Fires on the Web: Towards Efficient Exploring Historical Web Graphs

Zhenglu Yang [#1], Jeffrey Xu Yu [*2], Zheng Liu [*3], Masaru Kitsuregawa [#4]

[#]Institute of Industrial Science, The University of Tokyo, Japan
[1 4]{yangzl, kitsure}@tkl.iis.u-tokyo.ac.jp
[*]Chinese University of Hongkong, China
[2 3]{yu, zliu}@se.cuhk.edu.hk

Abstract. Discovery of evolving regions in large graphs is an important issue because it is the basis of many applications such as spam websites detection in the Web, community lifecycle exploration in social networks, and so forth. In this paper, we aim to study a new problem, which explores the evolution process between two historic snapshots of an evolving graph. A formal definition of this problem is presented. The evolution process is simulated as a fire propagation scenario based on the Forest Fire Model (FFM) [17]. We propose two efficient solutions to tackle the issue which are grounded on the probabilistic guarantee. The experimental results show that our solutions are efficient with regard to the performance and effective on the well fitness of the major characteristics of evolving graphs.

1 Introduction

Graphs represent the complex structural relationships among objects in various domains in the real world. While these structural relationships are not static, graphs evolve as time goes by. Evolving graphs are usually in the form of a set of graphs at discontinuous time stamps, where the period between two adjacent time stamps may be quite long. Take the Web archive for example. Due to its large size, the Web or a part of it is periodically archived by months or even by years. Mining evolving graphs is important in many applications including the detection of spam websites on the Internet [7], exploration of community lifecycle in social networks [20], and identification of co-evolution relationships between structure and function in bio-informatics [18].

While many of the existing studies have paid attentions to finding stable or changing regions in evolving graphs [1, 19, 10], only a few of them are about how graphs evolve. The researchers have proposed various generative models to capture the statistical properties of the graph evolution such as Power Law distribution [5], effective diameter [21], and so forth. In this paper, however, we study a new problem, which is to model the evolving process between two historical snapshots of an evolving graph. Fig. 1 briefly shows our idea. G is an evolving graph which evolves from time t to t'. Suppose we have the graph snapshots at time t and t', and the real evolution details between these two

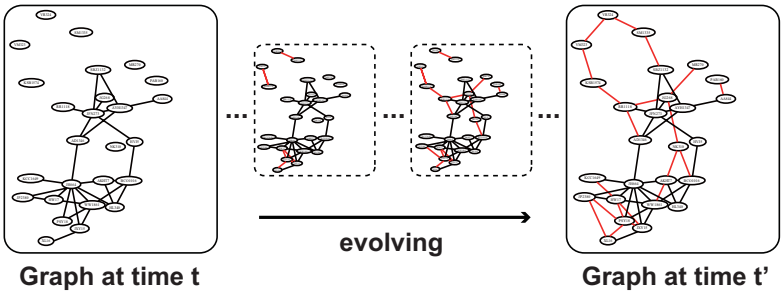


Fig. 1. Example Graphs (All the figures in this paper are made colorful for clarity)

snapshots are unknown. We would like to generate a series of virtual graph snapshots, as shown in the dotted parts in Fig. 1. What is important is that the statistical properties of the evolution must be maintained in these virtual graph snapshots. By doing this, we decompose the macro graph change of the two real snapshots into several micro changes. The benefits are twofold.

Firstly, by parsing the graphs into historical snapshots, we can learn the evolution of the different parts, and thus the future trends of these regions can be predicted. Fig. 2 shows a concrete example, which is conducted on the DBLP dataset[1]. The extracted virtual historical snapshot steps can help us understand the evolution of co-authorship relations. It seems that Web community evolution detection [2] can do the same thing, but the work in this paper is different from that. We aim to detect the changes throughout the whole graph and do not constrain the work on the boundary subgraph (community) that may be defined by the users (i.e., with keywords).

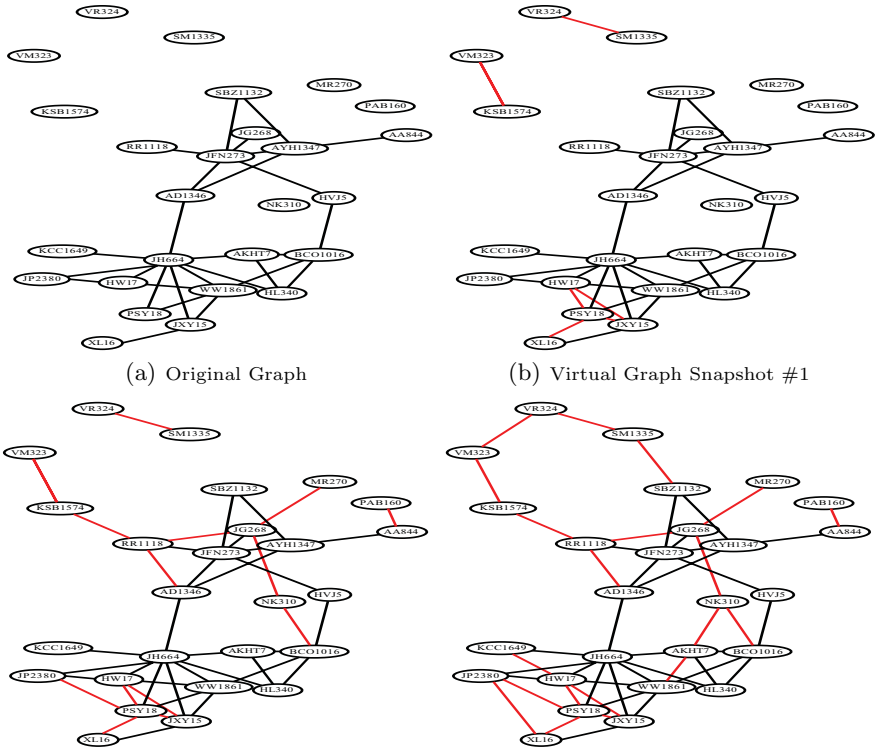
Secondly, successfully tackling the issue proposed in this paper can address the critical issue of the lack of intermediate order-based Web data between two historical Web graphs. Many existing studies on static/dynamic graph mining can profit from restoring these historical graphs such as frequent subgraph discovery [9], temporal graph cluster detection [1], micro view on social networks [13], and so forth. As such, this research work is orthogonal to the existing issues on graph mining in a complementary manner.

For ease of exposition and without loss of generality, we only consider the node/edge insertion scenario in this paper. It should be noted that our approaches can handle the scenario where both the insertion and the deletion of nodes/edges occur. The difficulty in generating these virtual graph snapshots is that there are numerous number of possibilities.

Our approach adopts the Forest Fire Model (FFM) [17], which has been demonstrated as successfully explaining the mechanism of dynamic systems [8]. The new edge linkage action can be thought of as fire propagation, and the nodes on the new edges are the origins of the fire. The virtual historical graph is then to be thought of as the snapshot on tracking how the fire propagates on the whole graph. We will give the formal definition of the problem shortly.

The contributions of our paper are as follows:

[1] www.informatik.uni-trier.de/~ley/db



(a) Original Graph (b) Virtual Graph Snapshot #1

(c) Virtual Graph Snapshot #2 (d) Existing Graph

Fig. 2. Expected Graph Evolution Process

- We propose a new problem of how to trace back the virtual snapshots of evolving graphs. The process can be simulated based on the FFM. The historical graph is deemed as a snapshot of tracking the fire propagation situation on the graph.
- We propose two approaches, *bottom-up* and *leap-search*. The *bottom-up* strategy examines the candidates from scratch in a global view, while the *lead-search* method applies a density-oriented candidate selection mechanism. We also explore the heuristics based on the properties of evolving graphs to improve the efficiency of the two approaches.
- We conduct comprehensive experiments on real large evolving graphs. The evaluation results demonstrate the effectiveness and efficiency of the proposed solutions.

The remainder of this paper is organized as follows. We introduce the preliminaries in Section 2. The bottom-up and leap-search solutions are presented in Sections 3 and 4, respectively. Section 5 reports the experimental results and Section 6 concludes the paper.

Notation	Definition
G	An evolving graph or a graph adjacent matrix
G_t	An evolving graph at time t
V_t	The set of vertices in the graph G_t
E_t	The set of edges in the graph G_t
$\|V_t\|$	Number of vertices in the graph G_t
$\|E_t\|$	Number of edges in the graph G_t
i, j, k	Vertices in a graph
e= (i, j)	Edges in a graph
$deg(u)$	Degree of vertex u
$v_{i,j}$	Fire propagation velocity between vertices i and j
$t_{i,j}$	Fire propagation time between vertices i and j
$d_{i,j}$	Distance or length between vertices i and j
c	Backward burning probability

Table 1. The summary of notation

2 Preliminary

In this paper, we deal with undirected evolving graphs. Let G denote an evolving graph. A snapshot of the graph G at time t is represented as $G_t = (V_t, E_t)$, where V_t is the set of vertices at time t; and $E_t \subseteq V_t \times V_t$ is the set of edges at time t. The notations used in this paper is summarized in Table 2. To trace back the historical snapshots of an evolving graph, we will introduce novel strategies based on the FFM [17].

2.1 Forest Fire Model

The FFM was first proposed in ecology [17], where the scholars were interested in how to control and predict wildfire in the real world environment. Henley [8] first introduced the FFM in studying the characteristics of self-organized systems. From then on, the FFM has succeeded in explaining many real dynamical systems such as Geographic Information Systems (GIS) [6], Affiliation Network [12], arXiv citation [12], and so forth. Most especially, the FFM fits in many properties of real Web data we would like to explore in this paper: (1) the rich get richer, which is called the attachment process (heavy-tailed degrees) [3]; (2) leads to community structure [11]; (3) densification [12]; and (4) shrinking effective diameter.

Note that the FFM is related to random walk and electric currents [4] with regard to the issue of evolving graphs. The difference is that the latter two have not taken all the aforementioned four properties of real Web graph into account. As far as we know, there is only one work [12] on studying evolving Web graph based on the FFM. There are three main differences between this work and [12]: (1) [12] aims to generate synthetic evolving graphs from scratch, while in this work we trace back the historical snapshots between two real graphs; (2) [12] randomly selects the initial fired nodes, while in this work we deliberately choose the initial fired nodes with probabilistic guarantee; and (3) we introduce the fire propagation velocity into our framework, while [12] does not consider this property. These distinct issues are due to the different purposes of the two works; [12] intended to generate synthetic evolving graph from scratch on the fly, while we aim to study the whole history of how an old graph evolves to a

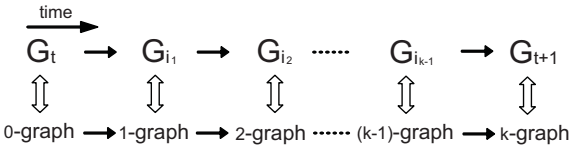


Fig. 3. Graph evolving process

newer graph. By parsing the evolving process into virtual historical snapshots, we can learn the micro evolution of the different regions in a large graph to make a navigational prediction on the evolving trend of the graph.

2.2 Discovery of the Historical Snapshot Graph Problem

Given two graphs, G_t and G_{t+1}, at time t and time $t+1$, the problem of discovering the historical graph snapshots involves tracing back the virtual graphs after inserting n new edges[2], where $1 \leq n \leq (|E(G_{t+1})| - |E(G_t)|) = k$, into the old graph G_t. Let *n-graph* denote the graph snapshot which has n new edges. Fig. 3 illustrates the evolving process of the graph. G_t and G_{t+1} can be mapped at 0-*graph* and *k-graph*, respectively. **The issue of discovering the historical graph snapshots is then equivalent to finding the n-graphs**, where $0 < n < k$ and $(|E(G_{t+1})| - |E(G_t)|) = k$.

For example, Fig. 2 (b)-(c) are the graph snapshots of Fig. 2 (a) by inserting 6 and 14 new edges, denoted as 6-*graph* and 14-*graph*, respectively. The new edge linkage action can be thought of as fire propagation, and the nodes on the new edges are the origins of the fire. The virtual historical graph is then thought of as the snapshot on tracking how the fire propagates on the whole graph. In this paper, we introduce how to set the initial fire energy and how fast fire propagates on the graph. As far as we know, this work is the first one to study these issues. **The problem of finding the n-graphs is then equivalent to discovering the burning out n-graphs**, which is formally defined in Definition 1.

Definition 1 Burning Out n-Graph (BOG) Problem
Given two undirected graphs $G_t = (V_t, E_t)$ and $G_{t+1} = (V_{t+1}, E_{t+1})$; a cost function $CF(i,j)$ on edge (i,j) where $i \in V_t \cup V_{t+1}$ and $j \in V_t \cup V_{t+1}$; a user preferred number n of new edges, find the subgraph H of $G_t \cup G_{t+1}$ such that

- *The number of new edges on H is n, and*
- *$\sum_{(i,j)\in E(H)} CF(i,j)$ is minimized.*

$E(H)$ is the edge set of graph H. The cost function CF can be considered as the time necessary to construct the subgraph H (or burning out it), as will be well defined shortly with the help of the FFM [17].

[2] New vertices are accompanied with new edges.

2.3 Discovery of the Burning Out n-Graph (BOG) Problem

We aim to address the issue of finding the first burning out n-graphs, which is equivalent to extracting the historical graph snapshots. In this section, we introduce the basic configuration of the fire model such as the initial fire energy, the velocity and time for fire propagation, and the update of the fire energy.

Initial fire energy: Consider an FFM, where the fires are caused by those changing edges (with vertex insertion), and each changing edge introduces one unit fire energy. The initial fire energy of a vertex is the accumulated energy of all its adjacent new edges.

Fire propagation: The cost function $CF(i,j)$ introduced in Section 2.2 is defined as the time $t_{i,j}$ of the fire propagation consumed on the edge between the two adjacent vertices i and j. We have

$$CF(i,j) = t_{i,j} = \frac{d_{i,j}}{\bar{v}_{i,j} + \bar{v}_{j,i}}, \tag{1}$$

where $d_{i,j}$ denotes the distance between the two adjacent nodes (i and j), and $\bar{v}_{i,j}$ and $\bar{v}_{j,i}$ denote the average fire propagation velocity from vertice i to j, and vice versa, respectively. It is interesting to note that the fire propagation has direction, which conforms to the common intuition that the effects of changing edges/vertices spread from the origins to the distant edges/vertices[3]. The average velocity $\bar{v}_{i,j}$ is mainly dependent on the initial burning energy[4] ℓ_i. Hence, simplify the model without loss of generality, we have $\bar{v}_{i,j} = \gamma_i * \ell_i$, where γ_i is a constant. In this paper, we assume γ_i=γ_j=...=1, and Eq. (1) can be deduced as

$$CF(i,j) = t_{i,j} = \frac{d_{i,j}}{\ell_i + \ell_j}. \tag{2}$$

Fire energy update: The fire energy ℓ should be updated after each successful propagation by using the following equation.

$$\ell_{j_{t'}} = \ell_{j_t} + \sum_{i \in Neigh_{suc}(j)} (1-c)\ \ell_{i \to j}, \tag{3}$$

where ℓ_{j_t} and $\ell_{j_{t'}}$ are the fire energy of j at time t and time t' respectively, c is a *backward burning probability* [12], and $\ell_{i \to j}$ is the fire energy transferred from i to j. Thus we have $\ell_{i \to j} = \ell_i / deg(i)$. $Neigh_{suc}(j)$ denotes the neighbour nodes of j that successfully transfer their fire energy to j between time t and time t'. Note that once a node successfully propagates fires to its neighbours, its own fire energy is reset to zero at the time.

[3] This mechanism will help to address the issue on directed graphs, which however, is out scope of this paper.

[4] In a real ecological environment, other effects such as wind, topography slope, and so forth, should also be taken into account.

Algorithm 1: Bottom-up algorithm

Input: The graph G_{i-1} and G_i at time t_{i-1} and t_i, a user preferred number n
Output: The fastest burning out n-graph(s)

```
1  H = G_{i-1} ∪ G_i;
2  for each vertex v_i ∈ H do //initial fire energy
3     fe[v_i] =num_of_adjacent_changing_edges;
4     can_graph_list=store v_i as a graph;
5  num_of_new_edge=0;
6  while num_of_new_edge < n do
7     for each graph g ∈ can_graph_list do
8        g'= appending g with new edge e of minimal spreading time;
9        t_local=fire propagation time on e;
10       if t_local < t_min_local do
11          t_min_local=t_local;
12          update g' in can_graph_list;
13          update fe[v_i] if new edge introduces a new vertex v_i;
14    num_of_new_edge++;
15 output the n-graphs with minimal fire propagation time;
```

In this paper, we propose two approaches to discover the fastest burning out n-graphs. The *bottom-up* approach examines the candidates from scratch in a global view with the dynamic threshold guaranteed, while the *leap-search* approach proposes a density-oriented candidate selection strategy.

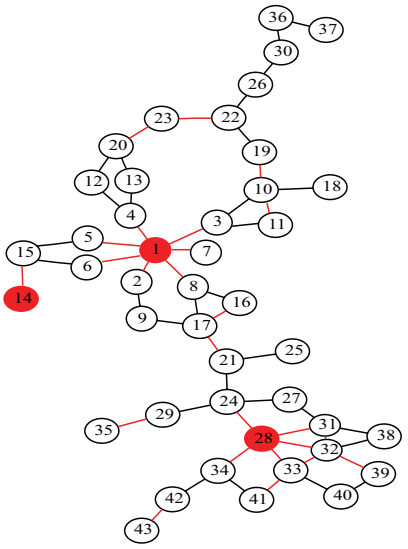
3 The Bottom-Up Approach

We develop a bottom-up greedy algorithm to extract the burning out n-graphs. The algorithm follows the candidate generation and verification iteration. To accelerate the process, the threshold-based technique is proposed to prune the candidates early.

Candidate generation: In each iteration, a k-graph g is grown up to a candidate (k+1)-graph g' by introducing a new edge e_{new}, where the following conditions hold: (1) e_{new} is connected to some vertices in g; and (2) the fire spreading time t from g to v_{new} is minimized[5].
Verification: If the burning out time of the candidate (k+1)-graph g' is greater by far than the best one, we turn to the next candidate subgraph. Otherwise, we update the fire energy of the new vertex, which is introduced by the new edge based on Eq. 3, and continually grow g' to a larger candidate graph with another candiate-generation-and-test iteration.

The basic bottom-up greedy algorithm is shown in Algorithm 1. We will introduce the pruning techniques in the next section. In the initial phase (line 1-5), the two graphs are joined. The changing edges of each vertex are counted while joining, stored in an array fe. The vertex is put into the candidate list as a graph. The candidate generation and test iteration is from line 6 to line 14.

[5] Specifically, t is computed based on Eq. 2. By default, $d_{i,j}$ is set to 1 for the unweighted graph in this paper. For the weighted graph, $d_{i,j}$ can be set according to the weight of the edge.



(a) Example graph data
(red color indicates insertion/removal edges)

# new edges	New edges	Virutal time$_{min}$
0	V_1	0
1	V_{28} — V_{32}	$\frac{1}{9}$
2 − 9	V_{28} — V_{33}; V_8, V_7, V_2, V_1, V_6, V_3, V_5, V_4	$\frac{1}{8}$
10 − 11	V_{32} — V_{33}; V_{28} — V_{31}	$\frac{1}{7}$
12 − 14	V_{28}, V_{24}, V_{34}, V_{31} — V_{32}	$\frac{1}{6}$

(b) Sequential inserted edges

Fig. 4. Example graph

Given a k-graph g, we generate its candidate (k+1)-graph, by finding a nearest edge to g, which may come from its nearest neighbor vertex or internal unlinked edge. The fire propagation time is computed based on Eq. 3. If the time is smaller than the least time of burning out graph time so far, we update the new graph information (line 12) and also update the fire energy of a new vertex if it exists (line 13). The iteration will be completed when the burning out n-graph is found.

3.1 Implementation Details

Pruning Techniques. We can prune many candidate graphs early based on the threshold-based technique. The threshold time t_{th} is dynamically updated based on the most optimal graph by far. Before a candidate graph g grows up to a larger candidate one, if we find the construction (burning out) time t_g of g is already greater than t_{th}, this round can be terminated and continued to the next round. The reason why we jump to the next round instead of the remaining candidates is that we can rank all the candidate graphs based on their burning out time in ascending order (by using minimal heap). If t_g is greater than t_{th}, then the time of all the remaining candidates (in the heap) should be greater than t_{th}; hence, this round can be safely terminated. The early pruning rule is justified based on the following lemma.

Lemma 1 (Anti-monotone) *Let g be a graph with k edges and g' be a connected supergraph of g with (k+1) edges. The burning out times of g and g' are t and t', respectively. Let t_{th} be a threshold time. If $t > t_{th}$, then $t' > t_{th}$.*

Algorithm 2: Pruning strategy for the bottom-up algorithm

```
1  sort fe in descending order;
2  min_heap ← store v_j ∈ fe with maximal value as a subgraph;
3  while min_heap is not empty do
4      g=subgraph with minimal value t_g in min_heap;
5      if #new edge(g) ≥ n do
6          break;
7      g'= appending g with edge of minimal spreading time;
8      t_local=fires propagation time of the new edge;
9      min_heap ← store g' with t_local_max;
10     update fe[v_i] if new edge introduces a new vertex v_i;
11     v_j=unvisited node in fe whose energy is the largest;
12     for each neighbor v_k of v_j do
13         if t(v_k, v_j) < t_local do
14             min_heap ← store v_j as a subgraph;
15             break;
```

Proof Sketch. (By contradiction) Suppose we have $t' \leq t_{th}$. We reduce g' to g'' by removing the edge not existing in g. Let t'' be the burning out time of g''. We have $t'' \leq t_{th}$. As t'' is equal to t, it results in $t \leq t_{th}$, which is contradictory to the assumption; thus the lemma holds. □

Lemma 1 can efficiently prune many candidate subgraphs, as will be demonstrated in the experimental results. The reason for this is that due to the Densification Power Law [12] property, the threshold of the most optimal subgraph (a "rich" one) by far will have high probability greater than most of the other subgraphs ("poor" ones); thus, the latter ones can be pruned earlier without testing. The optimized algorithm is shown in Algorithm 2, which replaces line 5-14 in Algorithm 1. The algorithm is self-explanatory, and we provide a concrete example to illustrate the process.

Example 1. Suppose we want to determine the snapshot with an insertion of 14 new edges. The graph in Fig. 4 (a) is our example graph, where the red lines indicate the changing edges. We first scan the graph to accumulate the number of the changing adjacent edges of each vertex and sort them. Therefore, we obtain a list $V_1 : 7, V_{28} : 5, V_{32} : 4, V_{33} : 3, \ldots$. Note that we only record the vertex which has at least one changing adjacent edge. Next, we push V_1 into the heap (because it has the largest initial fire energy and may propagate the fire faster). We traverse the adjacent vertices of V_1 and compute the time that the fire can be spread to the nearest neighbor, resulting in 1/8 unit time (w.r.t Eq. 2, where $\ell_{V_1} = 7$ and $\ell_{V_2} = 1$). Note that there are multiple nearest neighbors, e.g., V_2, V_3, V_4, etc. The new subgraph is pushed into the heap, and the fire energy of each vertex involved is updated (w.r.t Eq. 3). We also push the node V_{28} into the heap because its initial energy is the largest among the unvisited nodes. Recursively, we execute the process until we find that the total number of new edges is greater than or equal to the threshold (i.e., 14). The results are listed in Fig. 4 (b). Note that the historical snapshot should be the union of graph G' of the new edges g_n with the old graph G_t, where G'=$G_t \cup g_n$.

Algorithm 3: Leap-search algorithm

Input: The graph G_{i-1} and G_i at time t_{i-1} and t_i, a user preferred number n
Output: The fastest burning out n-subgraph(s)

```
1  H = G_{i-1} ∪ G_i;
2  for each vertex v_i ∈ H do //initial fire energy
3     fe[v_i] =num_of_adjacent_changing_edges;
4  sort fe in descending order;
5  for each v_j ∈ fe do //generate candidate core subgraphs
6     g=store v_j with energy value as a subgraph;
7     while #new edge(g) < n do
8        g'= appending g with edge of minimal spreading time;
9        t_local=fires propagation time of the new edge;
10    t_local_max=maximal value of t_local;
11    cand_graph_list=store g' with t_local_max;
12 for each subgraph g ∈ cand_graph_list do //test the candidate graph
      //update g if possible
13    for each edge e(i, j) ∈ g do
14       update fe[i] and fe[j] by checking neighbors of i and j;
15       update fire propagation time of e(i, j);
16    update least time t_least necessary to fire out g;
17 Output the n-graphs(s) with minimal fire propagation time;
```

4 The Leap Search Approach

In this section, we present a leap-search based method for the extraction of burning out n-graphs. The method is efficient in processing graphs with a large n, where growing up the candidate subgraphs from scratch by using bottom-up growth can be time consuming. The approach is based on the density-oriented candidate selection strategy. The intuitive idea is that fire transfers fast in those regions where the energy density is high. The extraction process is also composed of candidate generation and verification iteration.

Candidate generation: Starting from the nodes with the most fire energy, we grow them by selecting their nearest neighbors (w.r.t. the fire propagation time) until the number of the new edges in the subgraph graph is equal to n. In other words, we do not wait for other possible candidates to grow up. During the growing process, we record the least time necessary to transfer the fire.
Verification: We check the bottleneck nodes of the fire propagation in these subgraphs (as indicated by the least time), greedily find the neighbors which can remedy the weak edges on spreading the fire, and then update the least time value. Through a recursive process, we finally determine the first burning out regions with the least n new edges.

The leap-search algorithm is shown in Algorithm 3. The initial phase (line 1-4) is similar to that of Algorithm 1. The candidate core subgraphs are generated first (line 5-11). Starting from the nodes with most fire energy, we grow them by linking to their nearest neighbors (w.r.t. the fire propagation time) or their internal unlinked edges (line 8) until the number of the new edges in the subgraph graph is equal to or greater than n. Different from the bottom-up algorithm, we do not wait for other possible candidates to grow up. During the growing process, we record the least time necessary to transfer the fire among them (line 9). In

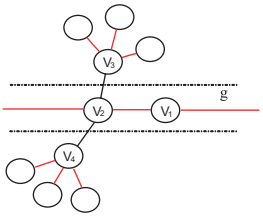


Fig. 5. Update mechanism for weak links (red line indicates the edge has changed and this figure is best viewed in color)

Algorithm 4: Pruning strategy for the leap-search algorithm

```
1  t_th=by far min time to burn out a n-candidate graph;
2  max_heap=candidate graphs with fire propagation time t;
3  while max_heap is not empty do
4      g=subgraph with maximal value in max_heap;
5      for all edges of g
6          e(i, j)=g's unvisited slowest edge on propagating fire;
7          update fe[i] and fe[j] by checking neighbors of i and j;
8          update fire propagation time t_{e_{i,j}} of e(i, j);
9          if(t_{e_{i,j}} > t_th)
10             break;
```

the candidate test phase (line 12-16), we check the bottleneck nodes of the fire propagation in these subgraphs (as indicated by the least time), greedily find the neighbors which can remedy the weak edges on spreading the fire (line 14-15), and then update the least time value[6] (line 16). The detail of the updating mechanism will be described shortly. Finally, we determine the first burning out regions with the least n new edges.

4.1 Implementation Details

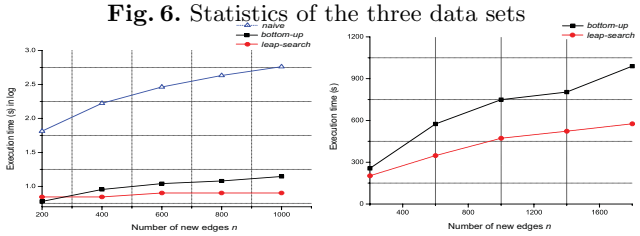
Weak Link Updating Mechanism. We introduce how to update the fire propagation time of the weak links. Suppose we have a subgraph as shown in Fig. 5. Nodes V_1 and V_2 have been included in the candidate graph g, but nodes V_3 and V_4 are outside of g. If we know that edge $e(V_1, V_2)$ is a weak edge of g (i.e., the fire propagation time is slow), then we start from nodes V_1 and V_2, and check whether their neighbors can transfer fire energy to them. For this example, node V_2 has two neighbors, V_3 and V_4, with a large fire energy and can propagate fire to V_2. Therefore, we update $fe[V_2]$ (line 14 in Algorithm 3) by using Eq. 3. The fire spreading time of $e(V_1, V_2)$ is also updated (line 15).

Pruning Strategy. When testing the candidate graphs (line 12-16 in Algorithm 3), we can prune many candidates early by using a time threshold, t_{th}, which is by far the fastest time to burn out an n-graph. Given a candidate graph, if after we update it by using the mechanism introduced in the last section the burning time is still smaller than t_{th}, then we can safely prune this candidate graph. To further improve the efficiency, the slowest edges on propagating fires in candi-

[6] The replacement should guarantee the number of new edge will not decrease.

Data set name	DBLP		Enron		Web	
Classifed type	old	new	old	new	old	new
Recorded time	2001	2007	2001-10-1	2001-12-31	2004-5	2005-7
Nodes	1849	5289	6310	10008	2446029	3078826
Edges	4732	16667	30637	67777	57312778	71400464

Fig. 6. Statistics of the three data sets



(a) **Enron** Dataset (b) **Web** Dataset

Fig. 7. Efficiency of the proposed solutions

date graphs are tested first. The optimized algorithm is shown in Algorithm 4.

5 Performance Evaluation

To evaluate our strategies, we conducted extensive experiments. We performed the experiments using a Itanium2 CPU (1.5GHz) server with a 128G memory[7], running Redhat linux. All the algorithms were written in C++. We conducted experiments on three real life datasets, *DBLP*, *Enron*, *Web*.

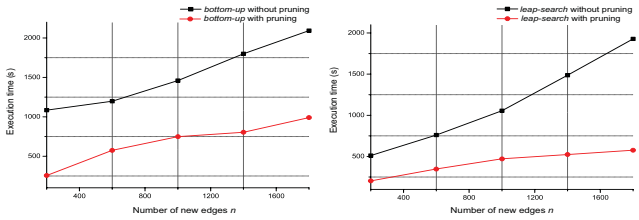
The first dataset, *DBLP*, is extracted from DBLP website[8] and focuses on the bibliography information from database community. It contains the co-authorship information of major database conferences from 2001 to 2007. The second dataset, *Enron*, records the email communication information of each day from 2001-10-01 to 2001-12-31. For detail of these two datasets refer [15]. The third dataset, *Web*, records two snapshots of Japanese web pages (in jp domain) in May 2004 and July 2005, respectively. Part of this dataset is reported in [22]. The basic statistics of the datasets are shown in Fig. 6. Refer [23] for more experimental results.

5.1 Efficiency of our solutions

We compare our two algorithms, *bottom-up* and *leap-search*, with a naive method [23]. The result is shown in Fig. 7. Note that the execution time on the *Enron* dataset is in logarithm format. We can see that our solutions are much faster than the naive one, about one to two orders of magnitude (as shown in Fig. 7 (a)). For the huge Web dataset, the naive algorithm can not finish in reasonable time. Between our two approaches, the *leap-search* performs better than the *bottom-up* when the number of new edges n becomes larger. The reason is that for large value of n, growing graphs from scratch by evaluating all the neighbors

[7] The actual used memory is smaller than 8G.

[8] www.informatik.uni-trier.de/∼ley/db



(a) **Web** Dataset (b) **Web** Dataset

Fig. 8. Pruning efficiency of the proposed strategies

is time consuming. However, for a small value of n, the cost for growing graphs becomes smaller and the cost for updating weak links becomes larger; thus, the *bottom-up* algorithm performs better.

5.2 Pruning Efficiency of our solutions

In this section, we evaluate the efficiency of the proposed pruning techniques. The result is illustrated in Fig. 8. We can see that with pruning techniques, the *bottom-up* algorithm can perform much better, as shown in Fig. 8 (a). The reason is that many of the candidate graphs can be pruned sharply with the anti-monotone rule and the refinement strategy. For the *leap-search* approach, the early pruning technique can remove many candidate graphs from the heap. Thus, the overall performance is improved. In summary, with the pruning techniques, both algorithms only need to test and update a small number of candidate graphs, which lead to good scalability with respect to the cardinality of the datasets and the value of n.

5.3 Effectiveness of our solutions

We examine whether the discovered virtual historical snapshots restore the real data with high precision and follow the important properties of evolving graphs.

– **Precision.** We compare three algorithms, *random*, *bottom-up* and *leap-search* on the precision metric[9]. The *random* method is implemented by randomly selecting n new edges combined with the old graph as the virtual snapshot. The quantitative metric is defined as

$$precision = \frac{|\Delta E_R \cap \Delta E_V|}{|\Delta E_R|}, \tag{4}$$

where ΔE_R denotes the set of actual new edges and ΔE_V represents the set of virtual generated new edges. Our methods can get rather high precision as illustrated in the figure. The reason why the precision decreases on restoring the older snapshots (i.e., 10/15), is due to the small number of the new edges, which leads to difficulty in locating the new edges. For our two algorithms, there is a trade-off between efficiency and effectiveness.

[9] Due to its prohibitive cost on execution compared with others, the *naive* algorithm has not been considered here.

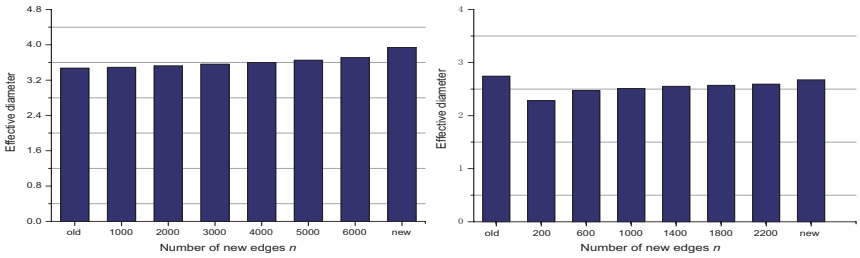
(a) Precision on **Enron** dataset

Date=	10/15	11/1	11/15	12/1	12/15
random	0.180	0.545	0.699	0.767	0.816
bottom-up	0.620	0.743	0.858	0.911	0.935
leap-search	0.517	0.654	0.779	0.863	0.907

(b) Precision on **DBLP** dataset

Year=	2002	2003	2004	2005	2006
random	0.129	0.271	0.415	0.622	0.731
bottom-up	0.546	0.644	0.700	0.773	0.902
leap-search	0.414	0.526	0.611	0.725	0.846

Table 2. Precision evaluation



(a) Effective diameter of **Enron** dataset (b) Effective diameter of **DBLP** dataset

Fig. 9. Effective diameter

- **Effective Diameter.** Fig. 9 shows the values of effective diameters [21] for the historical snapshots. We can see that the virtual snapshots mainly express the transition characteristic between the two real graphs. In Fig. 9 (b), the effective diameter drops after inserting a few edges. We argue that the reason is due to the relative small community of the DB scholars. The first few new edges may link to many others because these insertion edges may be caused by those influential people (with more initial fire energy).
- **Degree Distribution.** We also evaluate the degree distributions of the virtual historical snapshots. The temporal degree distribution follows the power law distribution and the changing edge degree distribution obeys the densification power law distribution. Refer [23] for detail.

6 Conclusion

In this paper we have studied a new problem of tracing back the virtual historical snapshots. Two solutions have been proposed, the *bottom-up* and the *leap-search*. We have conducted extensive experiments and the results show that our approaches can restore the historical graph snapshots efficiently while maintaining the evolution properties. In the future, we will evaluate some other predictors such as those proposed in [14, 16].

Acknowledgements

The work was done when the first author visited the Chinese University of Hongkong. The work was supported by grants of the Research Grants Council of the Hong Kong SAR, China No. 419008 and 419109.

References

1. N. Bansal, F. Chiang, N. Koudas, and F. W. Tompa, "Seeking stable clusters in the blogosphere," in *VLDB*, 2007.

2. R. Breiger, K. Carley, and P. Pattison, "Dynamic social network modeling and analysis: Workshop summary and papers," 2003.
3. A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener, "Graph structure in the web," *Comput. Netw.*, vol. 33, no. 1-6, 2000.
4. P. G. Doyle and J. L. Snell, "Random walks and electric networks," *Carus Mathematical Monographs, Mathematical Association of America*, vol. 22, 1984.
5. M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *SIGCOMM*, 1999.
6. P. Goncalves and P. Diogo, "Geographic information systems and cellular automata: A new approach to forest fire simulation," in *EGIS*, 1994.
7. Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen, "Combating web spam with trustrank," in *VLDB*, 2004.
8. C. Henley, "Self-organized percolation: A simpler model," *Bull. Am. Phys. Soc.*, vol. 34, 1989.
9. A. Inokuchi, T. Washio, and H. Motoda, "An apriori-based algorithm for mining frequent substructures from graph data," in *PKDD*, 2000.
10. A. Inokuchi and T. Washio, "A fast method to mine frequent subsequences from graph sequence data," in *ICDM*, 2008.
11. R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal, "Stochastic models for the web graph," in *FOCS*, 2000.
12. J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: densification laws, shrinking diameters and possible explanations," in *KDD*, 2005.
13. J. Leskovec, L. Backstrom, R. Kumar, and A. Tomkins, "Microscopic evolution of social networks," in *KDD*, 2008.
14. D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *JASIST*, vol. 58, no. 7, 2007.
15. Z. Liu, J. X. Yu, Y. Ke, X. Lin, and L. Chen, "Spotting significant changing subgraphs in evolving graphs," in *ICDM*, 2008.
16. M. E. J. Newman, "The structure and function of complex networks," *SIAM Review*, vol. 45, no. 2, 2003.
17. R. C. Rothermel, "A mathematical model for predicting fire spread in wildland fuels," USDA Forest Service, Ogden, UT, Tech. Rep., 1972.
18. B. E. Shakhnovich and J. M. Harvey, "Quantifying structure-function uncertainty: a graph theoretical exploration into the origins and limitations of protein annotation," *Jounal of Molecular Biology*, vol. 4, no. 337, 2004.
19. J. Sun, C. Faloutsos, S. Papadimitriou, and P. S. Yu, "Graphscope: parameter-free mining of large time-evolving graphs," in *KDD*, 2007.
20. C. Tantipathananandh, T. Berger-Wolf, and D. Kempe, "A framework for community identification in dynamic social networks," in *KDD*, 2007.
21. S. Tauro, C. Palmer, G. Siganos, and M. Faloutsos, "A simple conceptual model for the Internet topology," in *GLOBECOM*, 2001.
22. M. Toyoda and M. Kitsuregawa, "What's really new on the web? identifying new pages from a series of unstable web snapshots," in *WWW*, 2006.
23. Z. Yang, J. X. Yu, Z. Liu, and M. Kitsuregawa, "Fires on the Web: Towards Efficient Exploring Historical Web Graphs," University of Tokyo, Tech. Rep., 2009.