10-2011

Context-Aware Nearest Neighbor Query on Social Networks

Yazhe WANG
Singapore Management University, yzwang@smu.edu.sg

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

DOI: https://doi.org/10.1007/978-3-642-24704-0_15

Citation

WANG, Yazhe and ZHENG, Baihua. Context-Aware Nearest Neighbor Query on Social Networks. (2011). *Social informatics: Third International Conference, SocInfo 2011, Singapore, October 6-8: Proceedings*. 6984, 98-112. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/1412

Context-Aware Nearest Neighbor Query on Social Networks[⋆]

Yazhe Wang and Baihua Zheng

Singapore Management University
{yazhe.wang.2008,bhzheng}@smu.edu.sg

Abstract. Social networking has grown rapidly over the last few years, and social networks contain a huge amount of content. However, it can be not easy to navigate the 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 context specified. CANN considers both the structure of the social network, and the profile information of the nodes. We design a *hyper-graph* based index structure to support approximated CANN search efficiently.

1 Introduction

Social network websites and applications have grown rapidly over the last few years. Take Facebook as an example. From an initial website used by Harvard students to one of the most famous social networking websites, it has currently attracted more than 400 million active users worldwide [17]. Obviously, more and more people start using social networks to share ideas, activities, and interests with each other, 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.

We model the social network as undirected graph, and assume each node of the graph maintains some profile information. Take the co-authorship network G depicted in Fig. 1 as an example. Each node represents a researcher and a link between two nodes states that those two researchers have collaborated at least once. Some descriptive information (e.g., name, profession, and research topics) of each node is maintained, as depicted in Fig. 1. A *context-aware nearest neighbor (CANN)* query is defined to search over social network based on both network structure and the profile 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 reach the nearest professor working in data

⋆ We would like to acknowledge that this research/project was carried out at the Living Analytics Research Centre (LARC), sponsored by Singapore National Research Foundation and Interactive & Digital Media Programme Office, Media Development Authority..

Published in Social Informatics. SocInfo 2011. Lecture Notes in Computer Science, vol 6984. Springer, pp. 98-112.
https://doi.org/10.1007/978-3-642-24704-0_15

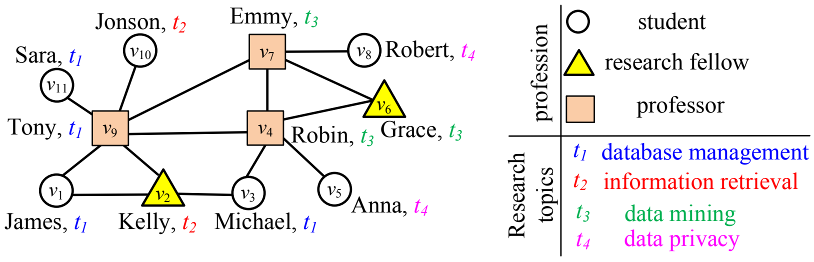


Fig. 1. A collaboration social network G

mining". Here, *distance* from the query node v_3 to a node v is evaluated by the shortest path distance; and the *context* is represented by keywords $\{$*professor, data mining*$\}$. The answer to Q_1 is node v_4 with its shortest path $\{v_3, v_4\}$. CANN query considers both the network distance and the context. It has a large application base. For example, researchers 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.

There are two naive approaches for CANN search. First, we can invoke traditional shortest path search algorithm to approach nodes based on ascending order of their distances to the query node until one that matches the queried keywords is found, denoted as *SPA-based approach*. Second, we can employ well-known information retrieval techniques to locate all the nodes that match the queried keywords, and then order them based on shortest path distances, denoted as *IR-based approach*. However, both approaches are *inefficient*, in terms of search performance and storage consumption. On one hand, SPA-based approach traverses the social network purely based on the distance but not context and hence it might have to visit many unnecessary nodes before the answer node is found. On the other hand, IR-based approach may find many nodes that match the queried keywords as intermediate results and hence the ranking process based on the distances between query node and *all* the intermediate nodes could be very costly. In addition, the inverted index used by IR-based approach might take up large storage space if the graph is big and/or the vocabulary of the context is large.

Given the fact that the exact search of CANN query is relatively expensive and some applications might be willing to trade in the accuracy for performance, we propose an approach, namely *hyper-graph based approximation*, that provides an approximated result to CANN queries with high performance and relatively cheap storage requirement. It tries to utilize the unique *power law degree distribution* feature of social network, and identifies some nodes with very high degree (whose number will be small) as *center nodes*. It then partitions the social network into disjoint sub-graphs with each centering around a center node. Based on the assumption that a path linking a node in one sub-graph G_i to a node in another sub-graph G_j is very likely to pass the corresponding center nodes of G_i and G_j, it builds a hyper graph to index i) among sub-graphs, the shortest paths between center nodes; and ii) within each sub-graph, the shortest paths between non-center nodes and the center node. Meanwhile, it attaches certain signature information, called *signature map*, at each center node that facilitates the space pruning based on queried keywords.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 defines CANN search and approximated CANN search. Section 4 presents the details of the hyper-graph based approximation technique. Section 5 analyzes the experimental results. Finally, Section 6 concludes this paper.

2 Related Work

In this section, we briefly review existing work related to CANN search, including (approximated) shortest path search, keyword query and signature technique used in information retrieval.

2.1 Shortest Path Search and Keywords Query

The most well-known shortest path search algorithm on graphs is the Dijkstra's algorithm [2]. It explores the graph in a best-first manner starting from the query node until the target node is reached. Some faster solutions were proposed to prune the graph exploration space based on domain-specific heuristic and pre-processing, such as A* search [5] and reach based method [6]. The algorithms mentioned above usually assume the searched graph can be stored in main memory, and do not scale very well for very large graphs.

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 [10] and HiTi [11] build index based on materializing local shortest paths of a number of disjoint subgraphs. The global shortest path is then obtained by combining selected local shortest paths. Recently, a novel tree decomposition based graph index structure has been proposed [20], which supports efficient shortest path query with even smaller index size. There are other works considering encoding all-pairs shortest paths of a graph in small-sized indexes. For instance, [19] proposes a quadtree-structured index utilizing the spatial coherence of the destination (or source and destination) nodes. Distance signature method [8] 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 [21] is another example. It exploits symmetry properties of graphs to reduce the index size of all-pairs shortest paths. However, it is only applicable to un-weighted graphs. All these discussed approaches support efficient shortest path search for given source and destination nodes. However, none of them considers the context of the nodes, or supports the queries that do not specify the destination at the query time.

There are some techniques designed for approximated shortest path/distance queries. Spanner [1] 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 should be deleted in order to generate a good spanner so that the distances between nodes do not change substantially. Spanners perform worse on dense graphs with large girth. Distant labeling and embedding techniques [4,18] 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.

Keyword query on graphs [7, 9, 12, 16] also considers both the distance and context information. It is to find closely connected clusters of nodes in the graph which contain specific keywords. Based on different query semantic, the result of the query could be rooted trees or subgraphs embedded in the graph. Obviously, the definition of keyword query is different from our CANN search.

2.2 Signature

Signature techniques have been studied extensively in information retrieval [13, 15]. A signature is basically an abstract of the keyword information of a data item. Given a set of keywords that index 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.

To decide whether a data item i matches/contains the query keyword Q, a *query signature* S_Q is generated first, based on the same hash function. Thereafter, S_Q is compared against the signatures S_i using bitwise-AND ($\wedge$). The signatures *match* if for every bit set in S_Q, the corresponding bit in the compared signature S_i is also set. If S_Q does not match S_i, then data item i does not match query Q. While, if a match happens, it could be a *true match* that the data item is really what the query searches for; or it could be a *false drop* that the data item in fact does not satisfy the search criteria.

3 Problem Definition

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 (ACANN)* query.

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 strength of the linkage. 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 are 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 distance, denoted as $||v_i, v_j||$ $(= |SP(v_i, v_j)|)$, is the shortest distance between v_i and v_j.* $\square$

Take the social network 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 $Q.k$, and 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 $Q.k \subseteq v_i.k \wedge ||Q.v, v_i|| < |P|$.* $\square$

As the exact search of CANN query is relatively expensive, we, in this paper, focus on supporting an 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 ratio of the length of the returned path of ACANN search to that of CANN query, i.e., $\frac{|ACANN(Q).P|}{|CANN(Q).P|}$.* $\square$

4 Hyper-Graph based Approximation

In this section, we present an index structure, namely *hyper-graph*, to support approximated CANN search. We first explain the basic idea of hyper-graph index based approximation, then present the structure of hyper-graph index and its construction algorithm, and finally explain the corresponding search algorithm.

4.1 Basic Idea

The idea of hyper-graph index comes from the intuition of how we search for information in the real social network. Usually, there are a small number of important persons who have strong connections with people in their local social network. For example, Prof. Jiawei Han is a distinguished researcher in the data mining field. If a UIUC graduate wants to build a connection with another data mining researcher, it is very likely that Prof. Han can provide great help.

Based on this finding, we first identify a small set of important persons as *center nodes* in the social network, and divide the social network into disjoint partitions P_i with each around one center node c_i. We then employ the center node as the knowledge hub of its partition, i.e., each center node carries distance information and context information of the nodes within its partition. We assume the center nodes serve as *glues* to connect nodes. In other words, a path linking nodes within a partition G_i will pass the center node c_i, and a path linking a node in partition G_i to a node in another partition G_j will pass the center nodes c_i and c_j, i.e., it is very likely that center nodes lie on the shortest paths between nodes. Consequently, we index the shortest paths from each node within a partition G_i to the center nodes, and the shortest paths from center nodes to the center nodes of their neighboring partitions, namely *hyper graph*. With the help of hyper graph, an ACANN query issued at a node v can be first forwarded to the center node c_i of the partition that covers v via a *local search* conducted by the center node c_i within its own partition. Meanwhile, c_i expands the search to its neighboring partitions via *expanded search*. The construction of hyper graphs, and the details of local search as well as expanded search will be presented in the following subsections.

4.2 Hyper Graph Index

The hyper graph index construction contains three steps, i.e., *center node selection*, *network partition*, and *hyper graph formation*, as detailed in the following.

Center Nodes Selection. There are multiple ways to select center nodes, such as random selection and betweenness-based selection. The former picks center nodes randomly while the latter selects those nodes with highest betweenness scores[1]. However, random selection may pick nodes that do not lie on many shortest paths, and betweenness based selection may suffer from very high computation cost. Consequently, we propose degree-based selection. The rationale is that in social network, the persons with wide social connections tend to exist on many shortest paths linking different nodes. We will evaluate those center node selection methods in Section 5.

Network Partition. Once the center nodes are fixed, we assign other nodes to their nearest center nodes for network partition, as formally defined in Definition 4. In case a node shares the same distance to multiple center nodes, it is randomly assigned to one of them. Accordingly, we need to locate the shortest paths from each node to center nodes. The graph partition could be computed in time $O(|E| + |V| + |V|\log(|V|)|)$ using the algorithm proposed in [3].

[1] The betweenness score of a node equals the number of shortest paths crossing it.

Definition 4 ***Network Partition.*** *Given a social network $G(V, E)$ and a set of center nodes $C = \{c_1, c_2, \cdots, c_r\}$ with $C \subset V$, a network partition $P_G = \{G_1(V_{G_1}, E_{G_1}), G_2(V_{G_2}, E_{G_2}), \cdots, G_r(V_{G_r}, E_{G_r})\}$ is a set of subgraphs G_i that i) $\forall c_i \in C, c_i \in V_{G_i}$; ii) $\forall v \in V$, $\exists G_i \in P_G$, $v \in V_{G_i}$; iii) $\forall v \in V_{G_i} \wedge \forall j (\neq i) \in [1, r]$, $||v, c_i|| \leq ||v, c_j||$; iv) $\forall v, v'(v \neq v') \in V_{G_i}$, if $e(v, v') \in E$, $e(v, v') \in E_{G_i}$; and v) $\forall i, j(i \neq j) \in [1, r]$, $V_{G_i} \cap V_{G_j} = \emptyset \wedge E_{G_i} \cap E_{G_j} = \emptyset \wedge \bigcup_{1 \leq i \leq r} E_{G_i} \subseteq E$.* $\square$

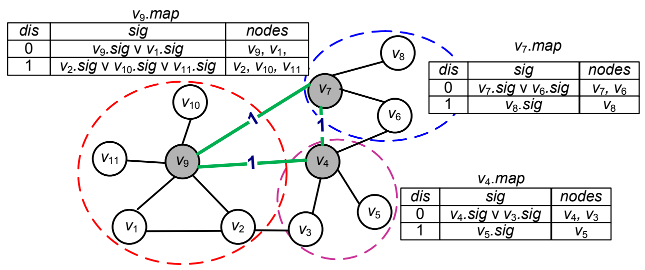
Hyper Graph Formation. As explained before, ACANN search contains local search and expanded search. In order to support local search, within each partition G_i, we store the shortest paths from each non-center node v to the center node c_i, via a two-tuple vector $\langle c_i, v_{next}\rangle$. Here, the shortest path from a non-center node v to the center node c_i is identified during the social network partition process, and v_{next} is the next-hop node on $SP(v, c_i)$.

In addition, to support space pruning based on queried keywords, each center node c_i maintains signatures representing the context of the nodes within its partition, via the *signature map*, denoted as c_i^{map}. To be more specific, within each partition G_i, we order the non-center nodes based on their distances to the center node c_i and cluster them into groups. For each group, a signature is generated by superimposing the signatures of the context of the nodes within the group. Thereafter, when a search reaches a center node c_i, we compare the queried keywords with the signatures of c_i's groups, and examine the nodes within a group only when its signature indicates a match.

Obviously, how to cluster the nodes into groups affects the search efficiency. In general, given a hash function for signature generation (i.e., a fixed signature size), the more the nodes clustered into a group are, the higher the false drop rate is. In this work, we pre-define a false drop rate threshold γ (e.g., 0.01) and decide the maximal number of distinct keywords, denoted as η, that could be represented by a signature with approximated false drop rate bounded by γ, based on Equation (1) [14]. Here, $|sig|$ is the length of the signature.

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

The clustering algorithm then works as follows. First, all the nodes v_j within a partition G_i are sorted based on ascending order of their shortest distances to c_i, maintained in a queue Que. Next, we dequeue the head node v_j from Que, insert v_j into set S, and check the total number of keywords associated with nodes in S, denoted as φ. There are three cases. Case (i) $\varphi > \gamma$: all the nodes in S, except v_j, form a new group g_l, with $S = \{v_j\}$; Case (ii) $\varphi = \gamma$: all the nodes in S form a new group g_l, with $S = \emptyset$; and Case (iii) $\varphi < \gamma$: no action. This process continues until Que is empty. Notation $c_i^{map}[l]$ is used to represent the signature map for the l-th group g_l w.r.t. the center node c_i, in the format of $\langle sig, dis, nodes\rangle$. Here, $c_i^{map}[l].sig$ is the signature generated based on all the keywords associated with nodes within the group g_l, $c_i^{map}[l].dis$ is the lower bound of the shortest distance from any node within group g_l to the center node c_i (i.e., $\forall v_j \in g_l$, $||v_j, c_i|| \geq c_i^{map}[l].dis \wedge \exists v' \in g_l$, $||v', c_i|| = c_i^{map}[l].dis$), and $c_i^{map}[l].nodes$ records all the nodes within group g_l.


v_9.map

dis	sig	nodes
0	v_9.sig ∨ v_1.sig	v_9, v_1,
1	v_2.sig ∨ v_{10}.sig ∨ v_{11}.sig	v_2, v_{10}, v_{11}

v_7.map

dis	sig	nodes
0	v_7.sig ∨ v_6.sig	v_7, v_6
1	v_8.sig	v_8

v_4.map

dis	sig	nodes
0	v_4.sig ∨ v_3.sig	v_4, v_3
1	v_5.sig	v_5

Fig. 2. An example of the hyper graph index

In order to support expanded search, we pre-compute the shortest paths between two center nodes whose partitions are adjacent. Two partitions G_i, G_j are adjacent, denoted as $G_i \wr G_j$, if there is an edge in G that connects a node in G_i to a node in G_j. Then, we build *hyper graph* which includes all the center nodes as vertexes, and the shortest paths between center nodes of adjacent partitions.

Definition 5 ***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 those center nodes with their corresponding partitions are adjacent, i.e., $V_H = C$, and $E_H = \cup_{G_i \wr G_j \wedge |SP(c_i,c_j)| \neq \infty} e(c_i, c_j)$ with $w(c_i, c_j) = |SP(c_i, c_j)|$.* □

An example of the hyper graph index is depicted in Fig. 2. Assume the number of center nodes is three. Using degree-based selection, nodes v_4, v_7, and v_9 with the top-three maximal degrees are selected as the center nodes. Thereafter, the network partition takes place. Each non-center node is attached to its nearest center node as demonstrated by the dashed circle in Fig. 2. Once the social network is partitioned, we proceed to form hyper graph. As all the partitions are adjacent, the hyper graph actually is a complete graph with vertices $V_H = C = \{v_4, v_7, v_9\}$ and edges $E_H = \{e(v_4, v_7), e(v_7, v_9), e(v_4, v_9)\}$. The content of each center node signature map is also depicted. Take center node v_7 as an example. Its partition has three nodes, and they are sorted based on ascending order of their shortest distances to the center node v_7. Suppose each signature contains up to four keywords (i.e., $\eta = 4$). Nodes v_6 and v_7 are clustered into the first group, and node v_8 is clustered into the second group. For each group, the signature is formed by superimposing the signature of each node and the distant is set to the shortest distance between the first node of the group to v_7.

4.3 Approximated Search Algorithm

The hyper graph based ACANN search assumes that a path from a node v within a partition G_i to node v' within a partition G_j $(i \neq j)$ must pass corresponding center nodes c_i, c_j, i.e., a center node serves as the only *entrance* to and the *exit* from its partition. To be more specific, a path from v to v' consists of three path segments, the one from v to c_i, the one from c_i to c_j, and the one from c_j to

v'. Algorithm 1 lists the pseudo code of ACANN search. 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 locate the center node c_q that covers $Q.v$ via $Q.v$'s two-tuple vector $\langle c_i, v_{next}\rangle$, with d being the shortest distance from $Q.v$ to c_q. We then enqueue c_q into Que, a priority queue maintaining center nodes of those partitions that might deserve examinations (line 6). All the entries in Que are two-tuple vectors $\langle c_i, ||c_i, c_q||\rangle$, ordered based on ascending order of the distance between center nodes c_i and c_q.

Algorithm 1: 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 $G_H(V_H, E_H)$, an ACANN query $Q$
Output: the approximated answer node $v_{ans}$, $d_{ans}$, and $P_{ans}$
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  for each $c_i \in V_H$ do
5      $d_{c_i} = \infty$;
6  $c_q = Q.v.c_i$, $Que = \langle c_q, 0\rangle$, $d = ||c_q, Q.v||$;
7  while Que is not empty do
8      $\langle c_i, ||c_q, c_i||\rangle = dequeue(Que)$;
9      if $(d + ||c_q, c_i||) \geq d_{ans}$ then
10         return $v_{ans}$, $d_{ans}$, $P_{ans}$;
11     for each $c_i^{map}[l] \in c_i^{map}$ do
12         if $(d + ||c_q, c_i|| + c_i^{map}[l].dis) \geq d_{ans}$ then
13             break;
14         else if $H(Q.k) \wedge c_i^{map}[l].sig = H(Q.k)$ then
15             for each $v_j \in c_i^{map}[l].nodes$ do
16                 if $Q.k \subset v_j.k$ and $(d + ||c_q, c_i|| + ||c_i, v_j||) < d_{ans}$ then
17                     $v_{ans} = v_j$; $d_{ans} = d + ||c_q, c_i|| + ||c_i, v_j||$;
18                     $P_{ans} = append(SP(Q.v, c_q), P(c_q, c_i), SP(c_i, v_j))$;
19     for each neighboring node $c_n$ of $c_i$ in $G_H$ do
20         if $d_{c_i} + ||c_i, c_n|| < d_{c_n}$ then
21             $enqueue(\langle c_n, d_{c_i} + ||c_i, c_n||\rangle)$; $P(c_q, c_n) = \text{append}(P(c_q, c_i), e(c_i, c_n))$;
22             $d_{c_n} = d_{c_i} + ||c_i, c_n||$;
```

Thereafter, we continuously dequeue the head entry from Que until it becomes empty. Every time when a head entry $\langle c_i, ||c_i, c_q||\rangle$ is dequeued, the lower bound of the approximated distance from $Q.v$ to any node in partition G_i centered at c_i (i.e., $d + ||c_i, c_q||$) is compared against the approximated distance d_{ans} from $Q.v$ to the current answer node. If the lower bound is longer than d_{ans}, the partition G_i can be safely discarded. Similarly, all the entries in Que, due to larger $||c_i', c_q||$

values, are pruned away to terminate the search (lines 9-10). Otherwise, partition G_i needs examination. We use c_i^{map} to filter out unnecessary nodes. The first filtering condition is based on distance. We calculate $(d + ||c_i, c_q|| + c_i^{map}[l].dis)$, the lower bound of the approximated distance from a node in $c_i^{map}[l].nodes$ to $Q.v$. If it is longer than d_{ans}, there is no need to examine nodes within this l-th group and the following groups (lines 12-13). The second filtering condition is based on the context. We could safely discard $c_i^{map}[l].nodes$ if $c_i^{map}[l].sig$ does not match the query context $Q.k$. If this l-th group is not filtered out by the previous two conditions, we need examine the nodes in this group one by one, and update the answer when the nodes $v_j \in c_i^{map}[l].nodes$ that match the search context are found(line 14-18). Up to this point, we have examined the partition centered at c_i, i.e., the local search is finished. We then start the extended search by inserting all the unexamined neighboring center nodes of c_i in G_H for further examination (lines 19-22).

5 Experiments

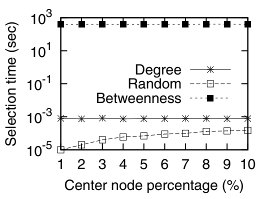
In this section, we report the experimental evaluation. First, we evaluate various center node selection schemes for the hyper graph index construction. Next, we test the hyper graph index based ACANN search performance, including preprocessing time, storage overhead, query time, and approximation quality.

Two real social network datasets are used, including **dblp** and **gamma**. The former is extracted from DBLP (http://dblp.uni-trier.de/xml/). We sample dblp graphs with number of nodes changing from $0.5K$ to $8K$. For each node, we extract 20 keywords from papers published by the author as the 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, including user's nickname, race, country and so on extracted from user's profile. For both datasets, the graphs are unweighted (i.e. the weight on every edges 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. Due to the space limitation, we skip some results w.r.t. **gamma** that share the similar trends as **dblp**.

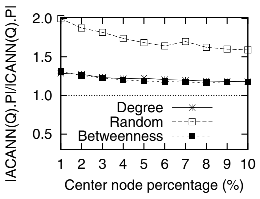
5.1 Evaluating Center Node Selection Schemes

As mentioned in Section 4, there are three center nodes selection schemes, including random selection, betweenness based selection, and degree based selection, denoted as Random, Betweenness, and Degree respectively. In the first set of experiments, we compare the performance of these three approaches in terms of selection time and the quality of approximation.

The test results on a $5K$ nodes dblp graph is reported in Fig. 3 as a representative. Fig. 3(a) shows the selection time when the number of center nodes,



(a) Selection time.



(b) Approximation quality.

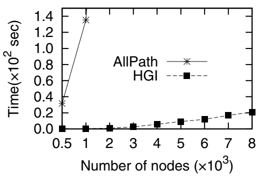
Fig. 3. Performance of the center node selection schemes (dblp, $|V| = 5K$)

presented as the percentage of the dataset size, changes. As we can see, Random is the most efficient in terms of selection time. Degree takes more time than Random, but is still efficient. However, Betweenness is very time consuming due to the high cost of computing nodes betweenness. Fig. 3(b) reports the approximation quality of ACANN under different schemes. The approximation quality is measured by $|ACANN(Q).P|/|CANN(Q).P|$, as defined in Definition 3. We run 200 random queries with each having 1 to 5 keywords randomly selected from the keywords vocabulary and report the average result. As shown in the figure, Random leads to very inaccurate results, while Betweenness offers the highest quality. The result on Degree is very close to that of Betweenness. Consider both the center node selection time and approximation accuracy, we set Degree as the default center node selection approach in the following evaluation.

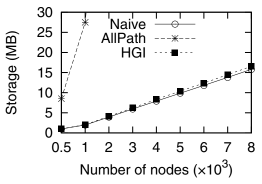
5.2 Performance of ACANN Search Algorithm

Next, we evaluate the performance of ACANN search with the help of hyper graph index. Two algorithms are implemented as the comparison in our evaluation. One is the naive SPA-based approach introduced in Section 1, referred as Naive. Started from the queried node, it explores the graph based on distance and does not rely on any index structure. The other method is based on the pre-computed all-pairs shortest paths, referred as AllPath. In AllPath, for each node v in G, we construct a signature map for each of its neighboring node n_i^v as described in Section 4.2. The signature map summarizes the context of the nodes u which can be reached from v via n_i^v (i.e. the shortest path from v to u passes n_i^v). When a query is performed on v, the signature map could efficiently direct the search towards the potential result node whose context actually matches the query. We also implement the hyper graph index method, referred as HGI.

Pre-processing Time. First, we evaluate the pre-processing time of different approaches vs. size of the datasets, as reported in Fig. 4(a). Note that Naive does not require any index structure and hence it is not reported. It is observed that as the graph size grows, the index construction time increases as well. AllPath takes longer construction time due to the need of computing all-pairs of shortest paths, and the construction time increases sharply with the increase of

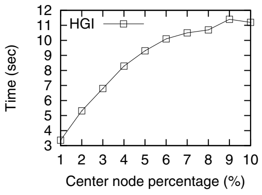


(a) Pre-processing time

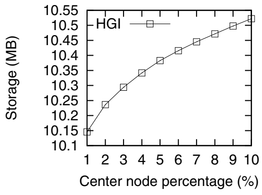


(b) Storage cost

Fig. 4. Performance vs. dataset size (dblp, 5% center nodes)



(a) Pre-computation cost.



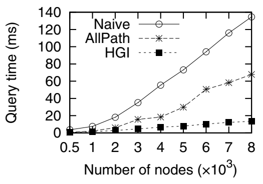
(b) Storage cost.

Fig. 5. HGI performance vs. # center nodes. (dblp, $|V| = 5K$)

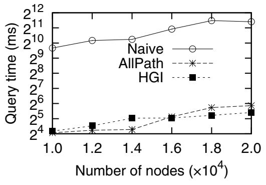
graph size. On the other hand, HGI takes much shorter construction time and hence the hyper graph based algorithm has a better scalability. We also report the preprocessing time of HGI with various number of center nodes selected, as depicted in Fig. 5(a). Generally, when the number of center nodes increases, the index construction time increases.

Storage Costs. Next, we evaluate the storage costs of various approaches in Fig. 4(b). Notice that Naive does not request any index. For other methods, we record the storage space taken by the social network and the corresponding indexes. We observe that for both datasets, the storage cost increases with the graph size growing, and HGI takes up much less space than AllPath. In addition, compared with Naive, the extra space consumed by HGI is smaller than 5% for both datasets. The storage cost of the hyper graph index is also affected by the number of center nodes selected. As shown in Figure 5(b), the more the selected center nodes are, the larger the hyper graph is.

Query Time. The query performance is evaluated by the query time and the approximation quality. We first test the query time of different approaches under different size of graphs, as reported in Fig. 6. Generally, Naive performs the worst, especially on large sized graphs. This is because it has to visit a large number of nodes with extremely long processing time. On the other hand, AllPath and HGI both significantly shorten the query time by precomputing certain information. For the dblp graphs, HGI even takes shorter query time than AllPath. This is probably because that there are more nodes in a dblp graph with their contexts matching the query keywords, thus it takes more time for AllPath to filter out

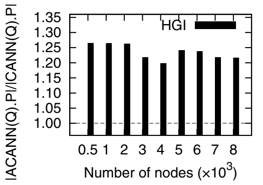


(a) dblp

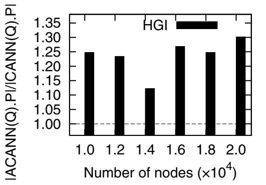


(b) gamma

Fig. 6. Query time vs. dataset size (5% center nodes)



(a) dblp



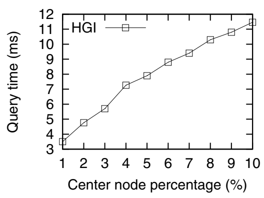
(b) gamma

Fig. 7. Approximation quality vs. dataset size (5% center nodes)

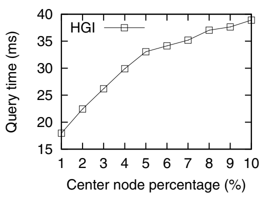
the non-result nodes based on distance. While, for the gamma graphs, HGI, in most cases, incurs similar query time as AllPath. Then, we fix the graph size and change the number of center nodes selected, and report its impact on the query time of HGI in Figure 8. Similar as previous observation, the more 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 under hyper graph index. First, we study the impact of dataset size on the approximation quality of HGI, as depicted in Fig. 7. For the dblp datasets, the approximated shortest path returned by HGI is 0.3 times longer than the real shortest distance as shown in Figure 7(a). Given that shortest distances between nodes of the dblp/gamma datasets are short (usually less than 5 for dblp datasets, and around 3 for gamma 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 can provide considerably good approximations with fast response time.

We further study the impact of the number of center nodes selected on the approximation quality of HGI, as reported in Figure 9. Again as observed from the results, the more the selected center nodes are, the better the approximation quality for HGI is. 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.

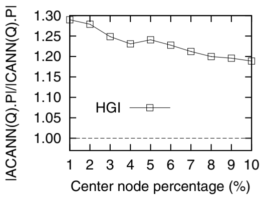


(a) dblp ($|V| = 5,000$).

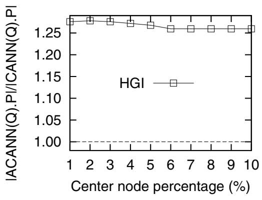


(b) gamma ($|V| = 16,000$).

Fig. 8. HGI query time vs. # center nodes ($\gamma = 0.01$, $|sig| = 128$)



(a) dblp ($|V| = 5,000$).



(b) gamma ($|V| = 16,000$).

Fig. 9. Approximation quality vs. # center nodes

To sum up, we evaluate the pre-processing time, storage overhead, query time, and approximation quality of the HGI method. The results demonstrate that HGI has relatively low preprocessing and storage overhead with certain sacrifice of the query accuracy. However, the average error factor is less than 1.3.

6 Conclusion

Motivated by the fact that social networking is growing rapidly, we, in this paper, formulate a new type of queries, namely *context aware nearest neighbor search*, over social networks. It returns a node that is closest to the query node, and meanwhile has its context matching the query condition. A *hyper graph* index structure is designed to support approximate CANN search. Through extensive evaluation tests, hyper-graph based approaches provide relative accurate results with low preprocessing and storage overhead.

References

1. Cohen, E.: Fast algorithms for constructing t-spanners and paths with stretch t. SIAM J. Comput. 28, 210–236 (1999)
2. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1(1), 269–271 (1959)
3. Erwig, M.: The graph Voronoi diagram with applications. Networks 36(3), 156–163 (2000)

4. Gaboille, C., Peleg, D., Pérennes, S., Raz, R.: Distance labeling in graphs. Journal of Algorithms 53(1), 85–112 (2004)
5. Goldberg, A.V., Harrelson, C.: Computing the shortest path: A search meets graph theory. In: SODA, pp. 156–165 (2005)
6. Gutman, R.: Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In: ALENEX, pp. 100–111 (2004)
7. He, H., Wang, H., Yang, J., Yu, P.S.: Blinks: ranked keyword searches on graphs. In: SIGMOD, pp. 305–316 (2007)
8. Hu, H., Lee, D.L., Lee, V.C.S.: Distance indexing on road networks. In: VLDB, pp. 894–905 (2006)
9. Hulgeri, A., Nakhe, C.: Keyword searching and browsing in databases using banks. In: ICDE, pp. 431–443 (2002)
10. Jing, N., Huang, Y.-W., Rundensteiner, E.A.: Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation. TKDE 10(3), 409–432 (1998)
11. Jung, S., Pramanik, S.: An efficient path computation model for hierarchically structured topographical road maps. TKDE 14(5), 1029–1046 (2002)
12. Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., Karambelkar, H.: Bidirectional expansion for keyword search on graph databases. In: VLDB, pp. 505–516 (2005)
13. Lee, D., Leng, C.: Partitioned signature file: Design considerations and performance evaluation. TOIS 7(2), 158–180 (1989)
14. Lee, D.L., Kim, Y.M., Patel, G.: Efficient signature file methods for text retrieval. TKDE 7(3), 423–435 (1995)
15. Leng, C., Lee, D.: Optimal weight assignment for signature generation. TODS 17(2), 346–373 (1992)
16. Li, G., Feng, J., Chin Ooi, B., Wang, J., Zhou, L.: An effective 3-in-1 keyword search method over heterogeneous data sources. Inf. Syst. 36, 248–266 (2011)
17. Burcher, N.:
`http://www.nickburcher.com/2010/03/`
`facebook-usage-statistics-march-2010.html`
18. Peleg, D.: Proximity-preserving labeling schemes. J. Graph Theory 33, 167–176 (2000)
19. Samet, H., Sankaranarayanan, J., Alborzi, H.: Scalable network distance browsing in spatial databases. In: SIGMOD, pp. 43–54 (2008)
20. Wei, F.: TEDI: efficient shortest path query answering on graphs. In: SIGMOD, pp. 99–110 (2010)
21. Xiao, Y., Wu, W., Pei, J., Wang, W., He, Z.: Efficiently indexing shortest paths by exploiting symmetry in graphs. In: EDBT, pp. 493–504 (2009)