Keyword-based Correlated Network Computation over Large Social Media

Jianxin Li[1], Chengfei Liu[2], Md. Saiful Islam[3]

Swinburne University of Technology
Melbourne, Australia
{[1]jianxinli, [2]cliu, [3]mdsaifulislam}@swin.edu.au

***Abstract*—Recent years have witnessed an unprecedented proliferation of social media, e.g., millions of blog posts, micro-blog posts, and social networks on the Internet. This kind of social media data can be modeled in a large graph where nodes represent the entities and edges represent relationships between entities of the social media. Discovering keyword-based correlated networks of these large graphs is an important primitive in data analysis, from which users can pay more attention about their concerned information in the large graph.**

In this paper, we propose and define the problem of keyword-based correlated network computation over a massive graph. To do this, we first present a novel tree data structure that only maintains the shortest path of any two graph nodes, by which the massive graph can be equivalently transformed into a tree data structure for addressing our proposed problem. After that, we design efficient algorithms to build the transformed tree data structure from a graph offline and compute the γ-bounded keyword matched subgraphs based on the pre-built tree data structure on the fly. To further improve the efficiency, we propose weighted shingle-based approximation approaches to measure the correlation among a large number of γ-bounded keyword matched subgraphs. At last, we develop a merge-sort based approach to efficiently generate the correlated networks. Our extensive experiments demonstrate the efficiency of our algorithms on reducing time and space cost. The experimental results also justify the effectiveness of our method in discovering correlated networks from three real datasets.

***Index Terms*—Social Media, Correlated Networks, Keyword Query, Large Graph**

I. Introduction

Recent years have seen an astounding growth of networks in a wide spectrum of application domains, ranging from sensor and communication networks to biological and social networks. And it becomes especially apparent as far as the great surge of popularity for Web 2.0 applications is concerned, such as Facebook, LinkedIn, Twitter and Foursquare. Typically, these networks can be modeled as large graphs with nodes representing entities and edges depicting relationship between entities [1]. To retrieve interesting information from large graphs, users often type in keywords as a request, which is known as *keyword search over graph data*. The problem of keyword search over graph data has been studied extensively, e.g., [2], [3], [4], [5], [6], [7], [8]. Most of existing works return top-k or all minimally matched subgraphs to the users. However, sometimes users not only want to see the individually matched subgraphs, but also expect to see bigger pictures consisting of multiple individual results with high correlations among them. This finds many applications that need to identify a dense part of a large network such as social media according to a specific topic defined by a set of keywords. For example, new events often happen every day in the world, which would lead to lots of discussions from related web pages/blogs or in social networks on the Internet. As another example, companies often launch advertisements before bringing their new products into market. The launched advertisements also receive comments from thousands of micro-blogs or persons on social media. Discovering such correlated networks among these related web pages/blogs or in social networks is helpful to analyze the influence, consequence and scope of the new events or products related to the given keywords. Therefore, in this paper we define and study the problem of so-called *keyword-based correlated network computation* over a large graph.

To do this, we can apply the dense subgraph discovering metrics to model our problem of keyword-based correlated network computation over a large graph. That is to say, the density of grouping the components into correlated networks can be utilized to evaluate the correlation of the components where each component is a full keyword matched subgraph and the maximal distance of any two keyword nodes in each component is bounded by a user-specified value γ. Dense subgraph discovering techniques have been studied in [9], [10], [11], [12], [13], [14], [15], [16], [17]. However, all these works concentrate their research on the efficiency of discovering the densest subgraph, or the top-k dense subgraphs from a graph. There is no existing work to allow users to find their interested dense subgraphs with a search request (e.g., a keyword query).

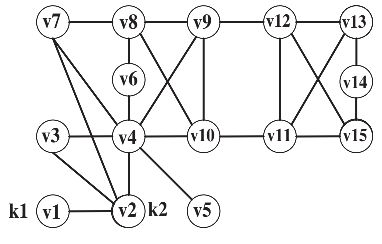


Fig. 1. An Example of Graph Data G

Authorized licensed use limited to: Griffith University. Downloaded on August 25,2020 at 06:00:36 UTC from IEEE Xplore. Restrictions apply.

Example 1: Figure 1 provides a part of large social network where the graph nodes may represent persons, or micro-blog posts in social network, and the graph edges may represent friendships of persons, or the communication relationships of these micro-blog posts over the social network. In addition, the keywords at the side of the nodes may represent the personal description information or the contents of the shared stories to be published in the micro-blog posts. Assume a user would like to see the correlated networks with the topic related to the set of keywords $\{k1, k2, k3\}$. To control the size of each component (subgraph) in the discovered network, the user can specify a parameter γ to bound the maximal distance of any two keyword nodes in each component. As such, our problem is to find a set of correlated networks where each network consists of multiple components (subgraphs) with high correlation above a threshold and each γ-bounded component of the network must contain the full keywords.

While there are lots of studies on searching subgraphs based on keywords, and efficiently discovering dense subgraphs, no existing work is available to take both aspects into account in the network analysis scenario. It is a big challenge to discover correlated networks with the consideration of the given keywords over a graph, especially over a large graph.

An easy way is to first compute the dense subgraphs using the conventional dense subgraph discovering techniques and then filter the dense subgraphs by checking if they contain the full keywords. For instance, we can compute the densest subgraph by using a representative approach in [11]. But often the densest subgraph does not contain the full keywords. It has to try the next densest subgraph until finding the right one with the full keywords or probing all the nodes in the graph. At one moment, even if we find a dense subgraph that contains the full keywords, we have to check the remaining nodes by repeating the above operation until all possible dense subgraphs with the density above a threshold have been completely identified. Figure 2 shows 3 densest subgraphs in the graph shown in Figure 1 by applying for the conventional dense subgraph discovering approaches. "or" in Figure 2 means only one of the first two subgraphs can be generated because a node can only appear in one densest subgraph based on [11]. It is obvious that they are false candidates because they do not contain the full keywords. Therefore, it has to repeatedly try the other subgraphs with less densities.

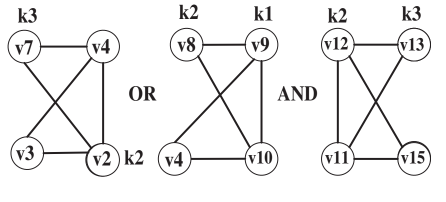


Fig. 2. The False Dense Subgraphs for $\{k1, k2, k3\}$ over G

Sometimes it is not easy to give a suitable density threshold value to select the dense subgraphs in a large graph. This case is discussed in [12] by proposing top-k dense subgraph discovering approach. However, both methods in [11] and [12] are not suitable to deal with our proposed problem in this paper. This is because (1) lots of unnecessary time is spent on the computation of the false dense subgraph candidates that do not contain the full keywords; (2) the components in each dense subgraph candidate have to be detected; (3) the computation of the distance between the keyword nodes in each component cannot be avoided. Therefore, although they are efficient to compute general dense subgraphs, they are not suitable to our problem where we expect to reply user's on-line request within short response time. In addition, we allow the correlated networks have overlaps in this paper, which is not allowed in traditional dense subgraph discovering methods. This is mainly because an entity in a social network may take important roles in different sub-networks at the same time while these sub-networks may not have strong correlation.

Another easy way is to incrementally generate a temporary subgraph by scanning the graph nodes one by one and check the density and the full keywords of the temporary subgraph. However, this method is also infeasible because even if a temporary subgraph can be identified as a false candidate due to low density, the nodes in the temporary subgraph may still have chance to involve in the other dense subgraphs. For instance, if we read in $v1$, $v2$, $v3$, and $v4$, then the density of the subgraph consisting of these four nodes is 66% where we use the simple density metrics (e.g., $\frac{2*|E|}{|V|*(|V|-1)}$) in [9], [11], [12]. However, the nodes $v2$, $v3$, and $v4$ can construct a more dense subgraph (density = 83%) with another node $v7$. Therefore, the incremental-based method does not work because the density metrics are not monotonic.

To address this challenging problem, in this paper we apply the semantics in the metrics [15], [16] to measure the correlation of the components in a network. The semantics of the metrics originates from the *clique* definition in which two nodes belonging to a clique share all nodes in the clique where a clique subgraph is a fully connected subgraph with its density as one. Obviously, if given two nodes share many adjacency nodes, then they have high probability of belonging to a dense subgraph. Based on the observation, we first compute the γ-bounded keyword matched subgraphs as the components over the large graph. And then, we measure the correlations of any two components by checking their overlapped neighbor nodes. Finally, we generate the correlated networks where each network consists of more than one component and the connection nodes among them. Different from previous graph keyword search approaches, in this paper we are required to efficiently find the *maximal covering* keyword matched subgraphs bounded by γ, not top-k minimal subgraphs. Our study can be considered as a complementary work that fills in the gap between graph keyword search and dense subgraph discovering.

The contributions of our work can be summarized as follows:

- We propose and define a new problem of keyword-based correlated networks computation based on the extended semantics of clique.

Authorized licensed use limited to: Griffith University. Downloaded on August 25,2020 at 06:00:36 UTC from IEEE Xplore. Restrictions apply.

- We design a new data structure, by which the graph data can be equivalently transformed into tree data with regard to computing the γ-bounded maximal covering keyword matched subgraphs.
- We develop a weighted shingling approach to improve the performance of discovering the correlated networks for a set of keywords over a large graph.
- We evaluate our methods on a variety of graph data and the experimental results demonstrate the effectiveness of our correlated network model and efficiency of evaluation algorithms.

The rest of this paper is organized as follows. We define our problem of keyword-driven correlated network computation in Section II. Section III provides our solution overview. In Section IV, we design a new data structure, develop an efficient algorithm of computing the γ-bounded maximal covering subgraphs, and present a weighted shingling algorithm to discover the correlated networks. Extensive experimental evaluations are provided in Section V. At last, we review the related work in Section VI and conclude the paper in Section VII.

II. Correlated Network Model

Many networks in real applications can be modeled as graphs. Given a graph $G(V, E)$ which consists of the vertex set V and the edge set E, we are interested in identifying the correlated networks of G for a given set of keywords. Each network candidate consists of a set of correlated individual subgraphs and a set of connection nodes, in which each individual subgraph should be a keyword search result candidate and any two individual subgraphs should have strong correlation. By adjusting the correlation ratio, we can control the density of the generated correlated networks.

Definition 1: (Keyword Matched Nodes) A graph node is a keyword matched node if it directly contains at least one of given keywords.

Definition 2: (γ-Bounded Keyword Matched Subgraph) A γ-bounded keyword matched subgraph consists of a set of keyword matched nodes, the corresponding connection nodes, and connection edges. It satisfies three conditions: (1) there is at least one occurrence of each given keyword matched node in the subgraph. (2) it keeps all the shortest paths of any two keyword matched nodes in the subgraph. (3) the distance of each shortest path of any two keyword matched nodes in the subgraph is no more than user-specified hop number γ.

Each keyword matched subgraph is a network component.

Example 2: Consider a keyword query $\{k1, k2, k3\}$ over the graph G in Figure 1. If γ is set as 2, then Figure 3 shows a 2-bounded keyword matched subgraph. In this subgraph, we can easily observe that the minimal distance of any two keyword nodes does not exceed the bound 2. Comparing the 2-bounded subgraph and its corresponding part in G, readers may question why we count the node $v4$, but do not count the node $v6$? This is because counting $v4$ makes the keyword matched nodes $v2$ and $v9$ to be connected within 2 hops, otherwise their distance would be 3 (i.e., $v9 - v8 - v7 - v2$)

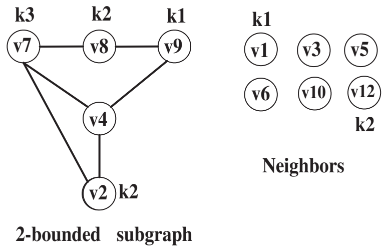


Fig. 3. A 2-Bounded Subgraphs for $\{k1, k2, k3\}$ and its neighbors over G

that exceeds the bound 2. However, deleting $v6$ does not affect the minimal distance of keyword nodes.

Definition 3: (Correlated Network) A network consists of multiple γ-bounded keyword matched subgraphs and their connection nodes where these subgraphs can be considered as the components of the network. The network is a correlated one if and only if all its components are correlated each other. The correlation of any two components G_1 and G_2 can be measured by $\frac{|G'_1 \cap G'_2|}{|G'_1 \cup G'_2|}$ where G'_1 contains the nodes in G_1 and the neighbor nodes of G_1, and G'_2 consists of the nodes in G_2 and the neighbor nodes of G_2.

The above definition extends from the clique semantics where two nodes have high probability of belonging to a dense subgraphs if they have many adjacency nodes. In this work, we can consider each component (γ-bounded keyword matched subgraph) as a "virtual" node and the nodes in or close to the component as the "neighbor" nodes of the "virtual" node. Therefore, if two components can occur in a correlated network, then their corresponding "virtual" nodes should share many neighbor nodes. In this paper, the neighbor nodes of a "virtual" node include not only the nodes in the component, but also the outlinked nodes of the nodes in the component.

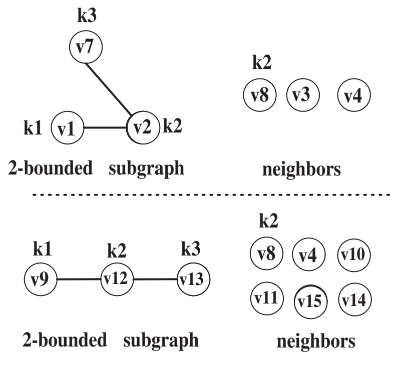


Fig. 4. Another Two 2-Bounded Subgraphs for $\{k1, k2, k3\}$ and their neighbors over G

Example 3: Figure 3 and Figure 4 present three 2-bounded keyword matched subgraphs for a keyword query $\{k1, k2, k3\}$ over the graph G in Figure 1 where γ is set as 2. Their corresponding neighbor nodes are listed at the right side of these subgraphs. Assume the three subgraphs are denoted as G_1, G_2 and G_3 and their extended subgraphs are denoted as G'_1, G'_2 and G'_3. E.g., G'_2 consists of G_2 and its neighbors $\{v3, v4, v8\}$.

Authorized licensed use limited to: Griffith University. Downloaded on August 25,2020 at 06:00:36 UTC from IEEE Xplore. Restrictions apply.

To measure the correlations of the three subgraphs, we have

$$\frac{|G_1' \cap G_2'|}{|G_1' \cup G_2'|} = 0.54; \quad \frac{|G_1' \cap G_3'|}{|G_1' \cup G_3'|} = 0.33; \quad \frac{|G_2' \cap G_3'|}{|G_2' \cup G_3'|} = 0.15.$$

Based on the calculation, we can construct the correlated networks by adjusting the correlation ratio. For instance, if the correlation ratio is set 0.5, we can return a correlated network consisting of G_1 and G_2 where all the nodes in G_2' (i.e., G_2 and its neighbor nodes) are overlapped with G_1'. If the ratio is reduced to 0.33, we can obtain the second correlated network consisting of G_1 and G_3 where $v9$, $v12$, $v8$, $v4$ and $v10$ are the overlapped nodes. Of course, if the ratio is further reduced to 0.15, then we are able to produce a large network that consists of all G_1, G_2 and G_3.

From Example 3, we can see both the nodes in components and their outlinked nodes may become connection nodes. These nodes should be weighted differently. That is to say, an overlapped node appearing in both components should contribute more to the correlation value of the two components than an overlapped node appearing in only one of the components or outside both components. Therefore, in query evaluation, we can assign 1 as the weight of each node v in G_1 because v is an inner node with regard to G_1. Based on the diffusion weighted model, each outlinked node v' in $G_1' \setminus G_1$, in regard to G_1, can be weighted by

$$weight(v', G_1) = 2^{-minDist(v', G_1)} \tag{1}$$

where $minDist(v', G_1)$ is the minimal distance of the shortest paths from the outlinked node v' to any node in G_1.

Definition 4: (Weighted Correlated Networks) Consider any two γ-bounded keyword matched subgraphs G_1 and G_2. Assume G_1' and G_2' are their corresponding extended and weighted subgraphs. If G_1 and G_2 can be grouped together as a correlated network, then the correlation of G_1' and G_2' should satisfy a given threshold value, i.e., $\frac{\sum\{weight(v,G_1)*weight(v,G_2)|v \in G_1' \cap G_2'\}}{|G_1' \cup G_2'|}$ is no less than the given threshold value.

Following Example 3, we have the calculated correlations based on Definition 4.

$$\frac{\sum\{weight(v,G_1)*weight(v,G_2)|v \in G_1' \cap G_2'\}}{|G_1' \cup G_2'|} = 0.34;$$
$$\frac{\sum\{weight(v,G_1)*weight(v,G_3)|v \in G_1' \cap G_3'\}}{|G_1' \cup G_3'|} = 0.18;$$
$$\frac{\sum\{weight(v,G_2)*weight(v,G_3)|v \in G_2' \cap G_3'\}}{|G_2' \cup G_3'|} = 0.03.$$

Due to the above more precise correlation values, it is much easier for us to judge the correlations of the three 2-bounded subgraphs.

In this paper, we are interested to efficiently discover all the correlated networks based on our formal definitions. In addition, we will also discuss the efficient solution to generate the γ-bounded keyword matched subgraphs and their corresponding extended and weighted subgraphs with regards to the given keyword query.

III. Solution Overview

Given a keyword query Q and an integer γ over a large graph G = (V, E), a naive solution is to first compute all the γ-bounded keyword matched subgraphs and their extended subgraphs containing their neighbor nodes. Then, it calculates the correlations of any two subgraphs. At last, it groups the subgraphs and their connection nodes based on the selection criteria, i.e., the given threshold value. However, such a naive solution may be too expensive to be practical due to the following three reasons.

- It is impossible to maintain all the γ-bounded keyword matched subgraphs and their extended subgraphs of a big graph in the main memory. It is worth noting that most existing keyword search methods focus on finding top-k subgraphs in large graph by making use of ranking functions due to the large number of possible keyword search results. However, in this work, we are interested in the correlations among all possible keyword matched subgraphs (the number N of possible keyword matched subgraphs is far greater than k), rather than top-k subgraphs. The naive solution has to maintain all the γ-bounded keyword matched subgraphs and their extended subgraphs of a large graph.
- To generate the correlated networks, we have to measure the correlation of any two keyword matched subgraphs by pair-wise comparisons. However, it is very expensive to do N^2 times of comparisons and load all the possible keyword matched subgraphs from hard disk (high I/O cost), especially when N is a big number.
- To generate all the γ-bounded keyword matched subgraphs and their extended subgraphs for a large graph, the graph may be passed many times because a graph node may appear in multiple γ-bounded keyword matched subgraphs and their extended subgraphs.

To address the above challenges, we are required to propose an efficient and effective approach that can incrementally generate γ-bounded keyword matched subgraphs. For each γ-bounded keyword matched subgraph G_1 and its extended subgraph G_1', we only need to record a fixed size of information through making a sample over G_1', i.e., G_1' can be represented by the fixed size of information. As such, we do not read G_1 and G_1' into memory at the run time, by which lots of time can be saved. At last, we can determine the correlation between any two γ-bounded keyword matched subgraphs by estimating the correlation between their corresponding fixed size of information. This idea is similar to the adapted shingling algorithm in [15], in which the authors group graph nodes based on their neighbor nodes by applying the shingling algorithm [16], [18]. Compared with [15] that only focuses on the grouping of general graph nodes, our work has to address more significant challenges:

- Computing γ-bounded keyword matched subgraphs is an NP-hard problem discussed below;
- To precisely group graph nodes based on user's request, we not only consider the nodes of γ-bounded keyword matched subgraphs (like [15]), but also take into account the outlinked nodes of the nodes based on the diffusion weighted model.

Authorized licensed use limited to: Griffith University. Downloaded on August 25,2020 at 06:00:36 UTC from IEEE Xplore. Restrictions apply.

- Since the weight of a node represents the relative correlation from the node to its related γ-bounded keyword matched subgraph, the shingling algorithm should be adapted to consider the weights of the nodes to be compared.

IV. Evaluation Algorithms

A. *Generating γ-bounded Keyword Matched Subgraphs*

To generate γ-bounded keyword matched subgraphs for a set of query keywords and a graph, a straightforward method is to run the Breadth-First Traversal algorithm for each keyword node up to γ hops. By doing this, we can generate all the γ-bounded keyword matched subgraph candidates where these candidates may contain duplicates or non-shortest paths. To refine these candidates, we have to check the path of every two nodes in each candidate and prune the non-shortest paths. And then, we filter the duplicate candidates in the final refined candidate set. However, the computational cost of this straightforward method is very expensive due to the following reasons:

- a large number of repeated scans on the graph for each keyword node;
- a large number of unqualified candidates to be produced;
- time spent for identifying the shortest path of any two nodes in every candidate.

To reduce the high computational cost, it is required to design an efficient approach to overcome the above shortcomings of the straightforward method. To do this, we devise a new tree data structure to record the shortest path of any two nodes in the graph. As such, generating γ-bounded keyword matched subgraphs for a set of query keywords over a graph can be realised by calculating γ-bounded maximal covering keyword matched subtrees over the equivalently transformed tree data structure. The transformation does not depend on any query, which can be done offline.

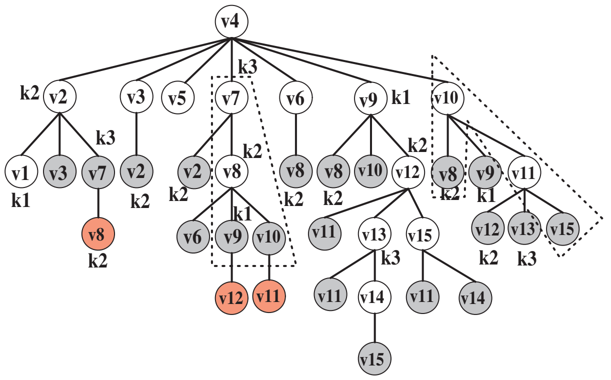


Fig. 5. The Transformed Tree Data Structure of Graph G

By making some node copies, we can use a tree data structure to maintain the graph nodes and their shortest paths. Figure 5 illustrates the transformed tree data structure of the graph data in Figure 1. As shown in Figure 5, the color nodes represent the copied nodes, where the gray color nodes are copied by only considering *directly-connected-edges* of graph nodes while the red color nodes are copied by considering the linked edges of graph nodes up to Γ hops (Γ=3 here) where Γ is set by system administrators, not the users issuing queries. Based on the general users' requirements or search favors, we can pre-build a transformed tree for a graph. Obviously, the larger the value Γ is, the higher space cost the transformed tree data structure consumes. Although the space cost increases when Γ becomes larger in theory, most of time, it only needs a little extra cost because the node information are maintained in a separate file and only node IDs are used to represent the node copies. In addition, many copied nodes of taking Γ' have been implicitly embraced when we deal with Γ'' ($< \Gamma'$). For example, we only need to add three red color copied nodes $v8$, $v12$ and $v11$ in Figure 5 in order to satisfy Γ (=3) requirement. This is because most of other paths ($\leq$ 2 hops) have already been pre-built when we consider the *directly-connected-edges* of graph nodes, e.g., as shown in the dashed line area, $v7$, $v9$ and $v10$ have been connected to the node $v8$; $v8$, $v10$, $v11$ and $v15$ have been connected in a sequential order.

With the help of transformed tree data structure, it becomes easy to compute the γ-bounded keyword matched subgraphs. The basic idea is to find a set of subtrees where each subtree should have a *maximal covering* of keyword nodes bounded by γ. Here, *maximal covering* means to include keyword nodes as many as possible, but the maximal distance of any two keyword nodes is bounded to γ. To efficiently find the set of subtrees, we can read the tree nodes in a Top-Down manner and incrementally generate the subtree candidates where each candidate is represented by its node set. Although it may still produce a few duplicate candidates due to the existence of copied nodes, the number of duplicate candidates is much less than that of the straightforward method.

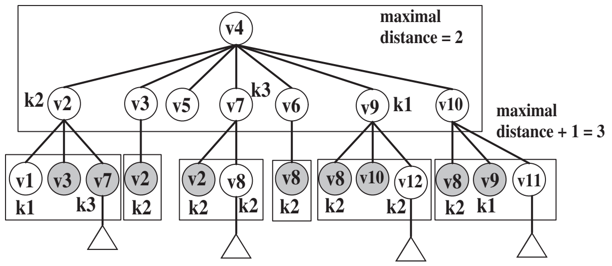


Fig. 6. 1^{st} and 2^{nd} steps of processing the tree nodes in Top-Down way

Example 4: Consider a keyword query $\{k1, k2, k3\}$ (γ = 3) over the transformed tree shown in Figure 5. Firstly, we deal with the nodes at the root ($v4$), and at the 1^{st} level, i.e., $v2$, $v3$, $v5$, $v7$, $v6$, $v9$, $v10$. Since $v2$, $v7$, $v9$ contain the full keywords together, $\{v4, v2, v7, v9\}$ is a candidate but it is not a maximal covering candidate. We also know the maximal distance of $\{v4, v2, v7, v9\}$ is 2, which allows us to probe the nodes at the 2^{nd} level. For each branch of nodes (in the rectangles), we first generate its corresponding subtree that does not count the non-keyword leaf nodes in, and then produce the corresponding candidate node set by filtering the repeated

Authorized licensed use limited to: Griffith University. Downloaded on August 25,2020 at 06:00:36 UTC from IEEE Xplore. Restrictions apply.

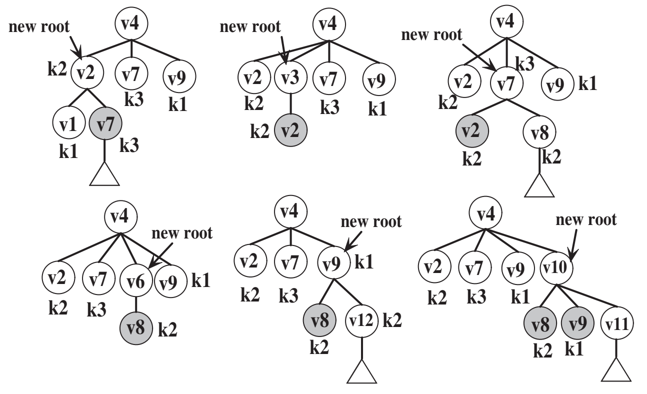


Fig. 7. Candidate set of running 1st and 2nd steps in Top-Down way

nodes. For instance, the candidate sets can be listed in the same order as shown in Figure 7: $R1 = \{v4, v2, v7, v9, v1\}$, $R2 = \{v4, v2, v7, v9\}$, $R3 = \{v4, v2, v7, v9, v8\}$, $R4 = \{v4, v2, v7, v9, v6, v8\}$, $R5 = \{v4, v2, v7, v9, v8, v12\}$, $R6 = \{v4, v2, v7, v9, v10, v8\}$. Since $R2$ and $R3$ are the subsets of $R1$ and $R4$ respectively, both $R1$ and $R3$ cannot become a result candidate.

At the next step, we take the child nodes of $v4$ (marked by arrow in Figure 7) as the new roots and check their corresponding subtrees. Since the subtrees rooted at $v2$, $v7$, $v9$ and $v10$ contain remaining nodes (marked by the triangle), we need to check these subtrees $R1$, $R3$, $R5$ and $R6$. Let's take $R5$ as an example. Since we need to probe the nodes at the new level in the subtree rooted at $v9$, the maximal distance between the nodes in the subtree and its root is 2. As such, we can discard $v2$ and $v7$ because they are outside of 3 hops. Subsequently, this leads to the deletion of $v4$ because $v4$ becomes a non-keyword leaf node. By probing the remaining nodes of $R5$ level by level, the candidate $R5' = \{v9, v8, v12, v13\}$ will be generated. Similarly, we can get another three candidates: $R1' = \{v1, v2, v7, v8\}$ from the branch of $R1$, $R3' = \{v2, v7, v8, v9\}$ (subset of $R4$) from the branch of $R3$ and $R6' = \{v10, v8, v9, v11, v12, v13\}$ (superset of $R5'$) from the branch of $R6$. By comparing these new generated four candidates and the previous candidates together, we can get the final result candidates, i.e., $R1$, $R1'$, $R4$, $R5$, $R6$ and $R6'$.

Now, we explain the detailed procedure of transforming graph G to the new tree structure T. The key idea is to record the shortest paths of graph nodes up to the given maximal number Γ of hops in the tree T. Firstly, we copy the directly-connected-edges from G to T, which guarantees the *1-Hop Correctness*. And then, we compare the connected edges with 2 hops between G and T. If some edges do not appear in T, then we need to copy them in T such that all the edges that are connected within 2 hops in G should also appear in T, which guarantees the *2-Hop Correctness*. Similarly, we can check the connected edges up to Γ hops, which can guarantee the Γ-*Hop Correctness*. As such, we have the following property.

Property 1: Given a graph G, its pre-built tree T can guarantee to correctly answer any keyword queries with the given hop number $\gamma \leq \Gamma$ where γ is a user specified hop number value while Γ is the maximal hop number bound set by system administrators.

The detailed procedure is provided in Algorithm 1.

Algorithm 1 Graph2Tree($G = (V, E)$, an integer Γ)

```
1: Take any node $v \in V$ with high degree from the graph $G$;
2: Take the node $v$ as the root of a new tree data $T$;
3: Scan $G$ from the node $v$ in the breadth-first traversal;
4: for each node $v \in V$ do
5:     Insert all its directly-connected-edges $\in E$ into $T$ (1-Hop Correctness);
6: hopNum = 2;
7: while hopNum $\leq \Gamma$ do
8:     for each node $v \in V$ do
9:         Get sets of $v's$ connected edges with hopNum hops in $G$;
10:        for each set of $v's$ connected edges do
11:            if they appear in $T$ and they are connected then
12:                {do nothing};
13:            else
14:                Copy and insert the necessary edges in $T$ to guarantee the set of edges appear and are connected in $T$;
15:    hopNum++;
16: Write $T$ into file system (hopNum-Hop Correctness);
```

From the above discussion in Example 4 and the implementation steps in Algorithm 1, we can see the benefits of utilizing our tree data structure to improve the performance of searching γ-bounded keyword matched subgraphs over graph data.

- It can avoid online identification of shortest path in the straightforward method because the shortest paths of nodes have been identified and pre-built in the new data structure. Therefore, the paths appearing in any candidate produced from the new data structure should be the shortest.
- It can reduce the unnecessary scanning cost of the straightforward method because we can produce all the candidates by scanning the new data structure just once.
- It can incrementally generate all the candidates, unlike the straightforward method that has to compute every candidate from the scratch every time (starting from every keyword node, and probing the connected nodes up to γ hops).
- The space cost of our transformed tree data structure can be controlled by varying the hops Γ.

Next, we show the brief procedure of computing the γ-bounded keyword matched subgraphs according to the pre-built tree data structure T. The key idea is to efficiently find all the subtrees, in which the maximal distance of any two keyword nodes is bounded by γ. To do this, we firstly tranverse the pre-built tree data level by level in a top-down strategy as shown in Example 4. For each node v, we probe all the possibilities that may generate the maximal covering subtrees by analyzing the relationships among the nodes at the current level, the other nodes at its following levels, and γ. After that, we check its child nodes until all nodes are reached.

Authorized licensed use limited to: Griffith University. Downloaded on August 25,2020 at 06:00:36 UTC from IEEE Xplore. Restrictions apply.

To improve the efficiency of identification, we can utilize the *Range Encoding Scheme* - (start, end, level) in [19] to quickly check if keyword nodes appear at some level or not.

To correctly and completely generate all the results, we have the following properties. Assume v is the node we are exploring in T at one moment. The nodes in T can be bounded to generate the node set of the γ-bounded subgraph in regard to the node v.

Property 2: If γ mod 2 = 0, then it is required to probe the ancestor nodes of v at the level in $[v.level - \frac{\gamma}{2}, v.level - 1]$. For each ancestor node v_a, we have to probe its descendant nodes at those levels in $[v_a.level+1, 2*v_a.level+\frac{\gamma}{2}-v.level]$. It also needs to consider the descendant nodes of v at those levels in $[v.level + 1, v.level + \frac{\gamma}{2}]$

Proof: Keep in mind that we are searching all the nodes where the maximal distance of any two nodes is no more than γ. The node v can be considered as the centroid. It is easy to see its descendant nodes at those levels in $[v.level + 1, v.level + \frac{\gamma}{2}]$ and ancestor nodes at those levels in $[v.level - \frac{\gamma}{2}, v.level - 1]$ are in the radius $\frac{\gamma}{2}$ in regard to the centroid v. Since its sibling nodes may also be in the radius, we need to probe the subtrees rooted at its ancestors. For any ancestor v_a at those levels in $[v.level - \frac{\gamma}{2}, v.level - 1]$, we have the distance of v_a and v by $v.level - v_a.level$. Due to the radius $\frac{\gamma}{2}$, we still have room distance $\frac{\gamma}{2} - (v.level - v_a.level)$ to check the descendant nodes of v_a. Therefore, the descendant nodes of v_a can be identified up to those levels $v_a.level + \frac{\gamma}{2} - (v.level - v_a.level) = 2 * v_a.level + \frac{\gamma}{2} - v.level$. ■

Property 3: If γ mod 2 $\neq$ 0, then it is required to probe the ancestor nodes of v at those levels in $[v.level - \lfloor\frac{\gamma}{2}\rfloor, v.level - 1]$. For each ancestor node v_a, we have to probe its descendant nodes at those levels in $[v_a.level + 1, 2 * v_a.level + \lfloor\frac{\gamma}{2}\rfloor - v.level]$. The descendant nodes of v have to be considered in different combinations based on the child branches of v. If we take the nodes in one branch up to the level $v.level + \lceil\frac{\gamma}{2}\rceil$, then the nodes in the other branches are limited to the level $v.level + \lfloor\frac{\gamma}{2}\rfloor$.

Proof: The proof is similar to the above property. The only difference is to probe the descendant nodes of v up to the level $v.level + \lceil\frac{\gamma}{2}\rceil$, instead of $v.level + \frac{\gamma}{2}$. ■

Example 5: Assume we have a keyword query $\{k1, k2, k3\}$ over the tree in Figure 5 and γ=3. We first take $v4$ as an example. Since the level of $v4$ is 0, it does not have ancestor nodes. So we only need to probe its descendant nodes. Since γ mod 2 $\neq$ 0, we need to explore the nodes up to level 2 at one child branch and the nodes up to level 1 at the other child branches. For Branch 1 up to level 2, we have {v4, v2, v1, v7}. And we also get the nodes {v4, v7, v9} up to level 1 at the other branches where the non-keyword nodes can be filtered. As such, we can get the final node set {v4, v2, v1, v7, v9}. Similarly, we can get a list of node sets if we consider the nodes up to level 2 at other branches individually.

Let us take an internal node $v8$ as an example. Due to $v8.level$ = 2, we first explore its ancestor nodes at the level in [1,1], i.e., $v7$. Because $v7.level + 1 = 2$ exceeds $2*v7.level + \lfloor\frac{\gamma}{2}\rfloor - v8.level = 1$, no satisfied descendant nodes exist. Since $v7$ is a keyword node, it will be included. The rest is to consider the combinations of the descendant nodes up to the level $v8.level + \lceil\frac{\gamma}{2}\rceil = 4$ or $v8.level + \lfloor\frac{\gamma}{2}\rfloor = 3$.

To make reading easy, we repeatedly scan the nodes up to the level $v.level + \lfloor\frac{\gamma}{2}\rfloor$ for generating different combinations as explained in the above properties and example. However, in our implementation, the repeating can be avoided easily because we only need to scan and keep the nodes within the level $v.level + \lfloor\frac{\gamma}{2}\rfloor$. At each time, these nodes can be used to combine with the nodes at the level $v.level + \lceil\frac{\gamma}{2}\rceil$ at one branch.

Since the keyword nodes may be connected in a cycle in the original graph data, it may result in incomplete result set because the cycle may be represented in the tree index in a long path. To do this, we can maintain the intermediate result candidates in a hierarchy and check the existence of cycles in the overlapped γ-bounded subgraph candidates. The algorithm of computing the γ-bounded keyword matched subgraphs according to the pre-built tree data structure is not provided due to the limited space. Once the γ-bounded keyword matched subgraphs are computed, their outlinked nodes can be accessed by graph adjacency lists.

B. General Shingling Algorithm

Shingling algorithm has been employed in many applications [20], [17]. In this work, we will employ the technique to efficiently measure the correlations among the γ-bounded keyword matched results where the results can be represented by their corresponding node sets. Each node set can be rewritten into a constant-size fingerprint. As such, these results can be compared by simply comparing their fingerprints. A simple and natural measure of the similarity of two sets is the Jaccard Coefficient, defined as the size of the intersection of the sets divided by the size of their union: $|A \cap B|/|A \cup B|$. Formally, if π is a random permutation of the elements in the ordered universe U from which A and B are drawn, then it can be shown that (see, e.g., [18])

$$Pr[\pi^{-1}(min_{a \in A}\{\pi(a)\}) = \pi^{-1}(min_{b \in B}\{\pi(b)\})] = \frac{A \cap B}{A \cup B}$$

That is, the probability that the smallest element of A and B is the same, where smallest is defined by the permutation π, is exactly the similarity of the two sets according to the Jaccard Coefficient. Using this observation, we can compute the fingerprint of A by fixing a constant number c of permutations π_1, ..., π_c of U, and producing a vector whose i-th element is $min_{a \in A}\pi_i(a)$. The similarity of two sets is then estimated to be the number of positions of their respective fingerprint vectors that agree. This formulation means to consider every 1-element set contained entirely in A and B, and measure the agreement by the fraction of these 1-element subsets that appear in both sets. To make generalization as [15], [21], we consider every s-element set contained entirely within either set, and measure similarity by the fraction of these s-element subsets that appear in both. It is identical to measuring the similarity of A and B by

Authorized licensed use limited to: Griffith University. Downloaded on August 25,2020 at 06:00:36 UTC from IEEE Xplore. Restrictions apply.

estimating the Jaccard Coefficient of two sets A_s and B_s, where $A_s = \{\{a_{x_1}, a_{x_2}, ..., a_{x_s}\} | a_{x_i} \in A \wedge 1 \leq i \leq s\}$ ($|A_s| = C^s_{|A|}$) and $B_s = \{\{b_{y_1}, b_{y_2}, ..., b_{y_s}\} | b_{y_i} \in B \wedge 1 \leq i \leq s\}$ ($|B_s| = C^s_{|B|}$). By tuning both s and c, we may arrive at an algorithm that accurately distinguishes between sets that are above or below a certain threshold of similarity. To avoid the consideration of all possible permutations π in our choice of c such permutations, we can employ the techniques of min-wise independent permutations, as [18], [16]. In this work, we utilize two-universal hash functions to simulate the permutations as [15].

Given a γ-bounded keyword matched subgraph G_1 and its neighbor node set $V_{neighbors} = \{v_1, v_2, ...\}$, we can generate c shingles by applying the general shingling algorithm and each shingle represents s-node subset. The accuracy of the sample can be guaranteed by seting different paramenters s and c as discussed in [15]. The brief procedure is provided in Algorithm 2.

Algorithm 2 BasicShingling($V_{neighbors}$, s, c)

```
1: let $H$ be a hash function from strings to integers;
2: let $p$ be a large random prime (say, 32 bits);
3: let $\lambda_1, \eta_1, ..., \lambda_c, \eta_c$ be random integers in [1...$p$];
4: for ($i = 1$; $i <= |V_{neighbors}|$; $i++$) do
5:     $x_i = H("V_{neighbors}[i]")$;
6: for ($j = 1$; $i <= c$; $j++$) do
7:     for ($i = 1$; $i <= |V_{neighbors}|$; $i++$) do
8:         $y_i = (\lambda_j \times x_i + \eta_j)$ mod $p$;
9:     let $y'_1, y'_2, ..., y'_s$ be $s$ minimal values of $\{y_1, y_2, ..., y_{|V_{neighbors}|}\}$;
10:    let $z_j = H("y'_1 \circ y'_2 \circ ... \circ y'_s")$;
11: return $z_1, z_2, ..., z_c$;
```

C. Weighted Shingling Algorithm

Given a γ-bounded keyword matched subgraph G_1 and its extended subgraph G'_1, the weights of the nodes in G'_1 can be quantified depending on the distance of each node v' to G_1 using our weight function $2^{-minDist(v', G_1)}$. For example, if a node v'_1 is located in G_1, i.e., $minDist(v'_1, G_1) = 0$, then $weight(v'_1, G_1) = 1$; if a node v'_2 is the neighbor node of G_1, i.e., $minDist(v'_2, G_1) = 1$, then $weight(v'_2, G_1) = 1/2$; Similarly, the outlinked nodes can be weighted based on their distances to G_1. After we assign the weights to the nodes in G'_1, the set of nodes can be classified into different subsets based on their weight values. Here, we assume we only consider the nodes with their weights equal to or above 0.25. As such, the nodes in G'_1 can be classified into three groups, i.e., $\{(V_1, 1), (V_2, 0.5), (V_3, 0.25)\}$. That is to say, we only consider the neighbor nodes up to two steps from the subgraph G_1. Given two γ-bounded keyword matched subgraphs, if they are considered to be correlated, then it is necessary to have many shared nodes between their extended subgraphs and the aggregated weight of these shared nodes should be enough big. To make a normalization, we can divide the aggregated weight by their total number of distinct nodes, by which the measure can be normalized as a value in [0,1]. Given a threshold in [0,1], we can select the correlated γ-bounded keyword matched subgraphs if the percentage of their shared weighted nodes is no less than the given threshold.

For each subset of nodes, we can call the BasicShingling algorithm and get the returned c shingles. For example, we have c shingles $Z^1 = \{z^1_1, z^1_2, ..., z^1_c\}$ for the first subset V_1 of nodes and the weight of each shingle is 1. Similarly, we have c shingles $Z^2 = \{z^2_1, z^2_2, ..., z^2_c\}$ for the second subset V_2 and the weight of each shingle is 0.5, and c shingles $Z^3 = \{z^3_1, z^3_2, ..., z^3_c\}$ for the third subset V_3 and the weight of each shingle is 0.25. Given another γ-bounded keyword matched subgraph G_2 and its extended subgraph G'_2, we can also generate three sets of shingles with different weights, e.g., $Z^{1'} = \{z^{1'}_1, z^{1'}_2, ..., z^{1'}_c\}$ where the weight of each shingle is 1; $Z^{2'} = \{z^{2'}_1, z^{2'}_2, ..., z^{2'}_c\}$ where the weight of each shingle is 0.5; $Z^{3'} = \{z^{3'}_1, z^{3'}_2, ..., z^{3'}_c\}$ where the weight of each shingle is 0.25.

The equation $\frac{\sum\{weight(v,G_1)*weight(v,G_2)|v \in G'_1 \cap G'_2\}}{|G'_1 \cup G'_2|}$ can be approximately estimated by evaluating the shared shingles of G'_1 and G'_2 by the following equation:

$$\rho = \frac{\sum\sum\{weight(z_x) * weight(z_y) | z_x = z_y\}}{\bigcup_{i=1}^{3} Z^i \cup \bigcup_{j=1}^{3} Z^j} \tag{2}$$

where $z_x \in Z^i \wedge z_y \in Z^j$, $1 \leq i \leq 3$, $1 \leq j \leq 3$.

If ρ is no less than the given threshold value, then we can say the two γ-bounded keyword matched subgraphs G_1 and G_2 are correlated, i.e., they can be considered in a dense subgraph that is constructed through the connection of their shared nodes of G'_1 and G'_2.

Equation 2 means that for each shingle element of Z^i, it chooses the minimum element of corresponding permutation and its weight. Weights can be normalized at the end. Therefore, it has the following property.

Property 4: The value ρ in Equation 2 is an unbiased estimate of the resemblance of G'_1 and G'_2 where the resemblance of the two subgraphs is computed by the equation $\frac{\sum\{weight(v,G_1)*weight(v,G_2)|v \in G'_1 \cap G'_2\}}{|G'_1 \cup G'_2|}$. The proof can be seen from [22].

D. MS-based Approach of Generating Correlated Networks

Here, we develop a *M*erge-*S*ort-based approach to find the correlated networks based on the generated shingles and their weights.

Firstly, we can apply the weighted shingling algorithm to the extended subgraph G'_1 of each γ-bounded keyword matched subgraph G_1 and generate a list of shingles {(z, weight(z,RID))} with their different weights where the corresponding γ-bounded keyword matched subgraph is represented using a result ID (RID). As such, for a subgraph result, we have RID $\rightarrow$ {(z, weight(z,RID))}, C where C is the total number of shingles generated for the result RID. That is to say, C is the sketch size of the result RID. And then, we produce a list of all the shingles, the results they appear in, and their weight in the corresponding results, sorted by shingle value. By this, the sketch of G'_1 can be transformed

Authorized licensed use limited to: Griffith University. Downloaded on August 25,2020 at 06:00:36 UTC from IEEE Xplore. Restrictions apply.

into a list of <shingle z, RID, weight(z,RID)>. After that, we generate a list of all pairs of keyword search results that share any shingles, along with the aggregated weight they have in common. To do this, we take as input the file of sorted <shingle z, RID, weight(z,RID)> and output list of <RID$_x$, RID$_y$, weight(z,RID$_x$)*weight(z,RID$_y$), 1>. At last, we can apply a merge-sort procedure (adding the weights for matching RID$_x$ - RID$_y$ pairs to produce a single file of all <RID$_x$, RID$_y$, AggregatedWeight, count> sorted by the first result ID RID$_x$. We normalize the value AggregatedWeight by $C_x + C_y - count$ where C_x is the sketch size of RID$_x$ and C_y is the sketch size of RID$_y$. To measure the correlation of any two results, e.g., RID$_x$ and RID$_y$, we can check if the calculated value $\frac{AggregatedWeight}{C_x + C_y - count}$ exceeds our threshold value. If it does, we add a link between the two results (RID$_x$ and RID$_y$). After we deal with all <RID$_x$, RID$_y$, AggregatedWeight, count>, we can build a set of correlated networks where each network can be output by attaching the corresponding node information.

V. Experimental Evaluation

In this section, we present experimental results evaluating the effectiveness and efficiency of our methods (i.e., no-shingling and shingling) on three real datasets, namely the CiteSeer, Wikipedia and CNET Forum datasets. All algorithms are implemented in Java and run on a 3.0 GHz Intel Pentium 4 machine with 3GB RAM running Windows 7. The graph data is stored in a node information file and an edge relationship file where the graph edges are maintained in adjacency list. Before we start to discover the correlated networks based on the given keywords, we offline construct the tree data from the edge relationship file for each dataset. By doing this, we can compute the correlated networks from the transformed tree data and the node information file directly. In the following experimental results of evaluating queries, we do not include the time cost for the transformation, since this cost is not related to any keyword query.

A. Datasets and Queries

CiteSeer Dataset: We construct a citation graph from the CiteSeer dataset [1], in which each record can be considered as a node, labeled by the element <identifier>, and each citation relationship is labeled by the element <dc:relation>. Since we do not consider "direction" in our discovered correlated networks, we denote the citation graph as an undirected graph where two nodes have a relationship if a node cites another one. In addition, we extract the title (<dc:title>...< /dc:title>), authors (<dc:creator>...< /dc:creator>) and description (<dc:description>< /dc:description>) as the node information for each record. The node information can be stored in a file by following a brief format: node ID $\rightarrow$ node content information; and the correlations of the nodes are stored in another file by following a format: node ID $\rightarrow$ list of related node IDs.

[1]http://csxstatic.ist.psu.edu/about/data

TABLE I
Extracted Undirected Graph Information

Dataset Name	Size	Number of Nodes	Number of Edges
CNET Forum	7MB	106,436	79,217
CiteSeer	89MB	230,470	406,784
Wikipedia	1,141MB	5,716,808	65,080,196

Wikipedia Dataset: We construct a page-to-page link graph from the Wikipedia dataset [2], in which each page can be considered as a graph node and the page title is used as the graph node's unique identifier, and the link relationships are discovered by extracting the link "to" or "from" information from the page. To assign a unique number to each node, we sort the total graph nodes by their full titles and each line number can be used as the ID of the corresponding node, which is stored in a node information file. By doing this, we utilize the ID numbers to replace the nodes in the link graph, which is stored in an edge information file. The same format is employed: node ID $\rightarrow$ title information, and node ID $\rightarrow$ list of linked node IDs.

CNET Forum Dataset: We construct a web post-based graph from the CNET forum dataset [3], in which each post can be considered as a node and the replylinks between posts can be taken as the edges. We take the message ID as the node ID from the element <url> and take the user and title information of each post as the node content. For each post, we generate a list of linked edges by taking the elements <replylink>. Similarly, the node and edge information are stored in two files using the same format as the above.

The detailed information of the three selected datasets are illustrated in Table I.

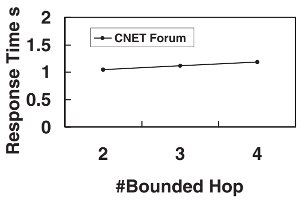
Keyword Queries: For each dataset, we can calculate the term frequencies by reading the node content information from the corresponding node file. After filtering the stop terms, e.g., "the", "and", "of", "for", "that", "have", etc., we can get top 200 most frequent terms for each dataset. Based on our analysis, the frequencies of the top 200 terms are about 500 for CNET Forum dataset, 4000 for CiteSeer dataset, and 8000 for Wikipedia, respectively. And then, we generate 20, 40, and 80 keyword queries for the three datasets by randomly choosing one or two or three terms from their own top 200 terms. The experimental results below show their average performance, which can make unbiased evaluation about our proposed methods.

B. Evaluating Graph2Tree Algorithm

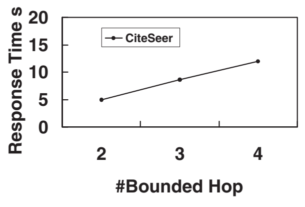
Figure 8 shows the time and space cost of transforming the graph datasets to the corresponding tree data structures when the value Γ is set as 2, 3, 4, respectively. The space cost does not count the node content because we use the node IDs to represent the nodes in graph and tree data. By doing this, even if the transformed tree may contain lots of copied nodes, its size can be controlled. From Figure 8(a), we can see that

[2]http://dbpedia.org/Datasets or http://haselgrove.id.au/wikipedia.htm
[3]http://sifaka.cs.uiuc.edu/ wang296/Data/index.html

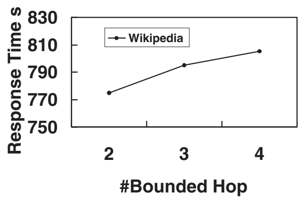
Authorized licensed use limited to: Griffith University. Downloaded on August 25,2020 at 06:00:36 UTC from IEEE Xplore. Restrictions apply.



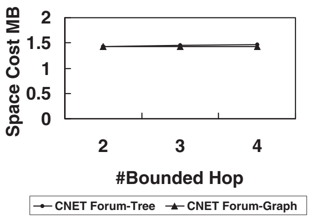
(a) Time Cost of Transformation



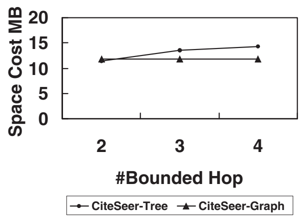
(b) Time Cost of Transformation



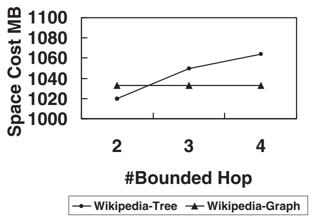
(c) Time Cost of Transformation



(d) Tree Size vs. Graph Size



(e) Tree Size vs. Graph Size



(f) Tree Size vs. Graph Size

Fig. 8. Transforming Graphs to Trees for Different Γ Values

the time of generating the three trees from the CNET Forum graph with Γ bounded by 2, 3, 4 is nearly same. And the three trees take the nearly same space as the graph, as shown in Figure 8(d). This is because the degree of most nodes in the graph is 2. In this case, there is no big difference between the graph and its transformed trees. Differently, we can see the time increases greatly with the increase of Γ for CiteSeer and Wikipedia datasets in Figure 8(b) and Figure 8(c). This is because the degree of most nodes in these two datasets exceeds 10. When Γ becomes large, the number of nodes to be considered would be increased greatly. Figure 8(e) and Figure 8(f) show that the trees consume less space than their graphs when Γ is 2. The main reason is that the adjacency list strategy makes each edge appear twice in the (undirected) graph edge file, but once in the corresponding tree file. By removing some edges from the graph adjacency list file based on Algorithm 1, we can realize the transformation from graph to tree with Γ bounded by 2. When Γ increases to 3 or 4, it will take a bit more space to cache the transformed tree data, but the increasing rate is still small in comparison with the original graph data size. Therefore, the experimental results in this section demonstrate the practical feasibility of our proposed tree data structure.

C. Time Cost of Query Evaluation

Figure 9, Figure 10 and Figure 11 show the time cost of evaluating our selected keyword queries over the three datasets when the number γ of user-specified hops takes 3, 4 and 5, respectively. Here, the *No-Shingling* approach is to calculate the correlated networks by making pair-wise comparisons over the total γ-bounded keyword matched subgraphs to be accessed from hard disk. From Figure 9, *No-Shingling* is better than our shingling approach because the number of keyword matched results is not large and the degree of most nodes is small in CNET Forum dataset. From Figure 10 and Figure 11, we can see that *Shingling* approach outperforms *No-Shingling* method by about 2 times for CiteSeer dataset and 16 times for Wikipedia dataset. The main reason is that we can load more results in an I/O operation and deal with them in a batch based on the merge-sort algorithm. The time cost of each query evaluation consists of four stages: keyword query $\rightarrow$ neighbor collection $\rightarrow$ shingle generation $\rightarrow$ correlated network generation. By analyzing the time cost components, we find up to 50% time cost is spent on the 4th stage. Therefore, the computation of correlated networks can be speeded up by comparing the shingles, rather than the keyword matched subgraphs. Only about 20% time cost is spent on the 1st stage, which is also the reason that the increasing rate is a bit when we increase the number of user-specified hops.

D. Memory Usage of Generating Correlated Networks

For a keyword query, the memory usage of generating its correlated networks is spent on maintaining the total keyword matched subgraphs, their extended subgraph nodes and the related node content information. To show the comparable performance of *Shingling* approach and *No-Shingling* method, we utilize *compressed ratio* measuring the value of dividing memory usage of *No-Shingling* approach by that of *Shingling* method. If the compressed ratio is lower than 1, then it says *No-Shingling* method consumes less space than *Shingling* method. The larger the compressed ratio is, the smaller the space to be consumed by the *Shingling* method. Figure 12 shows that the shingling method costs more space than the no-shingling method for CNET Forum dataset. Figure 13 shows that when the parameters (s, c) in shingling algorithm are set to be (3,5) or (3,10), the shingling method costs less space than the no-shingling one. But the space of shingling-(3,15) is more than that of the no-shingling one for CiteSeer dataset. From Figure 14, we can see that the compressed ratio is much

Authorized licensed use limited to: Griffith University. Downloaded on August 25,2020 at 06:00:36 UTC from IEEE Xplore. Restrictions apply.

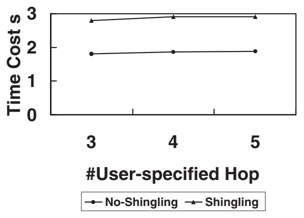


Fig. 9. Time of Evaluating CNET Forum

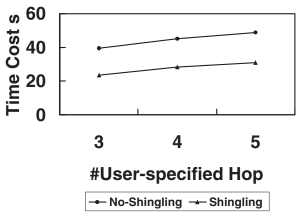


Fig. 10. Time of Evaluating CiteSeer

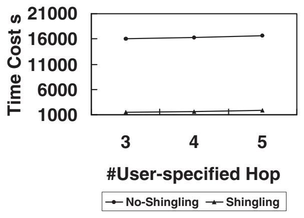


Fig. 11. Time of Evaluating Wikipedia

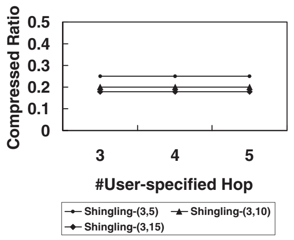


Fig. 12. Compressed Ratio of Results for CNET Forum

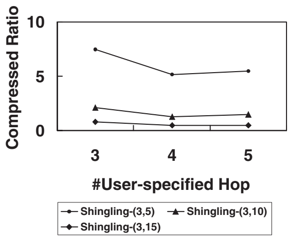


Fig. 13. Compressed Ratio of Results for CiteSeer

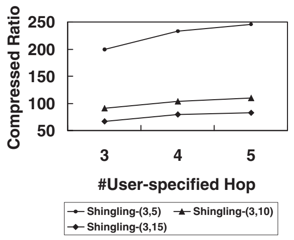


Fig. 14. Compressed Ratio of Results for Wikipedia

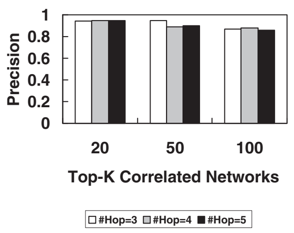


Fig. 15. Precision Evaluation for CNET Forum

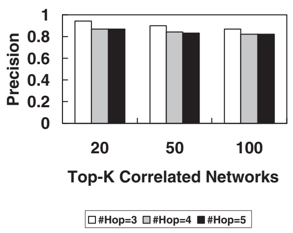


Fig. 16. Precision Evaluation for CiteSeer

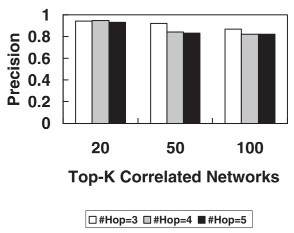


Fig. 17. Precision Evaluation for Wikipedia

larger in Wikipedia dataset due to the maintenance of the big number of results in no-shingling method.

We know it is not easy to verify the precision of discovering correlated networks. To measure the precision in this work, we take top 20, 50, 100 most correlated networks discovered by the no-shingling approach for a keyword query, which can produce three sample node sets, denoted as SNS_{20}, SNS_{40}, and SNS_{100}, respectively. And then, we apply our shingling algorithms to compute the top 20, 50, 100 most correlated networks and generate three node sets, denoted as NS_{20}, NS_{40}, and NS_{100}, respectively. The precision of the query evaluation can be calculated by

$$(\frac{|NS_{20} \cap SNS_{20}|}{|SNS_{20}|} + \frac{|NS_{40} \cap SNS_{40}|}{|SNS_{40}|} + \frac{|NS_{100} \cap SNS_{100}|}{|SNS_{100}|})/3$$

To make our evaluation as unbiased as possible, we use the average precision of the selected keyword queries as the precision of our method with regards to each dataset, as shown in Figure 15, Figure 16 and Figure 17, respectively. This experiment shows our proposed shingling method can retrieve about 80% of nodes of the no-shingling method, with less time and space in most cases.

VI. Related Work

Our work is related to graph keyword search. Most of the approaches to keyword search over graphs find trees as answers. In [2], a backward search algorithm for producing Steiner trees is presented. A dynamic programming approach for finding Steiner trees in graphs are presented in [23]. Although the dynamic programming approach has exponential time complexity, it is feasible for input queries with small number of keywords. In [3], the authors proposed algorithms that produce Steiner trees with polynomial delay. Due to the NP-completeness of the Steiner tree problem, producing trees with distinct roots are introduced in recent years [4]. BLINKS improves the work of [4] by using an efficient indexing structure [5]. There are other two methods that find subgraphs rather than trees for keyword search over graphs [6], [7]. The first method finds γ-radius Steiner graphs that contain all of the input keywords [6]. The second method finds multi-centered subgraphs, called communities [7]. In each community, there are some center nodes. There exists at least a single path between each center node and each content node such that the distance is less than R_{max}. Parameter R_{max} is used to control the size of the community. [8] further improves the semantics of results [7] by finding γ-cliques, in which all the content

Authorized licensed use limited to: Griffith University. Downloaded on August 25,2020 at 06:00:36 UTC from IEEE Xplore. Restrictions apply.

nodes are close to each other while it reduces the irrelevant nodes by producing a Steiner tree (instead of a graph) to reveal the relationship between the content nodes in a γ-clique. All the above keyword search methods mainly compute the minimal keyword matched subtrees or subgraphs as the returned results. However, in this work, we need to return the correlated networks, in which each correlated network consists of more than one maximal covering keyword matched subgraphs. Therefore, many indexes and pruning algorithms in the above works are not applicable to our work.

Our work is also related to dense subgraph. We have seen various techniques for discovering densest or top-k dense subgraphs in the recent data mining literature. [9] uses the methodology of graph partitioning/clustering - the matrix blocking method to obtain dense partitions/clusters. [11] proposes an approximation algorithm to compute the densest subgraphs using MapReduce. [14] presents the local algorithm to compute local dense subgraphs at many starting vertices. Based on the computed local dense subgraphs, the densest subgraph can be efficiently computed in an efficient global approximation algorithm. [15] computes large and dense subgraphs in massive graphs based on a recursive application of fingerprinting via shingles [16], [17]. [12] studies the dense subgraph discovery in a top-k fashion, by which users don't need to specify threshold. [10] addresses the maintenance issue of discovering dense subgraphs by maintaining a small number of sparse subgraphs under streaming edge weight updates. With the knowledge of a few important nodes as sources, the Web can be considered a flow network and max-flow/min-cut algorithms can be applied to identify communities centered at the source nodes [24]. In [25], bipartite graphs are considered and dense subgraphs are iteratively grown by using local search heuristics. However, all the above methods cannot be applied to the applications where users are only interested in the analysis of the graph nodes relating to a set of user-specified keywords. By discovering these keyword-based dense subgraphs (or communities or networks), our work can be used to present more specifically analyzed information to users.

VII. Conclusion

This paper proposed the problem of computing keyword-based correlated networks over a large graph G. Given a keyword query and a hop number γ, we can return a set of correlated networks in G, from which users can see big pictures about their interested information in G. To solve the problem, we firstly devised a novel tree data structure transformed from G by only maintaining the shortest paths with distance up to Γ ($\Gamma \geq \gamma$) in G. Based on the transformed tree, we can greatly improve the efficiency of computing γ-bounded keyword matched subgraphs. We then explored shingling algorithm to efficiently measure the correlations among these subgraphs. Finally, we demonstrated the efficiency and effectiveness of our proposed approaches based on the experimental results over the three real datasets, i.e., CNET Forum, CiteSeer and Wikipedia.

Acknowledgment

This work was supported by the ARC Discovery Projects under Grant No. DP110102407 and DP120102627. We thank all the reviewers for their helpful comments.

References

[1] C. C. Aggarwal and H. Wang, "Graph data management and mining: A survey of algorithms and applications," in *Managing and Mining Graph Data*, 2010, pp. 13–68.
[2] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, "Keyword searching and browsing in databases using banks," in *ICDE*, 2002, pp. 431–440.
[3] K. Golenberg, B. Kimelfeld, and Y. Sagiv, "Keyword proximity search in complex data graphs," in *SIGMOD Conference*, 2008, pp. 927–940.
[4] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar, "Bidirectional expansion for keyword search on graph databases," in *VLDB*, 2005, pp. 505–516.
[5] H. He, H. Wang, J. Yang, and P. S. Yu, "Blinks: ranked keyword searches on graphs," in *SIGMOD Conference*, 2007, pp. 305–316.
[6] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou, "Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data," in *SIGMOD Conference*, 2008, pp. 903–914.
[7] L. Qin, J. X. Yu, L. Chang, and Y. Tao, "Querying communities in relational databases," in *ICDE*, 2009, pp. 724–735.
[8] M. Kargar and A. An, "Keyword search in graphs: Finding r-cliques," *PVLDB*, vol. 4, no. 10, pp. 681–692, 2011.
[9] J. Chen and Y. Saad, "Dense subgraph extraction with application to community detection," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 7, pp. 1216–1230, 2012.
[10] A. Angel, N. Koudas, N. Sarkas, and D. Srivastava, "Dense subgraph maintenance under streaming edge weight updates for real-time story identification," *PVLDB*, vol. 5, no. 6, pp. 574–585, 2012.
[11] B. Bahmani, R. Kumar, and S. Vassilvitskii, "Densest subgraph in streaming and mapreduce," *PVLDB*, vol. 5, no. 5, pp. 454–465, 2012.
[12] E. Valari, M. Kontaki, and A. N. Papadopoulos, "Discovery of top-k dense subgraphs in dynamic graph collections," in *SSDBM*, 2012, pp. 213–230.
[13] A. J. T. Lee, M.-C. Lin, and C.-M. Hsu, "Mining dense overlapping subgraphs in weighted protein-protein interaction networks," *Biosystems*, vol. 103, no. 3, pp. 392–399, 2011.
[14] R. Andersen, "A local algorithm for finding dense subgraphs," in *SODA*, 2008, pp. 1003–1009.
[15] D. Gibson, R. Kumar, and A. Tomkins, "Discovering large dense subgraphs in massive graphs," in *VLDB*, 2005, pp. 721–732.
[16] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig, "Syntactic clustering of the web," *Computer Networks*, vol. 29, no. 8-13, pp. 1157–1166, 1997.
[17] K. Bharat, A. Z. Broder, J. Dean, and M. R. Henzinger, "A comparison of techniques to find mirrored hosts on the www," *IEEE Data Eng. Bull.*, vol. 23, no. 4, pp. 21–26, 2000.
[18] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," *J. Comput. Syst. Sci.*, vol. 60, no. 3, pp. 630–659, 2000.
[19] H. Jiang, W. Wang, H. Lu, and J. X. Yu, "Holistic twig joins on indexed xml documents," in *VLDB*, 2003, pp. 273–284.
[20] S. Chakrabarti, "Mining the web: Discovering knowledge from hypertext data," *Morgan Kaufmann*, 2002.
[21] A. Z. Broder, "Identifying and filtering near-duplicate documents," in *CPM*, 2000, pp. 1–10.
[22] Z. E. Gharghe and B. M. Bidgoli, "Weighted shingling: an adaptation of shingling for weighted shingles," in *Proceedings of the 6th international conference on Innovations in information technology*, ser. IIT'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 161–165. [Online]. Available: http://dl.acm.org/citation.cfm?id=1802274.1802307
[23] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin, "Finding top-k min-cost connected trees in databases," in *ICDE*, 2007, pp. 836–845.
[24] G. W. Flake, S. Lawrence, and C. L. Giles, "Efficient identification of web communities," in *KDD*, 2000, pp. 150–160.
[25] R. Kumar, U. Mahadevan, and D. Sivakumar, "A graph-theoretic approach to extract storylines from search results," in *KDD*, 2004, pp. 216–225.

Authorized licensed use limited to: Griffith University. Downloaded on August 25,2020 at 06:00:36 UTC from IEEE Xplore. Restrictions apply.