et al. deal with the problem of finding top-k highest aggregate values over a h-hop neighbors [9] Furthermore top-k aggregation queries are proposed that involves identifying top-k vertices with highest aggregate values over their h-hop neighbors [7]. With path-based metrics, the solution obtain the top-k most relevant items for a give query item set [10]. But these algorithms don't take subgraph queries into consideration.

The top-k subgraph match problem has attracted great attention recently. It can be solved by first finding all matches for the query using the existing graph matching methods and then ranking the matches. But the cost of exhaustive enumeration for all the matches can be prohibitive for large graphs. Some works propose more efficient approaches for top-k sub graph query. To find top-k subgraphs, Gupta et al. propose index-based approach [8]. Their solution exploits two low-cost index structures (a graph topology index and a maximum metapath weight index) to perform top-k ranking. But their approach considers the weighted data graph, while not query graph. Recently, diversified top-k subgraph query has attract some research interest [6] [5]. Subgraph isomorphism [5] and subgraph simulation [6] are used as the matching metrics for top-k sub graph query.

For edge-weight graph computation, some prior works consider weights in data graphs. They aim to find top-k matches in an uncertain graph. For example, Zou. et. al investigate the problem of finding top-k maximal cliques in an uncertain graph [11] [25]. The weight of each edge is denoted as a probability value such as the probability presenting the chance of the interaction existing in practice protein-protein interaction networks. However, they have considered the weights in data graph, but not in query graph.

SECTION 6.
Conclusion

We study the problem of top-k sub graph query in massive graph. Different from existing graph isomorphism, we present a new graph matching notion: approximate graph simulation. For the first time, sub graph query can be customized by weighting query graph, which provides good flexibility for different application scenarios. Moreover, we propose a parallel algorithm at the level of single vertex. Every single vertex can obtain its matching state separately without requiring global graph information. Therefore, it is very suitable for massive graph computing due to the strong scalability. Our experiments with real-life datasets show that the proposed algorithm is efficient in distributed environment.

Keywords

IEEE Keywords

Computational modeling, Algorithm design and analysis, Parallel algorithms, Approximation algorithms, Synchronization, Program processors

INSPEC: Controlled Indexing
query processing, approximation theory, graph theory, parallel algorithms

INSPEC: Non-Controlled Indexing
top-k vertices, parallel top-k subgraph query, massive graphs, parallel algorithm, graph matching, approximate graph simulation, weighted query graph, vertex level, filter mechanism, computation mechanism, aggregation mechanism

Authors

Jianliang Gao
School of Information Science and Engineering, Central South University, Changsha, China

Bo Song
College of Computing & Informatics, Drexel University, Philadelphia, USA

Ping Liu
School of Information Science and Engineering, Central South University, Changsha, China

Weimao Ke
College of Computing & Informatics, Drexel University, Philadelphia, USA

Jianxin Wang
School of Information Science and Engineering, Central South University, Changsha, China

Xiaohua Hu
College of Computing & Informatics, Drexel University, Philadelphia, USA

Related Articles

Joint trajectory generation for redundant robots
T.C. Hsia; Z.Y. Guo

Asynchronous leasing
R. Boichat; P. Dutta; R. Guerraoui

Dictionary design algorithms for vector map compression
S. Shekhar; Yan Huang; J. Djugash

Schedulability in model-based software development for distributed real-time systems
S.S. Yau; Xiaoyong Zhou

Performance evaluation of a probabilistic replica selection algorithm
S. Krishnamurthy; W.H. Sanders; M. Cukier

Computational complexity management of motion estimation in video encoders
Yafan Zhao; I.E.G. Richardson

Distributed object-oriented real-time simulation of the multicast protocol RFRM
Y.S. Hong

Wavelet-based lossy compression of barotropic turbulence simulation data
J.P. Wilson

GUI approach to programming of TMO frames
K.H. Kim; Seok-Joong Kang; Yuqing Li

Implementation of a TMO-based real-time airplane landing simulator on a distributed computing environment
Min-Gu Lee; Sunggu Lee

SECTION 1.
Introduction

Research of massive graph has attracted renewed interest in recent years due to increased interests in a number of network applications such as biological networks, social networks, communication networks [14]. Network data are often represented as graphs, where nodes are labeled entities and edges represent relations among these entities [1] An important processing for graph data application is that of sub graph query.

Subgraph query is to find out the subgraphs of data graph G which match a given query graph Q. It is essential for a wide range of emerging applications such as community discovery and neighbor query in social networks, biological data analysis [13], classification of web documents [16], software plagiarism [19] and so on. A number of matching algorithms have been developed to compute the set of matches of Q in G. In real world applications, query graph is usually small, but data graph is typically large, even including billions of nodes [3] The large scale graphs such as social graphs and protein-protein interactive networks (PPI) give rise to the following problems with the subgraph query algorithms.

1. The large size of data graphs makes matching costly. For matching defined by simulation, it takes $O(|G||Q| + |G|^2)$ time to compute graph match set [4]. It can be seen that the size of data graphs could greatly affect the computational complexity. Furthermore, the matching algorithms often return an excessive number of sub graphs. It could be a difficult task for the users to inspect such a large number of matching subgraphs and select the most desirable solution.

2. The top-k subgraph query is needed to filter unwanted matching subgraphs. Exiting top-k subgraph query approaches include two phases: 1) computing all matching subgraphs which satisfy the query in a sequential way; 2) ranking such results based on given metrics. However, previous works on the matching problem could face difficulty due to the lack of scalability for the increasing scale of graphs.

3. Subgraph query should take the weighted graph into consideration. With the ever-increasing popularity of entity-centric applications, it becomes very important to study the interactions between entities, which are captured using edges in entity-relationship networks [8]. Entity-relationships usually have different importance in various applications. For example, the entity-relationship in social network might be different according to various personal influence. In protein-protein interaction networks, the probability value to present the interaction existing in practice is an important parameter. The values between entities of a network can be mapped to the weights on edges of a graph. Therefore, subgraph query according to weighted graphs becomes important in massive graph computing.

Some researches aim to solve the above problems in subgraph query. For example, diversified top-k subgraph query is proposed recently [5] [6]. To simplify the subgraph query, “query focus” is introduced as an substitution of finding the entire set of matches [6]. By appointing a focus in query graph, sub graph query returns only those vertices which match the query focus, rather than the entire matching subgraph set. Query focus based subgraph query is widely needed in e.g. expert recommendation [20] and egocentric search [21]. In fact, 15% of social queries are to find matches of specific nodes [22]. In this paper, we also take the “query focus” as the subgraph query.