SkyGraph: Retrieving Regions of Interest using Skyline Subgraph Queries

Shiladitya Pande
Dept. of CSE, IIT Madras
Chennai, India.
spande@cse.iitm.ac.in

Sayan Ranu
Dept. of CSE, IIT Delhi
New Delhi, India.
sayanranu@cse.iitd.ac.in

Arnab Bhattacharya
Dept. of CSE, IIT Kanpur
Kanpur, India.
arnabb@cse.iitk.ac.in

ABSTRACT

Several services today are annotated with *points of interest (PoIs)* such as "coffee shop", "park", etc. A *region of interest (RoI)* is a neighborhood that contains PoIs relevant to the user. In this paper, we study the scenario where a user wants to identify the best RoI in a city. The user expresses relevance through a set of keywords denoting PoIs. Ideally, the RoI should be small enough in size such that the user can conveniently explore the PoIs. On the other hand, it should be as relevant as possible. How does one balance the importance of size versus relevance? To a user exploring the RoI on foot, size is more critical. However, for a user equipped with a vehicle, relevance is a more important factor. In this paper, we solve this dilemma through *skyline subgraph queries* on keyword-embedded road networks. Skyline subgraphs subsume the choice of optimization function for an RoI since the optimal RoI for any rational user is *necessarily* a part of the skyline set. Our analysis reveals that the problem of computing the skyline set is NP-hard. We overcome the computational bottleneck by proposing a polynomial-time approximation algorithm called *SkyGraph*. To further expedite the running time, we develop an index structure, *Partner Index*, that drastically prunes the search space and provides up to 3 orders of magnitude speed-up on real road networks over the baseline approach. The datasets and executables are available at http://www.cse.iitd.ac.in/~sayan/software.html.

1. INTRODUCTION

In recent years, the Internet has undergone a dramatic change from being used mostly from desktop-based devices to being predominantly mobile-based such as smartphones and tablets. Owing to this transformation, large volumes of *geo-textual objects* are available on the Web that represent *points of interest (PoIs)* such as restaurants, cafes, malls, etc. [4, 24] A geo-textual object contains the geo-location of the PoI, represented using latitude and longitude, and a textual description with tags. The availability of such rich geo-textual data can help identify *regions of interest (RoIs)* in a city depending on a user's interests. An RoI is a neighborhood, ideally small, that contains the PoIs a user is interested in.

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org.
Proceedings of the VLDB Endowment, Vol. 10, No. 11
Copyright 2017 VLDB Endowment 2150-8097/17/08.

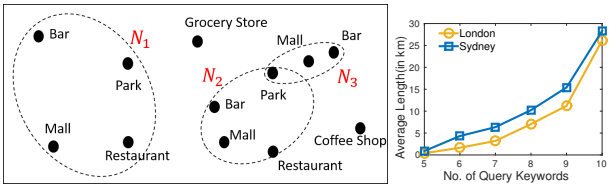


Figure 1: **(a) A motivating scenario for skyline RoI neighborhoods. (b) The growth rate of neighborhood size against the number of query keywords.**

To illustrate the idea of RoI, consider Fig. 1a that shows a hypothetical map with some PoIs. Suppose a visitor wants to explore this city and wants to visit a neighborhood that contains a "mall", "restaurant", "bar" and "park". Ideally, the neighborhood should contain all of the query keywords and be as small as possible. The extent of a neighborhood can be quantified using a number of metrics such as diameter, minimum spanning tree over the road-network that spans the entire neighborhood, etc. Three neighborhoods are shown in Fig. 1a. Here, N_1 and N_2 contain all the query keywords. Since N_2 is smaller than N_1 in size, N_2 is clearly preferable over N_1. Notice that the third neighborhood N_3 is significantly smaller than N_2. However, it does not contain the query keyword "restaurant". Therefore, an important question arises: *Would the user be willing to compromise on the absence of a restaurant in exchange for a smaller neighborhood?* Only the user can answer this question. To illustrate, if one plans to explore the neighborhood through walking, size of the neighborhood is a critical factor. On the other hand, for those with vehicles, size may not be as important a factor. Hence, it is desirable to present both N_2 and N_3 as RoIs. However, given N_2, N_1 can be safely omitted.

We capture these intricacies of identifying RoIs by performing *skyline subgraph queries* on keyword-embedded road networks. Informally, a skyline is an object that is not worse (or equal) in all the attributes of interest than another object. Thus, while N_2 is not worse in terms of number of keywords, N_3 is not worse in terms of size. Both of them are, thus, *skyline neighborhoods*. The neighborhood N_1, on the other hand, is worse than N_2 in size while being equal in number of keywords.

In our problem, the user provides a set of query keywords, and our goal is to compute all *skyline neighborhoods* in a keyword-embedded road network. Each node in a road network represents a location and edges correspond to roads connecting two locations. Additionally, a node contains a set of *keywords* depicting the services available in that location. A neighborhood, therefore, corresponds to a *subgraph* of the road network. A neighborhood subgraph is characterized by two features: (a) the *coverage* depicting

the number of query keywords covered by nodes in the subgraph, and (b) the *size* of the subgraph. (We characterize the size in more precise terms later.) A smaller size and a larger coverage is better in the context of our problem. A neighborhood subgraph is a *skyline RoI* if it is not dominated by any other neighborhood.

The interplay between the coverage of keywords and the size of the neighborhood forms the crux of defining what is of interest to the user. To understand this better, we study the relationship between keyword coverage and neighborhood size in two prominent cities: London and Sydney. Specifically, we choose 1000 random query sets of x keywords and plot the average size of the smallest neighborhood that covers all x keywords. The size of a neighborhood subgraph is defined by the length of the *minimum spanning tree (MST)* on that subgraph. Further details of the datasets are provided in Sec. 6. Fig. 1b presents the results. As can be seen, the size of the neighborhood grows exponentially with the number of keywords. Whether this exponential increase in size is worth at the expense of one additional keyword forms a key decision point for any user. At the same time, the optimal trade-off between size and coverage is an individual choice made by each user, and an effective querying system should not assume a preference function on the user's behalf. Skyline subgraphs fit the bill perfectly by allowing the user to retain this power by not discarding any neighborhood that has a chance of being the optimal choice.

The problem of querying RoIs based on user provided keywords has been studied previously. However, all of the existing techniques have at least one of the following weaknesses:

- *Assumption of Euclidean space:* It has often been assumed that the PoIs are distributed in an Euclidean space [11, 25, 26]. This assumption does not model the reality in road networks where the movements are restricted over roads.
- *Assumption of a preference function:* Several techniques quantify the quality of a RoI through a preference function defined over its coverage and size [3, 4, 20, 28]. As discussed earlier, all users may not optimize the same preference function and this choice must be left to the user itself. Computing skyline subgraphs overcomes this weakness. More specifically, the optimal answer to *any* monotonic preference function must lie within the skyline set. Thus, skyline neighborhoods are more generic and powerful since they can be used for any preference function.

Our work is also significantly different from those that aim to identify only the optimal PoIs (single nodes) for a set of user provided keywords [6,13,14,16–19,21,23,27,29], and subsumes them.

The core contributions of our work are as follows:

- We introduce the problem of querying RoIs through skyline subgraphs on keyword-embedded road networks (Sec. 2).
- We prove that the problem of finding skyline neighborhoods is NP-hard (Sec. 3). To enable fast query response times, we develop a technique called *SkyGraph* (Sec. 4). SkyGraph derives its efficiency by developing an approximation algorithm with quality guarantees. The efficiency is further enhanced through the usage of an index structure called the *Partner Index* (Sec. 5).
- We establish the efficiency and efficacy of SkyGraph through a comprehensive empirical evaluation on 4 real keyword-embedded road networks (Sec. 6). Our results demonstrate that SkyGraph is up to 3 orders of magnitude faster than baseline techniques and achieves quality that is close to the optimal.

2. PROBLEM FORMULATION

DEFINITION 1 (ROAD NETWORK). *A road network graph $G = (V, E, \tau, \lambda)$ consists of a set of nodes V that represents sites, a set of undirected edges $E \subseteq V \times V$ that represents roads between a pair of sites, a distance function $\tau : E \rightarrow \mathcal{R}$ that represents the distance along the road, and a spatial mapping $\lambda : V \rightarrow \mathcal{R}^2$ that represents the spatial locations of the sites.*

Each site v represents either a road intersection, a dead-end, or a PoI. If it is a PoI, it is associated with a text description, i.e., a set of keywords denoted by $v.\psi$. The keywords denote the type of entities present in that site such as "restaurant", "cafe", "mall", etc. Each edge $e_{ij} = (v_i, v_j)$ represents a road segment connecting the nodes v_i and v_j.

DEFINITION 2 (NEIGHBORHOOD). *A neighborhood N in a road network G is a* connected subgraph *of G. We denote the vertices and edges in N as $N.V$ and $N.E$ respectively. The keywords contained in N are*

$$N.\psi = \cup_{\forall v \in N.V} v.\psi \tag{1}$$

DEFINITION 3 (NEIGHBORHOOD SIZE). *The size of a neighborhood subgraph N is the* length *of its* minimum spanning tree *(MST). Formally, if T is the MST on N, where $T.E \subseteq N.E$ is the set of edges contained within T, then*

$$size(N) = \sum_{\forall e \in T.E} \tau_e \tag{2}$$

In an RoI query, a user U provides a set of keywords $Q = \{q_1, \cdots, q_n\}$ as input and the "best" neighborhood N^* in the road network G is output as the intended RoI. Mathematically,

$$N^* = \arg\max_{N \subseteq G} \{score_U(N, Q)\} \tag{3}$$

where $score_U(N, Q) = f_U\langle cov(N, Q), size(N)\rangle$ is a function that quantifies the quality of a neighborhood N with respect to query Q from user U's perspective. The function f_U is defined over two neighborhood aspects: the coverage of N with respect to Q, and the size of N. While Eq. (2) defines the size of N (note that it is independent of Q), the *coverage* is defined as follows.

DEFINITION 4 (NEIGHBORHOOD COVERAGE). *The coverage of a neighborhood N with respect to a query Q is the* number *of query keywords contained in N:*

$$cov(N, Q) = |N.\psi \cap Q| \tag{4}$$

The scoring function f_U changes from user to user. It is neither practical to ask every user to provide the exact function f_U, nor can we assume the function f_U on the user's behalf. In other words, we need to identify the best neighborhood *without* the explicit knowledge of f_U.

It can be assumed, however, that any rational user will prefer a neighborhood that has higher coverage and a smaller size. Thus, the *quality* (or score) of a neighborhood monotonically increases with coverage and monotonically decreases with size. This observation can be formalized as follows.

PROPERTY 1 (SCORING FUNCTION). *If N_1 and N_2 are two neighborhoods such that*
$cov(N_1, Q) = cov(N_2, Q)$ and $size(N_1) < size(N_2)$ or,
$cov(N_1, Q) > cov(N_2, Q)$ and $size(N_1) = size(N_2)$,
then $score_U(N_1, Q) > score_U(N_2, Q)$.

The RoI querying problem is to identify the best neighborhood under *any* possible scoring function f_U that follows Property 1.

PROBLEM 1 (ROI QUERYING). *Given a set of keywords Q, find the set of RoI neighborhoods that can potentially be the optimal neighborhood for some scoring function $score_U$ that follows Property 1. Mathematically,*

$$RoIs = \{N \subseteq G \mid \exists score_U \text{ s.t. } \forall N' \subseteq G, score_U(N, Q) \geq score_U(N', Q)\} \quad (5)$$

Under Property 1, Problem 1 reduces to identifying the *skyline subgraphs* of G.

DEFINITION 5 (SKYLINE SUBGRAPH). *A neighborhood N is a skyline subgraph of G with respect to query Q if and only if it is not dominated by any other neighborhood. Formally, N is a* skyline neighborhood *if and only if $\nexists N' \subseteq G$ such that*
(i) $cov(N', Q) \geq cov(N, Q)$ and $size(N') < size(N)$, or
(ii) $cov(N', Q) > cov(N, Q)$ and $size(N') \leq size(N)$.

We next establish the universality of skyline subgraphs.

THEOREM 1 (UNIVERSALITY). *For any scoring function that follows Property 1, the neighborhood that has the highest score must be a skyline subgraph.*

PROOF. (BY CONTRADICTION.) Assume that there is a neighborhood N' that is *not* a skyline but there exists a function $score_U$ for which N' has the highest score. Since N' is not a skyline subgraph, there exists a skyline subgraph N that dominates it. Therefore, at least one of the following conditions is true:
(i) $cov(N) > cov(N')$ and $size(N) \leq size(N')$, or
(ii) $cov(N) \geq cov(N')$ and $size(N) < size(N')$.
In either case, using Property 1, $score_U(N, Q) > score_U(N', Q)$, which is a contradiction. □

Using Thm. 1, Problem 1 can be equivalently defined as follows.

PROBLEM 2 (SKYLINE NEIGHBORHOOD QUERY). *Given a set of keywords Q, identify all skyline neighborhood subgraphs.*

The following conclusions follow from Thm. 1.

COROLLARY 1. *If no two subgraphs are of the exact same size, the number of skyline subgraphs for a set of query keywords Q is at most $|Q|$.*

While the assumption of uniqueness of every subgraph size is theoretically not be true, in practice the chance of having two subgraphs of exactly the same size is extremely low. Thus, the size of the answer set for a query Q is $\approx |Q|$. This has an implication on the efficacy of the proposed formulation. In an RoI query, the number of keywords is typically small and within 10. Consequently, our answer set is small enough to be explored and diagnosed manually.

COROLLARY 2. *The number of skyline neighborhoods increases with increase in the number of dimensions if there exists at least one dimension that always contains unique values (such as neighborhood size in our problem).*

Corollary 2 shows that adding more dimensions will lead to increase in the size of the answer set size and, therefore, may not be justified. Hence, we limit ourselves to only 2 dimensions, size and coverage, since the utility of these two dimensions have been well established in large volumes of work [2–9, 12–14, 16–23, 25–29].

Algorithm 1 Baseline algorithm(G, Q)

```
1: RoIs ← ∅;
2: for all integers k in range [1, |Q|] do
3:     N ← minimal sized subgraphs of coverage k
4:     RoIs ← RoIs ∪ N
5: return RoIs
```

3. COMPLEXITY ANALYSIS

Alg. 1 presents the pseudocode of the baseline algorithm to solve our problem. Given the set of query keywords Q, we iterate over each possible coverage in the range 1 to $|Q|$ (lines 2-5) and identify the minimal sized subgraphs for the corresponding coverage (line 3). Since they are all guaranteed to be skylines (using Cor. 1), we add them to the answer set.

We refer to these subgraphs as *k-skyline* subgraphs. Next we show that identifying the k-skyline subgraphs reduces to the problem of *k-minimum spanning tree (k-MST)*, which is NP-hard [10].

DEFINITION 6 (k-MST). *Given a graph $G = (V, E)$ with non-negative edge weights, the k-MST problem is to identify the tree of minimum weight that spans k nodes.*

THEOREM 2. *Identifying k-skyline subgraphs is NP-hard.*

PROOF. Given an arbitrary instance of the k-MST problem, we reduce it to an instance of the k-skyline subgraph problem through the following procedure. Let the given graph G for k-MST problem be the road network for our problem. Each node in the road network is annotated with a unique keyword. In addition, we assign spatial coordinates to each node such that the length of each edge is same as its weight in G. Let this road network constructed from G be called G'. Finally, let the query Q be a set of k keywords.

In G', the k-skyline problem is to identify the smallest subgraph of coverage k. Since each node in G' contains an unique keyword and the size of a subgraph is the total weight of its MST, if the k-skyline subgraph problem is solved, the MST of the k-skyline subgraph is same as the k-MST on G. □

4. SKYGRAPH

Sec. 3 establishes that identifying skyline subgraphs is NP-hard. Particularly, the bottleneck lies in line 3 of Alg. 1, where we need to find the k-skyline subgraph, which is the smallest subgraph of a specific coverage value k. Although the k-skyline problem has connections to the k-MST problem, approximation algorithms developed for the k-MST problem cannot be applied directly for identifying k-skyline subgraphs. Fig. 2 illustrates the reason. In the shown network, the 3-MST is the subgraph spanning nodes $\{1, 3, 4\}$. However, if one wants to span the three keywords of $\{a, b, c\}$, the smallest subgraph, i.e., 3-skyline, is the subgraph spanning nodes $\{1, 2\}$. The difference stems from the fact that although both nodes 4 and 1 have the same value initially (covers keyword a), once any of these nodes is included in the subgraph, the other node is no more useful since it does not contribute any new keyword to the coverage. Consequently, the value of a node is dependent on the other nodes that are part of the subgraph. In k-MST however, inclusion of every node contributes a value of 1 towards the final goal of spanning k nodes.

With this intuition in mind, we next modify and adapt an existing algorithm for k-MST [1] for our k-skyline problem. Our algorithm has two primary components: *Merge-cluster* and *Connect-cluster*.

4.1 Merge-Cluster

First, we describe the merge-cluster phase of the algorithm and then we analyze its properties. Alg. 2 presents the pseudocode. The input are number of keywords k, query Q, and the road network G.

The algorithm proceeds in a bottom-up manner similar to agglomerative clustering. More specifically, every node in the network is initially considered a cluster (line 1). A cluster is essentially a subgraph of the road network G.

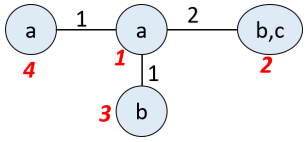


Figure 2: **Illustration of the difference between k-MST and k-skyline subgraph problem. The node labels indicate the keywords contained within it. The node IDs are indicated by the number next to each node in bold, red-italics font. The number next to each edge denotes its length. We follow the same convention for all subsequent graph figures.**

Next, it *joins* the two clusters, C_i, C_j, that minimizes the ratio

$$r = \frac{d(C_i, C_j)}{\min\{\text{unique}(C_i, C_j), \text{unique}(C_j, C_i)\}} \quad (6)$$

$$\text{where, } d(C_i, C_j) = \min_{\forall v_i \in C_i, \forall v_j \in C_j} \{\text{shortestPath}(v_i, v_j)\} \quad (7)$$

$$\text{unique}(C_i, C_j) = |\{C_i - C_j\} \cap Q| \quad (8)$$

Essentially, we want to merge clusters that are close to each other and after merging them the total coverage increases by a large margin. Thus, the numerator of r (Eq: 6) is minimized if the distance between two clusters is low. On the other hand, the denominator is maximized if C_i and C_j have high coverage and low overlap among the keywords covered (the *min* component ensures low overlap).

Joining clusters C_i and C_j creates a new cluster C where $C.V = C_i.V \cup C_j.V$ and $C.E = \{e = (v_i, v_j) \in G.E | v_i, v_j \in C.V\}$. C_i and C_j are then discarded and C is added to the set of clusters (line 4). This process continues iteratively till a cluster of coverage at least $k/4$ is created. The maximum possible coverage of a cluster created through MERGE-CLUSTER is less than $k/2$.

THEOREM 3. *The size of the cluster produced by* MERGE-CLUSTER *is at most* $(2^{k/4} - 1)4 \log k$ *times the size of the optimal k-skyline subgraph* OPT.

PROOF. Theorem 3 follows directly from the following 2 lemmas, whose proofs are in the Appendix.

LEMMA 1. *Let R be the largest ratio that has been observed in* MERGE-CLUSTER *(Eq. 6 in Alg. 2) till it terminates. Then any cluster C of coverage p has size less than $(2^{k/4} - 1)Rp$.*

PROOF. Refer to Appendix A.

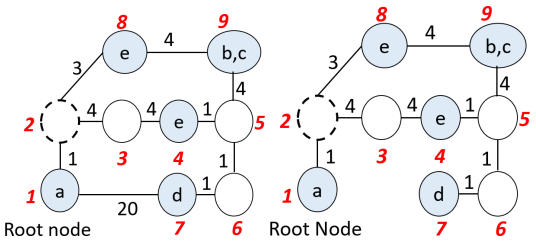
LEMMA 2. MERGE-CLUSTER *never uses a ratio (Eq. 6) larger than $(8L \log_2 k)/k$ where L is the size of the optimal k-skyline subgraph* OPT.

PROOF. Refer to Appendix B.

An obvious question that arises is: Why stop MERGE-CLUSTER at $k/4$? We take this decision purely from a theoretical standpoint: the quality guarantee provided by Thm. 3 does not hold if we let MERGE-CLUSTER run till k (Lemma 2 is violated). Further details are provided in Appendix C.

Algorithm 2 MERGE-CLUSTER

Input: Query Q containing k keywords, Graph G
Output: Subgraph C of coverage at least $k/4$
1: Initially, each node is considered as a cluster
2: **while** all clusters have coverage less than $k/4$ **do**
3: $C \leftarrow$ Join clusters C_i and C_j that minimize Eq. (6)
4: Remove C_i and C_j and add C
5: **return** C



(a) Road Network (b) Subgraph extracted by CONNECT-CLUSTER

Figure 3: **(a) A sample road network where the node with the dashed boundary is a fork node (node id 2). (b) The subgraph extracted by CONNECT-CLUSTER on the query keyword set $\{a, b, c, d\}$. Here $d_k = 12$ corresponding to node ID 7 containing the keyword 'd'.**

4.2 Connect-Cluster

To attain the target coverage of k, we devise the next algorithm called CONNECT-CLUSTER. Alg. 3 provides the pseudocode.

In addition to k, query keywords Q and the road network G, CONNECT-CLUSTER also takes a node $v \in G$ as input. Considering v as the root node, CONNECT-CLUSTER finds the shortest path to each of the query keywords from v. Let the distance to the k^{th} farthest query keyword from v be d_k (line 1). CONNECT-CLUSTER extracts the d_k-hop subgraph $S \subseteq G$ around v that contains all nodes and edges within distance d_k from v (line 2). An example of the extracted subgraph is shown in Fig. 3.

On S, MERGE-CLUSTER is performed iteratively (lines 5-9). Each iteration produces a cluster C' of some coverage k', $k' < k$ (line 6). C' is stored and nodes contained in it are removed from S (as well as edges between these nodes) so that they are not considered in any of the subsequent iterations (line 7). Furthermore, the produced cluster is "connected" to the cluster combination produced so far (line 9). The iterations of MERGE-CLUSTER continue till the coverage of this combined cluster is at least k.

To analyze the quality of the subgraph produced by CONNECT-CLUSTER, we first make the assumption that the root node v, which is an input to the algorithm, is also part of the optimal k-skyline subgraph OPT. Under this assumption, the following lemma holds.

LEMMA 3. *$d_k \leq L$, where L is the size of* OPT.

PROOF. By definition, d_k is the distance of the k^{th} nearest query keyword from the v. Thus, d_k is the minimum size that has to be added to the size of OPT. □

THEOREM 4. *The size of the tree $\mathcal{C}$ produced by* CONNECT-CLUSTER *is at most $O(2^k (\log k)^2)$ times the size of the optimal k-skyline subgraph* OPT.

Algorithm 3 CONNECT-CLUSTER

Input: Root node v, Query Q of size k, Graph G
Output: Cluster C
1: $d_k \leftarrow$ distance to the k^{th} farthest query keyword from v
2: $\mathcal{S} \leftarrow$ all nodes and edges within radius d_k from v
3: $\mathcal{C} \leftarrow \varnothing$ //set of clusters found by MERGE-CLUSTER
4: $Q' \leftarrow Q$
5: **while** $COV(\mathcal{C}, Q) < k$ **do**
6: $\mathcal{C}' \leftarrow$ MERGE-CLUSTER(k, S, Q')
7: Remove subgraph $\mathcal{C}'$ from $\mathcal{S}$.
8: Remove keywords covered in $\mathcal{C}'$ from Q'
9: $\mathcal{C} \leftarrow$ Join $\mathcal{C}$ with $\mathcal{C}'$ through the shortest path
10: **return** $\mathcal{C}$

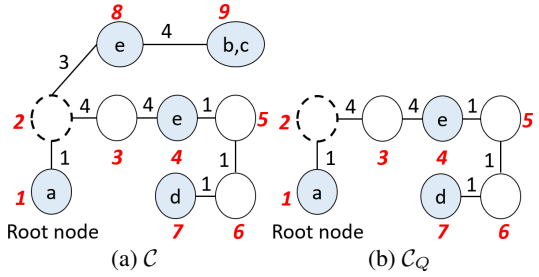


Figure 4: **(a) The minimum spanning tree covering all keyword nodes in Fig. 3b. (b) 3-skyline subgraph corresponding to $Q = \{a, d, e\}$ on the subgraph rooted at Node 1.**

PROOF. First, we analyze the number of MERGE-CLUSTER iterations in a single instance of CONNECT-CLUSTER. If k' is the number of query keywords covered by the invocations of MERGE-CLUSTER so far, then in the next invocation, the number of query keywords covered by MERGE-CLUSTER $\geq (k - k')/4$. So, the maximum number of iterations of MERGE-CLUSTER algorithm is at most $O(\log_{\frac{4}{3}} k)$.

From Theorem 3, the size of each cluster returned by MERGE-CLUSTER is $O(2^k \log k)$ times the size of OPT. Thus, the total size of all clusters returned by MERGE-CLUSTER is $O(2^k \log k) \times O(\log k) = O(2^k (\log k)^2)$ times the size of OPT. Denoting L as the size of OPT, the size of the tree produced by CONNECT-CLUSTER is therefore, $cLk(\log k)^2 + H$, where H is the total size of paths used to connect the clusters produced by MERGE-CLUSTER, and c is some constant.

Let C_i and C_{i+1} be the clusters produced by MERGE-CLUSTER in two consecutive iterations. From our construction of S (line 1-2 in Alg. 3), its diameter is at most $2d_k$. Thus, the size of the path connecting C_i with C_{i+1} is at most $2d_k$. Extending this further, the size of H is at most $2d_k \log k$. Combining all these results, the total size of the tree $\mathcal{C}$ produced by CONNECT-CLUSTER is

$$\begin{aligned} Size(\mathcal{C}) &\leq cL2^k(\log k)^2 + 2d_k \log k \\ &\leq cL2^k(\log k)^2 + 2L \log k \ (\because d_k \leq L \text{ from Lemma 3}) \\ &= O(2^k(\log k)^2) \text{ times the size of OPT.} \end{aligned}$$ □

4.3 SkyGraph Algorithm

The theoretical guarantees provided by CONNECT-CLUSTER assumes that the root node is part of the optimal k-skyline subgraph. This assumption can be easily removed by the following algorithm.

1. $\mathbb{N} = \{$nodes containing at least one query keyword$\}$
2. Iterate over each node $v \in \mathbb{N}$ and compute the tree returned by CONNECT-CLUSTER on v
3. Return the smallest tree returned by CONNECT-CLUSTER across all nodes in $\mathbb{N}$

It is easy to see that the quality guarantee of Theorem 4 holds for this algorithm since we iterate over all keyword containing nodes. Here on, we use the term k-skyline subgraph to refer to the smallest tree returned by SkyGraph.

4.3.1 Time Complexity

With the quality guarantees in place, we next analyze the time complexity of the SkyGraph algorithm. We denote the number of nodes in the network with n.

Merge-Cluster: Initially, all nodes are clusters on their own. Before the iterations start, for each cluster, we store the cluster that provides the minimum ratio if merged with. This pre-processing step requires $O(n^2)$ time. In each iteration, finding the global minimum ratio requires $O(n)$ time. Once two clusters are merged, the minimum ratio for the newly formed cluster is computed in another $O(n)$ time by comparing across all existing clusters. Since there are $O(n)$ iterations, the total time taken by a single call to MERGE-CLUSTER is $O(n^2)$.

Connect-Cluster: From our proof in Theorem 4, we know there will be $O(\log k)$ invocation of MERGE-CLUSTER (line 5). Since the computational cost of MERGE-CLUSTER supersedes the cost of all other operation in CONNECT-CLUSTER, the time complexity is bounded by $O(\log k(n^2))$.

Overall: Since CONNECT-CLUSTER is run on each node ($O(n)$) containing a query keyword, the overall time complexity of SkyGraph is $O(n^3 \log k) \approx O(n^3)$ as $k \ll n$.

4.3.2 Connection to k-MST Approximation [1]

The SkyGraph algorithm has a similar flow to the algorithm proposed in [1]. However, since the contribution to coverage by a node is dependent on the other nodes already added to the subgraph, several changes are required to [1]. First, the ratio being minimized in line 3 of Alg. 2 has been modified to capture the dependence among nodes. This leads to a different proof for Theorem 3. Furthermore, CONNECT-CLUSTER algorithm is different (Alg. 3, line 8) resulting in a different bound for Theorem 4.

5. INDEX STRUCTURE

The most expensive component of the SkyGraph algorithm is the need to run CONNECT-CLUSTER on each node of the network that contains a query keyword. Furthermore, if the subgraph $\mathcal{S}$ on which CONNECT-CLUSTER is run is large (line 2 in Alg. 3), then the subsequent calls to MERGE-CLUSTER would also be expensive. Notice that the larger the radius of this subgraph $\mathcal{S}$, the less are the chances of S containing a "small" k-skyline subgraph within it. This result follows directly from Lemma 4 since the radius d_k of $\mathcal{S}$ is a lower bound to the size of the k-skyline subgraph returned by CONNECT-CLUSTER. The above two observations reveal that most of the execution time is likely to be spent on those CONNECT-CLUSTER calls that are unlikely to be part of the answer set. *Can we devise a filter of low computation cost that prunes out most of those* CONNECT-CLUSTER *calls that are guaranteed to not provide the k-skyline subgraph?* We answer this question by computing fast lower bounds on the size of the tree returned by CONNECT-CLUSTER. The computation of the lower bound is facilitated by an index structure called *Partner Index*.

5.1 Partner Index

Let u be a root node and $\mathcal{S}$ the subgraph around u that contains all nodes and edges within some radius r. Furthermore, let $\mathcal{C}$ be the minimum spanning tree containing all keyword nodes in $\mathcal{S}$. For each keyword in $\mathcal{C}$, we now introduce the concept of a *partner node* and *partner distance*.

DEFINITION 7 (PARTNER NODE AND PARTNER DISTANCE). *The partner node of a keyword w is the closest node in $\mathcal{C}$ (in terms*

Table 1: **Partner nodes and distances for keywords in $\mathcal{C}$ (Fig. 4a).**

Keyword	Partner Nodes for $\mathcal{C}$	Partner Node in $\mathcal{C}_Q$
a	(2, fork, 1)	(2, fork, 1)
b	(8, e, 4)	NA
c	(8, e, 4)	NA
d	(4, e, 3)	(4, e, 3)
e	(7, d, 3)	(7, d, 3)

of edge weight distance) that either contains a keyword different from w or is a fork node, i.e., a node with degree ≥ 3, from any node containing w. The distance to the partner node is termed partner distance.

In essence, the partner distance is the minimum distance that needs to be covered to connect a keyword to the minimum spanning tree. Since partner node and partner distance come in pairs, we represent them as a tuple $p =$ (Node ID, keyword/Fork, Partner distance). Furthermore, $p.ID$, $p.keyword$, and $p.dist$ denote the node ID, keyword, and partner distance associated with tuple p.

EXAMPLE 1. *Let us revisit the network in Fig. 3a. Let radius $r = 12$ and node 1 with keyword 'a' be the root node. Consequently, the 12-hop subgraph S is identical to the network in Fig. 3b. The smallest tree $\mathcal{C}$ spanning all keyword nodes in S is shown in Fig. 4a. Column 2 in Table 1 presents the partner node and partner distance for each keyword in $\mathcal{S}$ (or $\mathcal{C}$). Note that we store a partner node for each keyword in $\mathcal{C}$ and not each keyword node. Thus, there is only one entry for keyword 'e' even though two nodes contain 'e'. Note that both node 7 (keyword node) and node 2 (fork node) are at a distance of 3 from a node containing 'e'. We break ties arbitrarily.*

Now, we define a function called *partnerSize($\mathcal{C}$)*, which is a lower bound on the size of $\mathcal{C}$.

DEFINITION 8 (PARTNERSIZE). *Let $P_{\mathcal{C}}$ be the set of (unique) tuples occurring as a partner node. The partnerSize is the combined distance to some keyword in $\mathcal{C}$.*

$$partnerSize(\mathcal{C}) = \sum_{\forall p \in P_{\mathcal{C}}} p.dist \tag{9}$$

EXAMPLE 2. *Let us revisit the partner nodes and distances of $\mathcal{C}$, shown in column 2 of Table 1. $P_{\mathcal{C}} = \{$(2,fork,1), (8,e,4), (4,e,3), (7,d,3)$\}$. Thus, $partnerSize(\mathcal{C}) = 11$. Note that $size(\mathcal{C}) = 19$.*

The fact that the partner size is less than the size of the minimum spanning tree is not a coincidence, but a property.

LEMMA 4. $partnerSize(\mathcal{C}) \leq size(\mathcal{C})$

PROOF. *Proof by contradiction.* Assume $partnerSize(\mathcal{C}) > size(\mathcal{C})$. Since $\mathcal{C}$ is the minimum spanning tree covering all keyword nodes, every keyword node must be connected by a path to another keyword node or a fork node. Since $partnerSize(\mathcal{C}) > size(\mathcal{C})$, there exists some keyword w in $\mathcal{C}$ with a partner distance greater than the length of the path that joins the corresponding keyword node to another keyword node or a fork node. However, from the definition of partner distance, this is a contradiction. □

While the above lemma provides us a lower bound, it is not useful since it considers all keywords in $\mathcal{C}$. In our problem, we need to consider only the query keywords provided by the user. *Suppose we have constructed the minimum spanning tree $\mathcal{C}$ covering all keyword nodes within radius r from root node u and the resultant partner nodes and distances. Can this information be used to obtain a lower bound on the size of the tree returned by* CONNECT-CLUSTER *on node u with some query set Q?* Our analysis reveals that a lower bound can be obtained if $Q \subseteq K$, where K is the set of all keywords in $\mathcal{C}$.

Before we derive the formal proof, we first discuss the intuition behind this bound. Consider Fig. 4b that shows the spanning tree C covering the query set $Q = \{a, d, e\}$. Q is a subset of keywords that occur in $\mathcal{C}$ of Fig. 4a. Column 3 in Table 1 shows the partner distance table for the keywords in Q. Notice that all of the partner distances for the keywords in Q are same as that for $\mathcal{C}$ shown in column 2. This result is not a coincidence. If Q is a subset of the keywords in $\mathcal{C}$, then the partner distance for any keyword in Q will be at least the partner distance of the same keyword in $\mathcal{C}$. This follows from the fact that any path that is feasible to connect a keyword node in Q to the minimum spanning tree, is also feasible in the spanning tree of $\mathcal{C}$. The formal proof for this is as follows.

LEMMA 5. $\sum_{\forall p \in P_{\mathcal{C}_Q}} p.dist_C \leq \sum_{\forall p \in P_{\mathcal{C}_Q}} p.dist_{C_Q}$.
$P_{\mathcal{C}_Q}$ is the set of tuples in the partner distance table for $\mathcal{C}_Q$, which is the minimum spanning tree covering all keywords in Q.

PROOF BY CONTRADICTION. Assume, there exists a query keyword $p.keyword$ in C_Q, such that $p.dist_{C_Q} < p.dist_C$.

Let node X contain the keyword $p.keyword$ in $\mathcal{C}_Q$. Since all keyword containing nodes in C_Q are also present in C, for $p.dist_{C_Q} < p.dist_C$, there must exist a shorter path that joins node X to some fork node or keyword node in C_Q than the path that joins X in C. The length of the path joining X to C_Q is stored as the partner distance for $p.keyword$. However, then this path would be used to join node X to C as well since, otherwise, C is not a minimum spanning tree. This is a contradiction. Hence,

$$\forall p \in P_{\mathcal{C}_Q}, p.dist_{\mathcal{C}} \leq p.dist_{\mathcal{C}_Q}$$
$$\text{or, } \sum_{\forall p \in P_{\mathcal{C}_Q}} p.dist_C \leq \sum_{\forall p \in P_{\mathcal{C}_Q}} p.dist_{C_Q}$$ □

THEOREM 5. $\sum_{\forall p \in P_{\mathcal{C}_Q}} p.dist_C \leq size(C_Q)$.

PROOF. Combining Lemmas 4 and 5, we get the proof. □

Theorem 5 gives us the platform to compute a lower bound on the tree returned by CONNECT-CLUSTER using the pre-computed partner distances. To apply Theorem 5 on any node of the network, we construct the *partner distance table* for every keyword node. This together constitutes the *partner index*.

5.1.1 Complexity Analysis

Time Complexity: The partner distance table is maintained as a *HashMap*. Consequently, computing the lower bound to a tree returned by CONNECT-CLUSTER requires $O(k)$ time where k is the number of keywords.

Storage Complexity: Let W be the set of all unique keywords in the road network. Since the partner distance table at each node could have at most W keywords, the storage complexity for a single partner distance table is $O(W)$. Since we store at most n partner distance tables with respect to each keyword containing node in the network, the total storage complexity is $O(nW)$.

5.2 Querying Algorithm using Partner Index

Empowered by the Partner Index, we complete the pipeline of the $SkyGraph$ algorithm by presenting the querying procedure.

The pseudocode is presented in Alg. 4. Similar to the baseline algorithm (Alg. 1), we compute the k-skyline subgraph for each value of k in the range $[1, |Q|]$ for a given set of query keywords Q (lines 2-3). Without Partner Index, the k-skyline subgraph is computed by executing CONNECT-CLUSTER on all nodes containing a query keyword and then returning the globally smallest tree.

We improve this approach in Alg. 5 by first computing the lower bound to CONNECT-CLUSTER on each of the nodes containing query keyword (lines 2-6). This information is stored as a tuple of the form $\langle u, lb(u) \rangle$, where u is a node containing a query keyword,

Algorithm 4 SkyGraph(G, Q, Partner Index $pIndex$)

```
1: RoIs ← ∅;
2: for all integers k in range [1, |Q|] do
3:     N ← getSkyline(k, G, Q, pIndex)
4:     RoIs ← RoIs ∪ N
5: return RoIs
```

Algorithm 5 getSkyline(k, G, Q, $pIndex$)

```
1: PQ ← empty min-heap
2: for all node u ∈ G.V do
3:     d_k ← distance to k^th farthest query keyword from u
4:     if d_k ≤ pIndex.r then
5:         lb(u) ← compute lower bound using pIndex
6:         PQ.insert(⟨u, max{lb(u), d_k}⟩)
7: θ ← ∞
8: T ← ∅ // stores smallest spanning tree found so far
9: while PQ.top().lb < θ and PQ is not empty do
10:    u ← PQ.pop().node
11:    C ← CONNECT-CLUSTER(u, k, G, Q)
12:    if size(C) < θ then
13:        T ← C
14:        θ ← size(C)
15: return T
```

and $lb(u)$ is the lower bound to the tree returned by CONNECT-CLUSTER on u as the root node (line 6). If Theorem 5 is not applicable on u due to violation of the constraint $d_k \leq r$, then $lb(u)$ is set to d_k based on Lemma 3 (lines 4-6).

Following the computation of lower bounds, all the tuples are inserted in a min heap, PQ. The most promising node with the smallest lower bound is first explored, i.e., this node is supplied as the root node to CONNECT-CLUSTER, which returns a tree spanning k query words (lines 10-11). The size of this tree is stored as a threshold θ, which is subsequently employed to prune the search space (lines 12-14). The next most promising node is then popped from the heap PQ and checked if its lower bound is smaller than θ (line 12). If this check passes, CONNECT-CLUSTER is executed on this node and θ is updated to the size of the smallest spanning tree found so far (lines 13-14). If the check fails, then we are guaranteed that none of the unexplored nodes can return a smaller spanning tree than what we have already found. Hence, the computation ends for the current value of k (first condition in line 9).

The process is then repeated with the remaining values of k.

6. EXPERIMENTS

In this section, we benchmark the proposed algorithms and establish the efficiency and efficacy of SkyGraph.

6.1 Experimental Setup

All experiments are implemented in Java and performed on a machine with Intel(R) Xeon(R) 2.5GHz CPU E5-2609 with 512 GB RAM running Ubuntu 14.04. All experiments are repeated 100 times for consistency and the average is reported.

Unless specifically mentioned, the number of query keywords, i.e., $|Q|$, is set to a random size between 4 and 10. Nevertheless, we have benchmarked the proposed system with query size of up to 10. The choice of the query keywords themselves is an important factor and we have experimented with various selection models, which will be discussed later in this section. Our default selection procedure is *frequency proportional sampling*. More specifically, if a keyword (or PoI) is found more frequently in the city, then its chances of being a query keyword is higher.

SkyGraph requires only one internal parameter, which is the radius parameter used to construct the Partner Index. We set the radius to 15Km, since for city-level queries, an RoI with a diameter of 30Km is likely to span almost the entire city. For all experiments in this section, we employ the usage of 2-hop indexing for fast shortest path computations [15].

Table 2: **Datasets used to benchmark SkyGraph.**

City	# nodes	# edges	# keyword nodes	# unique keywords	Bounding Box
London(L)	209,407	282,268	87,346	56,648	lat∈[51.342, 51.648] lon ∈[-0.448, 0.205]
Sydney(S)	236,041	317,266	23,103	14,063	lat∈[149.724, 151.880] lon ∈[-34.286, -33.250]
Dublin(D)	62,975	82,730	1351	170	lat∈[53.033, 53.777] lon ∈[-7.348, -5.192]
California(C)	1,752,951	2,157,459	172,197	399,238	lat∈[32.524, 42.112] lon ∈[-114.130, -124.406]

6.1.1 Datasets

We use real-life datasets from three large metropolitan cities in our experiments. The details of the datasets are listed in Table 2. The road network of the cities have been extracted from OpenStreetMaps (OSM)[1]. The bounding box for most major cities are provided by OSM in their website. We use this bounding box to extract the road network of the chosen city. The geo-textual objects are crawled from both OSM and Google Places to obtain as many keywords as possible. A geo-textual objects is represented as a tuple composed of geographical location and a list of keywords, such as "restaurant", "park" etc. The geographical location is the latitude and longitude of the place. Each object is mapped to its nearest node on the road network according to Euclidean distance. The Euclidean distance between two locations is calculated by converting the latitude and longitude pairs to the UTM (Universal Transverse Mercator coordinate system) format, using World Geodetic System 84 specification.

6.2 Efficiency and Scalability

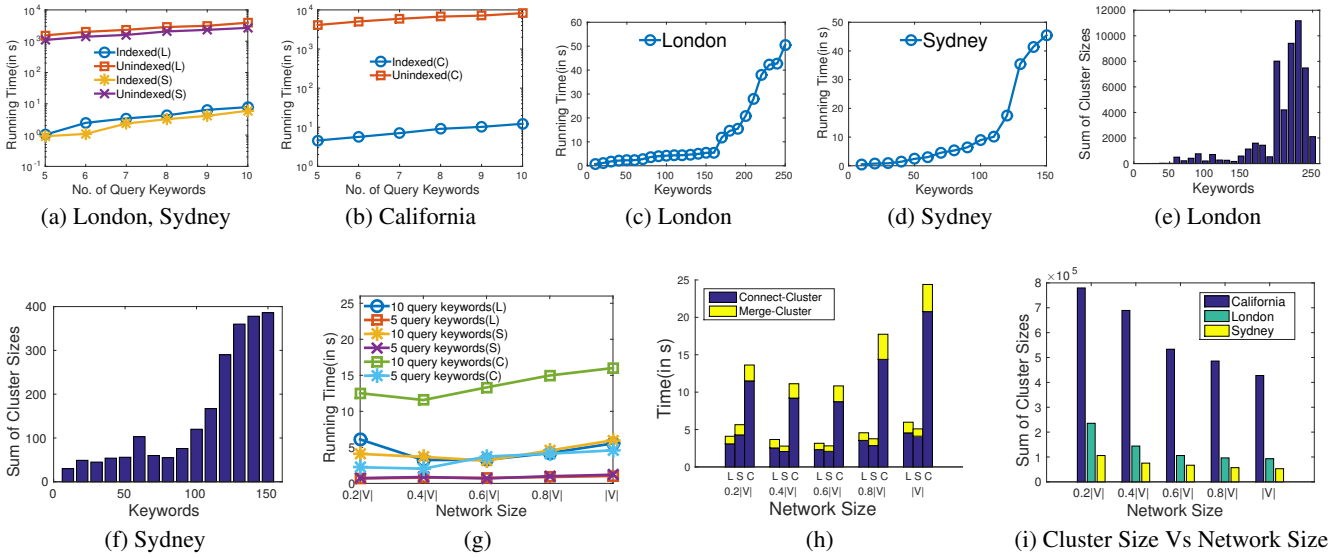
Figs. 5a-5b show the time taken to find the skyline neighborhoods as the number of query keywords is varied. We present the time taken by SkyGraph both with the help of Partner Index (indexed) as well as without it (unindexed). Note that the 2-hop label index is used in both cases and the only source of difference in the running time is the usage of Partner Index.

This experiment reveals several key aspects of the SkyGraph algorithm. First, across both Sydney and London, the running time is less than 3 seconds when boosted by Partner Index. Even in California, which contains more than 1.7 million nodes, the running time is only 13 seconds. This establishes that SkyGraph can be operationalized on any major city of the world. Second, Partner Index plays a major role in boosting the efficiency of the system. On average, Partner Index prunes more than 96% of the nodes in the network where the execution of CONNECT-CLUSTER is entirely skipped. Consequently, a 1000 times speed-up is achieved.

Other than the number of query keywords, the choice of the query keywords themselves play a role in the efficiency of the system. We next present results to understand this relationship better.

The basic question we ask in this experiment is the following: *How does the frequency of a query keyword impact the running time?* Since Connect-cluster needs to be run on all nodes containing a query keyword, a high frequency keyword would result in a large number of CONNECT-CLUSTER executions. On the other hand, if the query keywords are rare, then the existence probability of a small neighborhood (or subgraph) containing them is low. In other

[1] 1. Go to http://www.openstreetmap.org/. 2. Search for the desired city (or state). 3. Click on export to get bounding box. 4. Extract the sub-network within that bounding box using OSM API.



(a) London, Sydney (b) California (c) London (d) Sydney (e) London

(f) Sydney (g) (h) (i) Cluster Size Vs Network Size

Figure 5: **(a-b) Growth rate of querying time against the number of query keywords. (c-d) Growth rate of querying time against the frequency of the query keywords. (e-f) The change in cluster (subgraph) sizes on which CONNECT-CLUSTER is executed against the frequency of the query keywords. (g) Growth rate of running time against network size. (h) Profiling the time spent on MERGE-CLUSTER and CONNECT-CLUSTER as the network size is varied. (i) The impact of network size on the sizes of the clusters (subgraphs) on which CONNECT-CLUSTER is executed.**

words, CONNECT-CLUSTER needs to be run on a large subgraph and thereby significantly driving up the running time.

Figs. 5c-5d answer the above questions. To understand the impact of frequency, we extract the top-250 and top-150 most frequent keywords from London and Sydney, which constitutes 88% and 82% of all keywords in the two datasets respectively. Next, we group them into sets of 10 query keywords, where the top-10 query keywords are the 10 most frequent keywords and the keywords numbered from 241 to 250 are the least frequent keywords (in London). In Figs. 5c-5d, a value 50 in the x-axis corresponds to the query set containing keywords in the range top-40 to top-50 most frequent. As can be seen clearly, the running time increases with decrease in frequency. This behavior stems from the fact that when the query keywords are rare, larger subgraphs are required to span them. To establish this concretely, we study the subgraph sizes on which CONNECT-CLUSTER is executed as the frequency of query keywords is lowered in Figs 5e-5f. As expected, the subgraph sizes grow with decrease in frequency and thereby clearly indicating the reason behind the increase in running time.

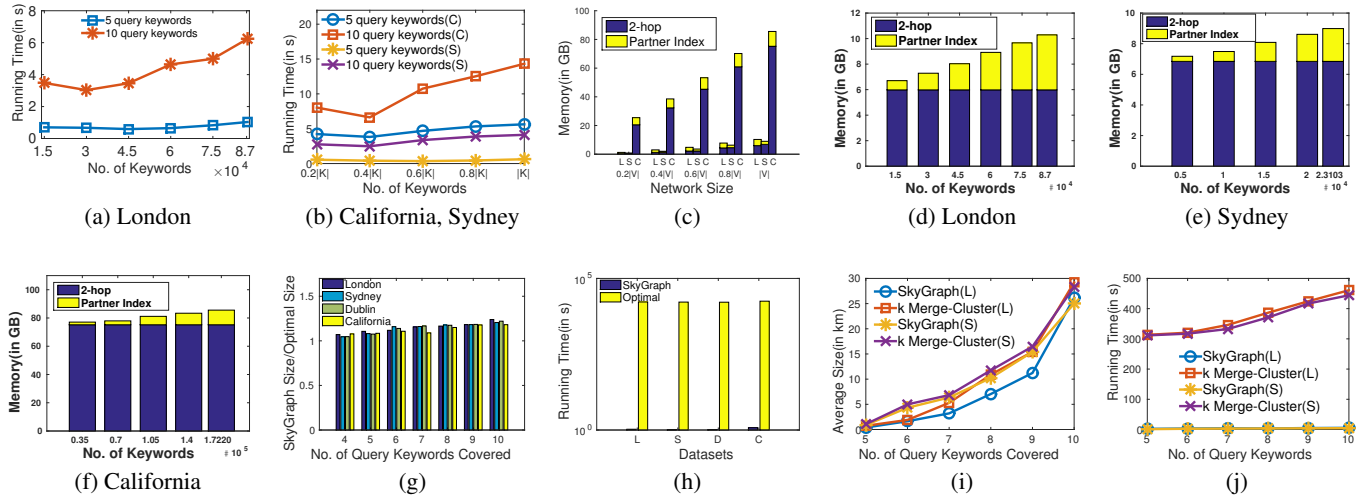
Next, we study the impact of network size on running time. Towards that end, we extract sub-networks from the London, Sydney and California datasets. To construct a sub-network of London containing X nodes, we pick a random node in the London network and expand through breadth first exploration till X nodes are added. This construction procedure ensures that when we further grow the network from X nodes to some $X + \Delta$ nodes, the previous network of size X is a subgraph of the $X + \Delta$-sized network. Whenever we add a node to the sub-network, we also add the keywords contained within that node. The same procedure is followed to construct sub-networks of Sydney and California as well.

Fig. 5g presents the results for query sets of sizes 5 and 10. For 5 query keywords, we do not observe much variation in the running time. However, for 10 query keywords, the running time decreases initially and then starts increasing again. The plot reveals that there is a "sweet spot" when it comes to the network size.

This behavior seems counter-intuitive at first. To further investigate, we plot a bar graph to profile the relative time spent in the MERGE-CLUSTER and CONNECT-CLUSTER algorithms as shown in Fig. 5h. We find that the majority of the time is spent in the CONNECT-CLUSTER algorithm. After the MERGE-CLUSTER algorithm returns the clusters, the CONNECT-CLUSTER algorithm joins these clusters to compute the connected k-skyline subgraph. The distance between two clusters C_1 and C_2 is the all pairs shortest distance between a node belonging to C_1 and a node belonging to C_2. If the sizes of the clusters returned by MERGE-CLUSTER algorithm are large, then a large number of shortest path computations need to be performed to determine optimal connecting path.

To establish that large cluster sizes are indeed the reason behind this behavior, in Fig. 5i, we plot the total sum of cluster sizes for the 100 test cases corresponding to each network size. We observe that the sum of cluster sizes decreases with network size. This justifies the large amount of time consumed by the CONNECT-CLUSTER algorithm and in turn the high running time for smaller networks.

This result brings us to an obvious question: *Why are the cluster sizes large when the network sizes are small?* The answer is tied to the frequency of the query keywords and how keywords are typically distributed in a city. Both in London and Sydney, there are certain pockets, such as the downtown region, with high density of keywords (or PoIs). If these pockets are not included in the sub-network, most keywords go into the "rare" category. Now, recall from our earlier analysis that when the query keywords are rare, larger subgraphs are required to span them all. Consequently we observe this counter-intuitive behavior. When the size of the network increases, it brings in more keyword nodes and the possibility of a small region that is densely populated with the query keywords. The "sweet spot" spot is, therefore, the scenario where the entire road network is not too large, but it encompasses the keyword-dense regions. A similar effect is also visible in California, but not as pronounced as London and Sydney.



(a) London (b) California, Sydney (c) (d) London (e) Sydney

(f) California (g) (h) (i) (j)

Figure 6: **(a-b) Growth rate of running time against the number of keywords in the network. |K| denotes total number of keywords (c) Variation in memory footprint against the network size and (d-f) number of keywords. (g) Comparison of the sizes of the skyline subgraphs retrieved by SkyGraph and the optimal brute-force algorithm. (h) Comparison of running time of SkyGraph and the optimal algorithm (i) Comparison of skyline subgraph sizes between SkyGraph and the modified k-MERGE-CLUSTER algorithm. (j) Comparison of the running time of SkyGraph and k-MERGE-CLUSTER algorithm.**

We next focus on the relationship between the total number of keywords in a network to the running time. Figs. 6a, 6b show the variation of running time with respect to the number of keywords for various datasets.

To construct a road network containing X keywords, we randomly select X keywords from the entire set; all remaining keywords are deleted. Next, to grow to larger set of $X + \Delta$ keywords, we add an additional Δ keywords while retaining the previous X keywords. This procedure ensure that each larger set of keywords is a superset of all the smaller sets of keywords.

As visible in Figs. 6a, 6b, for 5 query keywords the change in querying time is minimal. For 10 query keywords, initially, the running time decreases slightly, and then starts to increase again. The reason is similar to our analysis of running time with growth in network size. When the number of keywords is less, the cluster returned by MERGE-CLUSTER are large and consequently, the running time increases.

6.3 Memory Consumption

After thoroughly establishing the efficiency of the SkyGraph algorithm, we next focus on analyzing its memory footprint. The primary driving force behind memory consumption of SkyGraph algorithm is the Partner Index. Note, that we also employ *2-hop label* index structure [15] to index shortest path queries. In our next set of experiments, we analyze the impact of these index structures on the memory consumption.

First, we evaluate the growth rate of memory consumption against the network size in the London, Sydney and California datasets. Fig. 6c presents the results. The network size is increased in the same manner as in our scalability experiments against the network size (Fig. 5g). As expected the memory consumption increases with network size. Our theoretical analysis in Sec 5.1.1 derives a $O(nW)$ storage complexity where n is the number of nodes in the network and W is the number of unique keywords. Consequently, the linear growth rate in the storage complexity of partner index with respect to the network size is expected.

An interesting observation in Fig. 6c is that London consistently consumes more memory than Sydney. This is due to the fact that London has more number of keywords. Thus, the W factor in the $O(nW)$ complexity of Partner Index drives up the storage cost of London. With increase in network size, the number of keywords in the network also increases. Consequently, the gap in the storage of London and Sydney also increases. This behavior is more concretely established in Fig. 6d and Fig. 6e, where we observe the effect of the number of keywords on memory consumption. Observe that in California the proportion of memory consumption from 2-hop index is much larger than London and Sydney(Fig. 6f). This follows from the fact that keyword density in California is lower than in urban regions such as London and Sydney.

6.4 Comparison with the Optimal

In this section, we compare the quality and running time of SkyGraph with the optimal algorithm of exponential computation cost. Fig. 6g compares the size of the actual k-skyline to the subgraph returned by SkyGraph. In other words, we empirically verify the approximation quality of the proposed k-skyline subgraph algorithm. Since computing the optimal k-skyline subgraph is NP-Hard even on the smallest of networks, the running time is exorbitantly high. Nonetheless, to obtain some intuition, we extract 10 subgraphs spanning 20 nodes from each of the four road networks in Table 2 so that the optimal solution can be computed within 5 hours. The optimal technique does not scale for subgraphs of size greater than 20 because the number of subgraphs is exponential in the number of nodes in the road network graph. We observe that SkyGraph is able to find a neighborhood that is within 1.25 times the size of the optimal neighborhood for all query set sizes. The ratio of the size of the neighborhood found by SkyGraph to that of the optimal neighborhood increases with the increase in the number of query keywords covered. This result follows our theoretical analysis where the approximation quality is expected to deteriorate with increase in k. Nonetheless, this deterioration is mild.

Fig. 6h compares the running time to find all the skyline subgraphs covering 4 to 10 query keywords in different networks by

the optimal algorithm to SkyGraph. Even in a small 20 node subgraph, SkyGraph is 5 orders of magnitude faster than the optimal algorithm.

6.5 Varying Parameter k in Merge-Cluster

A question that we raised earlier in Sec. 6.4 was what if we try to find the k-skyline subgraph by allowing the MERGE-CLUSTER algorithm to run until k query keywords are discovered instead of $k/4$. We call this modified MERGE-CLUSTER algorithm the k-*Merge-Cluster* algorithm.

First, we compare the size of the k-skyline subgraphs retrieved by MERGE-CLUSTER when we let it run till a coverage of k. Fig. 6i presents the results. The value of k is varied from 5 to 10. For the k-MERGE-CLUSTER algorithm, the average size of the skyline subgraphs is slightly larger than the $k/4$ version used by SkyGraph.

While the quality is comparable for both versions of MERGE-CLUSTER, a major difference is observed in their running times. Fig. 6j presents the running time as the number of query keywords grows. We observe that for both cities the k-MERGE-CLUSTER algorithm is slower by up to 2 orders of magnitude.

In k-MERGE-CLUSTER, when trying to cover q query keywords, as the coverage of a cluster C_i becomes k-1, to get the final query keyword only one more unique query keyword is required. But due to the coverage of the cluster being k-1, the denominator of the ratio Eq. 6 that we are trying to minimize with respect to some other cluster C_j, $min\{unique(C_i, C_j), unique(C_j, C_i)\}$ tends to become 0 as the cluster C_i contains all but 1 query keyword. This leads to a high number of iterations in k-MERGE-CLUSTER where clusters of coverage of 1 or 2 query keywords are merged, thereby leading to increased running time.

6.6 Comparison with Existing Techniques

None of the existing techniques solve the skyline-subgraph problem in their native form. In this section, we explore possibilities of adopting LCMSR [2], the closest technique to our problem, and benchmark its performance. LCMSR [2] aims to find the smallest region in the road network whose size in terms of the minimum spanning tree is lesser that a user-given constraint and that best matches the query keywords. LCMSR expands a region greedily starting from a single node based on a preference function. The preference function is a weighted linear combination of subgraph size and coverage. μ is the weightage given to subgraph size and $1-\mu$ is the weightage for coverage. At each step, LCMSR chooses the node providing highest marginal gain. This process of expansion continues till the size constraint is violated. We explore two adaptations of the LCMSR query for our skyline subgraph problem.

• **LCMSR-k:** In contrast to our problem where we need to identify the smallest subgraph for each value of k (number of query keywords covered), LCMSR aims to find the smallest subgraph containing all query keywords. Thus, for a given set of query keywords Q, we perform LCMSR queries for each possible subset of k query keywords where $k \in [1, |Q|]$. LCMSR-k terminates when the desired coverage is reached. Note, that LCMSR-k does not necessarily find the smallest subgraph of coverage k. It is a heuristic where the preference function guides the expansion strategy. We set $\mu = 0.2$ since LCMSR [2] reports best quality at this value.

• **LCMSR-μ:** In LCMSR, a small μ results in an expansion strategy where coverage of keywords is of higher importance; a larger μ puts more emphasis on keeping the subgraph size small. Thus, instead of explicitly tuning k, we vary μ from 0.1, 0.2, …, 1.0, resulting in 10 pre-defined preference functions. A subgraph is returned for each value of μ, and we consolidate them to find the smallest subgraph for each unique coverage value.

Fig. 7a shows the running times of LCMSR-k, LCMSR-μ and SkyGraph against the number of query keywords in London and Sydney. SkyGraph outperforms LCMSR-k and LCMSR-μ by up to 3 orders and 2 orders of magnitude respectively. LCMSR-k is expensive since the LCMSR query needs to be performed on each possible subset of the query keywords. LCMSR-μ is relatively faster, but still slower than SkyGraph, since for each query, it needs to be run 10 times corresponding to each value of $\mu \in [0.1, 0.2, \cdots, 1]$.

As we mentioned earlier, the value of μ in LCMSR-k is set to 0.2 since to optimize quality. To investigate the impact of μ on LCMSR-k, we compare its running time to SkyGraph across various values in Fig. 7b. As can be seen, there is minimal variance in the running time. This is consistent with results reported in LCMSR [2].

Finally, we evaluate the quality of subgraphs returned by LCMSR-k and LCMSR-μ. Fig. 7c shows the growth rate of subgraph size against keyword coverage. LCSMR keeps expanding the neighbourhood till the required number of keywords are covered. When the number of query keywords to be covered is less, the expansion stops early, whereas for higher number of query keywords covered, the expansion continues longer, leading to larger neighbourhoods Since LCMSR is not designed to identify skyline subgraphs, even when adopted for our problem, the quality suffers.

6.7 Dimensions more than Two

Although we have considered only two dimensions to characterize a neighborhood subgraph so far, additional dimensions such as diameter, ratings, popularity, etc. can be easily incorporated as well. We next show how diameter can be handled through our framework.

When the diameter is added as the third dimension, we use d_k, the distance to k^{th} farthest query keyword from root node u (line 3, Algorithm 3), as the lower bound for the diameter of the k-skyline instead of the bound obtained from the partner index. Fig. 7d shows that we are able to report the skyline neighborhoods within a reasonable running time of less than 16 seconds.

We also explore whether the diameter of the k-skyline provides any additional information over the two features already used. To this end, we take answers from 100 skyline neighborhood queries and then generate two ranked lists by sorting the answers of each query based on the diameter and its size. Then, we compute the Spearman's rank correlation coefficient between the lists. Fig. 7e shows that the Spearman's rank correlation coefficient varies between 0.89 and 0.93 for London and Sydney. This indicates that diameter and size are correlated and adding diameter as the third dimension may not be much useful.

The above results however should not be generalized to other features (e.g., popularity, ratings) since each feature brings in their own complexity issues and characteristics. Any comment on the scalability and utility of additional dimensions can be made only after a thorough investigation by incorporating each one of them. We leave this aspect of our work as a future study.

7. RELATED WORK

Queries on geo-textual databases can be broadly classified into the following two types.

Point of Interest (PoI) Queries: These queries retrieve a set of PoIs, characterized by keywords and spatial attributes, from a geo-textual database instead of a region. Some queries assume a Boolean preference function for the keywords such as *Boolean range queries* [12] and *Boolean k-NN queries* [5, 9], and require all the query keywords to be covered. In *k-NN queries* [2, 7, 8], a

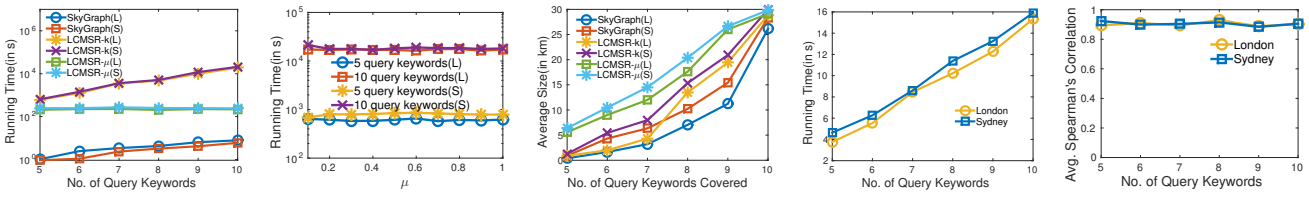


Figure 7: **(a) Comparison of the running time between SkyGraph and LCMSR. (b) Variation of running time against μ for LCMSR. (c) Comparison of skyline subgraph sizes between SkyGraph and LCMSR. (d) Running time against number of query keywords for 3-dimensional skyline neighborhoods. (e) Average Spearman's rank correlation between diameter and length of k-MST.**

weighted preference function is used to assign a relative importance between a distance function and a textual relevance function. Some works [2, 3, 8, 9, 20, 25, 26] assume an underlying Euclidean space while others assume a spatial network to estimate the cost of covering the query keywords more realistically [4, 6, 7, 22]. Recently, Zheng et al. addressed the problem of learning the user's preference automatically in [28]. Our problem differs from these works in that it finds a region, rather than a list of objects, that covers all or a subset of the query keywords.

Region of Interest (RoI) Queries: Most of the papers dealing with these queries assume the distance between two objects to be the Euclidean distance. The *m-closest keywords* (mCK) query finds a group of objects such that they cover all query keywords and have the smallest diameter, which is defined as the largest distance between any pair of objects in the group [11, 25, 26]. The *spatial group keyword query* (SGK) [3, 20] is similar to the mCK query. In addition to diameter, it also the considers the distance to a query location in the preference function. The weakness of these queries is that they return a region in the form of a geometric shape like a circle or a rectangle and consider the underlying space to be Euclidean. The road network distance between PoIs is a more accurate measure. Cao et al. propose the *length constrained maximum-sum region* (LCMSR) query that retrieves regions that are subgraphs of a road network graph [4]. The query returns a minimum spanning tree whose length is lesser than a size constraint given by a user and that maximizes a *tf-idf* based coverage function based on the number of query keywords in the region. However, this technique assumes the knowledge of a preference function which constraints the balance between relevance and neighborhood size. Our work does not depend on such a preference function and, therefore, provides a more flexible solution. Technically, our problem reduces to skyline subgraph queries in keyword-annotated graphs, a problem that has *not* been solved earlier.

8. CONCLUSIONS

In this paper, we studied the problem of identifying the best *Region of Interest (RoI)* for a given a set of user provided query keywords. Two factors influence the quality of an RoI: its relevance and size. While it is reasonable to assume that a smaller size and higher relevance is always preferred, the relative importance of these two factors vary from user to user. We propose a framework called *SkyGraph* where the best RoI would be part of the answer set regardless of the user's preference function. This remarkable property is achieved by leveraging the power of *skyline subgraph queries* on keyword-embedded road networks. Since the problem is NP-hard, we developed an approximation algorithm with provable quality guarantees. To enable fast response times, SkyGraph was boosted by an indexing technique called *Partner Index*. Extensive experiments on large road networks established the efficiency and efficacy of the proposed techniques.

9. REFERENCES

[1] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala. Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. In *STOC*, pages 277–283, 1995.

[2] X. Cao, G. Cong, and C. S. Jensen. Retrieving top-k prestige-based relevant spatial web objects. *PVLDB*, 3(1-2):373–384, 2010.

[3] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In *SIGMOD*, pages 373–384. ACM, 2011.

[4] X. Cao, G. Cong, C. S. Jensen, and M. L. Yiu. Retrieving regions of interest for user exploration. *PVLDB*, 7(9):733–744, 2014.

[5] A. Cary, O. Wolfson, and N. Rishe. Efficient and scalable method for processing top-k spatial boolean queries. In *SSDBM*, pages 87–95. Springer, 2010.

[6] H.-J. Cho and C.-W. Chung. An efficient and scalable approach to cnn queries in a road network. In *PVLDB*, pages 865–876. VLDB Endowment, 2005.

[7] H.-J. Cho, S. J. Kwon, and T.-S. Chung. Alps: an efficient algorithm for top-k spatial preference search in road networks. *Knowledge and Information Systems*, 42(3):599–631, 2015.

[8] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2(1):337–348, 2009.

[9] I. De Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *ICDE*, pages 656–665. IEEE, 2008.

[10] N. Garg. A 3-approximation for the minimum tree spanning k vertices. In *focs*, volume 96, pages 302–309, 1996.

[11] T. Guo, X. Cao, and G. Cong. Efficient algorithms for answering the m-closest keywords query. In *SIGMOD*, pages 405–418. ACM, 2015.

[12] R. Hariharan, B. Hore, C. Li, and S. Mehrotra. Processing spatial-keyword (sk) queries in geographic information retrieval (gir) systems. In *SSDBM*, pages 16–16. IEEE, 2007.

[13] H. Hu, D. L. Lee, and V. Lee. Distance indexing on road networks. In *PVLDB*, pages 894–905. VLDB Endowment, 2006.

[14] H. Hu, D. L. Lee, and J. Xu. Fast nearest neighbor search on road networks. In *EDBT*, pages 186–203. Springer, 2006.

[15] M. Jiang, A. W.-C. Fu, R. C.-W. Wong, and Y. Xu. Hop doubling label indexing for point-to-point distance querying on scale-free networks. *PVLDB*, 7(12):1203–1214, 2014.

[16] M. Kolahdouzan and C. Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *VLDB*, pages 840–851, 2004.

[17] K. C. Lee, W.-C. Lee, and B. Zheng. Fast object search on road networks. In *EDBT*, pages 1018–1029. ACM, 2009.

[18] K. C. Lee, W.-C. Lee, B. Zheng, and Y. Tian. Road: A new spatial object search framework for road networks. *TKDE*, 24(3):547–560, 2012.

[19] B. Liao, M. L. Yiu, Z. Gong, et al. Beyond millisecond latency nn search on commodity machine. *TKDE*, 27(10):2618–2631, 2015.

[20] C. Long, R. C.-W. Wong, K. Wang, and A. W.-C. Fu. Collective spatial keyword queries: a distance owner-driven approach. In *SIGMOD*, pages 689–700. ACM, 2013.

[21] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *PVLDB*, pages 802–813. VLDB Endowment, 2003.

[22] J. B. Rocha-Junior and K. Nørvåg. Top-k spatial keyword queries on road networks. In *EDBT*, pages 168–179. ACM, 2012.

[23] H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *SIGMOD*, pages 43–54. ACM, 2008.

[24] S. Srivastava, S. Pande, and S. Ranu. Geo-social clustering of places from check-in data. In *ICDM*, pages 985–990, 2015.

[25] D. Zhang, Y. M. Chee, A. Mondal, A. K. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *ICDE*, pages 688–699. IEEE, 2009.

[26] D. Zhang, B. C. Ooi, and A. K. Tung. Locating mapped resources in web 2.0. In *ICDE*, pages 521–532. IEEE, 2010.
[27] B. Zheng, K. Zheng, X. Xiao, H. Su, H. Yin, X. Zhou, and G. Li. Keyword-aware continuous knn query on road networks. In *ICDE*, pages 871–882. IEEE, 2016.
[28] K. Zheng, H. Su, B. Zheng, S. Shang, J. Xu, J. Liu, and X. Zhou. Interactive top-k spatial keyword queries. In *ICDE*, pages 423–434. IEEE, 2015.
[29] R. Zhong, G. Li, K.-L. Tan, and L. Zhou. G-tree: An efficient index for knn search on road networks. In *CIKM*, pages 39–48. ACM, 2013.

APPENDIX

A Proof of Lemma 1

PROOF. Suppose, we merge two clusters C_i and C_j to form a cluster C such that $min\{unique(C_i, C_j), unique(C_j, C_i)\} = p$. Then, from Eq. 6,

$$d(C_i, C_j) = r \times min\{unique(C_i, C_j), unique(C_j, C_i)\}$$
$$\leq Rp \text{ since } R \geq r$$

In other words, there exists a path of length at most Rp between C_i and C_j. Thus, if the sizes of C_i and C_j are S_i and S_j respectively, then $size(C) \leq S_i + S_j + Rp$. This follows from the fact that we can create a spanning tree on C by simply merging the MSTs on C_i and C_j through a path of length at most Rp. Since initially all clusters are single nodes of size 0, the maximum size of a cluster can be defined through the following recurrence.

$$Size(C_0) = 0 \tag{10}$$
$$Size(C_l) = 2 \times Size(C_{l-1}) + Rp \tag{11}$$

Here, C_l denotes the maximum possible size of a cluster that have undergone l merges. Solving this recurrence, we get

$$size(C_l) = (2^l - 1)Rp \tag{12}$$

For any C_i and C_j that are merged, $min\{unique(C_i, C_j), unique(C_j, C_i)\}$ is at least 1. Thus, there can be at most $k/4$ merges in any cluster provided by MERGE-CLUSTER. Therefore, for a cluster of coverage p, the maximum size is $(2^{k/4} - 1)Rp$. □

B Proof of Lemma 2

PROOF BY CONTRADICTION. Assume that the above statement is false. This means in some particular iteration of the MERGE-CLUSTER algorithm, $R > (8L \log_2 k)/k$. We next perform two tasks.

1. From the k-skyline subgraph OPT, in an arbitrary manner we remove all duplicate occurrences of keywords. Note we only delete the keyword and not the node. Following this operation, no query keyword in OPT is present more than once. The size of OPT remains unaffected.
2. Next, we extract all clusters that share at least one keyword node with OPT. We denote this set of clusters by $\mathbb{C}$.
3. Finally, we distribute clusters in $\mathbb{C}$ into $\log_2 k$ buckets such that bucket i contains all clusters where the number of keyword nodes that overlap with OPT is in range $(k/2^i, k/2^{i+1}]$, $i \geq 2$. Recall, that no cluster can exceed a coverage of $k/4$ in the MERGE-CLUSTER algorithm. Thus, all overlapping clusters would be restricted among the above buckets.

We observe that there must exist at least one bucket that contains more than one cluster since if every bucket contains exactly one cluster, then the maximum overlap with OPT achieved by merging all these clusters is $\sum_{i=2}^{\infty} k/2^i < k/2$. Thus, more than $k/2$ keywords from OPT remain to be covered, which means there is at least one bucket that has two clusters. Since we need to distribute more than $k/2$ keywords among $\log_2 k$ buckets, from *pigeonhole principle*, there is at least one bucket with two clusters that has more than $\frac{k}{2 \log_2 k}$ keywords. Let us say that this bucket contains clusters of overlap size in the range $(2p, p]$. In this bucket, the distance between any two clusters $d(C_i, C_j) \geq Rp$ since both C_i and C_j contains subgraphs C_i' and C_j' respectively that overlap with OPT and $unique(C_i', C_j') > p$. This means that if we draw balls of radius $\frac{Rp}{2}$ around any two clusters in this bucket, these balls will not overlap. Furthermore, since the maximum overlap with OPT of any cluster is $2p$, there are at least $\frac{k}{2 \log_2 k} \div 2p$ clusters in this bucket. The minimum length of OPT is the gap between all these clusters. In other words,

$$L \geq \frac{Rp}{2} \times \frac{k}{2 \log_2 k} \times \frac{1}{2p} \tag{13}$$
$$\text{or, } R \leq \frac{8L \log_2 k}{k} \tag{14}$$

The above result contradicts our assumption. Hence, proved. □

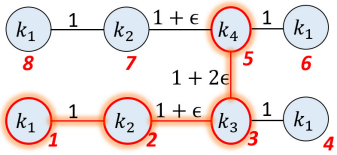


Figure 8: **Illustration of the sub-optimality of Merge-Cluster.**

C Why not run Merge-Cluster till k?

COROLLARY 3. *Lemma 2 does not hold for* $\frac{k}{4-\epsilon}$, $\epsilon > 0$.

PROOF: For any value, $\frac{k}{4-\epsilon}, \epsilon > 0$, point 3 in the proof of Lemma 2 is violated and hence it does not hold.

To understand how sub-optimal MERGE-CLUSTER can get if we let it run till k, we derive the following theorem.

THEOREM 6. *If we let Merge-Cluster run till k, in the worst case, the subgraph returned is at least* $O(\frac{2^k}{k})$ *times the size of the optimal. In other words, the worst-case quality is* $O(f(n))$, *where* $f(n) \in \Omega(\frac{2^k}{k})$.

PROOF. Consider Fig. 8, which follows a generic pattern. First, all nodes containing keyword k_i is connected to all nodes containing keyword k_j where $j > i$. Second, all edges of the form (k_i, k_j), $j > i$ has distance $1+(i-1)\epsilon$, where ϵ is a small positive number. Suppose the query contains the keywords $\{k_1, k_2, k_3\}$. The optimal answer is the subgraph spanning nodes $\{1, 2, 3\}$ of length ≈ 2. However, the subgraph returned by MERGE-CLUSTER if we let it run till k, which is 3 in this case, is $\{1, 2, 3, 4\}$ of size ≈ 3. This follows from the fact that MERGE-CLUSTER would first form clusters by joining all nodes containing k_1 to their adjacent nodes, followed by connecting k_2 to nodes containing k_i, $i > 2$ and so on. Let us now consider the query set $\{k_1, k_2, k_3, k_4\}$. In this case, the optimal subgraph spans the nodes $\{1, 2, 3, 5\}$ (highlighted in glowing red color) of size ≈ 3. In contrast, MERGE-CLUSTER returns the entire graph in Fig. 8 resulting in a size of ≈ 7. When Fig. 8 is generalized for k keywords, the ratio of the size of the subgraph returned by MERGE-CLUSTER to that of the optimal grows at $\frac{2^{k-1}-1}{k-1} = O(\frac{2^k}{k})$. □

While Theorem 6 provides a lower bound on the worst-case guarantee of MERGE-CLUSTER, it is not the worst-case guarantee itself. This is a limitation of our work, which requires further study.