Density Index and Proximity Search in Large Graphs

Nan Li
Computer Science
UC Santa Barbara
Santa Barbara, CA 93106
nanli@cs.ucsb.edu

Xifeng Yan
Computer Science
UC Santa Barbara
Santa Barbara, CA 93106
xyan@cs.ucsb.edu

Zhen Wen
Social Network Analytics
IBM Research
Hawthorne, NY 10532
zhenwen@us.ibm.com

Arijit Khan
Computer Science
UC Santa Barbara
Santa Barbara, CA 93106
arijitkhan@cs.ucsb.edu

ABSTRACT

Given a large real-world graph where vertices are associated with labels, how do we quickly find interesting vertex sets according to a given query? In this paper, we study *label-based proximity search* in large graphs, which finds the top-k query-covering vertex sets with the smallest diameters. Each set has to cover all the labels in a query. Existing greedy algorithms only return approximate answers, and do not scale well to large graphs. We propose a novel framework, called gDensity, which uses density index and likelihood ranking to find vertex sets in an efficient and accurate manner. Promising vertices are ordered and examined according to their likelihood to produce answers, and the likelihood calculation is greatly facilitated by density indexing. Techniques such as progressive search and partial indexing are further proposed. Experiments on real-world graphs show the efficiency and scalability of gDensity.

Categories and Subject Descriptors

G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms*

Keywords

Graph mining, Proximity search, Indexing

1. INTRODUCTION

Graphs and networks can model various types of interactions in a myriad of applications [12, 15, 18]. They are used to encode complex relationships such as chemical bonds, entity relations, social interactions, and so on. In contemporary graphs, vertices and edges are often associated with attributes. While searching the graphs, what is interesting is not only the connectivity, but also the attributes, such as labels and weights. Figure 1 shows a graph where vertices contain numerical labels. Consider a succinct yet fundamental graph query formula: given a set of labels, find vertex sets covering these labels and rank the sets by their connectivity. Viable connectivity measures include diameter, edge density, and minimum spanning tree. In Figure 1, if we want to find vertex sets that cover labels $\{1, 2, 3\}$, and the diameter of a vertex set is its longest pairwise shortest-path, we can return S_3, S_1 and S_2 in ascending order of diameters.

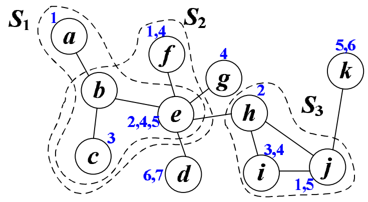


Figure 1: Label-Based Proximity Search

Applications of such a setting abound. The vertex labels can represent movies recommended by a user, functions carried by a gene, skills owned by a professional, and so on. Such queries help solve various interesting problems in real-world graphs: (1) in a protein network where vertices are proteins and labels are their annotations, *find a set of closely-connected proteins with certain annotations*; (2) in a collaboration network where vertices are experts and labels are their skills, *find a well-connected expert team with required skills* [18]; (3) in an intrusion network where vertices are computers and labels are intrusions they initiate, *find a set of intrusions that happen closely together*. The list of applications continues: finding a group of close friends with certain hobbies, finding a set of related movies covering certain genres, finding a group of well-connected customers interested in certain products, and many others.

We study *label-based graph proximity search*, to find the top-k vertex sets with the smallest diameters, for a query containing distinct labels. Each set covers all the labels in the query. The advantages of using diameter as a measure are shown in [16]. *Graph proximity query* is a general and simple way to query graphs. Lappas et al. [18] studied a similar problem called Diameter-Tf for expert team formation and adopted a greedy algorithm, `RarestFirst`, to return a 2-approximate answer (the returned set has a diameter no greater than two times of the optimal diameter). Diameter-Tf is NP-hard [18]. In this paper, we propose a scalable solution to find top-k answers efficiently in large graphs, for queries with moderate sizes. Our goals are: (1) finding the *exact* top-k answers, not approximate answers; (2) designing a novel graph index for fast query processing.

Other similar studies include [16] and [8]. Kargar and An [16] studied finding the top-k r-cliques with smallest weights, where an r-clique is a set of vertices covering all the input keywords and the distance between each two is constrained. Two algorithms are proposed: branch and bound and polynomial delay. The former is an exact algorithm, but it is slow and does not rank the answers; the latter ranks the answers, but is a 2-approximation. Gajewar and Sarma [8] studied the team formation problem with subgraph density as the objective to maximize and focused on approximation algorithms. The problem definition is different in our paper and we aim for exact and fast solutions.

A naive approach is to enumerate all query-covering vertex sets, linearly scan them and return the top-k with the smallest diameters. This is costly for large graphs. It is desirable to have a mechanism to identify the most *promising* graph regions, or local neighborhoods, and examine them first. If a neighborhood covers the query labels, and meanwhile has high edge density, it tends to contain vertex sets with small diameters that cover the query. We propose a framework, called gDensity, to address the proximity search problem using this principle. Empirical studies on real-world graphs show that gDensity improves the query performance over competing methods.

Our contributions. (1) We introduce density index in graphs and the workflow to answer graph proximity queries using such index. (2) We demonstrate that if the neighborhoods are sorted and examined according to the likelihood, the search time can be reduced. (3) We propose partial indexing techniques to significantly reduce index size and index construction time, with negligible loss in query performance. (4) Empirical studies on real-world graphs show that gDensity is effective and scalable.

2. PRELIMINARIES

For a vertex-attributed graph $G = (V, E, \mathbb{L})$, each vertex is attached with a set of labels from $\mathbb{L} = \{\alpha_1, \ldots, \alpha_l\}$. $L(u)$ denotes the label set of vertex u. For the ease of presentation, we focus on un-directed and un-weighted graphs. However, the proposed framework can be extended to directed and weighted graphs as well.

Definition 1 (Cover). *Given a vertex-attributed graph $G = (V, E, \mathbb{L})$, a vertex set $S \subseteq V$, and a query $Q \subseteq \mathbb{L}$, S "covers" Q if $Q \subseteq \bigcup_{u \in S} L(u)$. S is also called a query covering vertex set. S is a minimal cover if S covers Q and no subset of S covers Q.*

Definition 2 (Diameter). *Given a graph $G = (V, E)$ and a vertex set $S \subseteq V$, the diameter of S is the maximum of the pairwise shortest distances of all vertex pairs in S, i.e., $\max_{u,v \in S}\{dist(u, v)\}$, where $dist(u, v)$ is the shortest-path distance between u and v in G.*

The diameter of a vertex set S, denoted by $\texttt{diameter}(S)$, is different from the diameter of a subgraph induced by S, since the shortest path between two vertices in S might not completely lie in the subgraph induced by S.

Problem 1 (Graph Proximity Search). *Given a vertex attributed graph $G = (V, E, \mathbb{L})$ and a query Q containing q labels ($|Q| = q$), label-based graph proximity search finds the top-k vertex sets $\{S_1, S_2, \ldots, S_k\}$ with the smallest diameters. Each set S_i is a minimal cover of Q.*

In many applications, it might not be useful to generate sets with large diameters, especially for graphs exhibiting the small-world property. One might apply a constraint such that $\texttt{diameter}(S_i)$ does not exceed a threshold.

`RarestFirst` is a greedy algorithm proposed by [18] that approximates the top-1 answer. First, `RarestFirst` finds the rarest label in query Q that is contained by the smallest number of vertices in G. Secondly, for each vertex v with the rarest label, it finds its nearest neighbors that contain the remaining labels in Q. Let R_v denote the maximum distance between v and these neighbors. Finally, it returns the vertex with the smallest R_v, and its nearest neighbors containing the other labels in Q, as an approximate top set. `RarestFirst` yields a 2-approximation in terms of diameter, i.e., the diameter of the top set found by `RarestFirst` is no greater than two times that of the real top set.

`RarestFirst` can be very fast if all pairwise shortest distances are pre-indexed. This is costly for large graphs. gDensity does not have such prerequisite. Besides, the goal of gDensity is finding the *real* top-k answers (not approximate answers). gDensity works well for queries with small-diameter answers, which are common in practice. For small graphs where all pairwise shortest distances can be pre-indexed, or for some difficult graphs where optimal solutions are hard to derive, `RarestFirst` could be a better option. In Section 8, we implement a modified top-k version of `RarestFirst` using the proposed progressive search technique, whose query performance is compared with gDensity.

2.1 Cost Analysis and Observations

The naive solution in Section 1 scales poorly to large graphs because: (1) It entails calculating all-pairs shortest distances. It takes $O(|V|^3)$ time using the Floyd-Warshall algorithm and $O(|V||E|)$ using the Johnson's algorithm. (2) It examines all query-covering sets without knowing their likelihood to be a top-k set. The time complexity is $O(|V|^q)$, with q being the size of the query.

An alternative approach is to examine the local neighborhoods of promising vertices, and find high-quality top-k candidates quickly. The search cost is the number of vertices examined times the average time to examine each vertex. It is important to prune unpromising vertices. A possible pruning strategy is: let d^* be the maximum diameter of the current top-k candidates. d^* decreases when new query covers are found to update the top-k list. d^* can be used to prune vertices which do not locally contain covers with diameter $< d^*$. gDensity instantiates such idea using nearest label pruning and progressive search, to quickly prune vertices which are unable to produce qualified covers. The key is to find vertices whose neighborhoods are likely to produce covers with small diameters, so that the diameter of the discovered top-k candidates can be quickly reduced.

Definition 3 (d-Neighborhood). *Given a graph $G = (V, E, \mathbb{L})$ and a vertex u in G, the d-neighborhood of u, $N_d(u)$, denotes the set of vertices in G whose shortest distance to u is no more than d, i.e., $\{v|dist(u, v) \leq d\}$.*

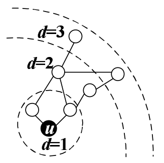


Figure 2: d-Neighborhood Example

Intuitively, the d-neighborhood of u, $N_d(u)$, can be regarded as a sphere of radius d centered at u. Figure 2 shows an example of the 1-neighborhood, 2-neighborhood and 3-neighborhood of vertex u. For each vertex u in G, we have to determine if its d-neighborhood is likely to generate vertex sets with small diameters to cover the query. The key question is: how do we estimate such likelihood?

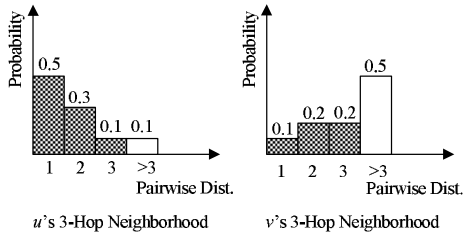


Figure 3: Pairwise Distance Distribution Example

We propose density index to solve the likelihood estimation problem. Figure 3 shows the intuition behind density index. Assume there are two regions, i.e., the 3-neighborhoods of vertices u and v. The distributions of pairwise shortest distances in both regions are plotted in Figure 3. The horizontal axis is the pairwise distance, which are 1, 2, 3 and greater than 3. The vertical axis shows the percentage of vertex pairs with those distances. Given a query Q, if both regions exhibit similar label distribution, which one has a higher chance to contain a query cover with smaller diameter? Very likely u's ! This is because there is a much higher percentage of vertex pairs in u's neighborhood that have smaller pairwise distances. Density index is built on this intuition. For each vertex, the pairwise distance distribution in its local neighborhood is indexed offline, which will later be used to estimate its likelihood online. Section 4 describes our indexing techniques in depth.

3. PROXIMITY SEARCH FRAMEWORK

gDensity consists of the following components.

Density Index Construction: We create a histogram-like profile for each vertex depicting the distribution of the pairwise shortest distances in its d-neighborhood, for $1 \leq d \leq d_I$. d_I is a user specified threshold.

Seed Vertex Selection: Instead of examining the entire vertex set V, we only examine the neighborhoods of the vertices containing the least frequent label in the query Q. These vertices are called *seed vertices*. Since a qualified vertex set must contain at least one seed vertex, we can solely focus on searching the neighborhoods of seed vertices.

Likelihood Ranking: Seed vertices are examined according to their likelihood to produce qualified vertex sets in their local neighborhoods. Vertices with the highest likelihoods are examined first.

Progressive Search: We maintain a buffer, $\mathbf{B}_k$, of the top minimal query covers discovered so far. A sequential examination finds qualified vertex sets with diameters $1, 2, \ldots$, until the top-k buffer is full (contains k answers). This mechanism enables early termination of the search. Once the top-k buffer is full, the algorithm stops, because all of undiscovered vertex sets will have diameter at least as large as the maximum diameter in the top-k buffer.

Nearest Label Pruning: Let d be the current diameter used in progressive search. Once d is determined, gDensity traverses seed vertices to find query covers with diameter exactly as d. The value of d increases from 1 and is used to prune seed vertices that are unable to generate qualified covers. Such seeds have their nearest neighbor with any query label further than d.

gDensity Algorithm. Algorithm 1 shows the overall work flow of gDensity. In the following sections, we elaborate the details regarding the above components.

Algorithm 1: gDensity Framework

```
Input: Graph $G$, indexing radius $d_I$, query $Q$, $k$
Output: The top-$k$ vertex sets with the smallest diameters
1  Indexing $G$ from 1 to $d_I$;
2  $\alpha_1 \leftarrow$ the least frequent label in $Q$;
3  Seed vertices $V_{\alpha_1} \leftarrow$ the vertices with label $\alpha_1$;
4  Top-$k$ buffer $\mathbf{B}_k \leftarrow \varnothing$, $d \leftarrow 1$;
5  while true do
6      Compute the likelihood for all the seed vertices;
7      Rank the seeds decreasingly using the likelihood;
8      for each seed vertex in the ranked list do
9          if it is not pruned by the nearest label rule then
10             Check its $d$-neighborhood for minimal query
               covers with diameter $d$;
11             Update $\mathbf{B}_k$ with discovered minimal covers;
12             if $\mathbf{B}_k$ is full then
13                 return $\mathbf{B}_k$;
14     $d$++;
```

4. INDEXING AND LIKELIHOOD RANK

In order to estimate the likelihood online fast, we propose density indexing to pre-compute indices that reflect local edge connectivity. How to utilize the density index to facilitate likelihood estimation is discussed in Section 4.2.

4.1 Density Index

Density indexing records the pairwise shortest distance distribution of a local neighborhood. In order to maintain a succinct index structure, the density index is solely based on topology. For each vertex u, we first grow its d-neighborhood, $N_d(u)$, using breadth-first search. The pairwise shortest distances for all vertex pairs in $N_d(u)$ are then calculated. Some pairwise distances might be greater than d (at most $2d$). Density index records the histogram of the discrete distance distribution, i.e., the percentage of pairs whose distance is h, for $1 \leq h \leq 2d$, as shown in Figure 3. Density index only needs to record the distribution, not all-pairs shortest distances. Section 6 will discuss how to derive density index approximately.

Let I be an indicator function and $P(h|N_d(u))$ be the percentage of vertex pairs with distance h. We have

$$P(h|N_d(u)) = \frac{\sum_{v_i,v_j \in N_d(u)} I(dist(v_i,v_j) = h)}{\sum_{v_i,v_j \in N_d(u)} I(1)}. \quad (1)$$

Users can reduce the histogram size by combining the percentage of pairs whose distance is greater than a certain threshold $\hat{h}$, as in Eq. (2). Usually $\hat{h} = d$.

$$P(> \hat{h}|N_d(u)) = \frac{\sum_{v_i,v_j \in N_d(u)} I(dist(v_i,v_j) > \hat{h})}{\sum_{v_i,v_j \in N_d(u)} I(1)}. \quad (2)$$

Since the distribution can change with respect to the radius of the neighborhood, we build the histograms for varying d-neighborhoods of each vertex, with $1 \le d \le d_I$, where d_I is a user-specified indexing locality threshold. Figure 2 shows the neighborhoods of vertex u with different radii. For each radius d, we build a histogram similar to Figure 3. Intuitively, if $N_d(u)$ contains a higher percentage of vertex pairs with small pairwise distances and it also covers Q, $N_d(u)$ should be given a higher priority during search. This intuition leads to the development of likelihood ranking.

Supplementary indices are also used to facilitate likelihood ranking and nearest label pruning (Section 5.2). (1) For each label α_i in G, global label distribution index records the number of vertices in G that contain label α_i. (2) Inspired by the indexing scheme proposed by He et al. [12], gDensity further indexes, for each vertex in G, its closest distance to each label within its d-neighborhood.

Since density index has histogram structure as in Figure 3, the space cost of density index is $\Sigma_{d=1}^{d=d_I} O(|V|d) = O(|V|d_I^2)$. For index time, suppose the average vertex degree in G is b, then for each vertex u, the expected size of its d-neighborhood is $O(b^d)$. If we use all pairwise distances within $d \in [1, d_I]$ to build the density index, the total time complexity will be $O(|V|b^{2d_I})$. The index time might be huge even for small d_I. This motivates us to design partial indexing (Section 6), which greatly reduces index time and size, while maintaining satisfying index quality.

4.2 Likelihood Ranking

Given a query $Q = \{\alpha_1, \alpha_2, \ldots, \alpha_q\}$, let $\alpha_1 \in Q$ be the label contained by the smallest number of vertices in G. α_1 is called the *rarest* or *least frequent* label in Q. Let $V_{\alpha_1} = \{v_1, v_2, \ldots, v_m\}$ be the vertex set in G containing label α_1. These vertices are referred to as the *seed vertices*. Algorithm 1 shows that the d-neighborhoods of all seed vertices will be examined according to their likelihood to produce minimal query covers with diameter exactly as d, while d is gradually relaxed. For each seed vertex $v_i(i = \{1, \ldots, m\})$, its likelihood depends on the pairwise distance distribution of its d-neighborhood, $N_d(v_i)$. The likelihood reflects how densely the neighborhood is connected and can be computed from the density index.

4.2.1 Likelihood Computation

Definition 4 (Distance Probability). *Randomly selecting a pair of vertices in $N_d(v_i)$, let $p(v_i, d)$ denote the probability for this pair's distance to be no greater than d. $p(v_i, d)$ can be obtained from the density index, $P(h|N_d(v_i))$,*

$$p(v_i, d) = \sum_{h=1}^{d} P(h|N_d(v_i)). \quad (3)$$

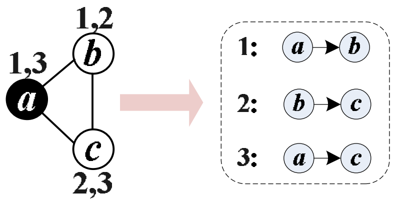


Figure 4: Minimal Cover Example

Definition 5 (Likelihood). *Randomly selecting a vertex set with q vertices in $N_d(v_i)$, let $\ell(v_i, d)$ denote the probability for this set's diameter to be no greater than d. With density index (Eq. (1)), $\ell(v_i, d)$ can be estimated as*

$$\begin{aligned}\ell(v_i, d) &\sim p(v_i, d)^{q(q-1)/2} \\ &\sim \big(\sum_{h=1}^{d} P(h|N_d(v_i))\big)^{q(q-1)/2}\end{aligned} \quad (4)$$

If the diameter of a vertex set is no greater than d, all the vertex pairs within this set must be at most d distance away from each other. If we assume independency of pairwise distances among vertex pairs, Eq. (4) can be obtained, given that the vertex set has size q. Certainly, it is an estimation, since pairwise distances should follow some constraints, such as triangle inequality in metric graphs. For a given query Q of size q, gDensity uses $\ell(v_i, d)$ as the likelihood to rank all the seed vertices. Apparently, seed vertices whose local neighborhoods exhibit dense edge connectivity tend to be ranked with higher priority. With the presence of density index, likelihood can be easily computed as in Eq. (4).

For all the seed vertices in V_{α_1}, we sort them in descending order of $\ell(v_i, d)$ and find minimal query covers with diameter d individually. For each seed vertex under examination, we first perform (unordered) cartesian product across query label support lists to get candidate query covers, and then select minimal covers from those covers. Such approach assures that all possible minimal query covers will be found from each seed vertex's d-neighborhood.

4.2.2 Cartesian Product and Query Covers

For each seed vertex v_i with label α_1, we generate a support vertex list for each label in the query $Q = \{\alpha_1, \alpha_2, \ldots, \alpha_q\}$ in v_i's d-neighborhood. Let n_j be the size of the support list for α_j. Let $\pi_d(v_i)$ denote the total number of possible query covers generated by performing a cartesian product across all label support lists, where each cover is an unordered vertex set consisting of one vertex from each support list.

$$\pi_d(v_i) = \prod_{j=1}^{q} n_j. \quad (5)$$

Not all such covers are minimal. In Figure 4, if $Q = \{1, 2, 3\}$, three support lists are generated in a's 1-neighorhood. For example, label 1 has two vertices in its list, a and b. One of the covers across the lists is $\{a, b, c\}$, which is not minimal. From $\{a, b, c\}$, we shall generate 3 minimal covers, $\{a, b\}$, $\{b, c\}$ and $\{a, c\}$. For each seed vertex, gDensity scans all candidate covers and generates those minimal ones to update the top-k list. Note that generating minimal covers from the supporting lists is an NP-hard problem itself. Here we find the minimal covers in a brute-force manner. It

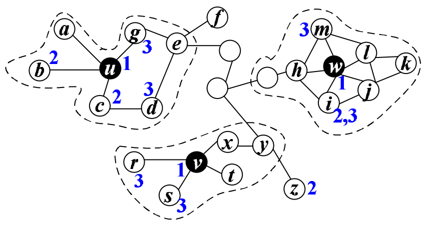


Figure 5: Pruning and Progressive Search Example

is a relatively a time-consuming process. However, with progressive search, which will be described later, we only need to do this locally in a confined neighborhood. Experiment results will show that gDensity still achieves good empirical performance on large graphs.

5. PROGRESSIVE SEARCH AND PRUNING

Progressive search enables search to terminate once there are k answers found. Nearest label pruning is used together with progressive search to prune unpromising seed vertices.

5.1 Progressive Search

The search cost increases exponentially when d increases. Instead of testing a large value of d first, we propose to check neighborhoods with gradually relaxed radii. A top-k buffer, $\mathbf{B}_k$, is maintained to store the top vertex sets with the smallest diameters found so far. We progressively examine the neighborhoods with $d = 1$, $d = 2$, and so on, until $\mathbf{B}_k$ is full. Such mechanism allows the search to terminate early. For example, if k answers are found while checking the 1-hop neighborhoods of all seed vertices, the process can be terminated without checking neighborhoods with $d \geq 2$. In Figure 5, suppose the query is $Q = \{1, 2, 3\}$, and we have three seed vertices $\{u, v, w\}$. Starting with $d = 1$, we explore the 1-hop neighborhoods of all three, looking for covers with diameter 1, which gives us $\langle\{w, i\}, 1\rangle$. Here, $\langle\{w, i\}, 1\rangle$ means the diameter of $\{w, i\}$ is 1. Moving onto $d = 2$, we explore the 2-hop neighborhoods of all the three vertices (in dashed lines), seeking covers with diameter 2, which gives us $\{\langle\{u, c, d\}, 2\rangle, \langle\{u, c, g\}, 2\rangle, \langle\{u, b, g\}, 2\rangle\}$. If $k = 4$, search process can terminate here.

5.2 Nearest Label Pruning

We further propose a pruning strategy called nearest label pruning. Used together with progressive search, it is able to prune unfavorable seeds from checking. Suppose the current diameter used in progressive search is d. For each seed vertex v_i, we calculate its shortest distance to each label in Q within its d-neighborhood, $N_d(v_i)$. If there is a label $\alpha \in Q$ such that the shortest distance between a vertex with α and v_i is greater than d, we skip checking v_i and its neighborhood, since $N_d(v_i)$ is not able to generate a query cover with diameter $\leq d$. Furthermore, v_i and the edges emanating from it can be removed. For example in Figure 5, if $Q = \{1, 2, 3\}$ and at certain point $d = 2$. Four query covers have been inserted into $\mathbf{B}_k$ together with their diameters, which are $\{\langle\{w, i\}, 1\rangle, \langle\{u, c, d\}, 2\rangle, \langle\{u, c, g\}, 2\rangle, \langle\{u, b, g\}, 2\rangle\}$. We no longer need to check the neighborhood of vertex v. This is because the shortest distance between v and label 2 is 3, which is greater than the current diameter constraint $d = 2$.

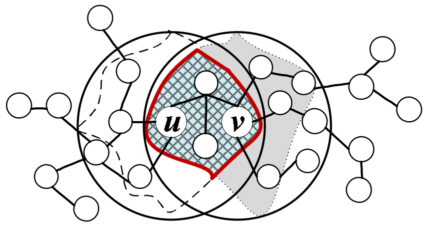


Figure 6: Partial Materialization Example

6. PARTIAL INDEXING

Building the complete density index for large graphs can be expensive. We propose partial indexing to build an approximate index using partial neighborhood information.

6.1 Partial Materialization

Using random sampling, *partial materialization* allows density index to be built approximately by accessing only a portion of the local neighborhoods. For each vertex u to index: (1) only a subset of vertices in u's d-neighborhood are used to form an approximate neighborhood; (2) only a percentage of vertex pairs are sampled from such approximate neighborhood to construct the partial density index. More specifically, the following steps are performed.

(a) Given a vertex u and an indexing distance d, a subset of vertices are randomly sampled from $N_d(u)$. An approximate d-neighborhood, $\tilde{N}_d(u)$, consists of those sampled vertices and their distances to u.

(b) Randomly pick a vertex v from $\tilde{N}_d(u)$.

(c) Get the intersection of $\tilde{N}_d(u)$ and $\tilde{N}_d(v)$, $\chi_d(u, v)$. For a random vertex x in $\chi_d(u, v)$, sample the pair (x, v) and record their distance as in $\tilde{N}_d(v)$.

(d) For a random vertex x in $\tilde{N}_d(u)$ but not in $\chi_d(u, v)$, sample the pair (x, v) and record their distance as $> d$.

(e) Repeat Steps (b) to (d) until a certain percentage, p, of vertex pairs are sampled from $N_d(u)$.

(f) Draw the pairwise distance distribution using sampled pairs to approximate the real density distribution in $N_d(u)$.

Figure 6 (better viewed in color) shows an example. The solid circles centered at vertices u and v are their actual 2-neighborhoods. The white free-shaped region surrounding u is its approximate 2-neighborhood, $\tilde{N}_2(u)$; similarly, the gray free-shaped region surrounding v is $\tilde{N}_2(v)$. The region with grid pattern circumscribed by a solid red line is the intersection of both approximate neighborhoods, $\chi_2(u, v)$. Each sampled vertex x from u's approximate 2-neighborhood forms a pair with v, (x, v). If x is in the intersection, $\chi_2(u, v)$, the pair (x, v) is sampled with a pairwise distance recorded as in $\tilde{N}_d(v)$; otherwise it is sampled with a pairwise distance recorded as $> d$. A localized version of Metropolis-Hastings random walk (MHRW) sampling [6, 9] is used to sample vertices from $N_d(u)$ (Step (a)).

6.2 Representative Vertices

Partial materialization reduces the indexing cost for an individual vertex. To further reduce the indexing cost, we can reduce the number of vertices to be indexed. The intuition is: if two vertices u and v have similar local topological

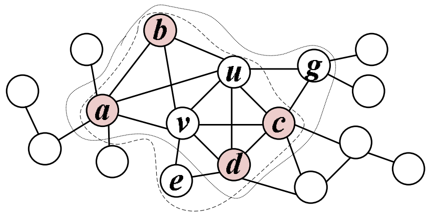


Figure 7: Representative Vertex Example

structure, there is no need to build the density index for u and v separately, given that the distance distributions in the neighborhoods of u and v are similar. For example, in Figure 7, the 1-hop neighborhoods of vertices u and v overlap each other to a great extent. The common adjacent neighbors of u and v in Figure 7 are $\{a, b, c, d\}$, which is 66.7% of u and v's 1-neighborhoods. Can we build the density index of v with the aid of the density index of u?

A simple strategy employed in gDensity is to use the density of u to *represent* that of v (or vice versa), if the percentage of common 1-hop neighbors of u and v exceeds a certain threshold in both u and v's neighborhoods. Let σ denote such threshold. In this case, vertex u is considered as the *representative vertex* of v. We only index those vertices which are representatives of some others, and use their density index to represent others'. Such strategy quickly cuts down the number of vertices to index, thus reduces the index time and index size. As experimented in Section 8, $\sigma \geq 30\%$ would suffice to produce effective partial index, which still yields good online query processing performance.

7. OPTIMALITY OF GDENSITY

Theorem 1 (Optimality of gDensity). *For a query, gDensity finds the optimal top-k answers. Partial indexing and likelihood ranking affect the speed of query processing, but not the optimality of the results.*

Proof Sketch. Since seed vertices contain the least frequent label in the query, all query covers contain at least one seed vertex. Confining the search to the neighborhoods of seed vertices does not leave out any answers. Progressive search assures that the diameters of unexamined vertex sets will be no less than the maximum diameter in the top-k buffer. Therefore the final top-k answers returned will have the smallest diameters. Indexing and likelihood ranking identify "promising" seed vertices and guide the algorithm to discover the top-k answers *faster*. If more promising seeds are ranked higher, the top-k buffer will be filled up faster. It is possible for a seed vertex, whose neighborhood contains good answers, to be ranked lower than other less promising seeds. However, this would only affect the *speed* of filling up the top-k buffer. It would not change the fact that the top-k buffer contains the top-k smallest diameters. Partial indexing further reduces the indexing cost by indexing only partial information. It approximates the *indexing* phase, and will not affect the optimality of the *query* phase. Therefore gDensity always returns the optimal answers. □

The likelihood computed in Eq. (4) for ranking seed vertices assumes independence among distances between vertices, which might not be valid for some seeds due to possible skewed distribution. However, as long as it is valid for some seed vertices, the top-k buffer can be quickly updated with answers discovered surrounding those seeds, thus speeding up the search. The goal of likelihood ranking is to locate promising regions containing many potential answers and fill up the top-k buffer quickly. Section 8.2.3 empirically confirms the effectiveness of likelihood ranking.

We reiterate that partial indexing only affects the estimated density and likelihood ranking. Only the speed of the top-k search will be affected by partial indexing. Partial indexing will not impair the optimality of gDensity in terms of returning the top-k answers with the smallest diameters.

8. EXPERIMENTS

The empirical evaluation contains: (1) comparison between gDensity and the modified `RarestFirst`; (2) evaluation of partial indexing; (3) scalability test of gDensity. All experiments are run on a machine that has a 2.5GHz Intel Xeon processor (only one core is used), 32G RAM, and runs 64-bit Fedora 8 with LEDA 6.0 [22].

8.1 Data Sets

DBLP Data. DBLP is a collaboration network in computer science. Each vertex is an author and each edge is a collaborative relation. We consider the keywords in the paper titles of an author as vertex labels. The DBLP graph used contains 387,547 vertices and 1,443,873 edges.

Intrusion Data. In this graph, each vertex is a computer and each edge is an attack. A vertex has a set of labels, which are intrusion alerts initiated by this computer. There are 1035 distinct alerts. Intrusion alerts are logged periodically. We use one daily data set (*IntruDaily*) with 5,689 vertices and 6,505 edges, and one annual data set (*IntruAnn*) with 486,415 vertices and 1,666,184 edges.

WebGraph Data. Web graph (`http://webgraph.dsi.unimi.it/`) is a collection of UK web sites. Each vertex is a web page and each edge is a link. A routine is provided to attach the graph with random integer labels following Zipf distribution [20]. Five subgraphs are used, whose vertex numbers are 2M, 4M, 6M, 8M and 10M, and whose edge numbers are 9M, 16M, 23M, 29M and 34M. The 2M graph is a subgraph of the 4M graph, and so on.

50 queries are generated for each graph used. Query time is averaged over all the queries. Table 1 shows some query examples. Indexing is conducted up to 3 hops for all the graphs. If not otherwise specified, partial indexing is the default indexing. The vertex pair sampling percentage is 40% and the 1-hop neighborhood similarity threshold in representative vertex selection is $\sigma = 30\%$.

8.2 gDensity vs. Baselines

8.2.1 Baselines

We discovered in our experiments that the original `RarestFirst` method does not scale well to large graphs. Thus we add a constraint D on the diameters of the top-k vertex sets in `RarestFirst`, limiting the search to each seed's D-neighborhood. We further use progressive search to speed up `RarestFirst`. Algorithm 2 outlines the customized top-k `RarestFirst`. Another baseline method is a variant of gDensity, called "gDensity w/o LR", which removes likelihood ranking from gDensity. All of the other components are still kept in gDensity w/o LR. gDensity w/o LR examines the seed vertices in a random order. The goal is to inspect the

Table 1: Query Examples

DBLP Graph

ID	Query	ID	Query
1	"Ranking", "Databases", "Storage"	4	"Intelligence", "TCP/IP", "Protocols"
2	"Bayesian", "Web-graphs", "Information"	5	"Complexity", "Ranking", "Router", "Generics"
3	"Mining", "Graph", "Stream"	6	"Image", "Allocation", "Statistical", "Multi-core"

Intrusion Graph

ID	Query
1	"HTTP_Fields_With_Binary", "HTTP_IIS_Unicode_Encoding", "MSRPC_RemoteActivate_Bo"
2	"FTP_Mget_DotDot", "HTTP_OracleApp_demo_info", "HTTP_WebLogic_FileSourceRead"
3	"Content_Compound_File_Bad_Extension", "HTTP_URL_Name_Very_Long", "HTTP_URL_Repeated_Dot"
4	"SMB_Startup_File_Access", "pcAnywhere_Probe", "HTTP_Viewsrc_fileread", "Failed_login-unknown_error"
5	"HTTP_Passwd_Txt", "DNS_Windows_SMTP_Overflow", "OSPF_Link_State_Update_Multicast", "POP_User"

Algorithm 2: RarestFirst With Progressive Search

```
Input: Graph $G$, Query $Q$, diameter constraint $D$, $k$
Output: Top-$k$ vertex sets with smallest diameters
1  $\alpha_1 \leftarrow$ the least frequent label in $Q$;
2  while the top-$k$ buffer is not full do
3      for $d$ from 1 to $D$ do
4          for each vertex $v$ with $\alpha_1$ do
5              $N_d(v) \leftarrow v$'s $d$-neighborhood;
6              $S \leftarrow \{v$ and $v$'s nearest neighbors in $N_d(v)$
               that contain other labels in $Q\}$;
7              Extract minimal covers from $S$;
8              for each minimal cover do
9                  If it is not yet in the top-$k$ buffer, and its
                   diameter $\leq D$, insert it into the buffer
                   according to its diameter;
10                 if the top-$k$ buffer is full then
11                     return top-$k$ buffer;
```

actual effect of likelihood ranking. Both methods are used for comparative study against gDensity.

8.2.2 Evaluation Methods

The comparison is done on two measures, query time (in seconds) and answer miss ratio (in percentage). RarestFirst could miss some real top-k answers since it is an approximate solution. Miss ratio is the percentage of real top-k answers RarestFirst fails to discover. For example, if the real top-5 all have diameter 2 and if 2 of the top-5 answers returned by RarestFirst have diameter greater than 2, the miss ratio is $2/5 = 40\%$. gDensity and gDensity w/o LR are able to find all real top-k answers.

We also examine the impact of label distribution. There is an interesting tradeoff here. If labels are densely distributed (the average number of vertices containing each label is high), each neighborhood might potentially contain many answers and the algorithm stops early; if the labels are sparsely distributed, the seed vertex list is shorter and the candidate set for each seed is smaller. We thus design a group of experiments where we synthetically regenerate labels for graphs *DBLP*, *IntruAnn* and *WebGraph* 10M, under certain label ratios. The ratio is measured as $|L|/|V|$, where $|L|$ is the total number of distinct labels in G. Each vertex is randomly assigned one of those synthetic labels.

8.2.3 Query Time Comparison

Figure 8 shows the query time comparison of gDensity, gDensity w/o LR and the modified RarestFirst. The left-most column shows how the average query time changes with k. The advantage of gDensity over the modified RarestFirst is apparent. The effectiveness of likelihood ranking is evident on *DBLP* and *IntruAnn*, where gDensity greatly outperforms gDensity w/o LR. Likelihood ranking does not work as well on *WebGraph* 10M. It is possible that *WebGraph* 10M does not contain many patterns or dense regions, rendering it difficult to rank seed vertices effectively.

The remaining columns depict how the average query time changes with the synthetic label ratio, $|L|/|V|$. The tradeoff between dense (small label ratio) and sparse (large label ratio) label distribution clearly shows on *DBLP*, where the gDensity query time first goes up and then goes down. It goes up because as label distribution becomes sparse, more seeds and larger values of d need to be examined to find the top-k, since each region contains less answers. It then goes down because the seed vertex list gets shorter and the set of candidate covers to check for each seed gets smaller. RarestFirst sometimes outperforms gDensity because the diameter constraint lets RarestFirst finish without finding all the optimal top-k sets. In the next section, we will show the percentage of answers missed by RarestFirst.

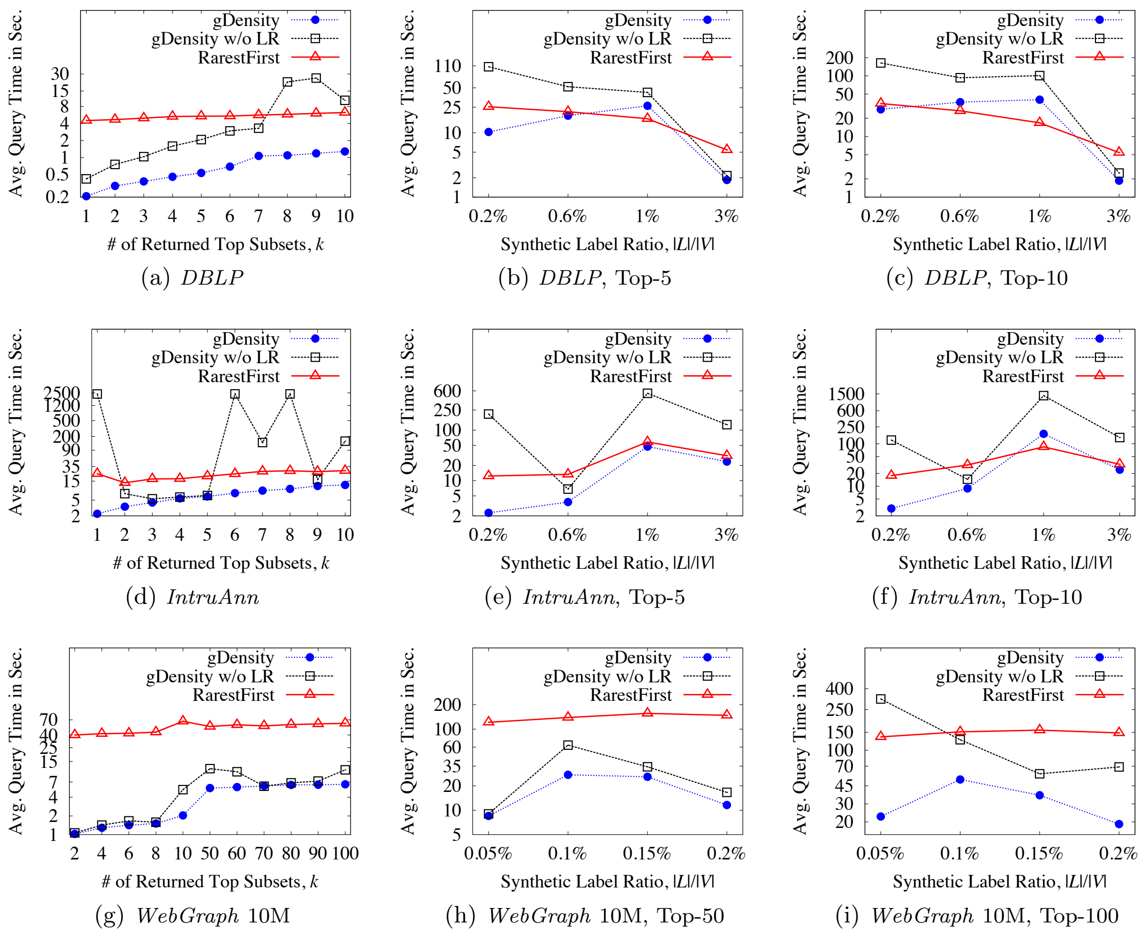
8.2.4 Query Optimality Comparison

Query optimality is measured by the answer miss ratio. gDensity discovers the real top-k answers, thus the miss ratio of gDensity and gDensity w/o LR will be 0. Figure 10 shows how the miss ratio of RarestFirst changes with k. Miss ratio gradually increases with k. In the worst case, RarestFirst misses 52.8% of the real top-k answers. On average, the miss ratio is around 30%. Clearly, compared to gDensity, RarestFirst suffers from the drawback of possibly leaving out the optimal answers.

We also observe how miss ratio changes with the synthetic label ratio, $|L|/|V|$, in Figure 9. RarestFirst again is disadvantageous in terms of top-k optimality. The average miss ratios are 46.6%, 23.6% and 35.1% for synthetically labeled *DBLP*, *IntruAnn* and *WebGraph* 10M, respectively.

8.3 Partial Indexing Evaluation

Partial indexing reduces cost by indexing a subset of vertices using their approximate neighborhoods. We refer to the alternative, indexing all vertices using their exact neighborhoods, as "all indexing". The query performances of both indexing techniques are compared. Three thresholds of representative vertex selection are used for partial indexing: $\sigma = \{10\%, 30\%, 50\%\}$. Since all indexing is very time-consuming for large graphs, we can only afford to conduct the comparison on a small graph, *IntruDaily*.



(a) *DBLP* (b) *DBLP*, Top-5 (c) *DBLP*, Top-10

(d) *IntruAnn* (e) *IntruAnn*, Top-5 (f) *IntruAnn*, Top-10

(g) *WebGraph* 10M (h) *WebGraph* 10M, Top-50 (i) *WebGraph* 10M, Top-100

Figure 8: gDensity vs. Baseline Methods, Query Time

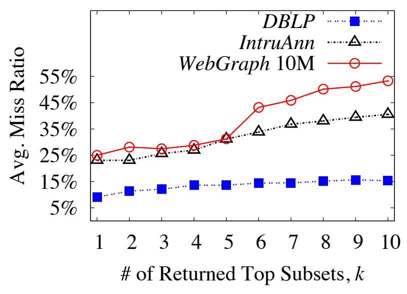


Figure 10: `RarestFirst` Miss Ratios vs. k

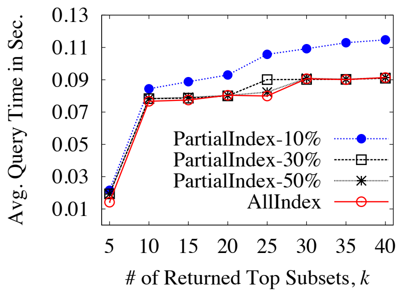


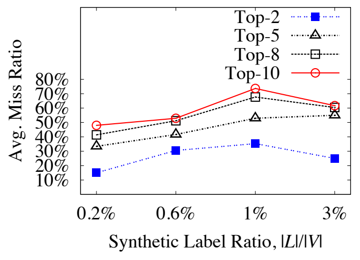
Figure 11: gDensity Partial vs. All Index: Query Time

As shown in Figure 11, the additional online query time partial indexing induces over all indexing is almost negligible, especially when $\sigma \geq 30\%$. The moderate performance margin between $\sigma = 10\%$ and $\sigma \geq 30\%$ might be attributed to the fact that there are not many vertex pairs whose neighborhood similarity falls between 10% and 30%.

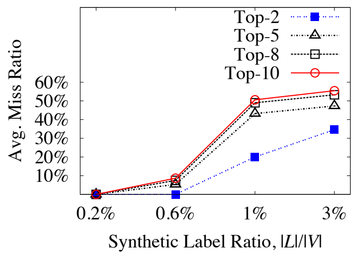
Table 2 shows the indexing time (seconds) and index size (MB) comparison. Partial indexing effectively reduces indexing cost. The indexing cost increases with the representative vertex threshold, because a higher threshold means a larger number of representative vertices to index.

8.4 gDensity Scalability Test

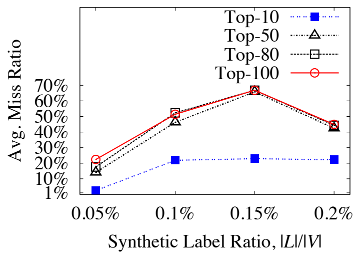
We conduct experiments on web graphs of increasing size to show the scalability of gDensity. Table 3 shows how the index time and size change when the graph increases from 2M to 10M vertices. Overall, the index time is satisfying and reasonable. The largest graph only takes 3.9 hours to index. The index size is no more than 30% of the graph size. Most importantly, the index time and size are approximately linear to the size of the graph. Therefore, partial indexing exhibits satisfying scalability over large graphs.



(a) *DBLP*



(b) *IntruAnn*



(c) *WebGraph* 10M

Figure 9: `RarestFirst` Miss Ratios vs. Synthetic Label Ratio

Table 2: Partial vs. All Index: Time & Size

Indexing Scheme	Time (Seconds)	Size (MB)
Partial Indexing, $\sigma = 10\%$	6.959	0.013
Partial Indexing, $\sigma = 30\%$	7.337	0.016
Partial Indexing, $\sigma = 50\%$	7.452	0.019
All Indexing	533.243	0.312

Table 3: gDensity Scalability Test: Index Time & Size

Vertex #	2M	4M	6M	8M	10M
Index Time (Hours)	0.92	1.65	2.53	3.23	3.85
Graph Size (MB)	234	390	528	678	786
Index Size (MB)	72	123	174	222	259

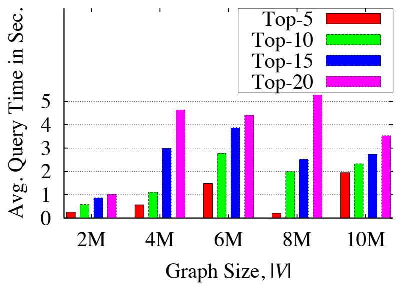


Figure 12: gDensity Scalability Test: Query Time

Figure 12 shows how query time changes when the graph size increases from 2M to 10M vertices, for various k values. We can see that the query time gradually increases and is still satisfying even for very large graphs. The query time is contingent on a number of factors such as the graph structure, the graph size, the label distribution, the query, k, and so on. Since the queries are randomly generated on each of the web graphs, it is possible for a smaller graph to encounter a more difficult query that entails longer processing time. Even for the same query, it is possible for it to be processed faster in a larger graph, since more answers might be found at an early stage. If we compare the runtime of top-5 for *WebGraph* 2M, top-10 for *WebGraph* 4M, top-15 for *WebGraph* 6M, and top-20 for *WebGraph* 8M (i.e., the value of k increases with the graph size), it is observed that the runtime increases linearly. Overall, gDensity is scalable with respect to the graph size.

9. RELATED WORK

Proximity Search. Typical proximity search in social networks includes link prediction [24], expert team formation [2, 8, 18, 26, 27]. The latter finds a team of experts with required skills. Existing methods include generic algorithms [26], simulated annealing [2], and so on. [18] adopts a 2-approximation algorithm to find a team of experts with the smallest diameter, where all-pairs shortest distances need to be pre-computed and no index structure is used to expedite the search. [8] presents approximation algorithms to find teams with the highest edge density. Proximity search is also studied in Euclidean space [1, 11], such as finding the smallest circle enclosing k points. Since the diameter of such a circle is not equal to the maximum pairwise distance between the k points, even with mapping methods such as ISOMAP [25], the techniques for the k-enclosing circle problem can not be directly applied here. The points here also do not contain label information.

Motif Finding in Graphs. Label-based graph proximity search is also related to the graph motif problem, introduced by [17] in the bioinformatics field. [7] further investigates three variants of the initial problem. A key difference is that diameter is not used to rank the discovered motifs.

Ranked Keyword Search in Graphs. Ranked keyword search in graphs [3, 12, 15] has focused on answers structured as rooted-trees. An answer is ranked mainly by the aggregate distances from the leaves to the root [12]. The distance among leaves is not considered. gDensity ranks an answer by its entire diameter. Finding subgraphs instead of trees is also studied in [16, 19]. Finding r-cliques that cover all the keywords is proposed in [16], which only finds answers with 2-approximation. [19] finds r-radius Steiner graphs that cover all the keywords.. [10] uses personalized PageRank vectors to find answers in the vicinity of vertices matching the query keywords in entity-relation graphs. [13] proposes XKeyword for efficient keyword proximity search in large XML graph databases. gDensity differs since it does not require a schema on the graphs. gDensity focuses on the diameter of a vertex set. and ensures optimality.

Top-k Query Processing. Top-k query processing is also studied for RDBMS [4, 10, 23] and middleware [5, 14, 21]. Supporting top-k queries in SQL is proposed in [4]. In middleware, top-k query is abstracted as getting objects with the top-k aggregate ranks from multiple data sources. Our work is different since gDensity answers top-k queries on a single source with graph data. Existing techniques for RDBMS and middleware are no longer applicable.

10. CONCLUSIONS

In this paper, we study the label-based graph proximity search problem, which finds the top-k query-covering vertex sets with the smallest diameters. The proposed gDensity framework introduces likelihood ranking of seed vertices to speed up the search. Fast pruning and early stopping are enabled by nearest label pruning and progressive search. Density indexing is proposed for fast likelihood estimation. Partial indexing is further proposed to reduce indexing cost. Empirical studies show the efficiency and scalability of gDensity. Our future works include: (1) extending gDensity to weighted and directed graphs; and (3) extending gDensity to handle other objective functions besides diameter.

11. ACKNOWLEDGMENTS

This research was sponsored in part by the U.S. National Science Foundation under grant IIS-0905084 and the Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053 (NS-CTA). The views and conclusions in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

12. REFERENCES

[1] A. Aggarwal, H. Imai, N. Katoh, and S. Suri. Finding k points with minimum diameter and related problems. *J. Algorithms*, 12(1):38–56, 1991.

[2] A. Baykasoglu, T. Dereli, and S. Das. Project team selection using fuzzy optimization approach. *Cybern. Syst.*, 38(2):155–185, 2007.

[3] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*, pages 431–440, 2002.

[4] M. Carey and D. Kossmann. On saying "enough already!" in SQL. In *SIGMOD*, pages 219–230, 1997.

[5] K. Chang and S. Hwang. Minimal probing: supporting expensive predicates for top-k queries. In *SIGMOD*, pages 346–357, 2002.

[6] S. Chib and E. Greenberg. Understanding the Metropolis-Hastings algorithm. *The American Statistician*, 49(4):327–335, Nov. 1995.

[7] R. Dondi, G. Fertin, and S. Vialette. Finding approximate and constrained motifs in graphs. In *CPM*, pages 388–401, 2011.

[8] A. Gajewar and A. D. Sarma. Multi-skill collaborative teams based on densest subgraphs. In *SDM*, pages 165–176, 2012.

[9] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. Walking in facebook: a case study of unbiased sampling of OSNs. In *INFOCOM*, pages 2498–2506, Piscataway, NJ, USA, 2010. IEEE Press.

[10] M. Gupta, A. Pathak, and S. Chakrabarti. Fast algorithms for top-k personalized PageRank queries. In *WWW*, pages 1225–1226, 2008.

[11] S. Har-Peled and S. Mazumdar. Fast algorithms for computing the smallest k-enclosing circle. *Algorithmica*, 41(3):147–157, 2005.

[12] H. He, H. Wang, J. Yang, and P. Yu. BLINKS: ranked keyword searches on graphs. In *SIGMOD*, pages 305–316, 2007.

[13] V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword proximity search on XML graphs. In *ICDE*, pages 367–378, 2003.

[14] I. Ilyas, G. Beskales, and M. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4):1–58, 2008.

[15] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, pages 505–516, 2005.

[16] M. Kargar and A. An. Keyword search in graphs: Finding r-cliques. *PVLDB*, 4(10):681–692, 2011.

[17] V. Lacroix, C. G. Fernandes, and M.-F. Sagot. Motif search in graphs: Application to metabolic networks. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 3(4):360–368, Oct. 2006.

[18] T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *KDD*, pages 467–476, 2009.

[19] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *SIGMOD*, pages 903–914, 2008.

[20] W. Li. Random texts exhibit Zipf's-law-like word frequency distribution. *IEEE Transactions on Information Theory*, 38(6):1842–, 1992.

[21] A. Marian, N. Bruno, and L. Gravano. Evaluating top-k queries over web-accessible databases. *ACM Trans. Database Syst.*, 29(2):319–362, 2004.

[22] K. Mehlhorn and S. Näher. *LEDA: a platform for combinatorial and geometric computing*. Cambridge University Press, 1999.

[23] L. Qin, J. X. Yu, L. Chang, and Y. Tao. Querying communities in relational databases. In *ICDE*, pages 724–735, 2009.

[24] P. Sarkar, A. W. Moore, and A. Prakash. Fast incremental proximity search in large graphs. In *ICML*, pages 896–903, 2008.

[25] J. Tenenbaum, V. Silva, and J. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, (5500):2319–2323.

[26] H. Wi, S. Oh, J. Mun, and M. Jung. A team formation model based on knowledge and collaboration. *Expert Syst. Appl.*, 36(5):9121–9134, 2009.

[27] A. Zzkarian and A. Kusiak. Forming teams: an analytical approach. *IIE Transactions*, 31(1):85–97, 1999.