Spotting Significant Changing Subgraphs in Evolving Graphs

Zheng Liu, Jeffrey Xu Yu, Yiping Ke
The Chinese University of Hong Kong
{zliu, yu, ypke}@se.cuhk.edu.hk

Xuemin Lin
The University of New South Wales
lxue@cse.unsw.edu.au

Lei Chen
Hong Kong University of Science and Technology
leichen@cs.ust.hk

Abstract

Graphs are popularly used to model structural relationships between objects. In many application domains such as social networks, sensor networks and telecommunication, graphs evolve over time. In this paper, we study a new problem of discovering the subgraphs that exhibit significant changes in evolving graphs. This problem is challenging since it is hard to define changing regions that are closely related to the actual changes (i.e., additions/deletions of edges/nodes) in graphs. We formalize the problem, and design an efficient algorithm that is able to identify the changing subgraphs incrementally. Our experimental results on real datasets show that our solution is very efficient and the resultant subgraphs are of high quality.

1. Introduction

Discovering patterns in graph data is a challenging task with many applications in Web analysis, social networks, telecommunication, sensor networks, etc. Due to the expressive power of graphs, graph patterns have to represent complex structural relationships among objects in various domains in the real world. In the literature, the existing work [5, 4, 7, 9] discovers complex relationships in a large graph that is static.

However, many applications show that graphs are evolving over time. In social science, large social networks are changing, and social network changes are caused by the proximity changes [9]. In bioinformatics, finding coevolution relationships of structure and function in the structural genomic is an important task to understand evolution progresses [8]. In network, traffic jam occurs at one link may affect the traffic routing in a large range. Monitoring the dynamic topology changes and their influences provides with network administrators the insights on network configuration [3]. Also, in wireless sensor networks, query processing is done by exchanging information between sensors where the communication range of a sensor is limited. The fact that some sensor runs out of power has impacts on the other sensors in terms of network routing and therefore query processing time. In a sensor network, it is important to note, even beforehand, which subgraphs will be affected significantly when such a change occurs.

Such changes can be modeled in a sequence of large graphs, $G = (G_1, G_2, \cdots)$, where nodes/edges can be added and/or deleted into/from G_i which results in another large graph G_{i+1}, and G is considered as an evolving graph. In this work, we take an edge-centric view regarding changes. We focus on edge changes (deletion/addition) which will cause structural changes. On the other hand, vertex changes also have impacts on structural changes. But adding isolated vertices before they are connected to any other vertices seems less important, while deleting vertices can be considered as removing edges connected to the deleted vertices.

Given two graphs G_i and G_{i+1} at time t_i and time t_{i+1}, there are many small subgraphs that change while the majority of the graph remains unchanged. A small changing subgraph can be a connected subgraph where every edge is changed (deleted from G_i or added into G_{i+1}), and such a small changing subgraph can be easily identified. However, the influence of a single edge change (deletion/addition) on the other parts of the large graph is more important than the physical change itself. For example, when a researcher A works with another researcher B for a new research issue, A's collaborators and B's collaborators may have new opportunities to work together. Consider the two researchers as two vertices. The newly added edge between them may change the closeness of the vertices that are directly/indirectly connected to the two vertices. Suppose that the closeness of two vertices can be measured. A changing subgraph is an induced subgraph in which the closeness be-

Symbol	Definition
G	An evolving graph
G_i	The snapshot of evolving graph G at time t_i
A_i	The adjacency matrix of graph G_i
P_i	The transition matrix of graph G_i
v_j	A vertex on a graph
$N(v_j)$	The set of neighbors of vertex v_j
$d(j)$	The sum of edge weights between vertex v_j and $N(v_j)$
D_i	The diagonal matrix where $d_{jj} = d(v_j)$ at time t_i
Π_i	The vertex closeness matrix at time t_i
VI_i	The vertex importance score at time t_i

Table 1. Notations

tween vertices changes. In this paper, we focus on the problem of spotting significant changing induced subgraphs in an evolving graph. The issues that we concentrate on include how to measure the closeness changes between two vertices that are caused by some edge changes, how to identify the boundary of the influences of a change, and how to determine a changing subgraph in which changing parts have influences on each other.

The main contributions of this paper are summarized below.

- We formalize the problem of spotting significant changing subgraphs in an evolving graph and propose to measure the vertex closeness with structure information using neighborhood random walks.
- We develop an incremental algorithm to speed up the vertex closeness computation, as well as a novel strategy about expanding the important vertex to acquire the connected induced subgraph which can reflect the closeness change between vertices.
- We present an evaluation of our proposed approach by using various large real data sets demonstrating that our method is able to find the suitable subgraph effectively and efficiently.

2. Changing Subgraph Discovery

We define an evolving graph as a sequence of undirected graphs, denoted as $G = (G_1, G_2, \cdots)$, where $G_i(V_i, E_i)$ is a snapshot of graph G at time t_i with a set of vertices V_i and a set of edges E_i. For simplicity, given two graphs $G_i(V_i, E_i)$ and $G_{i-1}(V_{i-1}, E_{i-1})$, the two sets of vertices, V_i and V_{i-1}, are identical, while the two sets of edges, E_i and E_{i-1}, are possibly different. The notations used in this paper are summarized in Table 1.

Consider an evolving graph. An edge change may make some vertices become closer and at the same time may make some other vertices become looser. As shown in Figure 1, there are two graphs G_1 and G_2 in an evolving graph at time t_1 and t_2, respectively. At time t_1, there is an edge between

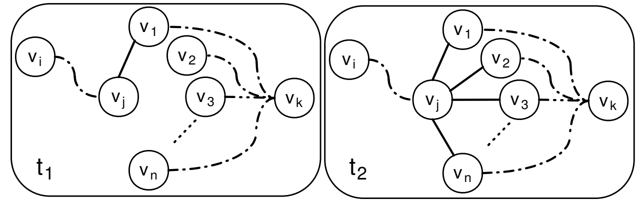


Figure 1. Changes

v_i and v_j, and there is a path between v_j and v_k. At time t_2, there are more paths from v_j to v_k, where the edge between v_i and v_j remains unchanged. In comparison with G_1 at time t_1, the closeness between v_j and v_k becomes closer, because the newly added edges make it easier for v_j to traverse to v_k. On the other hand, the closeness between v_j and v_i becomes looser, because v_j has more opportunities to traverse to other vertices. This fact motivates us to consider an accumulative score that measures the overall impacts of all changes on a vertex when a graph evolves. Reconsidering v_j in Figure 1, we need to consider the relationships between v_j and v_i, between v_j and v_k, and between v_j and any other vertex, in order to judge the change influence on v_j. Given such an accumulative score, it becomes possible to find significant changing subgraphs when a graph evolves.

In this paper, we explore this issue using neighborhood random walks on graphs to help spotting significant changing subgraphs. We first review some basic concepts of random walks on graphs. Let A_i denote the adjacency matrix of a weighted graph G_i, where $A_i(j, k)$ maintains the weight for the edge (v_j, v_k). A random walk on G_i is performed in the following way. A particle starts at a certain vertex v_0, which is a vertex involved in an edge change. Suppose it walks to a vertex v_s in the s-th step and it is about to move to one of the neighbors of v_s, denoted as $v_t \in N(v_s)$, with probability p_{st}, where p_{st} is $A_i(s,t)/\sum_{v_k \in N(v_s)} A_i(s,k)$, and $N(v_s)$ contains all neighbors of vertex v_s. The vertex sequence of the random walk is a Markov chain. Let D_i be the diagonal matrix with the diagonal value $d(s) = \sum_{v_k \in N(v_s)} A_i(s,k)$, then the transition probability matrix P_i of the Markov chain for graph G_i is

$$P_i = D_i^{-1} A_i.$$

The probability of going from v_j to v_k through a random walk of length l can be obtained by multiplying the transition probability matrix l times and is given as $P_i^l(j,k)$.

A fixed step approach with restart (fixed l and $0 < c < 1$): With a fixed l, we focus on the local structural information using neighbors of a vertex, v_j, from which the random walk starts. The vertex, v_j, to start random walks is the vertex that is involved in an edge change. The neighbors of v_j are the vertices that v_j can reach in l steps. Random walks are only conducted in the l-step neighborhood of the vertex v_j with a restart probability c. It is important to note that our

algorithm is designed in a way that a user can enlarge the l value if needed at run time. We adopt the similar expected f-distance in [6, 5]. In the expected f-distance, a parameter c is used. We prove that such a parameter c is the restart probability used in [7] with minor difference which can be ignored. Due to the space limit, we omit the proof. In short, neighborhood random walk distance, which is also called the *vertex closeness*, is the expected f-distance defined on random walks whose length is smaller or equal to l.

Definition 1 Neighborhood Random Walk Distance (Vertex Closeness)*: Let P_i be the $n \times n$ transition probability matrix of a graph G_i. Given l as the length that a random walk can go, the neighborhood random walk distance $\Pi^l(j,k)$ from v_j to v_k is defined as follows:*

$$\pi(j,k) = \sum_{\tau: v_j \rightsquigarrow v_k; length(\tau) \le l} p(\tau) c (1-c)^{length(\tau)}, \quad (1)$$

where $0 < c < 1$, and τ is a path from v_j to v_k whose length is $length(\tau)$ with transition probability $p(\tau)$.

The matrix form of the neighborhood random walk distance is as follows.

$$\Pi_i^l = \sum_{\gamma=1}^{l} c(1-c)^{\gamma} P_i^{\gamma}. \quad (2)$$

Here, P_i is the transition probability matrix for graph G_i, and Π_i is the neighborhood random walk distance matrix for graph G_i. We then define the *importance score* of a vertex as the accumulative change of its closeness to other vertices in Eq. (3).

$$VI_i(v_j) = \sum_{v_k \in V_i} |\Pi_{i-1}^l(j,k) - \Pi_i^l(j,k)|. \quad (3)$$

Here, $VI_i(v_j)$ is the importance score of a vertex v_j when a graph evolves from graph G_{i-1} to G_i.

We propose a two-step framework to spot significant changing subgraphs in an evolving graph. First, we compute the importance score $VI_i(v_j)$ for any vertex v_j in graph G_i that is involved in edge changes. Second, based on the importance scores, we find significant changing subgraphs.

3. Vertex Importance Score Computation

In this section, we discuss in detail how to calculate the difference of vertex closeness in two graphs G_{i-1} and G_i and the vertex importance scores.

3.1. The Straightforward Algorithm

We can develop a straightforward algorithm to compute the difference of vertex closeness and the vertex importance score based on the definitions. The straightforward algorithm iteratively calculates the respective closeness matrices Π_{i-1} and Π_i at time t_{i-1} and t_i based on Eq. (2) by the power method. The closeness difference matrix is simply computed as $C_i = \Pi_i - \Pi_{i-1}$, based on which the importance scores of vertices can be easily computed by Eq. (3). The time complexity of the straightforward algorithm is $O(ln^3)$. One can use the fast sparse matrix multiplication instead of the normal matrix multiplication to improve the speed, but usually that is not enough to lower the running time especially when G is large and there are a lot of edge changes.

3.2. A Novel Incremental Algorithm

In this section, we introduce a novel incremental algorithm that computes the closeness difference directly for those vertex pairs with changing closeness.

Let us start from a simple case. Suppose that there is only one edge e that is different between two graphs G_i and G_{i-1}. It can be either the addition of e to G_i or the deletion of e from G_{i-1}. The question is to identify those vertex pairs whose closenesses change due to the difference of e, as well as the quantities changed. Recall our closeness measure in Eq. (1). The answer to the above question is that if a vertex pair has at least one tour path passing through the edge e or one of the two vertices incident to edge e, the closeness of the vertex pair changes. By identifying those paths, we can find the vertex pairs with changing closenesses. Furthermore, the summation of the probability of these paths is exactly the quantity changed in the closeness of each vertex pair.

By Eq. (2), the iterative form of the vertex closeness is

$$\Pi_i^l = \sum_{\gamma=1}^{l} c(1-c)^{\gamma} P_i^{\gamma} = c(1-c)^l P_i^l + \Pi_i^{l-1}. \quad (4)$$

Therefore, the closeness difference matrix is

$$\Delta\Pi_i^l = c(1-c)^l (P_i^l - P_{i-1}^l) + \Delta\Pi_i^{l-1}. \quad (5)$$

By Eq. (5), we can see that the key step in computing $\Delta\Pi_i^l$ is to compute $(P_i^l - P_{i-1}^l)$, which is easy when $l = 1$. When $l > 1$, obviously we cannot compute it in a naive way by the power method since it is computational expensive. Recall that $P_i^l(j,k)$ is the probability of going from v_j to v_k through random walks of length l on graph G_i. We now show how to calculate $\Delta P_i^l = P_i^l - P_{i-1}^l$ in an efficient way. Apparently,

$$P_i^l(j,k) = \sum_{\tau: v_j \rightsquigarrow v_k} p_i(\tau), \quad (6)$$

where τ is a tour path from vertex v_j to v_k, and $p_i(\tau)$ is the probability of path τ in G_i. Suppose $\tau =$

$\langle v_1, v_2, ..., v_l \rangle$, where $v_1 = v_j$ and $v_l = v_k$, then $p_i(\tau) = \prod_{m=1}^{l-1} A_i(v_m, v_{m+1})/d(v_m)$. The sum of the probability of all these distinct tour paths is $P_i^l(j,k)$.

In order to compute ΔP_i^l, we only need to consider the different paths on G_i and G_{i-1}, as well as the difference in the probability of the same paths. For simplicity, we only discuss the case when there are only additions of edges or increase of edge weights. We will show later that our algorithm can handle deletions of edges and decrease of edge weights as well.

Let (v_m, v_n) be one of the added edges or one of the edges whose weights increase. For any vertex pair $\{v_j, v_k\}$, if there is a tour path τ of the maximum length l starting from v_j, passing through the edge (v_m, v_n) and ending at v_k, then the vertex closeness $\Pi_i(j,k)$ will increase by $p_i(\tau)$, since this path does not exist in G_{i-1}. On the other hand, if there is a tour path τ of the maximum length l starting from v_j, passing v_m or v_n or both, and ending at v_n, but without passing through (v_m, v_n), then the vertex closeness $\Pi_i(j,k)$ will decrease by $(p_{i-1}(\tau) - p_i(\tau))$, since the path τ exists in both G_i and G_{i-1}, and with the increase of $d(m)$ and $d(n)$ in G_i, the probability of the path τ decreases. We formalize the above analysis in Theorem 3.1.

Theorem 3.1 *Given two graphs G_i and G_{i-1} of an evolving graph G, let (v_m, v_n) denote the changing edge, then $\Delta P_i^l(j,k)$ can be computed as follows:*

$$\Delta P_i^l(j,k) = \sum_{\tau: v_j \rightsquigarrow v_k; (v_m,v_n) \in \tau} p(\tau) + \sum_{\tau: v_j \rightsquigarrow v_k; (v_m,v_n) \notin \tau; v_m \text{ or } v_n \in \tau} (p_i(\tau) - p_{i-1}(\tau)). \quad (7)$$

Theorem 3.1 suggests an effective way to calculate the change quantity of the closeness between vertex pairs. The key is to find all the related paths distinctly and completely so that the change quantity is computed correctly. To enumerate all the possible positions of the edge (v_m, v_n) in a path τ is obviously not a good solution due to the exponential number of combinations with respect to the number of changing edges and the range l.

We first discuss the case of the path $\tau : v_j \rightsquigarrow v_k$ when $(v_m, v_n) \in \tau$. We can calculate the closeness difference in the following way. For the changing edge $e = (v_m, v_n)$, we first calculate the probability of a path τ_1 from v_j to v_m with length l_1, where $l_1 \leq l - 1$. We then calculate the probability of a path τ_2 from v_n to v_k with length $l_2 = l - l_1$. In this way, we ensure that the computed paths from v_j to v_k passing the edge (v_m, v_n) is of length l. The closeness difference that is accounted for such paths can be computed as

$$\pi_1(j,k) = \sum_{\tau_1: v_j \rightsquigarrow v_m; \tau_2: v_n \rightsquigarrow v_k} p(\tau_1) P_i(m,n) p(\tau_2), \quad (8)$$

where $\pi_1(j,k)$ denotes the first term of $\Delta P_i^l(j,k)$ in Eq. (7).

In order to compute $p(\tau_1 : v_j \rightsquigarrow v_m)$ and $p(\tau_2 : v_n \rightsquigarrow v_k)$ correctly without missing and double-computing any path, we do not allow a path τ_1 from v_j to v_m to pass e , while we do not have this restriction on path τ_2.

As for the other case of the path $\tau : v_j \rightsquigarrow v_k$ when v_m or $v_n \in \tau$ but $(v_m, v_n) \notin \tau$, the closeness difference can be computed in a similar way:

$$\pi_2(j,k) = \sum_{\substack{\tau_3: v_j \rightsquigarrow v_m; \tau_4: v_m \rightsquigarrow v_k \\ \text{or } \tau_3: v_j \rightsquigarrow v_n; \tau_4: v_n \rightsquigarrow v_k}} p(\tau_3) p(\tau_4), \quad (9)$$

where $\pi_2(j,k)$ denotes the second term of $\Delta P_i^l(j,k)$ in Eq. (7). For the correctness of computation, we do not allow τ_3 to contain the vertex v_m (or v_n when τ_3 is from v_j to v_n) and we do not allow τ_4 to contain the edge e.

We handle all the changed vertices together in matrix form instead of one by one. We use six arrays to store the corresponding probability of the four types of paths τ_1 to τ_4 discussed above, the total number of whose entries is much small than n^2.

We have discussed how to handle the additions of edges and the increase of edge weights. In fact, our algorithm can also handle the situation when there are deletions of edges and decrease of edge weights. Let us first suppose that there are only deletions of edges and decrease of edge weights from G_{i-1} to G_i. It is easy to see that this is exactly the same as the evolvement from G_i to G_{i-1}, where only additions of edges and increase of edge weights happen. The only difference is that the closeness difference matrix should be multiplied by -1. In general, we can first handle all the additions of edges, together with the increase of edge weights, and then handle the deletions of edges and decrease of edge weights. In order to do this, we can add a ghost graph, G'_i, such that $(G'_i - G_{i-1})$ contains all the edges added or with increased weights and $(G_i - G'_i)$ contains all the edges deleted or with decreased weights. The sum of these two closeness difference matrices gives the closeness difference matrix from G_{i-1} to G_i.

4. Spotting Significant Subgraphs

With the closeness difference matrix Π_i^l at time t_i and the vertex importance score vector VI, we now explain how to expand those vertices of high importance scores to obtain significant changing subgraphs. As mentioned, a changing subgraph is significant if the vertex closeness in the subgraph changes a lot. In our experiments, we find that the vertex importance scores follow the power law distribution. Therefore, instead of defining an absolute threshold for the score, we use the value ξ as the threshold such that more than 80% of the scores are smaller than it. Apparently, significant changing subgraphs should contain all the important vertices (i.e., those with high importance scores) and

Table 2. Dataset Characteristics

Datasets	Vertices	Avg. Added Edges	Time Steps
DB	5492	1734	10
DM	5574	1079	10
Enron2001	16639	320	184
Enron2002	16639	203	164

most of the vertices whose closenesses with the important vertices change a lot. We develop an expanding strategy which is similar to the density clustering. The basic idea is to include the vertices whose closeness differences with the important vertices are high.

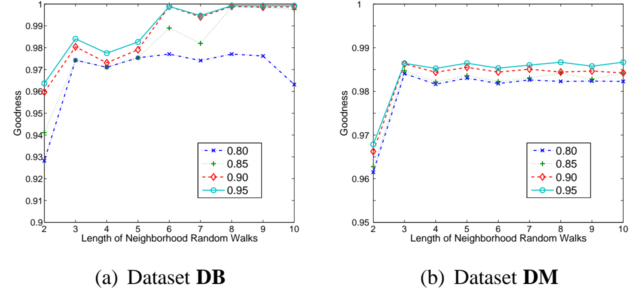
Let the union graph of G_{i-1} and G_i keep the information of connectivity. The algorithm starts from an important vertex v_j with the maximum importance score in each loop to generate a significant changing subgraph. First, it includes v_j into the subgraph. Next, the algorithm adds all the neighbors of v_j into a maximum heap H. It then repeatedly includes a vertex v_k in the heap to the subgraph as long as $\max(\Pi_i(\mathcal{I}(V(g)), k))$ is larger than the current bound ϵ, where ϵ is set to 1/10 of the maximum transition probability of the important vertex lastly included into the subgraph. $\mathcal{I}(V(g))$ is the index of all the vertices in g. When $\max(\Pi_i(\mathcal{I}(V(g)), k))$ is smaller than ϵ, we clear the heap and output the current significant changing subgraph. In the final result set of the significant changing subgraphs, two subgraphs are merged if they are directly connected.

5. Experiments

We evaluate both the effectiveness and efficiency of our proposed algorithms.

5.1. Datasets

The four real datasets are extracted from the DBLP [1] co-authorship dataset and the Enron email dataset [2]. In the DBLP co-authorship dataset, each author is represented by a vertex and there is an edge between two authors if they co-authored some paper. In the Enron email dataset, each email sender or receiver is considered as a vertex and there are edges between senders and receivers. The first two datasets **DB** and **DM** are from the DBLP co-authorship dataset. **DB** contains the co-authorship information of six major database conferences from 1998 to 2007, including SIGMOD, PODS, VLDB, ICDE, EDBT and ICDT. **DM** contains the co-authorship information of five major data mining conferences from 1998 to 2007, including KDD, ICDM, PKDD, SDM and PAKDD. The other two datasets **Enron2001** and **Enron2002** are extracted from the Enron email dataset. **Enron2001** contains the email communication information of each day from 2001-07-01 to 2001-12-31, while **Enron2002** contains the email communication information of each day from 2002-01-01 to 2002-7-31. The characteristics of these datasets are summarized in Table 2.



(a) Dataset **DB** (b) Dataset **DM**

Figure 2. The Goodness

Tong in [10] introduced three aggregation methods: global aggregation, exponential aggregation and sliding window. It is worth noting that our proposed approaches can cooperate with all these three aggregation methods. In this paper, we choose global aggregation to perform our experiments, which aggregates the new edges or edge weights to the adjacency matrix of previous time. Let ΔA_i be the adjacency matrix of the graph at time step t_i, then $A_i = \sum_{t=1}^{i} \Delta A_t$. The average number of added edges per time step is presented in Table 2.

5.2. Effectiveness

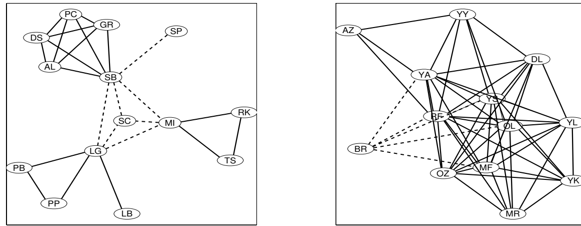
Let us first introduce our criterion of the significant subgraphs. Let g_i denote a significant subgraph found at time t_i. We evaluate the goodness of significant subgraphs as

$$Goodness = \frac{\sum_{v_j, v_k \in g_i} \Delta\Pi_i(j,k)}{\sum_{v_j \in g_i, v_k \in G_i} \Delta\Pi_i(j,k)}, \qquad (10)$$

where $\Delta\Pi_i(j,k) = |\Pi_i(j,k) - \Pi_{i-1}(j,k)|$, is the closeness difference for v_j and v_k between G_{i-1} and G_i. The goodness is essentially the fraction of the closeness differences between G_{i-1} and G_i that are captured by significant subgraphs.

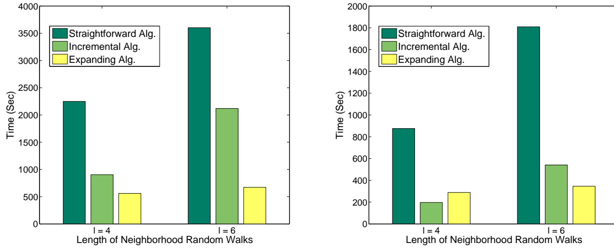
We use $c = 0.15$ in all experiments. Figures 2(a) and 2(b) present the average goodness for different values of ξ, when varying the length of neighborhood random walks l from 2 to 10. For dataset **DB**, our algorithm captures 92% changes in vertex closeness, while for dataset **DM**, our algorithm captures more than 96%. For a higher value of ξ and longer length of l, the goodness scores increase.

Two significant subgraphs found are presented as examples in Figures 3(a) and 3(b), which is from the experiments with $l = 4$ and $\xi = 0.8$. For privacy, we replace author names by abbreviations. The newly added edges are dotted in both subgraphs. Figure 3(a) shows the subgraph from dataset **DB**. There are originally three communities (dense areas) and the newly added edges make three communities connected, which usually indicates that there is a joint



(a) Dataset **DB** (b) Dataset **DM**

Figure 3. Significant Subgraphs



(a) Dataset **Enron2001** (b) Dataset **Enron2002**

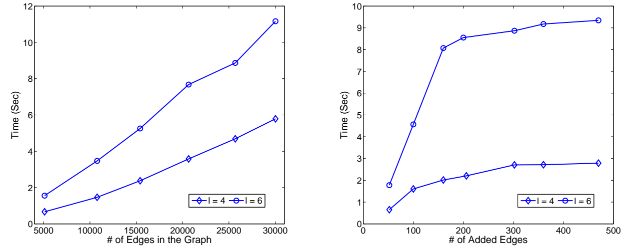
Figure 4. Overall Running Time

research work involving multiple research groups. Apparently, only the subgraph consisting of the added edges cannot provide this information. There are other vertices connecting to some of the vertices in three communities, and these vertices are not included in the significant subgraph because the difference of the vertex closeness between them and the vertex importance scores are small. In Figure 3(b), the researcher BR co-authored papers with researchers in a very dense community. Researchers in the same research group tend to co-author a lot and form a very dense community. Therefore, it is obvious that BR should be a new member to some research group.

5.3. Efficiency

We perform our efficiency testing on datasets **Enron2001** and **Enron2002**. Figures 4(a) and 4(b) show the overall running time for the three algorithms: the straightforward algorithm, the incremental algorithm to compute the vertex importance scores, as well as the expanding algorithm to generate the significant subgraphs. Each figure shows two groups of running time for $l = 4$ and $l = 6$.

Figure 5(a) presents the average running time of the straightforward algorithm versus the average number of edges in the graphs at each time spot, where we can see that the running time of the straightforward algorithm is proportional to the total number of edges in the graph at current time spot. The average running time of the incremental algorithm versus the average number of the newly added edges in the graphs is shown in Figure 5(b). The running time of the incremental algorithm is proportional to the



(a) The Straightforward Algorithm (b) The Incremental Algorithm

Figure 5. Average Running Time

total number of edges added. This explains why the incremental algorithm is faster.

6. Conclusions

We study the challenging problem of spotting significant changing subgraphs in evolving graphs in this paper. We propose to use the neighborhood random walk distance to measure the vertex closeness, as well as a novel incremental algorithm for fast computation. The significant subgraphs are generated based on the vertex importance score. Experimental results show that our approach can solve this problem effectively and efficiently.

7. Acknowledgment

This work was supported by a grant of RGC, Hong Kong SAR, China (No. CUHK419008).

References

[1] http://www.informatik.uni-trier.de/∼ley/db/.
[2] http://www.cs.cmu.edu/∼enron/.
[3] H. Bunke, P. J. Dickinson, M. Kraetzl, and W. D. Wallis. *A Graph-Theoretic Approach to Enterprise Network Dynamics*. Birkhauser, 2006.
[4] C. Faloutsos, K. S. McCurley, and A. Tomkins. Fast discovery of connection subgraphs. In *KDD*. ACM, 2004.
[5] G. Jeh and J. Widom. Simrank: a measure of structural-context similarity. In *KDD*. ACM, 2002.
[6] G. Jeh and J. Widom. Scaling personalized web search. In *WWW*. ACM, 2003.
[7] J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu. Automatic multimedia cross-modal correlation discovery. In *KDD*. ACM, 2004.
[8] B. E. Shakhnovich and J. M. Harvey. Quantifying structure-function uncertainty: A graph theoretical exploration into the origins and limitations of protein annotation. *Journal of Molecular Biology*, 4(337), 2004.
[9] H. Tong, C. Faloutsos, and Y. Koren. Fast direction-aware proximity for graph mining. In *KDD*. ACM, 2007.
[10] H. Tong, S. Papadimitriou, P. S. Yu, and C. Faloutsos. Proximity tracking on time-evolving bipartite graphs. In *SDM*, 2008.