1-2013

Hypergraph Index: An Index for Context-aware Nearest Neighbor Query on Social Networks

Yazhe WANG
Singapore Management University, yazhe.wang.2008@smu.edu.sg

Baihua ZHENG
Singapore Management University, bhzheng@smu.edu.sg

Citation

WANG, Yazhe and ZHENG, Baihua. Hypergraph Index: An Index for Context-aware Nearest Neighbor Query on Social Networks. (2013). *Social Network Analysis and Mining*. 3, (4), 813-828.
Available at: https://ink.library.smu.edu.sg/sis_research/1835

Noname manuscript No.
(will be inserted by the editor)

Hyper-Graph Index: an Index for Context-Aware Nearest Neighbor Query on Social Networks

Yazhe Wang · Baihua Zheng

Received: date / Accepted: date

Abstract Social network has been touted as the No. 2 innovation in a recent IEEE Spectrum Special Report on "Top 11 Technologies of the Decade", and it has cemented its status as a bona fide Internet phenomenon. With more and more people starting using social networks to share ideas, activities, events, and interests with other members within the network, social networks contain a huge amount of content. However, it might not be easy to navigate social networks to find specific information. In this paper, we define a new type of queries, namely *context-aware nearest neighbor (CANN)* search over social network to retrieve the nearest node to the query node that matches the textual context specified. The textual context of a node is defined as a set of keywords that describe the important aspects of the nodes. CANN considers both the network structure and the textual context of the nodes, and it has a very broad application base.

Two existing searching strategies can be applied to support CANN search. The first one performs the search based on the network distance, and the other one conducts the search based on the node context information. Each of these methods operates according to only one factor but ignores the other one. They can be very inefficient for large social networks, where one factor alone normally has a very limited pruning power. In this paper, we design a *hyper graph* based method to support efficient approximated CANN search via considering the network structure and nodes' textual contexts simultaneously. Experimental results show that the hyper graph based method provides approximated results efficiently with low preprocessing and storage costs, and is scalable to large social networks. The approximation quality of our method is demonstrated based on both theoretical proofs and experimental results.

School of Information Systems, Singapore Management University
Tel.: +65-68280915
Fax: +65-68280919
E-mail: {yazhe.wang.2008, bhzheng}@smu.edu.sg

1 Introduction

Social network has been touted as the No. 2 innovation in a recent IEEE Spectrum Special Report on "Top 11 Technologies of the Decade", and it has cemented its status as a bona fide Internet phenomenon. Reported by Nielsen, the world's leading marketing and media information company, the social networks and blogs reach nearly 80% of active Internet users and represent the majority of users' time online in U.S. Similarly, it was reported that 73% of UK Internet users rely on social networks. Take Facebook, one of the most famous and successful social networking websites, as an example. The average Facebook user spends around 8 hours per month on Facebook, and Facebook is rapidly approaching 1 billion users soon. Obviously, more and more people start using social networks to share ideas, activities, events, and interests with other members within the network, and social networks contain a huge amount of content. However, it might not be easy to navigate social networks to find specific information. Consequently, we focus this paper on querying social networks.

Without loss of generality, the social network is modeled as an undirected weighted graph, with nodes representing the network users and edges representing the social connections between the users. The weight on each edge represents the inverse strength of the social connection. Besides the graph structure, we also consider the textual content related to the users, e.g., the key features of the users' profile, the keywords of the users' publication, and the tags of the users' shared resource. We abstract these textual information into a set of keywords as the context of the nodes. Take the co-authorship network G depicted in Figure 1 as an ex-

ample. Each node represents a scientist. An edge between two nodes states that those two scientists have collaborated at least once, and the weight on the edge is the inverse number of collaborations. For the simplicity of discussion, we assume the weight on every edge is 1 in this example. The contexts of the nodes include the users' name, profession, and research keywords, as depicted in Figure 1. Although majority of the nearest neighbor algorithms focus on the structure of the graph, we believe that both the structure and the context information of the nodes are important on social networks. Therefore, in this paper, we propose a *context-aware nearest neighbor (CANN)* query to search over social network based on both the network structure and context information. It retrieves the nearest node to the query node that matches the context specified, as well as the shortest path between them. For example, Michael (i.e., node v_3) may issue a CANN query Q_1 "finding me the shortest path to the nearest professor working on data mining". Here, *distance* from the query node v_3 to a node v is evaluated by the shortest path distance. A smaller distance indicates a stronger connection between the query node and the target node, and the shortest path gives the information of how the two nodes are connected. The queried *context* is represented by keywords $\{$*professor*, *data mining*$\}$. The answer to Q_1 is node v_4 with the shortest path $\{v_3, v_4\}$ as v_4 is the node which is the nearest to v_3 and contains the queried keywords in its context. CANN query considers two key factors on social network, i.e., network distance and the context, and it has broad application base. For example, scientists can issue CANN to find potential collaborators to start new research and employers can issue CANN to locate qualified employees to work on specific tasks.

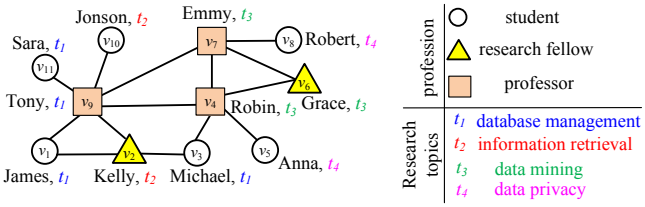


Fig. 1 A collaboration social network G.

The newly proposed CANN search shares some similarity with the social web search problem studied in [36] that considers both the network structural and textual factors in their objective function to retrieve the relevant user generated documents on the social web. The corresponding approaches proposed to conduct social web search, including *SI-based search* and *TR-based search*, are also applicable to CANN query. SI-based search considers distance factor first and examines the network users based on their distances to the queried node. To be more specific, it invokes traditional shortest path search algorithm (e.g., Dijkstra's algorithm) to visit nodes based on ascending order of their distances to the query node, and determines the corresponding textual relevancy on the run. TR-based search conducts the search based on the textual factor. It locates the textually relevant users first as the intermediate results, and then orders them based on their distances to the queried node. Both approaches are *inefficient* especially for large social networks, as they only consider one factor which results in very limited pruning power. On the one hand, SI-based approach traverses the social network purely based on the distance but not context. Consequently, when the answer node is relatively far from the queried node, it has to visit many unnecessary nodes before the answer node is reached. On the other hand, TR-based approach may find many nodes that match the queried keywords as intermediate results, especially when the queried keywords are popular, and hence the ranking process based on the distances between query node and *all* the intermediate nodes could be very costly.

Given the fact that existing approaches cannot support CANN search efficiently, we, in this paper, propose a *hyper-graph index* to support *high-performance* approximated CANN (ACANN) searches via considering the distance and the context factors simultaneously. The hyper-graph index consists of a *hyper graph* and *local signature-maps*. The hyper graph is formed among a set of selected *center nodes* that are very likely to be crossed by many shortest paths in the network, with hyper edges created between center nodes. Each center node maintains a local signature-map for a local cluster of nodes that center around it. The local signature map records the abstracted context and distance information of the nodes within each cluster. Compared with existing methods, the hyper-graph based search mainly conducts the search by traversing the hyper graph, which is much smaller than the whole network, and thus is more efficient. In addition, the local signature map facilitates the search space pruning based on queried keywords and distance simultaneously.

In order to demonstrate the efficiency of hyper-graph based search, we first conduct a theoretical study to derive the upper bound of approximation error, and then conduct a comprehensive simulation study to evaluate the performance of the proposed approach in terms of the index construction cost, storage cost, query time, and approximation quality.

The rest of the paper is organized as follows. Section 2 defines CANN search and approximated CANN search, together with a review of the related work. Section 3 presents the details of the hyper-graph based index, and the ACANN search algorithm is described in Section 4. Section 5 introduces a simplified variant of hyper-graph based index. Finally, Section 6 reports the experimental results, and Section 7 concludes this paper.

2 Preliminary

In this section, we first describe the graph model of the social network, and then formally define the *context-aware nearest neighbor (CANN)* query and approximated CANN query (ACANN). In addition, we also briefly review existing work related to CANN search, including (approximated) shortest path search, keyword query and signature technique used in information retrieval.

2.1 Problem Definition

In general, we model a social network as an undirected graph $G(V, E)$, with V being a set of nodes and E being the set of edges. An edge $e(v_i, v_j) \in E$ represents that nodes v_i and v_j are connected in the network. The weights of edges are captured by W. A non-negative weight $w(v_i, v_j) \in W$ of edge $e(v_i, v_j) \in E$ represents the inverse strength of the linkages, i.e. smaller weight indicates stronger social connection. In this paper, we assume that the context of each node $v_i \in V$ is maintained as a set of keywords, denoted as $v_i.k$. The domain of keywords for a graph G is represented by L with $L = \cup_{v_i \in V} v_i.k$. Given two nodes v_i and v_j of a graph $G(V, E)$, a path and the shortest path connecting them is described in Definition 1.

Definition 1 ***Path and Shortest Path.*** *Given a social network $G(V, E)$ and two nodes $v_i, v_j \in V$, a path $P(v_i, v_j)$ connecting v_i and v_j sequentially passes nodes $v_{p_1}, v_{p_2}, \cdots, v_{p_m}$, denoted as $P(v_i, v_j) = \{v_{p_0}, v_{p_1}, v_{p_2}, \ldots, v_{p_m}, v_{p_{m+1}}\}$, with $v_{p_0} = v_i$ and $v_{p_{m+1}} = v_j$. The length of $P(v_i, v_j)$, denoted as $|P(v_i, v_j)|$, is $\sum_{n=0}^{m} w(v_{p_n}, v_{p_{n+1}})$. The shortest path $SP(v_i, v_j)$ is the one with the shortest distance among all the paths between v_i and v_j, and its disance, denoted as $||v_i, v_j||$ $(= |SP(v_i, v_j)|)$, is the shortest distance between v_i and v_j.* □

Take the social network shown in Figure 1 as an example. Path $P(v_1, v_3) = \{v_1, v_9, v_4, v_3\}$ is a path from v_1 to v_3 via nodes v_9 and v_4, and path $P'(v_1, v_3) = \{v_1, v_2, v_3\}$ is another one via v_2. Assume $G(V, E)$ is an unweighted graph with $\forall e(v_i, v_j) \in E$, $w(v_i, v_j) = 1$, the path $P'(v_1, v_3)$ is the shortest path between v_1 and v_3, i.e., $SP(v_1, v_3) = \{v_1, v_2, v_3\}$ and $||v_1, v_3|| = |SP(v_1, v_3)| = w(v_1, v_2) + w(v_2, v_3) = 2$.

With $v_j.k$ capturing the context of v_j, CANN search is to locate the nearest node with its context matching the queried keywords, as given in Definition 2.

Definition 2 ***Context-aware Nearest Neighbor Search*** *(CANN). Given a graph $G(V, E)$, a CANN search Q specifies a query node $Q.v$ and a set of queried keywords $Q.k$. It asks for a shortest path P to a node $v_j \in V$ such that the context of v_j matches queried keywords and its distance to $Q.v$ is the shortest among all the nodes with context matching $Q.k$. In other words, CANN(Q) = $\langle v_j, P \rangle \Rightarrow v_j.k \supseteq Q.k \wedge P = SP(Q.v, v_j)$, and meanwhile $\nexists v_i \in V$ such that $Q.k \subseteq v_i.k \wedge ||Q.v, v_i|| < |P|$.* □

Generally, the exact CANN search on large social networks is very expensive. In this paper, we explore approximated CANN search as defined in Definition 3.

Definition 3 ***Approximated CANN Search (ACANN).*** *Given a graph $G(V, E)$, an ACANN search Q specifies a query node $Q.v$ and a set of queried keywords $Q.k$. It returns a path P to a node $v_j \in V$ such that the context of v_j matches queried keywords. However, it does not guarantee that i) v_j is the nearest node that satisfies the query; or ii) P is the shortest path from $Q.v$ to v_j. The quality of the approximation is measured by the error rate of the length of the returned path of ACANN search to that of CANN query, i.e., $\frac{|ACANN(Q).P| - |CANN(Q).P|}{|CANN(Q).P|}$.* □

2.2 Related Work

2.2.1 Shortest Path Search

The point-to-point shortest path problem has been extensively studied. It retrieves the shortest path from a source node to a *given* destination node in the network. However, our CANN query requires the shortest path from the source node to an *unknown* destination that is nearest to the source node and the context matches the queried keywords. Thus many efficient shortest path algorithms cannot be directly applied to our problem. However, some of them could be utilized as the sub-function for the distance related calculation.

The most well-known shortest path search algorithm on graphs is the Dijkstra's algorithm [8]. It explores the graph in a best-first manner starting from the query node until the target node is reached. The existing SI-based search and TR-based search use this algorithm for distance based exploration. Some more efficient solutions were proposed to reduce the graph exploration space based on domain-specific heuristic and pre-processing. For example, A* search [13] uses estimation on distance to the destination to guide node selection in the search from the source, thus is more efficient than the Dijkstra's algorithm. In our simulation, we use A* instead of Dijkstra in the TR-based search for retrieving the shortest paths from the query node to the intermediate result nodes. Many of these approaches use heuristics that are datasets-dependent (e.g. GIS data), which cannot be extended to other graphs, e.g., the reach based method in [14]. A comprehensive survey of these algorithms is reported in [12].

In recent years, efficient indexing techniques have been designed for shortest path search on large graphs. Some index techniques are designed based on partial pre-computation. For example, HEPV [18] and HiTi [19] build index based on partition graphs, and maintain local shortest paths related to the boundary nodes. The global shortest path is then obtained by combining selected local shortest paths. ROAD [24, 25] organizes a large road network as a hierarchy of interconnected regional sub-networks (called Rnets) and augments Rnets with shortcuts and object abstracts to accelerate network traversals and provide quick object lookups, respectively. These methods work well on planar graphs (e.g. road networks), where a small number of boundary nodes can be constructed easily. However, they could be very *inefficient* on non-planar graphs (e.g. social networks), where the number of boundary nodes is large. TEDI [34] is another graph partition based index approach. It partitions the graph based on the tree decomposition that is in general expensive on graphs. TEDI supports efficient shortest path retrieval for any given pair of source and destination nodes by a bottom-up operation. However, its construction and storage costs are expensive on large graphs, and it cannot be easily extended to support efficient network exploration from a given source node to find an "unknown" destination which satisfies certain context constraint.

There are other works considering encoding all-pairs shortest paths of a graph in small-sized indexes. For instance, a quadtree-structured index is proposed in [31] to utilize the spatial coherence of the destination (or source and destination) nodes. Distance signature method [16] pre-computes the distance from each node v to a set of objects of interests, and maintains this information as a signature at v. Compact BFS-tree [35] is another example. It exploits symmetry properties of graphs to reduce the index size of all-pairs shortest paths, but is only applicable to unweighted graphs. These discussed approaches require pre-computing the shortest paths between all-pairs of nodes, which is extremely expensive on large or even middle sized graphs. Therefore, in this work, we try to avoid these all-pair shortest paths based methods.

Approximated shortest path/distance problem has been studied as well. Spanner [6] is a subgraph obtained by deleting edges from the original graph. Due to the smaller size, the search performed on the spanner is much faster. However, it is hard to decide which edges to delete to generate a good spanner with the distances between nodes not changing substantially. Spanners do not perform well on dense graphs with large girth. Distant labeling and embedding techniques [11, 30] assign each node of a graph a label such that the (approximated) distance between two nodes can be directly computed based on the corresponding labels. However, these approaches can only provide distance information but not the paths. For the CANN, the shortest path from the query node to the destination node gives the information about how the two users are connected, thus is important for many applications. Therefore, these approaches are not applicable to the CANN problem.

2.2.2 Keyword Query

Keyword query on graphs considers both the structure and context information. Given a set of queried keywords, it is to find closely connected clusters of nodes in the graph which contain all the specific keywords. Some of the works search for rooted subtrees with the leaves containing the queried keywords. The results are ranked based on the distances from the roots to the leaf nodes. For instance, backward expansion algorithm and bidirectional search are proposed in [17, 20] respectively to solve this problem; while a bi-level index is designed in [15] to improve the search efficiency. Differently, [27] performs the keyword query on graphs to retrieve connected subgraphs which contain the queried keywords, then the results are ranked based on the keywords matching score and the compactness of the sub-graph. The issue of processing the keyword query on external memory graphs has been also discussed in [7]. Obviously, keyword query problems studied by the above works are different from our CANN search.

Recently, the idea of integrating social networks to keywords based document search has drawn much attention [4, 32, 33, 36]. The social affinity between the users is incorporated to improve the quality of the document search results. Some of the works consider this problem under complex social tagging systems, and the social affinity is evaluated based on the number of common tags [4, 32]. Those works usually emphasize on the effectiveness of the results more than the efficiency of the search process. Some other works define social affinity based on the network distance [33, 36]. They retrieve documents based on two factors, i.e., the relevancy of the documents to the queried keywords, and the network distances from the query node to the authors of the documents. Although their purposes are different, their query conditions are quite similar to our CANN search. Consequently, some of the proposed search strategies can be extended to CANN search, e.g., the SI-based search and the TR-based search proposed in [36]. The SI-based search examines the documents' authors based on their distance to the query node, while the TR-based strategy conducts the search by locating the most textually relevant documents first, and then examining the shortest path distances from their authors to the query node. However, in this paper, we conduct the CANN search by considering the distance and textual relevancy simultaneously. To avoid the expensive shortest path distance computation, a distance labeling technique is adopted in [33] to quickly estimate the distance between any two users. However, distance labeling does not provide

shortest path information that is requested by CANN search and hence is not applicable to CANN search.

2.2.3 Node betweenness centrality

The betweenness centrality of a node in a graph is defined as the number of shortest paths from all nodes to all others that pass through that node. It is an important measurement in graph theory that determins the relative importance of a node in the graph. The betweenness centrality and its variances have been wildly studied [10, 29, 21], and proved to be useful for many network analysis applications, such as finding keystone species in pollination networks [1], understanding the interaction patterns of the players of massive online games [2], and identifying significant nodes in wireless ad hoc networks [28]. In this work, we explore the usage of the node betweenness as the criteria to select the center nodes to build efficient index structure on social networks to support CANN queries.

2.2.4 Signature

Signature techniques have been studied extensively in information retrieval [22, 26]. A signature is basically an abstract of the keywords information of a data item. Given a set of keywords that represent the data item i, the signature S_i is typically formed by first hashing each keyword in the set into a *bit string* and then *superimposing* (i.e., bitwise-OR, $\vee$) all these bit strings into a signature. Note that the size of a signature equals to the size of the bit string. An example of signature generation is depicted in Figure 2(a), in which each keyword is hashed into a 12-bit string.

Data Item:	***Attr. 1: Security***	***Attr. 2: Pervasive***
	Security	***001 100 001 001***
	Pervasive ∨)	***101 000 100 001***
Data Signature	***Si***	***101 100 101 001***

(a) Signature generation

Query Q	***Query Signature S_Q***	***$S_Q \wedge S_i$***	***Results***
Hacker	***000 101 000 101***	***000 100 000 001***	***No Match***
Security	***001 100 001 001***	***001 100 001 001***	***True Match***
Mobile	***100 100 001 001***	***100 100 001 001***	***False Match***

(b) Query comparision

Fig. 2 Signature generation and comparison

To decide whether a data item i matches/contains the queried keyword Q, a *query signature* S_Q is generated first, based on the same hash function. Thereafter, S_Q is compared against the signature S_i using bitwise-AND ($\wedge$). There are two possible outcomes of the comparison:

- $S_Q \wedge S_i \neq S_Q$: data item i does not match query Q.
- $S_Q \wedge S_i = S_Q$: a match has two possible implications:
  - *true match*: the data item is really what the query searches for; and
  - *false drop* (or *false match*): the data item in fact does not satisfy the search criteria although the signature comparison indicates a match.

As shown in Figure 2(b), three queries are issued and their corresponding signatures are produced based on the same hash function. According to the result of $S_Q \wedge S_i$, the examined data item is not qualified for the first query, Q_1 = Hacker, but qualified for the other two queries, Q_2 = Security and Q_3 = Mobile. It is a true match for Q_2 as the data item does contain the keyword Security while it is a false drop for Q_3 because the data item does not contain the keyword Mobile. *Bloom Filter*, commonly used in networking, shares some similarities as signature [3]. However, it adopts multiple hash functions to generate bit strings.

3 Hyper-Graph based Index

To support efficient ACANN search, we design a novel index structure, namely *hyper-graph based index*. It consists of two parts, a *hyper graph* and the *local signature maps*. In this section, we present the detailed structures of these two parts and their constructions respectively.

3.1 Hyper Graph

Given the fact that the retrieval of shortest paths is expensive on large graphs and some applications are willing to trade efficiency for accuracy, we design a *hyper graph structure* based on graph partition and partial pre-computation to support approximated shortest path retrieval. To be more specific, we first identify a small set of center nodes in the graph, and divide the graph into disjoint partitions P_i with each around one center node c_i. The shortest paths from a center node c_i to every node of its corresponding partition P_i are computed. Then, we form the hyper graph with the center nodes as the hyper nodes, and generate hyper edges between every pair of center nodes. Each hyper edge represents the shortest path between the corresponding center nodes. The formal definition of hyper graph is given in Definition 4.

Definition 4 ***Hyper Graph.*** *Given a social network $G(V, E)$ and a set of center nodes $C = \{c_1, c_2, \ldots, c_r\}$, the hyper graph $G_H(V_H, E_H)$ consists of the set of center nodes, and the connections between them, i.e., $V_H = C$, and $E_H = \cup_{c_i, c_j \in C \wedge |SP(c_i,c_j)| \neq \infty} e(c_i, c_j)$ with $w(c_i, c_j) = |SP(c_i, c_j)|$.* □

An example hyper graph is depicted in Figure 3, with nodes v_4, v_7, and v_9 being the center nodes. Given a hyper graph, the approximation of the shortest path from a source node v to a destination node u is performed as follows. We assume that v belongs to the partition P_i with the

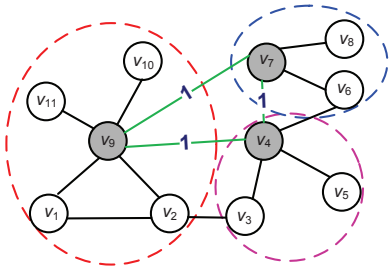


Fig. 3 An example of the hyper graph.

corresponding center node c_i, and u belongs to the partition P_j with the corresponding center node c_j. First, we get the shortest path $P_1(v, c_i)$ from v to the center nodes c_i within partition P_i. Then, based on the hyper graph, we can find the shortest path $P_2(c_i, c_j)$ from c_i to c_j. Finally, we get the shortest path $P_3(c_j, u)$ from c_j to u inside the partition P_j. The combination of $P_1(v, c_i)$, $P_2(c_i, c_j)$ and $P_3(c_j, u)$ then forms the approximated shortest path from v to u. We embed certain information in each center node to facilitate the retrieval of $P_1(v, c_i)$/$P_3(c_j, u)$ (to be explained in Section 3.2), and the hyper graph carries information of $P_2(c_i, c_j)$. Consequently, this approximation will be much faster than deriving the real shortest path $SP(v, u)$.

In the following, we explain three key steps of forming a hyper graph, namely *center node selection*, *graph partition*, and *hyper graph formation*.

Center Nodes Selection. We assume each shortest path will cross at least one center node in our shortest path approximation. Consequently, the center nodes have a direct impact on the quality of the shortest path and we need to carefully select the center nodes in order to improve the approximation quality. In this paper, we propose to choose the center nodes based on their betweenness scores. The betweenness score of a node counts the number of shortest paths that the node falls on. Intuitively, by selecting the nodes laying on many shortest paths (i.e. having highest betweenness score) as the center nodes, the approximated shortest path between two nodes, which passes their corresponding center nodes, will be close to the real shortest one. We will evaluate this intuition later in Section 6.

However, the betweenness scores are computationally expensive to derive as it requires computing all-pair shortest paths of a graph. Instead of computing the shortest paths for every pair of nodes, we adopt the algorithm proposed in [5] to efficiently fetch the set of nodes with highest betweenness. It chooses a set of nodes as pivots, and only computes the shortest paths from pivots to the other nodes. Then, it uses those partial shortest paths to estimate the nodes with highest betweenness. The algorithm runs this process iteratively. At each iteration, some new pivots are added and the highest betweenness node set is updated. The algorithm terminates if the membership of the highest betweenness node set remains largely stable.

Graph Partition. Once the center nodes are fixed, we partition the graph into disjoint sub-graphs with each centered at one center node. In this work, we assign each non-center node to its nearest center node based on the shortest path distance. When there is a tie (i.e., a non-center node has more than one closest center node), the non-center node is randomly assigned to one closest center node. Definition 5 gives a formal definition. It is noted that finding the closest center node for each non-center node is expensive if we derive the distances from each non-center node to all the center nodes. We adopt the parallel expansion algorithm proposed in [9]. It initiates graph expansion from all the center nodes concurrently. Let Exp_i represent the i-th expansion initiated at center node c_i. All the nodes reached by Exp_i first will be included into the partition corresponding to c_i. The worst time complexity of this algorithm is $O(|E|+|V|+|V-r|\log(|V-r|))$, where r is the number of center nodes.

Definition 5 ***Graph Partition.*** *Given a graph $G(V, E)$ and a set of center nodes $C = \{c_1, c_2, \cdots, c_r\}$ with $C \subset V$, a graph partition $\mathcal{P}_G = \{P_1, P_2, \cdots, P_r\}$ is a set of node sets P_i that i) $\forall c_i \in C, c_i \in P_i$; ii) $\forall i, j (i \neq j) \in [1, r]$, $P_i \cap P_j = \emptyset \wedge \bigcup_{1 \leq i \leq r} P_i = V$; and iii) $\forall v \in P_i \wedge \forall j (\neq i) \in [1, r]$, $||v, c_i|| \leq ||v, c_j||$.* □

Hyper Graph Formation. Once the graph is partitioned, we need to build the hyper graph. A hyper graph has the center nodes as hyper nodes and has hyper edges between every pair of center nodes that are reachable. Each hyper edge represents the shortest path between the corresponding center nodes. The weight on a hyper edge equals to the shortest path distance between the corresponding hyper nodes.

Figure 3 depicts an example of the hyper graph. Assume the number of center nodes is 3, and nodes v_4, v_7, and v_9 are selected as center nodes based on betweenness scores. Thereafter, the network partition takes place. Each non-center node is attached to its nearest center node as demonstrated by the dashed circle in Figure 3. Then, we proceed to form the hyper graph. Three hyper edges are built among the hyper nodes, i.e. $e(v_4, v_7)$, $e(v_7, v_9)$ and $e(v_4, v_9)$. As the shortest path distances between these nodes are all 1, the weight on every hyper edge is assigned as 1.

3.2 Local Signature Map

The hyper graph builds a graph skeleton that can facilitate the approximation on shortest paths by assembling some pre-computed shortest paths. As discussed earlier, if the source node v and the destination node u are known (i.e., the corresponding partitions P_i and P_j that v and u are located respectively are known), its shortest path can be approximated by the combination of the shortest path from v to the center node c_i, the shortest path from c_i to c_j, and the shortest path

from u to c_j. However, for a given ACANN search, the destination node is unknown. In other words, we do not know who the destination node is and where it is located. We have to traverse the hyper graph to visit the center nodes by certain order, and then at each center node c_i, we scan the nodes within the corresponding partition P_i to see whether any node satisfies the query condition. To speedup this checking process of each partition, we introduce a new structure, namely *local signature map*, to carry some abstract information about the shortest distances from non-center nodes to the center node and textual contexts of the non-center nodes in the corresponding partition. It can effectively filter out those partitions that do not contain any answer node.

Given a center node c_i with corresponding partition P_i, its local signature map incorporates two parts, i.e., *textual signature* and *distance lower bound*. The textual signature sig represents the keyword set L_i in a compact manner, with L_i ($= \cup_{v \in P_i} v.k$) summarizing the textual contexts of the nodes in P_i. As explained in Section 2.2.4, for a given queried keyword $Q.k$, we can compare sig against the signature of $Q.k$. If the signatures do not match, it is certain that L_i does not contain $Q.k$ (i.e., no node within P_i satisfies the textual condition of the query) and P_i can be safely filtered out. Otherwise, a signature match occurs which implies two possible implications, i.e., true match and false drop. Since each false drop results in *unnecessary* exploration of the corresponding partition, a longer signature is preferred in order to maintain a low false drop rate. However, for a large social network, each partition contains different number of nodes. Some partitions could be very large in size, and the corresponding L_i could be large as well. To make our approach scalable, we propose to use multiple relatively small signatures, instead of one single long signature, to represent L_i. Given the pre-defined false drop rate threshold γ and the length $|sig|$ of the signature, we can derive the maximum number of keywords that a signature can represent based on Equation (1) [23], denoted as η. Then, we partition the nodes of P_i into subgroups g_i^l such that the number of keywords corresponding to the nodes inside each subgroup does not exceed η. A signature is generated based on each subgroup g_i^l, and the local signature map contains all these signatures.

$$\eta = \lfloor \frac{|sig| \cdot (log_e 2)^2}{-log_e \gamma} \rfloor \quad (1)$$

Now the issue is how to group nodes into subgroups. In this work, we group the nodes in P_i based on their distances to c_i. To be more specific, nodes in P_i are clustered into x_i groups with each group g_i^l ($1 \leq l \leq x_i$) associated with a distance parameter d_i^l. All the nodes within the group have their shortest distances to c_i bounded by the range $[d_i^l, d_i^{l+1})$ with $d_i^{x_i+1} = \infty$, i.e., $\forall v_j \in g_i^l$, $||v_i, v_j|| \in [d_i^l, d_i^{l+1})$. The distance parameter d_i^l is stored with the textual signature sig of the nodes group. The advantage of attaching the distance information is that it can be used to estimate the (approximated) distance from the center node c_i to the potential qualified node in the partition, thus helps to further prune the search space. To be more specific, during ACANN search, when a temporary answer node is found with distance d, the following search can discard the node groups whose estimated lower bound distances are no smaller than d, even when their contexts match the queried keywords. This is because these node groups cannot contain an answer node whose distance is smaller than the current answer d. In order words, only the node group whose lower bound distance is smaller than the current answer node is worth further examination.

Algorithm 1 Local Signature Map Construction

Input: a center node c_i and the partition P_i, η, a hash function $h()$;
Output: map_i;
Procedure:
1: order the nodes in P_i based on ascending order of their shortest distances to c_i, so that $\forall v_j \in P_i$, with $||v_j, c_i|| \leq ||v_{j+1}, c_i||$;
2: **for** each $v_j \in P_i$ **do**
3: **if** $j == 1$ **then**
4: $l = 1$;
5: $g_i^l = \{v_j\}$;
6: **else if** $|\cup_{v \in g_i^l} v.k \cup v_j.k| \leq \eta$ **then**
7: $g_i^l = g_i^l \cup \{v_j\}$;
8: **else**
9: $l = l + 1$;
10: $g_i^l = \{v_j\}$;
11: **for** each g_i^l **do**
12: $map_i[l].sig = \vee_{v \in g_i^l} h(v.k)$;
13: $map_i[l].dis = MIN_{v \in g_i^l} ||v_i, v||$;
14: return map_i;

To sum up, given a partition P_i that is divided into x_i disjoint node groups, the local signature map of the center node c_i, denoted as map_i, is the union of an array of x_i two-tuple vectors, denoted as $\langle map_i[l].sig, map_i[l].dis \rangle$, ($1 \leq l \leq x_i$). $map_i[l].sig$ is the textual signature of the lth group of P_i, and $map_i[l].dis$ is the distance lower bound representing the minimum distance from c_i to a node within the l-th group.

Algorithm 1 lists the pseudo-code of the local signature map construction. For a given center node c_i and its partition P_i, we first order the nodes $v_j \in P_i$ based on ascending order of their shortest distances to c_i (line 1). To simplify the discussion, we use v_j to represent nodes within P_i with $1 \leq j \leq |P_i|$, and assume that $||v_j, c_i|| \leq ||v_{j+1}, c_i||$. Next, we cluster nodes into groups (lines 2-10). To be more specific, suppose v_j is within group g_i^l, and now we are examining v_{j+1}. If the total number of distinct keywords associated with nodes within g_i^l and node v_{j+1} does not exceed η, v_{j+1} is clustered into group g_i^l (lines 6-7). Otherwise, a new group g_i^{l+1} is generated with v_{j+1} as the initial value (lines 9-10). Finally, for each group g_i^l, we generate the signature $map_i[l].sig$ and record the distance lower bound

Table 1 Local signature map associated with v_7 ($\eta = 4$)

sig	dis	group member
$v_7.sig \vee v_6.sig$	0	v_7, v_6
$v_8.sig$	1	v_8

$map_i[l].dis$ (lines 11-13). We use this algorithm to generate signature map for every center node. Table 1 depicts the local signature map of center node v_7. Here, $v_i.sig$ means the signature of node v_i's context (i.e. $v_i.k$).

4 Approximated Search Algorithm

In this section, we discuss the detailed approximated search algorithm based on hyper-graph based index, and then provide a theoretical analysis on the approximation quality.

4.1 Basic Idea

As explained earlier, the shortest path from a node v in partition P_i to a node u in partition P_j can be approximated by three segments, i.e., the shortest path from v to c_i, the shortest path from c_i to c_j, and the shortest path from c_j to u. Since the hyper edge between c_i and c_j carries $||c_i, c_j||$, and $||v, c_i||$ and $||c_j, u||$ are known, the approximation is fast. Recall that the answer node of an ACANN search is unknown. Consequently, given an ACANN search issued at node $Q.v$, the challenge is to quickly find a center node c_j on the hyper graph, whose partition contains an answer node u that matches the queried keywords, and approximated distance $||Q.v, c_i|| + ||c_i, c_j|| + ||c_j, u||$ is the shortest.

In order to address this issue, we propose the *distance based network expansion* on the hyper graph. It starts the expansion from c_i with corresponding partition containing the query node $Q.v$. Every time a center node c_j is visited, a *local examination* is performed in its corresponding partition P_j to find the answer node. In the following, we explain the details of the network expansion and local examination.

Algorithm 2 presents the pseudo code of ACANN search, and Table 2 lists the useful notations. First, we initialize the temporary answer node v_{ans} as empty and its approximated distance d_{ans} to be infinity (line 1). For an ACANN query Q issued at node $Q.v$, if the query node matches the queried keywords $Q.k$, the search terminates (lines 2-3). Otherwise, we perform a distance based expansion on the hyper graph. We first locate the center node c_i whose partition contains $Q.v$, as well as the the shortest distance from $Q.v$ to c_i, denoted as d_0 (line 4). Then, the local expansion starts from c_i. To enable the distance based expansion, we maintain two important structures, a distance set and a priority queue Que. Each element $d_H(c_i, c_j)$ in the distance set corresponds to the distance from c_i to another hyper node c_j with initial value ∞ (line 5-6). As the expansion proceeds, $d_H(c_i, c_j)$ will be updated as well. The priority queue Que carries two-tuple vectors $\langle c_j, d_H(c_i, c_j)\rangle$ that are ordered based on ascending order of $d_H(c_i, c_j)$, and it is initialized as $\langle c_i, d_H(c_i, c_i) = 0\rangle$ (line 7). Thereafter, we traverse the hyper graph by continuously dequeuing the head entry from Que until it becomes empty (line 8-24).

Table 2 Notation table

$G(V, E)$	social network with node set V and edge set E
$G_H(V_H, E_H)$	hyper graph constructed on G with hyper(center) node set V_H and hyper edge set E_H
Q, $Q.v$, $Q.k$	an ACANN query issued at query node $Q.v$, and the queried keywords $Q.k$
$h()$	hash function used to generate signature for a set of keywords
c_i	center node of partition P_i
map_i, $map_i[l]$	local signature map attached to c_i, and the map entry for the lth group
$map_i[l].sig$, $map_i[l].dis$, $map_i[l].nodes$	the signature, distance lower bound, and group members information recorded in the map entry of the lth group
$\|\|v_i, v_j\|\|$	the shortest distance between nodes v_i and v_j
$P(v_i, v_j)$	a path between nodes v_i and v_j
$SP(v_i, v_j)$	the shortest path between nodes v_i and v_j
Que	a priority queue to assist the distance based hyper graph expansion
d_0	the shortest distance from the $Q.v$ to its corresponding center node
$d_H(c_i, c_j)$	the shortest distance from c_i to c_j based on hyper graph
$e_H(c_i, c_j)$, $w_H(c_i, c_j)$	the hyper edge between c_i and c_j on the hyper graph, and the corresponding weight

Every time when a head entry $\langle c_j, d_H(c_i, c_j)\rangle$ is dequeued, the lower bound of the approximated distance from $Q.v$ to any node in partition P_j centered at c_j (i.e., $d_0 + d_H(c_i, c_j)$) is compared against d_{ans} (i.e., the approximated distance from $Q.v$ to the current answer node). If the lower bound is larger than d_{ans}, the partition P_j can be safely discarded as none of the nodes inside P_j will be closer to $Q.v$ than the current answer node. In addition, all the remaining entries $\langle c'_j, d_H(c_i, c'_j)\rangle$ in Que have their $d_H(c_i, c'_j)$ larger than $d_H(c_i, c_j)$ and hence their corresponding partitions $P_{j'}$ can be pruned away. In other words, none of the nodes in the unexplored partitions will satisfy the query condition $Q.k$ and meanwhile be closer to $Q.v$ than the current the answer node. We can safely terminate the current expansion and return the current answer node (lines 10-11). Otherwise, partition P_j needs local examination. We use the local signature map map_j attached to the center node c_j to facilitate the pruning. As explained before, the nodes in the partition P_j are clustered into groups g_j^l, and the pruning process is to filter out those groups that do not contain any node satisfying the query condition. The first filtering criterion is based on distance. We calculate $(d_0 + d_H(c_i, c_j) + map_j[l].dis)$, the lower bound of the approximated distance from a node in the lth group of P_j to $Q.v$. If it is longer than d_{ans}, there

Algorithm 2 ACANN Search based on Hyper-Graph Index

Input:a social network $G(V, E)$ with corresponding context L and weight W, a hash function $h()$, hyper graph $G_H(V_H, E_H)$ with local signature maps, an ACANN query Q.
Output:the answer node and the approximated shortest path.
Procedure:

1: $v_{ans} = \emptyset$, $d_{ans} = \infty$
2: **if** $Q.k \subseteq Q.v.k$ **then**
3: return $v_{ans} = Q.v, d_{ans} = 0, P_{ans} = \{Q.v\}$
4: locate the corresponding center node c_i of $Q.v$, and $d_0 = ||c_i, Q.v||$
5: **for** each $c_j \in V_H$, $(j \neq i)$ **do**
6: $d_H(c_i, c_j) = \infty$
7: $Que = \langle c_i, d_H(c_i, c_i) = 0\rangle$
8: **while** Que is not empty **do**
9: $\langle c_j, d_H(c_i, c_j)\rangle = dequeue(Que)$
10: **if** $(d_0 + d_H(c_i, c_j)) \geq d_{ans}$ **then**
11: return v_{ans}, P_{ans}
12: **for** each $map_j[l] \in map_j$ **do**
13: **if** $(d_0 + d_H(c_i, c_j) + map_j[l].dis) \geq d_{ans}$ **then**
14: break
15: **else if** $h(Q.k) \wedge map_j[l].sig == h(Q.k)$ **then**
16: **for** each $v_k \in map_j[l].nodes$ **do**
17: **if** $Q.k \subseteq v_k.k$ and $(d_0 + d_H(c_i, c_i) + ||c_j, v_k||) < d_{ans}$ **then**
18: $v_{ans} = v_k$;
19: $d_{ans} = d_0 + d_H(c_i, c_i) + ||c_j, v_k||$
20: $P_{ans} = append(SP(Q.v, c_i), P(c_i, c_j), SP(c_j, v_k))$
21: **for** each neighboring node c_n of c_j in G_H **do**
22: **if** $d_H(c_i, c_j) + w_H(c_j, c_n) < d_H(c_i, c_n)$ **then**
23: $enqueue(\langle c_n, d_H(c_i, c_j) + w_H(c_j, c_n)\rangle)$; $P(c_i, c_n) =$ append$(P(c_i, c_j), e_H(c_j, c_n))$
24: $d_H(c_i, c_n) = d_(c_i, c_j) + w_H(c_i, c_i)$

is no need to examine nodes within this group and the following groups (lines 13-14). The second filtering criterion is based on the textual context. We can safely discard the lth group if $map_j[l].sig$ does not match the query context $Q.k$. If this group g_j^l passes these two criteria, we examine the nodes in this group one by one, and update the answer when the nodes v_j in the group that match the search context are found (lines 15-19). Up to this point, we have examined the partition centered at c_j. We then continue to expand the search by inserting all the unexamined neighboring center nodes c_n of c_j in G_H for further examination (lines 21-24).

4.2 Approximation Quality

Compared with CANN search, ACANN search can effectively improve the search performance. However, the result node returned by ACANN search might not be the closest node, and the returned path might not be the shortest path. Consequently, it makes sense to replace CANN search with ACANN search only if a high approximation quality can be achieved. In the following, we conduct a theoretical study to analyze the approximation quality of our ACANN search. For a given query Q issued at the node $Q.v$ located in the partition centered at center node $c_{Q.v}$, we assume ACANN search returns answer node u that is located in the partition centered at center node c_u with corresponding approximated shortest path $ASP(Q.v, u)$. First, we analyze the difference between $|ASP(Q.v, u)|$ returned by ACANN and the real distance $||Q.v, u||$ between $Q.v$ and u, with its upper bound presented in Lemma 1. Next, we analyze the difference of $|ASP(Q.v, u)|$ compare to $|CANN(Q).P|$ in Lemma 2.

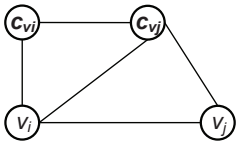


Fig. 4 A demonstration of the shortest paths between two nodes v_i and v_j, $(i \neq j)$ and their corresponding center nodes c_{v_i}, c_{v_j}.

Lemma 1 *Given a social network $G(V, E)$, a set of center nodes C, the corresponding social network partitions, and two nodes v_i, v_j, we assume v_i is within partition P_i and v_j is within partition P_j with $i \neq j$, and c_{v_i}, c_{v_j} are the corresponding center nodes respectively.* $(||v_i, c_{v_i}|| + ||c_{v_i}, c_{v_j}|| + ||v_j, c_{v_j}||) - ||v_i, v_j|| \leq 2 \times (||v_i, c_{v_i}|| + ||v_j, c_{v_j}||)$.

Proof. To facilitate the proof of this lemma, we construct a graph as shown in Figure 4. It includes nodes v_i and v_j, and center nodes c_{v_i} and c_{v_j}. An edge between two nodes in Figure 4 represents that those two nodes are reachable with the edge weight set to their shortest distance[1]. Based on the triangular inequality on triangles $\Delta_{v_i, c_{v_i}, c_{v_j}}$ and $\Delta_{v_i, c_{v_j}, v_j}$, the following inequalities are established.

$$\begin{aligned} ||c_{v_i}, c_{v_j}|| &\leq ||v_i, c_{v_i}|| + ||v_i, c_{v_j}|| \\ ||v_i, c_{v_j}|| &\leq ||v_j, c_{v_j}|| + ||v_i, v_j|| \end{aligned} \tag{2}$$

By adding the two inequalities in (2), we derive that

$$||c_{v_i}, c_{v_j}|| \leq ||v_i, c_{v_i}|| + ||v_j, c_{v_j}|| + ||v_i, v_j||$$

Obviously, the inequality below stands.

$$\begin{aligned} &(||v_i, c_{v_i}|| + ||c_{v_i}, c_{v_j}|| + ||v_j, c_{v_j}||) - ||v_i, v_j|| \\ &\leq 2 \times (||v_i, c_{v_i}|| + ||v_j, c_{v_j}||) \end{aligned}$$

■

Lemma 2 *Given a query node $Q.v$, and a corresponding hyper graph G_H, suppose the real nearest node which matches $Q.k$ is u^*, and the approximated nearest node which ACANN algorithm returns is u with corresponding approximated shortest path $ASP(Q.v, u)$, we define $\Delta = |ASP(Q.v, u)| - ||Q.v, u^*||$, then $\Delta \leq 4 \times \max_{1 \leq k \leq |V|} ||v_k, c_{v_k}||$ where c_{v_k} is v_k's nearest center node.*

[1] If there is no edge between two nodes, it does not mean these nodes are not reachable but mean that edge is not needed to prove this lemma.

Proof. As mentioned previously, the approximated shortest path from node $Q.v$ to node u^* passes their corresponding center nodes, i.e., center node $c_{Q.v}$ and center node c_{u^*}. In other words, the following equation holds. $|ASP(Q.v, u^*)| = ||Q.v, c_{Q.v}|| + ||c_{Q.v}, c_{u^*}|| + ||c_{u^*}, u^*||$. Based on Lemma 1, we have

$$|ASP(Q.v, u^*)| - ||Q.v, u^*|| \leq 2 \times (||Q.v, c_{Q.v}|| + ||u^*, c_{u^*}||).$$

Because that u is the nearest node found based on the approximated shortest path distance, $|ASP(Q.v, u)| \leq |ASP(Q.v, u^*)|$. Obviously, we have

$$\begin{aligned} \Delta &= |ASP(Q.v, u)| - ||Q.v, u^*|| \\ &\leq |ASP(Q.v, u^*)| - ||Q.v, u^*|| \\ &\leq 2 \times (||Q.v, c_{Q.v}|| + ||u^*, c_{u^*}||). \end{aligned}$$

For every node v_i, $||v_i, c_{v_i}|| \leq \max_{1 \leq k \leq |V|} ||v_k, c_{v_k}||$. Therefore, in the worst case,

$$\Delta \leq 4 \times \max_{1 \leq k \leq |V|} ||v_k, c_{v_k}||.$$

■

5 Simplified Hyper Graph

The hyper graph defined above contains all pairs shortest paths between any two center nodes that are reachable. When a large social network is considered, the number of center nodes might be big and consequently, the computation cost and storage overhead for the hyper graph is not negligible. Actually, we observe that the shortest path from a center node c_i to another center node must cross the boundary between c_i's partition P_i and one of its adjacent partitions. Partition adjacency is formally defined in Definition 6. In other words, the shortest path between two center nodes of two non-adjacent partitions could be approximated by the shortest paths between center nodes of adjacent partitions. Based on this observation, we propose *simplified hyper graph* that only includes the links between the center nodes of two adjacent partitions, as defined in Definition 7.

Definition 6 ***Partition Adjacency.*** *Given a social network partition $\mathcal{P}_G$ of a social network $G(V, E)$ and two partitions $P_i, P_j (i \neq j) \in \mathcal{P}_G$, P_i and P_j are adjacent to each other if and only if $\exists e(v_i, v_j) \in E$ such that $v_i \in P_i$ and $v_j \in P_j$, denoted as $P_i \wr P_j$.* □

Definition 7 ***Simplified Hyper Graph.*** *Given a social network $G(V, E)$ and a set of center nodes $C = \{c_1, c_2, \ldots, c_r\}$, the simplified hyper graph $G_{SH}(V_{SH}, E_{SH})$ consists of the set of center nodes, and the connections between those center nodes with their corresponding partitions being adjacent, i.e., $V_{SH} = C$, and $E_{SH} = \cup_{P_i \wr P_j \wedge |SP(c_i, c_j)| \neq \infty} e(c_i, c_j)$ with $w(c_i, c_j) = |SP(c_i, c_j)|$.* □

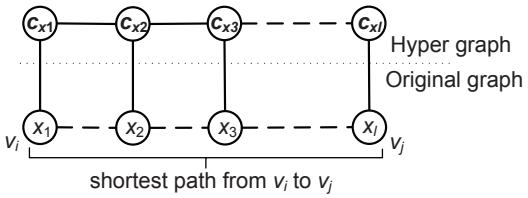


Fig. 5 A demonstration of the shortest path between nodes v_i and v_j, $(i \neq j)$ on simplified hyper graph.

The ACANN search algorithm presented in Algorithm 2 is still applicable on the simplified hyper graph. However, Lemma 2 is no longer valid for the simplified hyper graph. This is because, for two non-adjacent partitions, the path between those two center nodes c_i and c_j derived based on simplified hyper graph, denoted as $SP_{SH}(c_i, c_j)$, might not be the real shortest path, i.e., $|SP_{SH}(c_i, c_j)| \geq ||c_i, c_j||$. We re-derive the upper bound of the difference between the shortest distance returned by ACANN based on simplified hyper graph and the real shortest distance returned by CANN in Lemma 3.

Lemma 3 *Given a social network $G(V, E)$, and the simplified hyper graph G_{SH}, we assume the real nearest node that matches $Q.k$ is u^*, and the approximated nearest node returned by ACANN algorithm via simplified hyper graph is u. Let $ASP_{SH}(Q.v, u)$ denote the approximated shortest path from $Q.v$ to u based on G_{SH}. $\Delta = |ASP_{SH}(Q.v, u)| - ||Q.v, u^*|| \leq 2 \times p \times \max_{1 \leq k \leq |V|} ||v_k, c_{v_k}||$, here p is the maximum number of adjacent partitions that a shortest path on G can cross.*

Proof. First, given two nodes $v_i, v_j \in V$, $(i \neq j)$, we prove the difference between their approximated shortest path based on G_{SH} and their real distance. To simplify our discussion, we assume v_i is associated with the center node c_{x_1}, and v_j is associated with the center node c_{x_l}. In addition, we assume the real shortest path from v_i to v_j passes several adjacent partitions. In other words, $SP(v_i, v_j)$ crosses partitions $P_{x_1}, P_{x_2}, \cdots, P_{x_l}$ in sequence, with their corresponding center nodes represented as $c_{x_1}, c_{x_2}, \cdots, c_{x_l}$, as shown in Figure 5. Since $SP(v_i, v_j)$ passes P_{x_t} $(t \leq l)$, there must be at least one node x_t within partition P_{x_t} that contributes to $SP(v_i, v_j)$. Let the dashed lines depicted in Figure 5 represent the real shortest paths between x_t and x_{t+1}. According to Lemma 1, we have $(||x_1, c_{x_1}|| + ||c_{x_1}, c_{x_2}|| + ||x_2, c_{x_2}||) - ||x_1, x_2|| \leq 2 \times (||x_1, c_{x_1}|| + ||x_2, c_{x_2}||)$, which means $||c_{x_1}, c_{x_2}|| - ||x_1, x_2|| \leq ||x_1, c_{x_1}|| + ||x_2, c_{x_2}||$. Similarly, the following inequalities hold:

$$\begin{aligned} &||c_{x_1}, c_{x_2}|| - ||x_1, x_2|| \leq ||x_1, c_{x_1}|| + ||x_2, c_{x_2}||; \\ &||c_{x_2}, c_{x_3}|| - ||x_2, x_3|| \leq ||x_2, c_{x_2}|| + ||x_3, c_{x_3}||; \\ &\cdots \\ &||c_{x_{l-1}}, c_{x_l}|| - ||x_{l-1}, x_l|| \leq ||x_{l-1}, c_{x_{l-1}}|| + ||x_l, c_{x_l}||. \end{aligned}$$

Because $\{c_{x_1}, c_{x_2}, \cdots, c_{x_l}\}$ are the center nodes of a sequence of adjacent partitions, $P_{SH}(c_{x_1}, c_{x_l}) = \{c_{x_1}, c_{x_2}, \ldots, c_{x_l}\}$ is a path on simplified hyper graph G_{SH} from c_{x_1} to c_{x_l}. $|P_{SH}(c_{x_1}, c_{x_l})| = ||c_{x_1}, c_{x_2}|| + ||c_{x_2}, c_{x_3}|| + \ldots + ||c_{x_{l-1}}, c_{x_l}||$. Notice that P_{SH} is not necessarily the shortest path from c_1 to c_l based on G_{SH}, i.e., $P_{SH}(c_1, c_l) \neq SP_{SH}(c_1, c_l)$. Also, we know that $||x_1, x_2|| + ||x_2, x_3|| + \ldots + ||x_{l-1}, x_l|| = ||x_1, x_l||$. By summing up above inequalities, we have

$$\begin{aligned}&|P_{SH}(c_{x_1}, c_{x_l})| - ||x_1, x_l|| \\ \leq &||x_1, c_{x_1}|| + ||x_l, c_{x_l}|| + 2 \times \sum_{2 \leq t \leq (l-1)} (||x_t, c_{x_t}||)\end{aligned}$$

Consequently,

$$\begin{aligned}&||x_1, c_{x_1}|| + |P_{SH}(c_{x_1}, c_{x_l})| + ||x_l, c_{x_l}|| - ||x_1, x_l|| \\ \leq &\ 2 \times \sum_{1 \leq t \leq l} (||x_t, c_{x_t}||). \end{aligned} \quad (3)$$

Suppose $ASP_{SH}(v_i, v_j)$ is approximate shortest path from v_i to v_j derived based on simplified hyper graph G_{SH}, then $|ASP_{SH}(v_i, v_j)| = |ASP_{SH}(x_1, x_l)| = ||x_1, c_{x_1}|| + |SP_{SH}(c_{x_1}, c_{x_l})| + ||x_l, c_{x_l}|| \leq ||x_1, c_{x_1}|| + |P_{SH}(c_{x_1}, c_{x_l})| + ||x_l, c_{x_l}||$. Based on (3) we have

$$\begin{aligned}&|ASP_{SH}(v_i, v_j)| - ||v_i, v_j|| \\ = &\ |ASP_{SH}(x_1, x_l)| - ||x_1, x_l|| \\ \leq &\ ||x_1, c_{x_1}|| + |P_{G_H}(c_{x_1}, c_{x_l})| + ||x_l, c_{x_l}|| - ||x_1, x_l|| \\ \leq &\ 2 \times \sum_{1 \leq t \leq l} (||x_t, c_{x_t}||) \end{aligned} \quad (4)$$

Next, we derive the lower bound of Δ. Based on (4), for the given $Q.v$ and u^*,

$$|ASP_{SH}(Q.v, u^*)| - ||Q.v, u^*|| \leq 2 \times \sum_{1 \leq t \leq l} (||x_t, c_{x_t}||),$$

given that $\{c_{x_1}, c_{x_2}, \ldots, c_{x_l}\}$ are the center nodes of the adjacent partitions that $SP(Q.v, u^*)$ crosses, and $x_t (1 \leq t \leq l)$ is a node on $SP(Q.v, u^*)$ which belongs to the partition of c_t, $x_1 = Q.v, x_l = u^*$. Because that u is the nearest node found based on the approximated shortest path distance, $|ASP_{SH}(Q.v, u)| \leq |ASP_{SH}(Q.v, u^*)|$. Straightforwardly, we have

$$\begin{aligned}\Delta &= |ASP_{SH}(Q.v, u)| - ||Q.v, u^*|| \\ &\leq |ASP_{SH}(Q.v, u^*)| - ||Q.v, u^*|| \\ &\leq 2 \times \sum_{1 \leq t \leq l} (||x_t, c_{x_t}||).\end{aligned}$$

For every node v_i, $||v_i, c_{v_i}|| \leq \max_{1 \leq k \leq |V|} ||v_k, c_{v_k}||$. Therefore, in the worst case,

$$\Delta \leq 2 \times p \times \max_{1 \leq k \leq |V|} ||v_k, c_{v_k}||.$$

p is the maximum number of adjacent partitions that a shortest path on G can cross. ■

6 Experiments

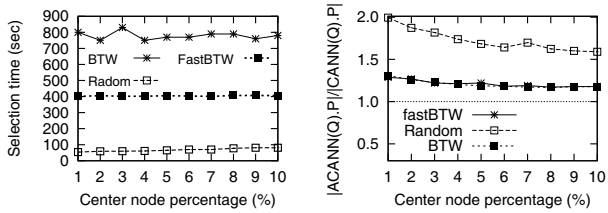
In order to evaluate the performance of proposed ACANN search on hyper-graph/simplified hyper-graph, we conduct a comprehensive experimental study and report its results in this section. First, we evaluate various center node selection schemes for the hyper-graph based index construction. Next, we test the performance of the ACANN search based on hyper graph index, including pre-processing time, storage overhead, query time, and approximation quality.

Two real social network datasets are used, including **Dblp** and **MyGamma**. The former is extracted from DBLP (http://dblp.uni-trier.de/xml/). We sample dblp graphs with the number of nodes changing from $0.5K$ to $8K$. For each node, we extract 20 keywords from the papers published by the author as the textual context. The latter is provided by MyGamma, a mobile social networking service provider (http://m.mygamma.com/). We sample mygamma graphs with node number changing from $10K$ to $20K$. Each node has on average 10 keywords extracted from user's profile, including user's nickname, race, country and so on. For both datasets, the graphs are unweighted (i.e. the weight on every edge is 1). We implemented all the evaluated schemes in C++, running on an AMD 2.4GHz Dual Processors server with 4GB RAM. In addition, the false drop rate γ is set to 0.01 and the size of the signature $|sig|$ is set to 128 in our implementation.

6.1 Evaluating Center Node Selection Schemes

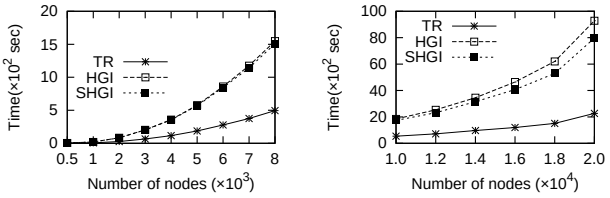
As mentioned in Section 3.1, the center nodes have a direct impact on the approximation quality, and we propose to select the nodes with the highest betweenness score as the center nodes. Because the betweenness score is computationally expensive to derive, we adopt a fast betweenness ranking algorithm proposed in [5] to retrieve the approximated top-k nodes with the highest betweenness score, and we refer this method as fastBTW. For comparison purpose, we implement another two center node selection methods, including Random that selects the center nodes randomly, and BTW that selects the nodes based on the real betweenness score. In our first set of experiments, we evaluate the performance of different center node selection methods, including the selection time, and the approximation quality.

First, we report the selection time of different center node selection methods under different numbers of center nodes, presented as the percentage of the dataset size, on a $5K$ nodes dblp graph in Figure 6(a). The results on different sized graphs show similar trend, and are omitted to the interest of space. As we can see, Random is very efficient as its running time is almost negligible. However, BTW is very time consuming due to the high cost of computing the betweenness for each node. fastBTW does not perform as well



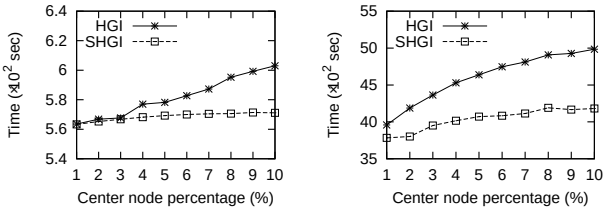
(a) Selection time. (b) Approximation quality.

Fig. 6 Performance of the center node selection schemes (dblp, $|V| = 5K$).



(a) Dblp (b) MyGamma

Fig. 7 Pre-processing time vs. dataset size (5% center nodes).



(a) Dblp($|V| = 5K$) (b) MyGamma($|V| = 16K$).

Fig. 8 HGI/SHGI pre-processing time vs. # center nodes.

as Random, but it is much more efficient than BTW. Generally, it takes only 10%-20% of BTW's time.

Next, we report the approximation quality of ACANN under different center node selection schemes in Figure 6(b). The approximation quality is measured by the error rate, i.e. $\frac{|ACANN(Q).P|-|CANN(Q).P|}{|CANN(Q).P|}$, as defined in Definition 3. We run 200 random queries with each having 1 to 5 keywords on each graph and report the average result. As shown in the figure, Random leads to very inaccurate results, while BTW offers the highest quality. The result on fastBTW is very close to that of BTW. Based on the above observation, we believe that fastBTW achieves a good trade-off between the center node selection time and approximation accuracy. Consequently, we adopt fastBTW to generate hyper-graph based index in the following experiments.

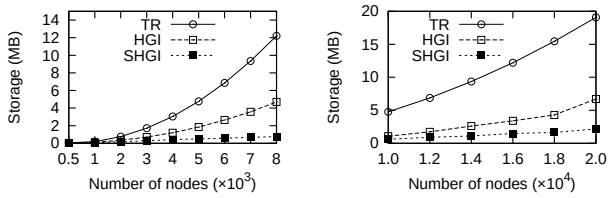
6.2 Performance of ACANN Search Algorithm

In the second set of experiments, we evaluate the performance of ACANN search based on the hyper-graph based index. We implement both the hyper-graph index and the simplified hyper-graph index, referred as HGI and SHGI, respectively, together with another two existing algorithms as comparison, i.e., the SI-based approach, referred as SI, and the TR-based approach, referred as TR. SI starts from the query node, and explores the graph based on distance until an answer node matching the queried keywords is reached (i.e. the classic Dijkstra's algorithm [8]). This method does not rely on any index structure. TR locates the nodes that match the queried keywords as the intermediate results, then performs network exploration to get the shortest paths from them to the query node.

It is reported in [36] that TR performs worse than SI. In our implementation, we change TR to improve its query efficiency. Different from original TR, we implement a more efficient algorithm, i.e., A* algorithm [13], to support the shortest path search from the query node to the intermediate results. A* algorithm first locates a small set of landmark nodes, and calculates the distance from these landmark nodes to every other node in the graph. Base on the pre-computed information, it estimates the distance from any given node to a destination. During the network exploration, it first visits the node with smallest estimated distance to the destination. This heuristic enables the search to quickly reach the destination, thus greatly reduces the search time. Generally, the more the selected landmark nodes are, the more accurate the distance estimation is. Thus, less nodes will be visited during the network exploration. However, a large number of landmark nodes will cause high pre-computational costs. Therefore, in our implementation, we choose 5% nodes as landmarks. Based on some initial tests that we performed, the implemented A* algorithm can reduce the shortest path search time by 1.5-30 times. In the following, we evaluate the performance of ACANN search via four aspects, i.e., pre-processing time, storage costs, query time, and approximation quality.

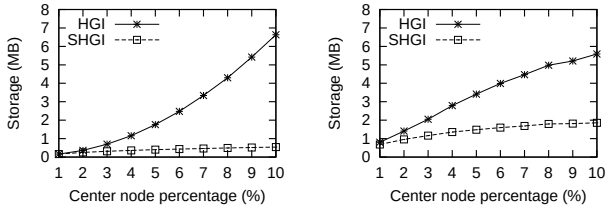
Pre-processing Time. First, we evaluate the pre-processing time of different approaches on different sized graphs, as reported in Figure 7. Note that SI does not require any index structure and hence it is not reported. The pre-processing cost of TR mainly comes from finding landmarks and computing the distance information from landmarks to other nodes. Generally, it incurs more index construction time as the graph size grows. Compared with both HGI and SHGI, TR spends less time on index construction. This is because the landmark based distance computation is less time consuming than the approximated betweenness score calculation.

We also report the pre-processing time of HGI and SHGI with various number of selected center nodes, as depicted in Figure 8. Generally, when the number of center nodes increases, the index construction time for both methods increases. We observe that SHGI takes less time than HGI, and the construction time of HGI grows faster than that of SHGI with the number of center nodes increasing. It is because HGI has to compute shortest paths between every pair of center nodes, while SHGI only maintains the shortest paths of those center nodes having their partitions adjacent.



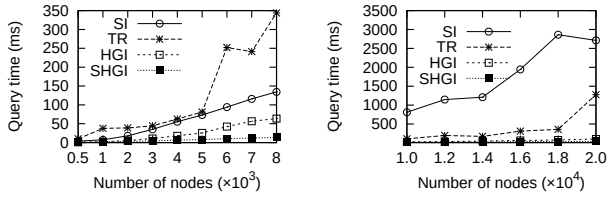
(a) Dblp. (b) MyGamma.

Fig. 9 Storage cost vs. datasets (5% center nodes).



(a) Dblp ($|V| = 5K$). (b) MyGamma ($|V| = 16K$).

Fig. 10 HGI/SHGI storage cost vs. # center nodes.



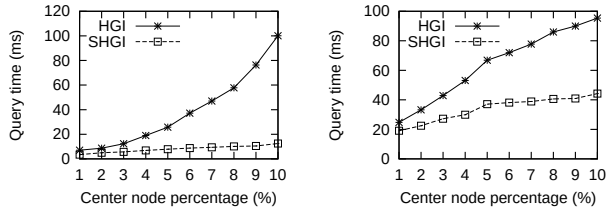
(a) Dblp (b) MyGamma

Fig. 11 Query time vs. dataset size (5% center nodes).

Storage Costs. Next, we evaluate the storage costs of various approaches, with the results reported in Figure 9. Notice that SI does not request any index, thus does not cause additional storage cost. On the other hand, TR records the shortest distances from each landmark node to every other node to facilitate fast shortest path search. Intuitively, all the approaches take more storage as the graph size grows. It is observed that TR takes up more space than HGI and SHGI. For example, on the Dblp graph with $8K$ nodes, TR consumes 2.5 times more space than HGI, and 16 times more space than SHGI. Similarly, on the MyGamma graphs, the space cost of TR is around 2-10 times more than that of HGI and SHGI.

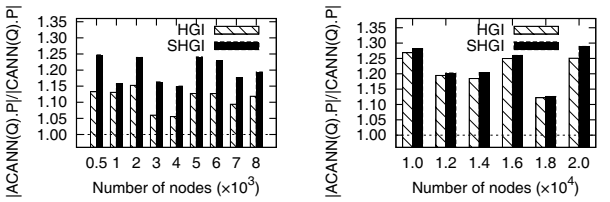
The storage cost of the hyper graph indexes is also affected by the number of center nodes selected. Obviously, the more the selected center nodes are, the larger the hyper graph is, thus the larger storage space it takes (see Figure 10). We can also observe that the storage cost of HGI is much larger than that of SHGI. The more the selected center nodes are, the larger the differences are. It is because that HGI maintains a complete hyper graph on the center nodes, while the hyper graph of SHGI is much sparser.

Query Time. We also study the query time of different approaches with different sized graphs, as reported in Figure 11. The query time is the average running time of 200 randomly generated queries. Generally, on both datasets, HGI and SHGI



(a) dblp ($|V| = 5K$). (b) gamma ($|V| = 16K$).

Fig. 12 HGI/SHGI query time vs. # center nodes ($\gamma = 0.01$, $|sig| = 128$).



(a) Dblp (b) MyGamma

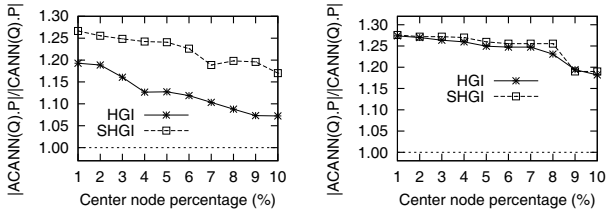
Fig. 13 Approximation quality vs. dataset size (5% center nodes).

perform significantly better than both SI and TR. In addition, we find that the query time increases as the graph size increases. However, SHGI is more resilient to the change of graph size than others and hence it has a better scalability to large sized graphs.

On the Dblp graphs, SI is faster than TR. Whereas, on the MyGamma graphs, TR, in most cases, incurs smaller query time than SI. This is mainly caused by different query keywords selectivity. The textual contexts of the nodes of Dblp graphs consist of the keywords of the users scientific publications. There are many common keywords shared by different users. Therefore, for the CANN queries with the common keywords, TR usually generates a large number of intermediate answer nodes, i.e., those nodes matching the queried keywords. TR needs to run A* algorithm several times to retrieve the shortest path from each intermediate node to the query node, thus causing longer running time. While, for the MyGamma graphs, the node context contains the unique user names. Many generated CANN queries with the unique user name as keywords has selectivity of 1. In these cases, TR runs faster than SI. We will examine the impact of the query keywords selectivity to the query performance in detail in the next set of experiments.

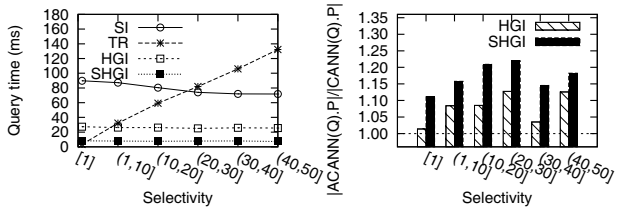
In addition, we also fix the graph size and change the number of center nodes selected, and report its impact on the query time of HGI and SHGI in Figure 12. Similar as previous observation, the more the selected center nodes are, the larger the index is, thus the longer the search time is.

Approximation Quality. We then evaluate the approximation quality of the ACANN search based on the hyper-graph index. First, we study the impact of dataset size on the approximation quality of HGI and SHGI, as depicted in Figure 13. We observe that the approximation quality of HGI is generally better than that of SHGI. For example, for the Dblp



(a) Dblp ($|V| = 5K$). (b) MyGamma ($|V| = 16K$).

Fig. 14 Approximation quality vs. # center nodes.



(a) Query time changes with selectivity. (b) Query accuracy changes with selectivity

Fig. 15 Query performance vs. query keywords selectivity.

graphs, the approximated shortest path returned by SHGI is under 0.25 times longer than the real shortest distance, while, the one returned by HGI is around 0.15 times longer than the real shortest path (see Figure 13(a)). Given that shortest distances between nodes of the Dblp/MyGamma datasets are short (usually less than 5 for Dblp datasets, and around 3 for MyGamma datasets), the approximated shortest paths are usually only one or at most two steps further, compared to the real shortest paths. Consequently, for those applications with high demand on search performance, our ACANN search algorithm provides considerably good approximations with fast response time.

Next, we study the impact of the number of center nodes selected on the approximation quality, as reported in Figure 14. Again as observed from the results, the more the selected center nodes are, the better the approximation quality is, for both HGI and SHGI. It is because that when more center nodes are selected, the graph is partitioned into finer partitions. Consequently, each partition contains less non-center nodes and the average distance from a non-center node to its nearest center node is shorter.

6.3 Query Keywords Selectivity

In this set of experiments, we further study the impact of the query keywords selectivity on the query performance of different methods. Due to the similarity among the results w.r.t. different parameter settings, we only present the results on the Dblp graph with $5K$ nodes. We generate several groups of CANN queries such that the queries in each group have different query keywords selectivity. The average query performance of the queries with different selectivity is reported in Figure 15. In the figure, the x-axis is the range of the number of nodes whose textual context matches the queried keywords (i.e. selectivity). Figure 15(a) demonstrates the query time change with different query selectivities. We find that as the query selectivity increases, the running time of SI decreases slightly. This is probably because that when there are more nodes containing the query keywords in a graph, the distance based network expansion of SI is likely to meet the answer nodes earlier, thus terminates faster. On the opposite, the query time of TR increases drastically with the selectivity increasing. It is easy to understand, because the number of times that TR issues A* shortest path search is equal to the number of intermediate nodes in the graph, whose context matches the queried keywords. However, because that the HGI and SHGI methods prune the search space based on distance and context simultaneously during the search, their running time is not very sensitive to the query selectivity changes, and so is their approximation quality (see Figure 15).

To sum up, in our experimental study, we evaluate the pre-processing time, storage overhead, query time, and approximation quality of the proposed HGI and SHGI methods. Compared with the existing SI and TR methods, our methods take more pre-processing time, and they require less storage overhead than TR[2]. However, HGI and SHGI significantly accelerate the CANN search with an average error factor less than 0.3. In addition, the query efficiency of HGI and SHGI is not sensitive to the query keywords selectivity changes.

7 Conclusion and Future Works

Social networking has been gaining popularity and growing rapidly in recent years. The problem of efficiently navigating and searching social networks has attracted more and more attentions. In this paper, we formulate a new type of queries, namely *context aware nearest neighbor search* (CANN) on social networks. It returns a node that is closest to the query node, and meanwhile has its context matching the query condition. CANN considers two important factors of social networks simultaneously in its search condition: the network structure and node context . We design a *hyper-graph* based index structure and a simplified hyper-graph variant to support approximate CANN search. According to extensive evaluation tests, hyper-graph based approaches support efficient query processing with limited sacrifice on the query accuracy.

In our definition of CANN query, we use shortest path distance to evaluate the connection strength of two users in a social network, and consider finding the nearest user based on boolean keywords match. In the future works, we would like to study other network distance metrics, e.g. the random walk distance, and other keywords matching scores e.g. TF/IDF score. In addition, besides the betweenness based

[2] The additional storage consumption of SI is none.

center nodes selection scheme, we want to explore other center node selection criteria. Finally, we want to study efficient ad-hoc search on dynamic social networks, where the network structure and node context are updated rapidly.

Acknowledgements This research is supported by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office, and Baihua Zheng is supported by Singapore Management University.

References

1. Ana M. Martn Gonzlez Bo Dalsgaardb, J.M.O.: Centrality measures and the importance of generalist species in pollination networks. Ecological Complexity **7**(1), 36–43 (2010)
2. Ang, C.: Interaction networks and patterns of guild community in massively multiplayer online games. Social Network Analysis and Mining **1**, 341–353 (2011)
3. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM (CACM) **13**(7), 422–426 (1970)
4. Carmel, D., Zwerdling, N., Guy, I., Ofek-Koifman, S., Har'el, N., Ronen, I., Uziel, E., Yogev, S., Chernov, S.: Personalized social search based on the user's social network. In: Proceedings of the 18th ACM conference on Information and knowledge management (CIKM '09), pp. 1227–1236 (2009)
5. Chong, W.H., Toh, W.S.B., Teow, L.N.: Efficient extraction of high-betweenness vertices. In: Proceedings of the 2010 International Conference on Advances in Social Networks Analysis and Mining (ASONAM '10), pp. 286–290 (2010)
6. Cohen, E.: Fast algorithms for constructing t-spanners and paths with stretch t. SIAM Journal on Computing **28**, 210–236 (1999)
7. Dalvi, B.B., Kshirsagar, M., Sudarshan, S.: Keyword search on external memory data graphs. VLDB Endow. **1**(1), 1189–1204 (2008)
8. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1**(1), 269–271 (1959)
9. Erwig, M.: The graph Voronoi diagram with applications. Networks **36**(3), 156–163 (2000)
10. Freeman, L.C.: A Set of Measures of Centrality Based on Betweenness. Sociometry **40**(1), 35–41 (1977)
11. Gaboille, C., Peleg, D., Pérennes, S., Raz, R.: Distance labeling in graphs. Journal of Algorithms **53**(1), 85–112 (2004)
12. Goldberg, A.V.: Point-to-point shortest path algorithms with preprocessing. In: Proceedings of the 33rd conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'07), pp. 88–102 (2007)
13. Goldberg, A.V., Harrelson, C.: Computing the shortest path: A search meets graph theory. In: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms (SODA'05), pp. 156–165 (2005)
14. Gutman, R.: Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In: Proceeding of the sixth Workshop on Algorithm Engineering and Experiments (ALENEX'04), pp. 100–111 (2004)
15. He, H., Wang, H., Yang, J., Yu, P.S.: Blinks: ranked keyword searches on graphs. In: Proceedings of the 2007 ACM SIGMOD international conference on Management of data (SIGMOD'07), pp. 305–316 (2007)
16. Hu, H., Lee, D.L., Lee, V.C.S.: Distance indexing on road networks. In: Proceedings of the 32nd international conference on Very large data bases (VLDB'06), pp. 894–905 (2006)
17. Hulgeri, A., Nakhe, C.: Keyword searching and browsing in databases using banks. In: Proceedings of the 18th International Conference on Data Engineering (ICDE'02), pp. 431–443 (2002)
18. Jing, N., Huang, Y.W., Rundensteiner, E.A.: Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation. IEEE Transactions on Knowledge and Data Engineering (TKDE) **10**(3), 409–432 (1998)
19. Jung, S., Pramanik, S.: An efficient path computation model for hierarchically structured topographical road maps. IEEE Transactions on Knowledge and Data Engineering (TKDE) **14**(5), 1029–1046 (2002)
20. Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., Karambelkar, H.: Bidirectional expansion for keyword search on graph databases. In: Proceedings of the 31st international conference on Very large data bases (VLDB'05), pp. 505–516 (2005)
21. Kourtellis, N., Alahakoon, T., Simha, R., Iamnitchi, A., Tripathi, R.: Identifying high betweenness centrality nodes in large social networks. Social Network Analysis and Mining pp. 1–16 (2012)
22. Lee, D., Leng, C.: Partitioned signature file: Design considerations and performance evaluation. ACM Transactions on Information Systems (TOIS) **7**(2), 158–180 (1989)
23. Lee, D.L., Kim, Y.M., Patel, G.: Efficient signature file methods for text retrieval. IEEE Transactions on Knowledge and Data Engineering (TKDE) **7**(3), 423–435 (1995)
24. Lee, K.C.K., Lee, W.C., Zheng, B.: Fast object search on road networks. In: Proceedings of the 12th International Conference on Extending Database Technology (EDBT'09), pp. 1018–1029 (2009)
25. Lee, K.C.K., Lee, W.C., Zheng, B., Tian, Y.: Road: A new spatial object search framework for road networks. IEEE Transactions on Knowledge and Data Engineering (TKDE) **24**(3), 547–560 (2012)
26. Leng, C., Lee, D.: Optimal weight assignment for signature generation. ACM Transactions on Database Systems (TODS) **17**(2), 346–373 (1992)
27. Li, G., Feng, J., Chin Ooi, B., Wang, J., Zhou, L.: An effective 3-in-1 keyword search method over heterogeneous data sources. Information Systems **36**, 248–266 (2011)
28. Maglaras, L.A., Katsaros, D.: New measures for characterizing the significance of nodes in wireless ad hoc networks via localized path-based neighborhood analysis. Social Network Analysis and Mining pp. 97–106 (2012)
29. Newman, M.: A measure of betweenness centrality based on random walks. Social networks **27**(1), 39–54 (2005)
30. Peleg, D.: Proximity-preserving labeling schemes. Journal of Graph Theory **33**, 167–176 (2000)
31. Samet, H., Sankaranarayanan, J., Alborzi, H.: Scalable network distance browsing in spatial databases. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of Data (SIGMOD'08), pp. 43–54 (2008)
32. Schenkel, R., Crecelius, T., Kacimi, M., Michel, S., Neumann, T., Parreira, J.X., Weikum, G.: Efficient top-k querying over social-tagging networks. In: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR'08), pp. 523–530 (2008)
33. Vieira, M.V., Fonseca, B.M., Damazio, R., Golgher, P.B., Reis, D.d.C., Ribeiro-Neto, B.: Efficient search ranking in social networks. In: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management (CIKM'07), pp. 563–572 (2007)
34. Wei, F.: TEDI: efficient shortest path query answering on graphs. In: Proceedings of the 2010 international conference on Management of Data (SIGMOD'10), pp. 99–110. New York, New York, USA (2010)
35. Xiao, Y., Wu, W., Pei, J., Wang, W., He, Z.: Efficiently indexing shortest paths by exploiting symmetry in graphs. In: Proceedings of the 12th International Conference on Extending Database Technology (EDBT'09), pp. 493–504 (2009)

36. Yin, P., Lee, W.C., Lee, K.C.: On top-k social web search. In: Proceedings of the 19th ACM international conference on Information and knowledge management (CIKM '10), pp. 1313–1316 (2010)