Exploring Index and Query Optimization Techniques for Database Applications

by

Liang Tang

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
May 7, 2016

Keywords: Database Query Optimization, Spatial Database, Social Network Analysis, MapReduce

Abstract

Database index can be regarded as a data structure that speed up the data retrieval operations on a database table. The cost of indexing in database is additional writes and storage space to maintain the data structure. The created data structures in database are used to quickly query data without having to search every row of table in most relational database management system (RDBMS). The read and write performance of database is elevated by bringing in appropriate indexing technique, given the specific data type. As a result, indexing technique plays a significant role in database applications.

After index is built completely, database will be able to answer the query. Generally, a query is a request for information from a database. It can be as simple as "finding the address of the head-quater of company ZZ," or more complex like "finding the average total amount of penalties for football players who live in Auburn or Opelika, incur more than 3 penalties, and captain less than 2 teams." In order to quickly resolve the query result, we raise the definition of query optimization. The query optimization techniques try to determine the most efficient way to execute a specific query by considering all the possible query strategies. The goal of the query optimization is to find the way to process a given query in minimum time. The main contribution of this dissertation is to explore and study out efficient indexing and query optimization techniques regarding the specific problem. Three concrete database applications will be analyzed and explored, and indexing and querying techniques will be proposed respectively in order to enhance the database performance.

First, two watchtower-based parameter-tunable indexing methods are introduced for efficient spatial processing with sparse distributions of Points of Interest (POIs) by exploiting mobile users' check-in data collected from the location-aware social networks. In the proposed frameworks, the network traversal can terminate earlier by retrieving the distance information. More important, by observing that people's movement often exhibit a strong spatial pattern, we employ Bayesian

Information Criterion-based cluster analysis to model mobile users' check-in data as a mixture of 2-dimensional Gaussian distributions, where each cluster corresponds to a geographical hot zone. Afterwards, POI watchtowers are established in the hot zones and non-hot zones discriminatorily. Moreover, the optimal watchtower deployment mechanism is discussed in order to achieve a desired balance between the off-line pre-computation cost and the on-line query efficiency. Finally, the superiority of our solutions over the state-of-the-art approaches is demonstrated using the real data collected from Gowalla with large-scale road networks.

Next, a novel probabilistic inference query machine is introduced to do the statistical inference on social network. Social network analysis will be carried out, because it attracted more and more interest from researchers due to the rapidly increasing availability of massive social network data. The link prediction problem is one of the most fundamental problems in social network analysis, and therefore has been extensively studied. In this dissertation, LinkProbe, a framework to quantitatively predict the existence of links in large-scale social networks based on Markov Logic Networks (MLNs) is designed and implemented. Differing from other probabilistic graph models, such as Bayesian Networks (BNs) or Conditional Random Fields (CRFs), MLNs allow undirected relationship with cycles and long-range (more than one hop) dependency, which are essential and abound in social networks. The extensive experiments with real social datasets verify the effectiveness and efficiency of our proposed framework.

Finally, a method to resolve the probabilistic skyline query problem is proposed by using MapReduce programming model in a distributed machine environment. The method is able to filter unqualified objects (not needed by users) during the early MapReduce cycle. Only qualified objects is passed to compute the real probabilistic skyline probabilities. In order to let multiple machines filter objects simultaneously, high dimensional data is fetched by hyper-angular partitioning. Several efficient filtering strategies are proposed based on the feature of the angular partitioning. Comparisons against other State-of-the-art approaches are demonstrated in the experiments.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my advisor, Prof. Wei-Shinn Ku. I have been fortunate to have an advisor who gave me the freedom to explore on my own, and at the same time the guidance to recover when my steps faltered. I appreciate all his contributions of time, ideas, and funding to make my PH.D. experience fun and productive. I could not have had a better supervisor.

Besides, my co-advisor, Dr. Xiao Qin, has been always there to give me insightful comments and encouragements for my research. I want to thank him for the weekly discussions that helped me clarify tons of research issues and problems.

In addition, I must thank Dr. Haiquan Chen, who is a recent graduate in our research group. He is one of the best teachers that I have had in my PH.D. life. He sets high standards for research work and and guides me to meet those standards. My research manuscripts can not be accepted without his constructive criticism, warm encouragement, and countless idea sparkles in our discussions. Also, he carefully read endless revisions of my papers, correcting the grammar and verbal problems. I am alway able to learn from him at every discussion.

My sincere thanks also goes to Dr. Ji Zhang, Dr. Min-Te Sun,who gave me a lot of supervision and instructions. Dr. Zhang eagerly taught me much when I began touching the MapReduce problem. At the very beginning of my PH.D. semester, Dr. Sun carefully helped me resolve algorithm problems regarding spatial indexing.

Next, I am grateful to two amazing software engineering managers, Vijay Swamidass and Ranjith Prubu, who recruited me to two awesome companies (Salesforce and Linkedin) respectively as a software engineer intern, for their trust in my programming abilities.

My heartful thanks to my fellow lab mates, Jeff Wang, Jiao Yu, Zhitao Gong, Yusheng Ding, Ting Shen, Wenlu Wang, Ya Chi Kuo for always being there during my wonderful days of PH.D.

life. I also thank people who were not part of my lab but helped me out, including Song Gao, Yuanqi Chen, Cong Xu, Zhuo Liu, Kang Sun, Ming Fang for all your support, and relevant academic discussions. I want to also thank my friends (Li Lin, Junchao Wei, Jue Wang, Hao Wu, Matthew Golson, Madeline Arnall, Josh, etc. too many to list here but you know who you are!) for providing support and friendship that I needed.

The best stuff throughout the past five years is finding my girl. Jiayi, who always loved and supported me, made me understand what true love could be. She has selflessly give more to me than I every could have asked for.

Lastly, I thank my mom, dad for all their love, dedication, support, and encouragement in helping me achieve today of my own. They are always supportive with unconditional love whenever times are rough.

Table of Contents

List of Figures

List of Tables

Chapter 1

Parameterized Spatial Query Processing based on Social Probabilistic Clustering

1.1 Introduction

Due to the recent advances in wireless communication technology, mobile devices (e.g., smart phones, tablets, etc.) with Internet access and positioning chips are significantly increasing in popularity. On the other hand, many vendors are providing various map and navigation services (e.g., MapQuest and Google Maps). As a result, we are witnessing the fast growth of location-based services (LBS), which allow mobile users to issue spatial queries from their mobile devices based on user-specified locations in a ubiquitous manner. Among all spatial queries, the k nearest neighbor (kNN) query [71, 12, 46] and range query [65] are the important building blocks used to realize LBS with more complex queries [51, 10, 27]. Therefore, the question of how to evaluate kNN queries and range queries has received considerable attention from both industry and academia in the past decade. One of the major challenges in the above problem is how to provide efficient evaluation of the queries for large road networks with massive Points of Interest (POIs), especially when the desired POIs are far away from the query point. Two of the well-known solutions that attempt to address this challenge are ROAD [45] and Islands [36]. However, the significant overhead limits their applications in practice, as listed in Table 1.1. For the North America Road Network, ROAD requires about one hour for index setup while Islands needs more than 30 GB for index storage (see Section 2.8 for details). Consequently, neither of them scale well towards large road networks with massive POI data.

1.1.1 The Motivating Problem

Our motivating problem is twofold.

Approach	Setup time	Storage cost
ROAD	$\approx$ 60 mins	$\approx$ 35 MB
Islands	$\approx$ 243 mins	$>$ 30 GB

Table 1.1: The overheads of the ROAD and Islands approaches on the North America Road Network (NA) (179,178 edges and 175,812 nodes) with 100,000 random synthetic POIs.

- How can we efficiently process spatial queries on large road networks with massive POIs? The inspiration of our solution is the road sign utilized in transportation systems. Such signs are erected on the sides of roads, providing users the distance information from the current location to a certain destination. Inspired by the above idea, we deploy watchtowers (serving as distance signature) on road networks. As a result, without the need to expand the network search to distant POIs or traversing any tree-based structure [32, 71] (e.g., the R-tree), the network search can terminate earlier by utilizing the distance information stored in the watchtowers in the proximity of the query point, which leads to a significant reduction in search time. Unlike what Islands [36] does, we store distance information only on selected anchor points in a parameter-tunable way for scalability.

- How can we take full advantage of the mobile users' check-in data acquired from the popular location-aware social networks to accelerate spatial query processing? By observing that people's movement often exhibits a strong spatial pattern, we use the probabilistic clustering algorithm to model the mobile user check-in data as a mixture of 2-dimensional Gaussian distributions where each cluster corresponds to a hot zone. Afterwards, watchtowers are established in the hot zones and non-hot zones discriminatorily.

1.1.2 Our Contributions

In this chapter, we propose two watchtower-based, parameter-tunable frameworks (the uniform watchtower framework and the hot zone-based watchtower framework) to speed up spatial query processing on large road networks. What distinguishes our work from other research efforts is that in our proposed frameworks (1) watchtowers are deployed on road networks as distance signatures in a tunable granularity without the need to maintain any tree-based structure for network traversal, (2) we incorporate mobile users' movement information (i.e., geo-social data) into the construction of watchtowers by using probabilistic clustering, and (3) we derive the optimal distance bound for setting up watchtowers to achieve a desirable balance between pre-computation cost and on-line query efficiency. The major contributions of this research are as follows:

- We propose the Uniform Watchtower (UW) framework to deploy watchtowers in a parameter-tunable manner for efficient evaluation of spatial queries.
- We provide the Hot zone-based Watchtower (HW) framework by incorporating mobile users' movement information into the construction of watchtowers. To be more specific, We employ Bayesian Information Criterion (BIC)-based hierarchical clustering to model the mobile users' check-in data as a mixture of 2-dimensional Gaussian distributions without prior knowledge on the number of geographic clusters. Based on the derived clusters (each corresponding to a hot zone), we deploy POI watchtowers in hot zones more densely than in non-hot zones in a discriminatory manner.
- We derive the optimal distance bounds for watchtower distribution with regard to kNN and range queries, respectively, so that (1) the POI watchtowers are deployed only in their spatial proximity instead of throughout the entire road network and (2) a desirable balance between pre-computation cost and on-line query efficiency can be achieved.
- We study the maintenance of our watchtower-based index structures and propose efficient strategies regarding both network update and POI update. The experiment section verifies the maintenance efficiency.
- We evaluate the performance of our proposed watchtower-based frameworks with ROAD [45] and Islands [36] by extensive experiments using geo-social data and various real road networks. The experiments show the superiority of our solutions over the state-of-the-art approaches.

The rest of this chapter is organized as follows. Section 2 surveys related works. The uniform watchtower framework is detailed in Section 3. In Section 4, we introduce the hot zone-based watchtower framework. The on-line query algorithms under our proposed frameworks are presented in Sections 5. In Section 6, we discuss the performance difference between the two proposed frameworks. In Section 7, we derive the optimal deployment of watchtowers to achieve a desirable balance between pre-computation and on-line query performance. In Section 8, we elaborate on index maintenance. The experimental validation of our design is presented in Section 9. Section 10 concludes this chapter.

1.2 Related Work

During the past decade, kNN query has been extensively studied and applied in various location-based service applications. Jensen *et al.* [38] proposed a data model with definitions of abstract functionality needed for moving kNN queries in road networks. For answering spatial queries on road networks, Papadias *et al.* [65] developed an Euclidean restriction framework and a network expansion framework to efficiently prune the search space. Based on the proposed frameworks, solutions for nearest neighbor queries are designed in the context of spatial network databases. In addition, because, for most users, the final goal of performing a kNN search is often to travel to one of the POIs of the search result, Ku *et al.* [43] introduced a solution for finding nearest neighbors in terms of travel time. They designed a novel travel time network, which integrates both spatial networks and real-time traffic event information. Based on the travel time network, algorithms were developed to provide adaptive nearest neighbor search results.

However, many of the existing kNN solutions [46, 12, 71, 38, 65] have been shown to be unscalable to large networks. Consequently, several approaches which pre-computed query results for potential searches in the future were proposed to improve kNN query efficiency. For example, a network Voronoi diagram-based approach, named Voronoi-based Network Nearest Neighbor (VN^3), was presented in [41], which partitions a large network into small Voronoi regions and pre-computes distances both within and across the regions. VN^3 can tackle sparse datasets; however, it cannot handle medium and dense datasets efficiently due to its high pre-computation and storage overhead. A kNN query solution named SPIE was presented in [34], which indexes the network topology based on a novel network reduction technique. The approach simplifies the network by replacing the graph topology with a set of interconnected tree-based structures with pre-computed NN results stored in tree nodes. kNN queries can be answered by accessing pre-computed results maintained at some of the nodes in the tree hierarchy. Hu *et al.* [33] proposed an efficient index (distance signature) for distance computation and query processing over long distances. Their technique discretizes the distances between POIs and network nodes into categories and then encodes these categories to accelerate the kNN search process. However, the index construction and index storage costs of the approach are very high because every network node needs to create and store abundant information. Therefore, the technique is not scalable. Sankaranarayanan *et al.* [72] presented a framework, termed SILC, for resolving the shortest path and the distance between every pair of vertices on a spatial network. SILC employs path coherence between the shortest path and the locations of vertices on the spatial network, resulting in an encoding that is compact in representation and fast in path and distance retrievals. Furthermore, in order to speed up kNN search, Samet *et al.* [71] designed an algorithm to explore the entire network by pre-computing the shortest paths between all the vertices in the network and employing a shortest path quadtree [30] to capture spatial coherence. With the algorithm, the shortest paths between all possible vertices can be computed only once to answer various kNN queries on a given spatial network. However, similar to [33], the solution in [71] incurs high I/O and computation costs and does not scale to large road networks. To overcome the shortcomings of solution-based approaches, Lee *et al.* [46] proposed a query framework named ROAD, which organizes a large road network as a hierarchy of interconnected regional subnetworks (called Rnet). ROAD maintains POIs separately from a given network and adopts an effective search space pruning technique to enhance search performance. Nevertheless, ROAD still needs to expand Rnets blindly towards all directions before retrieving all the kNN results, which deteriorates its search performance.

Huang *et al.* [36] proposed the Islands framework which pre-computes the distance between POIs and vertices with a given radius and stores the distance information on every vertex. However, the paper does not discuss how to select an optimal radius in practice. The main differences between our work and the work in [36] are listed as follows: (1) our frameworks provide a parameter-tunable tradeoff between pre-computation overhead and on-line query efficiency based on our watchtower deployment scheme and the setup of anchor points on road networks, (2) a maximum distance bound for creating watchtowers can be derived to achieve an optimal balance between pre-computation and query efficiency, and (3) we take into account mobile users' movement information during the construction of watchtowers. Our previous work [79] focuses only on kNN query precessing. However, in this dissertation, we extend our frameworks in support of range queries with the detailed algorithm, derive the optimal distance bounds for practical watchtower deployment, discuss the efficient strategies on index maintenance, and compare the performance of our proposed query frameworks with that of the-state-of-the-art approaches using extensive experiments.

Symbol	Meaning
p	An anchor point
t	A watchtower
$\mathbb{P}$	A set of anchor points
λ	The maximum distance between two adjacent anchor points
$SP(.,.)$	The shortest path between two points
$\|SP(.,.)\|$	The distance of a shortest path
$PSP(.,.)$	The possible shortest path between two points
$\|PSP(.,.)\|$	The distance of $\mathbb{SP}(.,.)$
l	The number of anchor points between two adjacent watchtowers (the selecting parameter)
r_d	The network distance threshold defined for all hot zones

Table 1.2: Symbolic notations.

1.3 The Uniform Watchtower Framework

The fundamental idea of using watchtowers for spatial query processing is to distribute and store the distance information (distance signature) of each POI on road networks so that the POI lookup can terminate once it encounters enough watchtowers to answer the query. For any POI o_i, we use its watchtowers to store the distance information from this current watchtower to o_i. With watchtowers, a query can be answered efficiently by checking only a few watchtowers close to the query point. In this section, we elaborate on how to distribute watchtowers to road networks in an approximately uniform fashion, which involves two steps, (1) anchor point deployment and (2) watchtower setup. Table 1.2 summarizes the notations in this chapter.

1.3.1 Anchor Point Deployment

A road network can be modeled as an undirected weighted graph $G = (V, E)$, where V denotes the set of nodes (road intersections or end points) and E signifies the set of edges. The weight of an edge ($w(v_1, v_2)$) indicates its length, and O is the set of POIs in G. Given two locations p and q, we use $SP(p, q)$ to denote the shortest path between them and $|SP(p, q)|$ to represent its distance.

Given a road network G, we distribute anchor points over G based on the following two rules. (1) For any node with a degree greater than 2 or equal to 1, we create an anchor point on it. (2) For two adjacent anchor points i and j obtained in (1), if their network distance is greater than λ, we add an anchor point every λ distance starting from i (or j) along the adjacent road segments so that there are no two anchor points with a distance greater than λ. Therefore, in G, any query point is able to find at least one anchor point within the distance of $\lambda/2$, where λ is a tunable parameter set by applications or users.

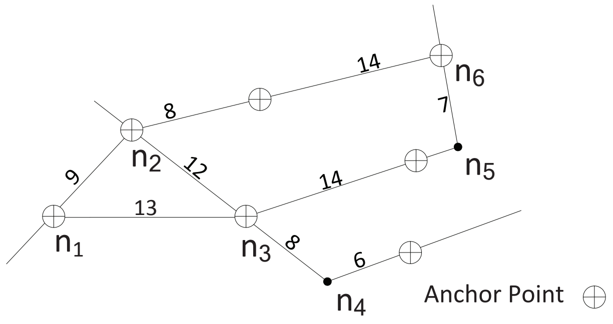
Algorithm 1 describes how anchor points are constructed on a graph. First of all, every edge is marked unvisited in the graph G. Then, a set $\mathbb{P}$ is initialized as empty. In addition, a set V_s is created for all the nodes with a degree unequal to two. The deployment process starts from any node v_r in V_s. Then, an anchor point is deployed at v_r and added to $\mathbb{P}$. Afterward, we check all the neighboring edges of v_r to see if any of them has not been visited yet. If there is any unvisited edge, exploration is executed from v_r along the edge, until a node in V_s is reached. We split the explored edge into several segments of length λ (the last segment could be shorter than λ), deploy an anchor point every λ distance, add the anchor points into $\mathbb{P}$, and mark the edge as visited. For other nodes in V_s, the process repeats for exploring unvisited edges. Take Figure 1.1(a) as an example. We first establish anchor points at n_1, n_2, n_3, and n_6, since their degree is greater than 2. Suppose λ is 14. For any segment longer than λ, anchor points are deployed every λ distance along the road segment.

1.3.2 Watchtowers Construction

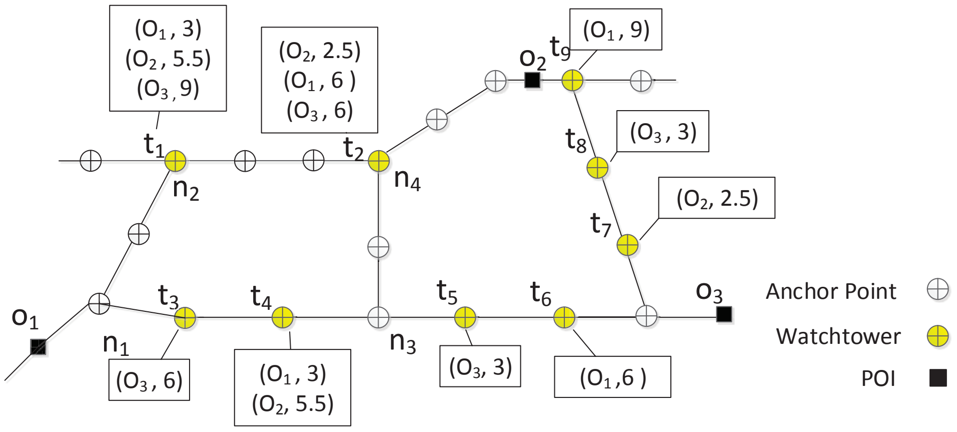
The naive way to set up watchtowers is to treat each anchor point, which we obtained in the above subsection, as a watchtower to store distance information of POIs. However, this design is neither practical nor efficient because it requires a prohibitively huge amount of storage space.

Algorithm 1: Anchor point deployment

```
     Input:  A graph $G = (V, E)$, a distance parameter $\lambda$
     Output: The anchor point set $\mathbb{P}$ for $G$
1.1  foreach $e \in E$ do $e$.visited ← FALSE ;
1.2  $\mathbb{P} \leftarrow \emptyset$, $V_s \leftarrow \emptyset$ ;
1.3  foreach $v$ with a degree unequal to 2 do $V_s \leftarrow v$;
1.4  foreach $v_r \in V_s$ do
1.5      $\mathbb{P} \leftarrow v_r$ ;
1.6      foreach edge $(v_r, v_e)$ do
1.7          if $(v_r, v_e)$.visited == TRUE then continue;
1.8          $e \leftarrow (v_r, v_e)$ ;
             /* set up one anchor point $p$ every $\lambda$ distance along $e$
                */
1.9          repeat
1.10             deploy an anchor point $p$ every $\lambda$ on $e$, $\mathbb{P} \leftarrow p$ ;
1.11             $e$.visited ← TRUE ;
1.12             $e \leftarrow (v_e, v_j)$ ;
1.13             $v_e \leftarrow v_j$ ;
1.14         until $v_e \in V_s$;
```



(a) Anchor point deployment.



(b) Watchtower construction.

Figure 1.1: Anchor point deployment and watchtower construction on example road networks.

Here, we propose a parameter-tunable approach to establish watchtowers. Specifically, for every l adjacent anchor points, we set up a watchtower.

The following describes in detail how we distribute watchtowers. Given a POI o_i, a Dijkstra-based expansion from u is launched. The expansion searches the whole graph and sets up a watchtower for every l anchor points. When an anchor point is selected as watchtower t_j for o_i, a distance tuple $(o_i, |SP(o_i, t_j)|)$ is added to this watchtower, where o_i is the POI's ID and $|SP(o_i, t_j)|$ is the shortest distance from o_i to watchtower t_j. We repeat the above expansion process for every POI until we set up watchtowers for all POIs over the entire graph. Note that there could be many such distance tuples inserted into a watchtower because an anchor point might be selected as a watchtower by more than one POI. If more than one tuple is added to a watchtower (i.e., this watchtower is shared by more than one POI), we sort all the distance tuples by their distance $|SP(o_i, t_j)|$ in ascending order. Take Figure 1.1(b) as an example. The distance between two adjacent anchor points is 1 and l is set to 3, i.e., only one anchor point serves as the watchtower for every three adjacent anchor points. As a result, the distance information for o_1, o_2, and o_3 are distributed along the network. In watchtower t_2, three distance tuples are stored, indicating that the distances from t_2 to o_1, o_2, and o_3 are 6, 2.5, and 6, respectively.

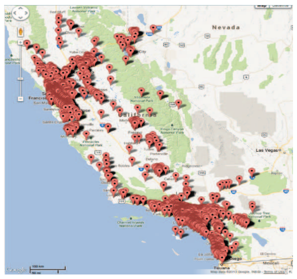
To save the storage cost, we actually do not need to store all distance tuples because there usually exists an upper bound of k, for example b_k, for kNN queries. If $b_k = 10$, only the top-10 distance tuples are necessary for storage in each watchtower because those tuples are sufficient to answer the query. Therefore, the storage complexity of our watchtower-based approach can be estimated as $O(b_k \frac{\sum w_e}{\lambda})$, where $\sum w_e$ is the sum of the distances of all the edges in the road network.

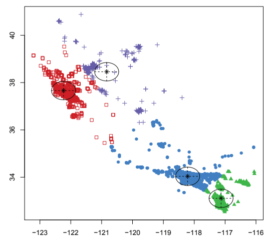
1.4 The Hot Zone-based Watchtower Framework

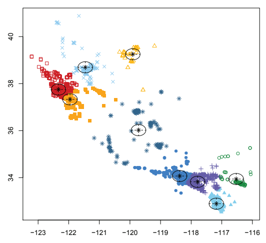
Many papers on social networks, such as [14, 54], reveal the fact of "Location Sparsity", which means that most users who subscribe location-based services move only within *limited areas*, i.e., people's movement often exhibit a strong spatial pattern. This idea inspires us to refine the Uniform Watchtower (UW) framework into the Hot zone-based Watchtower (HW) framework. In this design, we first collect users' movement data from a popular geo-social service, Gowalla, and then cluster those data to derive hot zones of people's movements (i.e., the geographic areas where people are most likely to appear and launch spatial queries). The clustering process yields a number of hot zones. Afterwards, based on the hot zones, we build watchtowers discriminatorily for hot zones and non-hot zones over road networks.

1.4.1 Probabilistic Clustering on Social Data

People's movement information can be obtained from popular geo-social services, e.g., Gowalla and FourSquare. Here, we elaborate on how to apply Model-based Clustering (Mclust) [29, 28]


(a) Check-ins


(b) $K^+ = 5$


(c) $K^+ = 10$

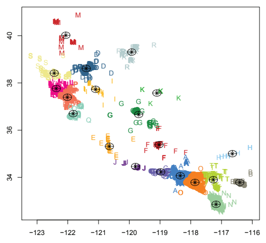

(d) $K^+ = 20$

Figure 1.2: Mclust with various K^+ values.

to analyze the structure of user check-in data to identify hot zones. We do not adopt the k-means (where k stands for the number of clusters) algorithm because we have no prior knowledge of the number of clusters. We employ Mclust, where clustering models are compared using an approximation to the Bayes factor based on the Bayesian Information Criterion [73] for selection while expectation-maximization (EM) algorithm [21] is used to maximize likelihood in the presence of incomplete data. Mclust can also be considered as an efficient algorithm to estimate k, the number of clusters. Given the upper bound of the number of the clusters, denoted as K^+, Mclust iteratively decides whether a cluster should be split further into smaller clusters by calculating the Bayesian Information Criterion. The details of this splitting process are illustrated in [29]. When Mclust runs, k increases gradually until it is steady. The steady k value is then returned and reported as the number of clusters found by Mclust.

In our approach, the check-in information is modeled as a mixture of 2-dimensional Gaussian distributions $\mathcal{N}(\mu_i, \sum_i)$, where $\mu_i = (\mu_i.x, \mu_i.y)$ is the center of the Gaussian distribution and $\sum_i$ is the covariance matrix. Since there is no clear evidence for "non-symmetric" distributions on X and Y dimensions in geo-social networks [54], $\sum_i$ can be assumed to be diagonal and isotropic. The probabilistic density function of a cluster $C_i(\mu_i, \sigma_i)$, returned by Mclust, for a particular location L can be represented by Equation 1.1.

$$f(L.x, L.y) = \frac{1}{2\pi\sigma_i^2} e^{\frac{(\mu_i.x - L.x)^2}{-2\sigma_i^2} + \frac{(\mu_i.y - L.y)^2}{-2\sigma_i^2}} \tag{1.1}$$

We cluster users' check-in locations using Mclust to determine the hot zones for a road network. We treat each cluster as a hot zone. Specifically, a hot zone is the region centering at the point μ_i for the cluster C_i with a fixed population density parameter p_d, which is able to compute a radius for different cluster C_i regarding different σ_i.

Figure 1.2 displays the clustering results of the check-ins for California road networks (where 8,523 check-ins were sampled from Gowalla[1]). In Figure 1.2, we varied K^+ from 5, 10, to 20, leading to distinct numbers of resulting clusters. Each cluster, highlighted with a distinct color, corresponds to a hot zone.

1.4.2 Hot Zone-based Watchtower Construction

To build a hot zone-based watchtower framework, we deploy all watchtowers in such a manner that watchtowers in hot zones are constructed densely to speed up the query processing while watchtowers in non-hot zones are distributed sparsely to save storage space. This design leads to a significantly higher query performance without the need of more storage space. Here, we illustrate the difference between the uniform watchtower framework and the hot zone-based watchtower framework. In the former, for each POI, we deploy watchtowers uniformly on the road network. In the latter, we analyze and leverage social data with the objective of allocating watchtowers to more important (i.e., populated) areas where queries are more likely to be launched.

The index construction process is as follows. For each cluster C_i, we obtain the node subset V_{ci} ($V_{ci} \subset V$), where for any node v in V_{ci}, the network distance between μ_i and v must be less than or equal to r_d. As a result, we obtain a graph $G^{'} = (V^{'}, E^{'})$, where $V^{'}$ is the union of V_{ci} for all clusters, and $E^{'}$ is the subset of E consisting of all edges whose two end points are both in $V^{'}$. Next, we launch a Dijkstra-based expansion from each POI and check if the currently visited edge e is in $G^{'}$. If e is in $G^{'}$, we set up watchtowers on e with higher density (by using a relatively small l). In addition, if e is not in $G^{'}$, we deploy watchtowers on e sparsely (by applying a relatively large l).

[1]http://snap.stanford.edu/data/loc-gowalla.html

The boundary of each cluster is determined by the population density parameter p_d. Given a p_d and a cluster C_i, we check whether a location v, ($v.x$, $v.y$), is in C_i by applying the equation

$$P_{C_i}(v.x, v.y) = \frac{1}{2\pi\sigma_i^2} e^{\frac{(v.x-\mu_i.x)^2}{-2\sigma_i^2} + \frac{(v.y-\mu_i.y)^2}{-2\sigma_i^2}} \geq p_d \tag{1.2}$$

v is in cluster C_i, centered at ($\mu_i.x$, $\mu_i.y$), if and only if $P_{C_i}(v.x, v.y) \geq p_d$. In other words, the population density value of a location for a cluster C_i must be greater than or equal to p_d for that location to be in C_i. An alternative way to determine the boundary of each cluster is to use a pre-defined network distance threshold r_d.

1.5 Spatial Query Processing

In this section, we elaborate on how spatial queries can be processed in our watchtower-based frameworks. To facilitate our discussion, we define a search operator, $Dijkstra(u, w_e)$, which represents all the watchtowers encountered by the Dijkstra search [22] in the range of distance w_e from the node u on a road network.

1.5.1 kNN query

Evaluation of kNN queries on road networks has drawn a lot of interest [46, 12, 71] during the past decade. However, most of the approaches are not scalable to large-scale road networks. In this subsection, we focus on how kNN queries can be processed in our watchtower-based frameworks.

Given a query point q, the search $Dijkstra(q, l\lambda)$ will return one watchtower for every POI o_i that must be in the shortest path from q to o_i, represented as $SP(o_i, q)$. The proof is as follows. Because the distance between two adjacent anchor points is at most λ, the distance of two adjacent watchtowers must be smaller than or equal to $l\lambda$ (recall that we set up watchtowers every l anchor points). For any POI o_i, $Dijkstra(q, l\lambda)$ is able to return at least one distance tuple for o_i, which must be in the shortest path from q to o_i, denoted as $SP(o_i, q)$.

The evaluation of kNN queries is based on the bidirectional Dijkstra's algorithm. At each watchtower, backward search results (POI distance information) are stored. A priority queue Q_o of size k is maintained as well. Because $Dijkstra(q, l\lambda)$ will return *at least two* watchtowers for any POI, we calculate the top k tuples for each watchtower and decide if Q_o needs to be updated. The updating rule is based on computing the shortest distance from q to o_i, which can be represented by:

$$\begin{aligned} |SP(o_i, q)| &= min\{|PSP(o_i, q)|\} \\ &= min\{|SP(o_i, t)| + |SP(t, q)| | t \in Dijkstra(q, l\lambda)\} \end{aligned} \tag{1.3}$$

In Equation 1.3, $|SP(o_i, t)|$ denotes the distance between o_i and t stored for the o_i tuple in t, and $|SP(t, q)|$ represents the distance from the query point q to watchtower t. The sum of the two items corresponds to a possible path $PSP(o_i, q)$ between the POI o_i and q while $|PSP(o_i, q)|$ is their network distance. Assume that from some watchtower t, we retrieve the top k POI tuples and one of them is o_i. Now we need to decide if the tuple of o_i needs to be inserted into Q_o. First, we check to see if $|PSP(o_i, q)|$ is less than the maximum distance in Q_o. If it is true, we add o_i into Q_o in the case that o_i is not in Q_o, and update the shortest distance accordingly. We update Q_o for every POI tuples in t until the search is expanded to $l\lambda$ distance. Since the search $Dijkstra(q, l\lambda)$ is able to encounter at least two watchtowers for any POI from any query point, and one of them must be in $SP(o_i, q)$, we can calculate the distance $|SP(o_i, q)|$ for all POIs. In other words, once the search expands to the distance of $l\lambda$, it terminates. The top k POIs in Q_o are the result of the kNN query.

It is worth noting that some POIs might be within the distance of $l\lambda$ from the query point q. In this case, the Dijkstra search actually might not have to expand as far as $l\lambda$ to answer a kNN query. Therefore, for those POIs, we also insert their distance information into the priority queue Q_o during the search. The search stops once there is enough POI distance information to answer the query. This leads to the early termination of the search.

Algorithm 2: kNN Query Processing

```
    Input: Graph $G = (V, E)$, watchtower set $WT$, $k$, $q$, $l$, $\lambda$
    Output: $k$ POIs
2.1  foreach vertex $u \in V$ do
2.2      $dist[u] \leftarrow infinity$ ;
2.3      $visited[u] \leftarrow false$ ;
2.4      $previous[u] \leftarrow undefined$ ;
2.5  A priority queue $Q_v \leftarrow \emptyset$ ;
2.6  A priority queue of size $k$, $Q_o \leftarrow \emptyset$ ;
2.7  $dist[q] \leftarrow 0$ ;
2.8  $Q_v \leftarrow Q_v \cup (q, dist[q])$ ;
2.9  $numObj \leftarrow 0$ ;
2.10 while $Q_v$ is not empty do
2.11     $u \leftarrow Q_v.remove()$ ;
2.12     if $visited[u] == TRUE$ then continue;
2.13     if $dist[u] > l\lambda$ then break;
2.14     DijkstraExpansion(G,u,visited,dist,previous,$Q_v$) ;
2.15     if $u == q$ then continue;
2.16     foreach watchtower t in $(u, previous[u])$ do
2.17         for top k objects $o_i$ in t do
2.18             $|PSP(o_i, q)| \leftarrow |SP(o_i, t)| + |SP(t, q)|$ ;
2.19             if $|PSP(o_i, q)| <$ the largest distance in $Q_o$ then
2.20                 $Q_o \leftarrow Q_o \cup (o_i, |PSP(o_i, q)|)$ ;
2.21     foreach object $o_i$ in $(u, previous[u])$ do
2.22         $|PSP(o_i, q)| \leftarrow dist[previous[u]] + w(o_i, previous[u])$ ;
2.23         $Q_o \leftarrow Q_o \cup (o_i, |PSP(o_i, q)|)$ ;
2.24         $numObj++$ ;
2.25     if $numObj \geq k$ then break;
2.26 return $Q_o$;
```

The kNN query evaluation is implemented based on the Dijkstra search, as shown in Algorithm 2. The input of the algorithm is the watchtower set WT, query point q, k, and distance threshold $l\lambda$. Lines 2.1 to 2.9 are the initialization for query process. Dijkstra search finds the shortest path by exploring the nodes in ascending order of distance away from the source. Here $dist[u]$ denotes the network distance from q to u. At any time, u is chosen from a priority queue Q_v. $visited[u]$ tracks if u has been visited or not while $previous[u]$ stores u's adjacent node. Lines 2.10 to 2.25 correspond to the Dijkstra-based search. When the search is expanded, we update

$dist[]$, $visited[]$, and $previous[]$. For an edge $(u, previous[u])$, we fetch all watchtowers in the edge and two end points u and $previous[u]$. For every watchtower t on the edge $(u, previous[u])$, top-k POI distance tuples are retrieved. For every found POI o_i , the possible shortest path distance $|PSP(o_i, q)|$ is computed. If $|PSP(o_i, q)|$ is shorter than the largest distance in Q_o, o_i is inserted to Q_o. The search stops once (1) k POIs have been found (early termination) or (2) the Dijkstra search has expanded to the distance of $l\lambda$ from the query point (the worst case).

To answer a kNN query, in the worst case, $Dijkstra(q, l\lambda)$ is needed. The time complexity includes the Dijkstra search cost and the priority queue update cost. $O(VlogV)$ is the complexity of the Dijkstra search while the maximum number of watchtowers encountered by $Dijkstra(q, l\lambda)$ can be estimated as $\frac{\sum w'}{\lambda}$, where $\sum w'$ is the sum of the distances of all the edges traversed by $Dijkstra(q, l\lambda)$. If we keep only the top b_k POI distance tuples in each watchtower, $O(\frac{b_k \sum w'}{\lambda})$ is the cost of the priority queue update. Therefore, the time complexity of our proposed algorithm is $O(V'logV' + \frac{\sum w'}{\lambda} b_k)$, where V' is the number of the vertices traversed by $Dijkstra(q, l\lambda)$. Notice that thanks to the employment of watchtowers, any kNN query can be answered by searching a distance of $l\lambda$ in the worse case.

1.5.2 Range Query

Given a distance e and query point q, a range query retrieves all the POIs that are within network distance e from q. For any encountered watchtower t, we compute the network distance of each POI o_i and query point q, $|PSP(o_i, q)|$. If the distance is not larger than e, we insert o_i into a priority queue Q_o. If the search expands to a POI, the POI is also inserted into Q_o. This process is repeated until a distance of $min(e, l\lambda)$ has been searched from q. The POIs in the final Q_o constitute the result for the range query. The details are as shown in Algorithm 3.

1.6 Performance Comparison of UW and HW

In this section, we discuss the superiority of HW over UW in terms of query performance. Without loss of generality, we assume the road network G is a 2D Manhattan network in a square

Algorithm 3: Range Query Processing

```
    Input: Graph $G = (V, E)$, watchtower set $WT$, $q$, distance range $e$
    Output: POI set O
3.1 foreach vertex $u \in V$ do
3.2     $dist[u] \leftarrow infinity$ ;
3.3     $visited[u] \leftarrow false$ ;
3.4     $previous[u] \leftarrow undefined$ ;
3.5 A priority queue $Q_v \leftarrow \emptyset$ ;
3.6 A priority queue $Q_o \leftarrow \emptyset$ ;
3.7 $dist[q] \leftarrow 0$ ;
3.8 $Q_v \leftarrow Q_v \cup (q, dist[q])$ ;
3.9 while $Q_v$ is not empty do
3.10    $u \leftarrow Q_v.remove()$ ;
3.11    if $visited[u] ==$ TRUE then continue;
3.12    if $dist[u] > min(e, l\lambda)$ then break;
3.13    DijkstraExpansion(G,u,visited,dist,previous,$Q_v$) ;
3.14    if $u == q$ then continue;
3.15    foreach watchtower $t$ in $(u, previous[u])$ do
3.16        foreach object $o_i$ in $t$ do
3.17            $|PSP(o_i, q)| \leftarrow |SP(o_i, t)| + |SP(t, q)|$ ;
3.18            if $|PSP(o_i, q)| \leq e$ then
3.19                $Q_o \leftarrow Q_o \cup (o_i, |PSP(o_i, q)|)$ ;
3.20    foreach object $o_i$ in $(u, previous[u])$ do
3.21        $|PSP(o_i, q)| \leftarrow dist[previous[u]] + w(o_i, previous[u])$ ;
3.22        if $|PSP(o_i, q)| \leq e$ then $Q_o \leftarrow Q_o \cup (o_i, |PSP(o_i, q)|)$ ;
3.23 return $Q_o$;
```

area consisting of only horizontal and vertical edges. The same assumption is also made in [33] [46].

Suppose that l_* is the selecting parameter in the UW framework while l_h and l_{nh} ($l_h < l_{nh}$) represent the selecting parameters in hot zones and non-hot zones in the HW framework. For a kNN query, if the query point is in a hot zone, in the worst case, we have to search $l_h\lambda$ to answer the query. On the contrary, if the query point is outside any hot zone, in the worst case, the search for $l_{nh}\lambda$ distance is needed to get the result. Let W_G be the sum of the weights of all the edges in G. Assuming that α is the ratio of W_G to the sum of the weights of edges in all hot zones, $\frac{W_G}{\alpha}$ is the sum of the weights of edges in all hot zones while $W_G - \frac{W_G}{\alpha}$ is the sum of the weights of edges in all non-hot zones. Because the index size is proportional to the number of watchtowers, assuming that UW and HW occupy the same amount of storage space, we can obtain Equation 1.4

$$\begin{cases} \frac{W_G}{l_*} = \frac{\frac{W_G}{\alpha}}{l_h} + \frac{W_G - \frac{W_G}{\alpha}}{l_{nh}} \\ l_{nh} = \beta l_h \end{cases} \tag{1.4}$$

where β is the ratio of l_{nh} to l_h. Based on Equation 1.4, the ratio of l_* to l_h, denoted as H, can be calculated as Equation 1.5.

$$H = \frac{l_*}{l_h} = \frac{\alpha\beta}{\alpha + \beta - 1} \tag{1.5}$$

Recall that the time complexity of our watchtower-based query processing algorithm is $O(V' logV' + \frac{\sum w'}{\lambda} b_k)$, where V' is the number of the vertices traversed by $Dijkstra(q, l\lambda)$. Therefore, to answer a query launched at point q, in the worst case UW requires $Dijkstra(q, l_*\lambda)$ while HW needs $Dijkstra(q, l_h\lambda)$. Therefore, if the query point is inside a hot zone, HW is able to reduce the query response time by a factor of H^2, compared with UW. According to Equation 1.5, we can approximate H^2 using Equation 1.6.

$$\begin{aligned} speedup &= (\frac{\alpha\beta}{\alpha + \beta - 1})^2 > (\frac{\alpha\beta}{\alpha + \beta - 1 + |\beta - \alpha| + 1})^2 \\ &= \frac{(\alpha\beta)^2}{4(MAX\{\alpha, \beta\})^2} = \frac{(MIN\{\alpha, \beta\})^2}{4} \end{aligned} \tag{1.6}$$

1.7 Pre-Computation Overhead versus Query Efficiency

In this section, we are investigating a problem: can we deploy the watchtowers of a POI only in its spatial proximity without much sacrifice of the query efficiency? If the answer is yes, how should we deploy watchtowers so that an optimal or near-optimal tradeoff between the pre-computation cost (index setup time or the storage cost) and the query efficiency is achieved?

We observed that in most cases, the POIs, which are significantly far away from the query point, are quite unlikely to be queried because users are more likely to exhibit interest in POIs closer to their current locations. For example, a user living in Los Angeles may show much less interest in a new cafeteria which opened in New York than the ones in Los Angeles. This fact inspires us to extend both our UW and HW indices by introducing another tuning parameter, the distance bound b_d. The distance bound b_d restricts the distribution of watchtowers for a POI o_i to only those anchor points within a network distance of b_d to o_i (here b_d should be no less than $l\lambda$ so that at least one watchtower can be established for o_i). The use of this distance bound aims to achieve a more desirable balance between storage cost (index size) and query efficiency. Another advantage of using the distance bound is that it can reduce the index construction time significantly.

Assume the road network G is a 2D Manhattan network in a square area consisting of only horizontal and vertical edges. If POIs are uniformly distributed on the road networks, to answer a kNN query, the Dijkstra search terminates once the following condition is met:

$$W_r/W_A = k/|O| \tag{1.7}$$

where W_r is the total weights of all edges in the searched area, W_A is the total weights of all edges in G, k is the number of POIs queried, and $|O|$ is total number of POIs. However, with the deployment of the watchtowers, if we assume that the POI is at a certain anchor point, with a network expansion of distance s from the query point q, we can discover all the POIs whose shortest distances to the query point are less than or equal to $s + \lfloor \frac{b_d}{l\lambda} \rfloor * l\lambda$. Given the search distance l_e, the weight sum of all the searched edges can be represented as $4l_e^2$. Therefore, based on Equation 1.7, if b_d is divisible by $l\lambda$, we can derive Equation 1.8.

$$4(s + b_d)^2/W_A = k/|O| \tag{1.8}$$

When s is 0, we can derive the optimal value of b_d for kNN queries, $OPT^n(b_d)$, as represented in Equation 1.9.

$$OPT^n(b_d) = \sqrt{\frac{kW_A}{4|O|}} \tag{1.9}$$

Similarly, to answer a range query with query distance e specified, the network expansion from the query point can stop when $s + b_d = e + l\lambda$ is met by using watchtowers. When s is 0, we can compute the optimal value of b_d for range queries, $OPT^r(b_d)$, using Equation 1.10.

$$OPT^r(b_d) = e + l\lambda \tag{1.10}$$

Notice that according to Section 5, when we deploy watchtowers along the entire road network, we need to search the distance of $l\lambda$ from the query point to answer a kNN or a range query in the worst case. However, when we establish watchtowers only in the area within the distance b_d from each POI, the search should stop once the priority queue Q_o has k POIs for a kNN query or terminate once Equation 1.11 is met for a range query.

$$s = \begin{cases} MIN\{e, e - b_d + l\lambda\}, & \text{if } e > b_d. \\ MIN\{e, l\lambda\}, & \text{if } e \leq b_d. \end{cases} \tag{1.11}$$

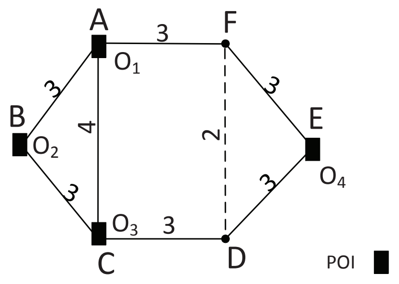
1.8 Index Maintenance

In this section, we focus on the maintenance of the our proposed frameworks with regard to (1) network update and (2) POI update.

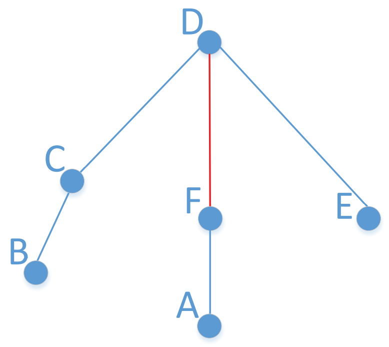
1.8.1 Network Update

The addition/removal of an edge causes the network topology to be changed. For each affected POI, we update its watchtowers. The affected POIs can be efficiently discovered by launching two shortest-path tree searches.

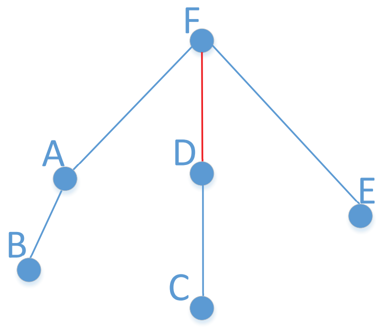
Addition of a new edge. If an edge (v, v') is added, We first retrieve two shortest-path trees (SPTs), T_v and $T_{v'}$ from v and v', respectively. For a POI o which is affected by the addition of edge (v, v'), its shortest path to v must cover (v, v') or its shortest path to v' must cover (v, v'). Take Figure 1.3 as an instance where black squares denote POIs and dashed line represents the newly added edge. According to the shortest path trees from F and D, the addition of edge (D, F)



(a) Graph G



(b) SPT from D



(c) SPT from F

Figure 1.3: Network update example.

affects only POIs o_1 at node A and o_3 at node C while POIs o_2 at node B and o_4 at node E will not be affected. Notice that the shortest-path trees are expanded only up to the distance of b_d because the watchtowers are deployed only within the range of b_d from each POI. Next we set up anchor points on edge (v, v') if v or v' does not have anchor points and re-distribute watchtowers for each affected POI.

Removal of an edge. If an edge (v, v') is removed, we initialize two shortest-path tree searches from v and v' up to the distance of b_d to identify affected POIs. For each affected POI, we re-distribute its watchtowers to reflect the network topology change.

Parameters	Default Values
λ	1 mile
k (kNN)	10
e (query range)	500 miles for NA, 60 miles for CA, and 20 miles for SF
Number of POIs ($\|O\|$)	1000
l	266 for CA, 358 for NA, and 107 for SF
l_h/l_{nh}	21/420 for CA, 44/594 for NA, and 15/298 for SF
K^+ (cluster)	50 for NA, 10 for CA, and 5 for SF
r_d (hot zone)	50 miles for NA, 10 miles for CA, and 3 miles for SF
b_d (Islands, UW, and HW)	1252 miles for NA, 105 miles for CA, and 9.2 miles for SF

Table 1.3: Experimental parameter values.

1.8.2 POI Update

For a new POI o, we establish its watchtowers in the range of b_d based on our discussion in Sections 3 and 4 (initialize a shortest path expansion from o and set up its watchtowers based on the selecting parameter l). On the other hand, to delete a POI, we also remove all its distance tuples in according watchtowers.

1.9 Experimental Validation

In this section, we compared the performance of UW and HW with that of ROAD [45] and Islands [36] using real road networks and geo-social data. We implemented UW and HW in C++, and all the experiments were conducted on an Ubuntu Linux server with an Intel Xeon 2.67GHz processor and 24GB memory, with a disk of 4KB page size. Three real road networks[2], namely CA, NA, and SF, were used [51], including road networks in the state of California (21,692 edges and 21,047 nodes), in North America (179,178 edges and 175,812 nodes), and in San Francisco (223,001 edges and 174,956 nodes), respectively. In addition, we collected the geo-social data

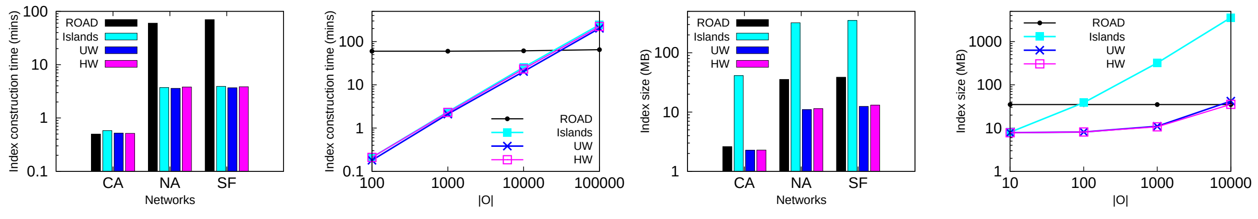
[2]http://www.cs.utah.edu/ lifeifei/SpatialDataset.htm

from Gowalla [15], consisting of 196,591 users and 6,442,890 check-ins around the world. We assumed that POIs were distributed uniformly on the networks and the query points were randomly generated from people's check-in locations. Each experimental result was averaged over 100 random queries. Table 1.3 lists important experimental parameters and their default values, where unless otherwise specified, their default values were used.

As listed in Table 1.3, the default for λ is 1 mile and the number of POIs $|O|$ is 1000. The default of the selecting parameter l for UW is 266 on CA, 358 on NA, and 107 on SF. We set the distance bound b_d as one fifth of D_{max}, where D_{max} denotes the maximum distance of two nodes in each referred network. The defaults of the selecting parameters l_h and l_{nh} for HW are 21 and 420 on CA, 44 and 594 on NA, and 15 and 298 on SF. The default of K^+ for HW is 5 on CA, 20 on NA, and 5 on SF, respectively.

1.9.1 Index Construction Overhead

The first experiment evaluates the index setup time and size. In ROAD [45], the number of levels of Rnet is set to 4 for CA and 8 for NA and SF while the partition factor is 4. Besides, the radius (b_d) in the Islands approach took the same value as in UW and HW. We first measured the index setup time on various networks. As Figure 1.4(a) shows, for large (or dense) road networks (like NA and SF), ROAD required a much longer construction time than HW, UW, and Islands. The reason is that the Rnet structure is based on the hierarchy computation over the entire network while our watchtower-based designs only store distance information in spatial proximity. Figure 1.4(b) shows the impact of the number of POIs ($|O|$) on index construction time on NA. The construction



(a) Various networks (b) Various $|O|$ on NA (c) Various networks (d) Various $|O|$ on NA

Figure 1.4: Index construction time and index size.

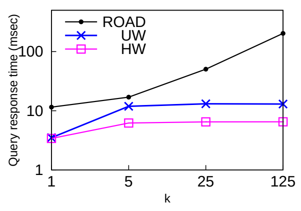
time required by ROAD was constant while that of UW, HW, and Islands expanded almost linearly with the increase of $|O|$. This is because the creation of the ROAD index is purely based on the network topology while UW, HW, and Islands establish distance information for each distinct POI. Specifically, when $|O|$ was 10,000, the construction time for ROAD was about 60 minutes while either UW or HW had a quicker setup time (about 23 minutes).

Figure 1.4(c) reveals that, for large (or dense) networks, like NA or SF, UW and HW required much less storage space than ROAD and Islands (the Islands approach is the most storage-expensive one). For example, the index size of UW and HW for NA were 10.99 MB and 10.67 MB, which are much smaller than that of ROAD ($>$ 35 MB) and that of Islands ($>$ 300 MB). Figure 1.4(d) shows the impact of $|O|$ on index size on NA. As depicted in Figure 1.4(d), the index size jumped dramatically in the Islands approach when $|O|$ increased. When $|O|$ is 10,000, the index size of Islands is 3.5 GB. On the other hand, the index size of ROAD, UW and, HW is 35.0 MB, 42.0 MB, and 35.3 MB, respectively. This is because Islands establishes distance information at every intersection while our approaches are parameter-tunable, with watchtowers set up only every l anchor points. Therefore, our approaches provide a higher scalability than Islands. Because Islands does not work well for large road networks, in the following experiments, we use ROAD as the only baseline.

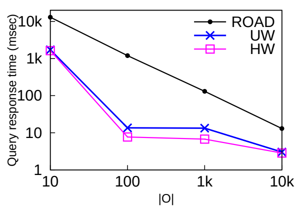
Notice that according to this experiment, ROAD is sensitive to the number of network edges while Islands is sensitive to the number of POIs. However, our UW and HW approaches are able to achieve a higher scalability towards large road networks with an affordable cost on the index setup time and size.

1.9.2 Query Efficiency of ROAD, UW, and HW

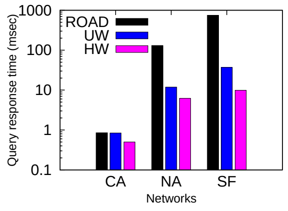
In this subsection, we evaluated the query efficiency of our proposed frameworks. First, we examined the performance of UW, HW, and ROAD in terms of kNN queries. In Figure 1.5(a), we varied the k value from 1, 5, 25 to 125. As shown in Figure 1.5(a), UW and HW were much faster than ROAD in query response time. When k is 50, the query time of UW was less than



(a) Various k on NA



(b) Various $|O|$ on NA

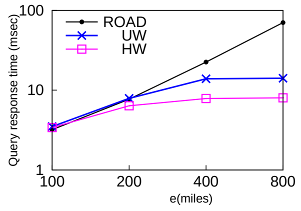


(c) Various networks

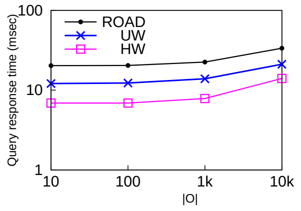
Figure 1.5: kNN Query efficiency comparison.

14 ms, more than 6 times quicker than that of ROAD. In addition, HW had a quicker query time than UW because HW allocates more watchtowers in hot zones (populated areas). Figure 1.5(b) exhibits the impact of $|O|$ on query response time. With the increase of $|O|$, the query time of all the three approaches dropped accordingly. Besides, UW and HW always required a much shorter query time than ROAD. In Figure 1.5(c), we evaluated their query time on various road networks. The query time of HW was always the fastest among the three approaches, while that of ROAD was always the slowest.

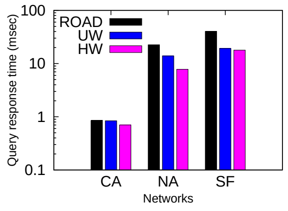
Second, we investigate the performance of UW, HW, and ROAD in terms of range queries. As depicted in Figure 1.6(a), the query time increased gradually with the increase of the query distance e. However, UW and HW always required a much shorter query time than ROAD. In Figure 1.6(b), when $|O|$ increased, the query time extended accordingly for all the approaches. Figure 1.6(c) shows that the query time of UW and HW for range queries is consistently quicker than that of ROAD on various road networks.



(a) Various e on NA



(b) Various $|O|$ on NA



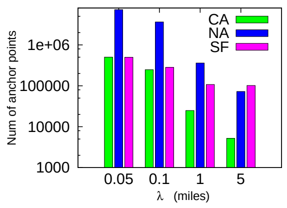
(c) Various networks

Figure 1.6: Range Query efficiency comparison.

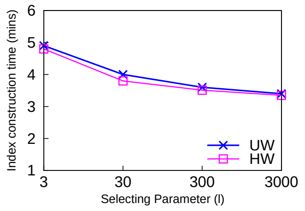
1.9.3 The Impact of λ and l on UW and HW

This experiment studies the impact of λ and l on the performance of UW and HW on NA. By varying λ from 0.05, 0.1, 1 to 5 miles, as shown in Figure 1.7(a), the number of anchor points dropped accordingly. This is because a larger λ represents a longer distance between most adjacent anchor points, leading to a sparser distribution. Specifically, the number of anchor points on NA decreased from 7.1 million to 3.6 million when λ increased from 0.05 miles to 1 mile. One interesting observation is that the number of anchor points in SF is greater than that in NA when $\lambda = 5$. This is because the majority of nodes in SF have a degree greater than 2.

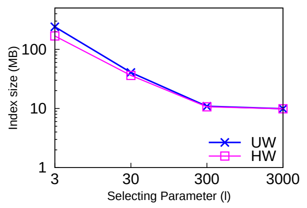
Next, we varied l (the selecting parameter) to observe its impact on index construction time, index size, and the query time for UW and HW on NA. In Figure 1.7(b), with the increase of l, the index construction time decreased gradually. This is because the increase of l leads to a smaller number of watchtowers to be built. In Figure 1.7(c), when l increased from 3 to 3,000, the size of UW index dropped dramatically from 240 MB to 9 MB. In Figure 1.7(d), the query time of UW and HW increased significantly with the enlargement of l because a larger l results in a sparser distribution of watchtowers. It can be also observed that HW is always faster than UW with respect to query response time. For example, when $l = 3000$, the query time for UW and HW was 122 ms and 25.3 ms, respectively. This is because HW deploys more watchtowers in the hot zones than UW.



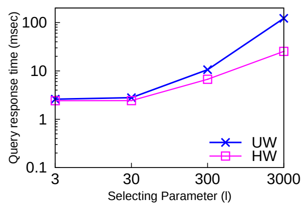
(a) The impact of λ on NA



(b) The impact of l on index construction time



(c) The impact of l on index size

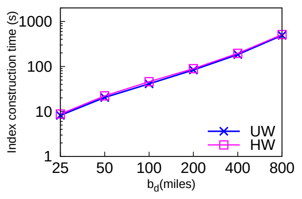


(d) The impact of l on query time

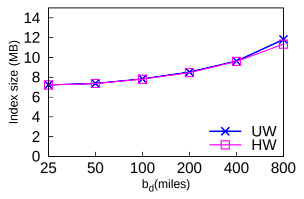
Figure 1.7: Effects of λ and l.

1.9.4 The Impact of Distance Bound b_d

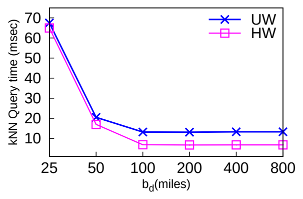
This experimental set studied the impact of b_d on the performance of UW and HW on NA. b_d increased from 25, 50, 75, 125, 250, 500, to 1000 miles. In Figure 1.8(a), with the increase of b_d, the index construction time extended. This is because the indices needed to cover more area. In Figure 1.8(b), the index size was elevated while b_d increased. In Figure 1.8(c), when we gradually raised b_d, the kNN query time dropped accordingly. However, It is worth mentioning that in Figure 1.8(c), when we increased b_d to more than 50 miles, the improvement on query time was very limited. This fact was consistent with our analysis on the optimal distance bound for kNN queries as represented in Equation 1.9. Based on Equation 1.9, the optimal distance bound can be derived as 41.89 miles, which means that setting b_d as around 40 will provide the optimal balance between index construction cost and the online query efficiency. Figure 1.8(d) shows that the query time for range queries decreased when we raised b_d. Similarly, when b_d was increased to more than 200 miles, the resulting improvement on query time was very limited, which accorded with our analysis on the optimal distance bound for range queries as represented in Equation 1.10.



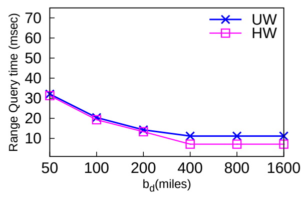
(a) The impact of b_d on index construction time



(b) The impact of b_d on index size



(c) The impact of b_d on query time for kNN queries

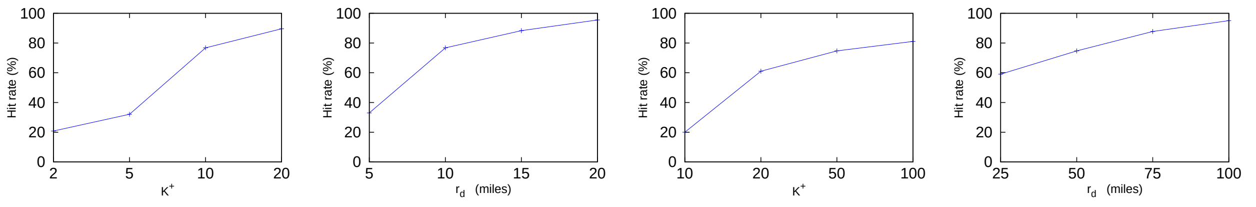


(d) The impact of b_d on query time for range queries

Figure 1.8: Effects of b_d.

1.9.5 The Impact of K^+ and r_d on HW

Next we evaluated the performance of HW on CA and NA in terms of the kNN query time and the hit rate by varying K^+ and r_d. The hit rate is defined as the probability that a user-generated query is launched from any hot zone. In Figure 1.9(a) and Figure 1.10(a), we fixed r_d as 10 miles

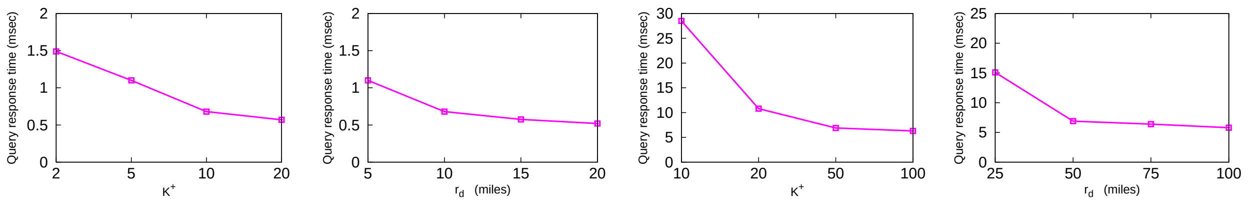


(a) The Hit Rate of K^+ change on CA (b) The Hit Rate of r_d change on CA (c) The Hit Rate of K^+ change on NA (d) The Hit Rate of r_d change on NA

Figure 1.9: Hit Rate Comparison.

and varied K^+ from 2 to 20. Figure 1.9(a) shows that the hit rate increased accordingly with the increment of K^+ while Figure 1.10(a) revealed that the query time decreases accordingly with the elevation of K^+. This is because the increase of K^+ will lead to more hot zones. In Figure 1.9(b) and Figure 1.10(b), K^+ was set to 10 and we varied r_d from 5 to 20 miles. It was observed that raising r_d resulted in an increase in hit rate and a quicker query time. This is because when r_d expands, each hot zone has a larger size and therefore more query points actually fell into the hot zones.

In Figure 1.9(c) and Figure 1.10(c), we fixed r_d as 50 miles and changed K^+ from 10 to 100. The hit rate was at its peak when K^+ was 100 and the corresponding query time was the fastest. In Figure 1.9(d) and Figure 1.10(d), K^+ was fixed to 50 and r_d varied from 25 to 100 miles. In Figure 1.9(d), the hit rate jumped when r_d increased. When r_d was raised to 100 miles, the hit rate reached almost 95%. On the other hand, as shown in Figure 1.10(d), the query response time dropped gradually from 15.1 ms to 7.0 ms when we raised r_d from 25 miles to 100 miles. The



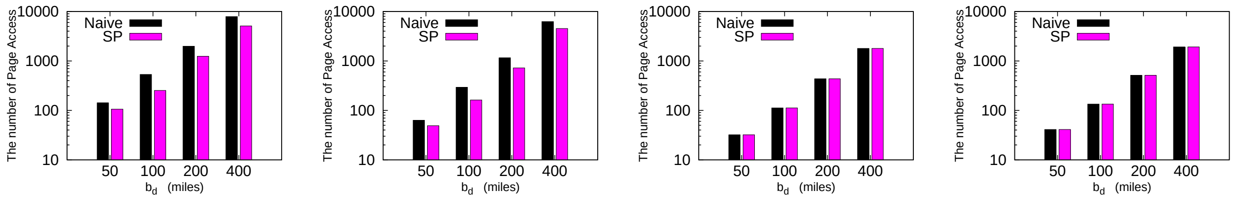
(a) The impact of K^+ on CA (b) The impact of r_d on CA (c) The impact of K^+ on NA (d) The impact of r_d on NA

Figure 1.10: Query efficiency comparison.

reason is that with a smaller size of all hot zones, the query required a longer response time to be answered.

1.9.6 Index Maintenance

In this section, we measured the page access of HW and UW regarding the index update due to the addition and removal of POIs and edges in NA. The page size was set to be 4 KB and all road nodes and segments and watchtowers were stored continuously in pages. We use SP to represent our proposed algorithm. In Figure 1.11(a), the number of required page accesses was recorded. When the distance bound b_d decreases from 800 to 100 miles, the number of required page accesses dropped down. The same trend can also be observed in Figure 1.11(b). Also, the number of required page accesses in SP was always fewer than that in the corresponding naive updating strategy which does not employ the shortest-path tree based searches proposed in Section 8. In Figures 1.11(c) and 1.11(d), we measured the number of required page accesses for adding or removing a POI. As shown in Figures 1.11(c) and 1.11(d), the decrease of b_d resulted in the fewer number of required page accesses for POI addition or removal.



(a) Edge addition on NA (b) Edge removal on NA (c) POI addition on NA (d) POI removal on NA

Figure 1.11: Index maintenance performance.

1.10 Conclusion

In this chapter, we introduce two novel parameter-tunable frameworks for efficient spatial query processing by exploiting the user check-in data collected from location-aware social networks. We first propose the uniform watchtower framework for efficient query evaluation with an affordable storage cost. Next, in order to further elevate the query efficiency, we provide the hot zone-based watchtower framework by incorporating mobile users' movement information (geo-social data) into the construction of watchtowers. Moreover, we derive optimal watchtower deployment distance in order to achieve a desired balance between the off-line pre-computation cost and the on-line query efficiency. Our experiments using various real-world road networks and geo-social data demonstrate that both of our solutions outperform the state-of-the-art approach in terms of the query efficiency. For future work, we intend to extend our frameworks to support more complex spatial query types, such as continuous kNN and continuous range queries, and investigate how other social data, such as social relationships between mobile users, can be utilized to speed up spatial query processing.

Chapter 2

LinkProbe: A Probabilistic Inference query machine on Large-Scale Social Networks

2.1 Introduction

As one of the most important Sematic Web applications, social network analysis [56, 86, 44, 78, 76] has attracted more and more interest from researchers due to the rapidly increasing availability of massive social network data for various Web 2.0 applications such as Facebook, YouTube, Flickr, and Wikipedia. In these applications, users are not only data consumers but also data producers. Friend-Of-A-Friend (FOAF) [23, 31] is an RDF/XML ontology especially designed to describe basic attributes of people and relationships among them, including name, mailbox, homepage URL, friends, interests, affiliations, etc. The friend relationships described in a FOAF data set can be depicted as a social graph, where each person is represented by a node and each friendship is denoted as an edge between two nodes as demonstrated in Figure 2.1. In this chapter, we are interested in answering the following query: given two arbitrary nodes, x and y, what is the possible probability that a specific relationship (link) exists between x and y given G as evidence? In social network analysis, evaluating the existence of links is crucial for predicting relationships among people, inferring profiles, clustering people for community discovery, criminal network detection, etc.

2.1.1 Challenges

However, to answer the aforementioned query is difficult.

- First, the desired solution should support partially correct inference rules. This is because most inference rules existing in social network analysis do not always hold in reality. For example, the transitive propriety in a friendship can be captured by the following rule:

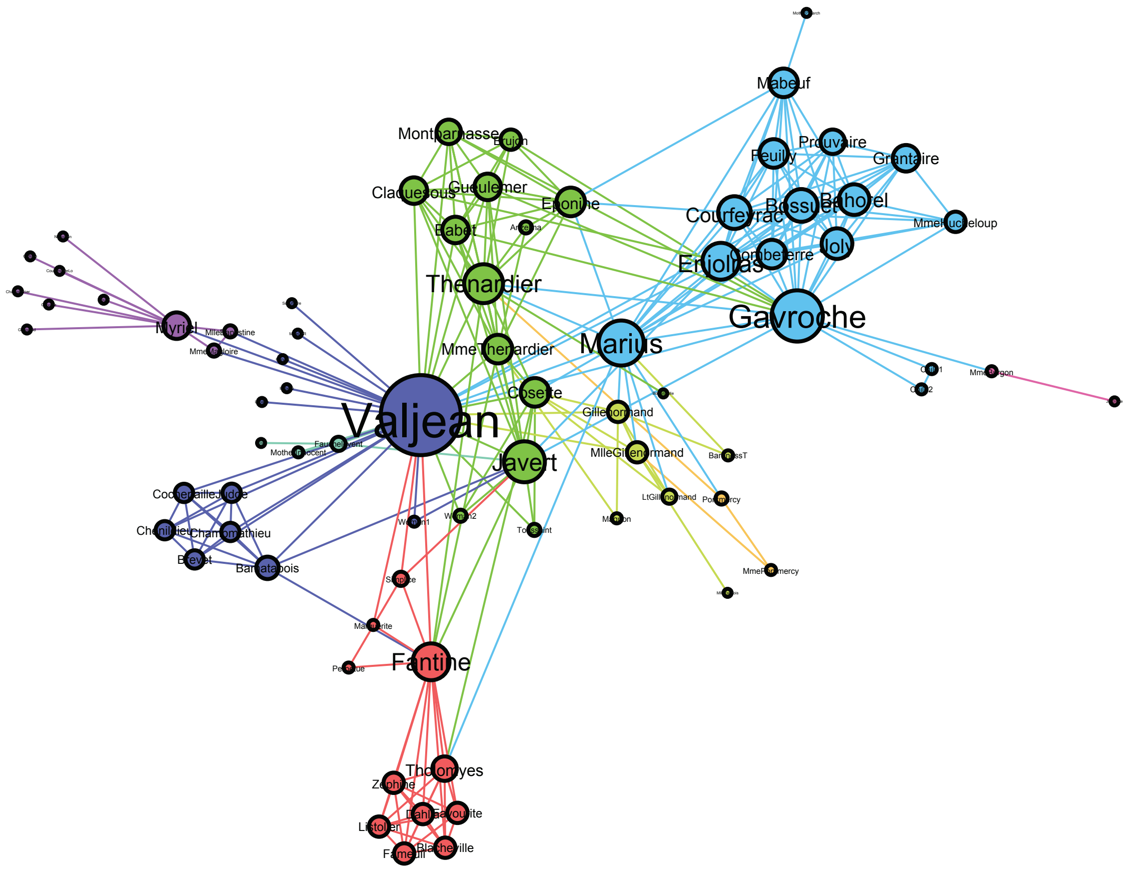
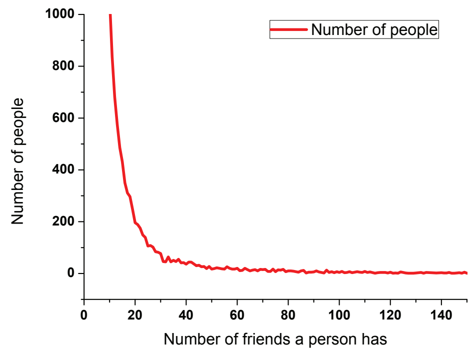
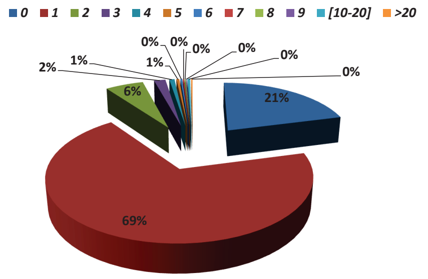


Figure 2.1: An example social graph.



(a) Power law curve.



(b) Distribution of the number of friends.

Figure 2.2: Statistics on the Billion Triple Challenge (BTC) 2009 data set.

$\forall x, y, z \in \mathrm{P} : knows(x, y) \wedge knows(y, z) \longrightarrow knows(x, z)$, where P means the set of people. However, this rule is not always true, i.e., it can be violated in some cases. Although many inference mechanisms have been proposed in the literature, most of them fail to support inference on partially correct rules, i.e., they require that underlying rules must be satisfied all the time and cannot be violated during the entire inference procedure.

- Second, the ideal solution should scale well towards very large data volume, i.e., the inference cost should be affordable and tractable when coping with massive data sets. This is because most social networks in practice often consist of a tremendous number of nodes and edges, featuring a huge amount of semi-structured data.
- Third, the desirable solution should be able to handle uncertain social data as evidence, where links between nodes are described probabilistically. In most real applications, social data (i.e., user profile and friend information) are often uploaded voluntarily by users themselves or obtained from various prediction techniques. As a consequence, impreciseness and uncertainty may arise in social data. Therefore, the desired method should support inference on such uncertain social data as well.

2.1.2 Observations

MLNs Support Partially Correct Inference Rules

As a unified inference method to combine probabilistic graph models and first-order logics, Markov Logic Networks (MLNs) [25, 26, 83] have shown their theoretical potentials in reasoning over partially correct rules [63] and modeling multi-variate structured dependency [94]. The inference on MLN is based on the inference over the resulting grounded Markov network (Markov random field) [93].

Social Graphs Tend to Be Sparse Globally and Dense Locally

Although MLNs are commonly employed to deal with partially correct inference rules, the major obstacle in applying MLNs in practice is that their grounded Markov network would be prohibitively huge. In other words, Markov logic networks become extremely inefficient in coping with large-scale data due to their highly demanding nature in terms of inference time and memory consumption [74]. This is because Markov logic networks require a complete grounding of rules in order to count the number of grounded rules (i.e., formula groundings) that are true given a particular possible world. Such counting becomes impractical when dealing with large-scale data because the number of formula groundings grows exponentially as the number of individuals in the investigated domain increases. However, unfortunately, social networks in reality usually contain a massive number of social relations, leading to the infeasibility of applying MLN directly on social networks in practice.

To tackle this issue, we observed that most social networks exhibit typical characteristics of scale-free networks. First, vertex degrees follow power law distributions. Second, vertices tend to cluster together. Third, the average length of all shortest paths between any two vertices is logarithmically small. We calculated statistics over a 1 % random sample on the Billion Triple Challenge (BTC) 2009 data set[1] with up to more than 13 million people and 35 million friend relations. Figure 2.2(a) illustrates the relationship between the number of friends (i.e., it can be interpreted as the degree of a vertex) and the number of people who have the number of friends in the BTC data set. The corresponding pie chart is demonstrated in Figure 2.2(b). Figures 2.2(a) and 2.2(b) reveal a fact that an overwhelming majority of people have a very limited number of friends while only a few number of people have a considerable number of friends (i.e., they are highly connected and can be considered as hubs in a given social graph). As depicted in Figure 2.2(b), 21% of the people do not have any friends, 69% of the people have exactly one friend, and 6% of the people have two friends, according to our sampled BTC data set.

In this dissertation, we attempt to take advantage of such characteristics to devise an MLN-driven inference engine with an affordable cost for searching massive social networks. We employ a k-backbone graph to discover the global topology of a given social network and conduct inference on both the most globally influencing nodes and the most locally related nodes.

[1]http://vmlion25.deri.ie/

2.1.3 Our Approach

This chapter presents LinkProbe, a prototype to estimate link existence in uncertain social networks based on probabilistic reasoning. LinkProbe is empowered by MLN inference but manages to provide a tunable balance between MLN inference accuracy and inference efficiency. First, in order to maintain the infrastructure of social graphs, we sort all the nodes by their degrees and construct the corresponding k-backbone graph. Then, we issue two independent runs of the random walk Metropolis sampling to explore local social graphs. Subsequently, LinkProbe applies MLN inference on the union of all the nodes in the k-backbone graph and all nodes discovered locally. In addition, in order to support uncertain/probabilistic evidence data, LinkProbe adopts MC-SAT$^+$, a probabilistic extension of the well-known MLN inference method MC-SAT.

2.1.4 Our Contributions

We summarize our contributions as follows:

- We define probabilistic social graphs by injecting uncertainty into social graphs and formalizing the link queries on them. Afterwards, LinkProbe, a prototype for predicting probabilistic links, is proposed to answer such queries.

- We propose LinkProbe, which is driven by Markov logic networks and is often utilized in handling partially correct inference rules which appear commonly in social networks.

- In order to handle large data volumes, LinkProbe takes full advantage of the fact that most social graphs tend to be sparse globally and dense locally. LinkProbe conducts the MLN inference on both the most globally influencing nodes and the most locally related nodes. Compared to the “entire graph" based naive implementation, LinkProbe reduces the time and space costs by several orders of magnitude and, in the meanwhile, maintains the inference accuracy in an acceptable level. With a much higher scalability than the naive MLN implementation, LinkProbe is applicable to large social networks for probabilistic reasoning.

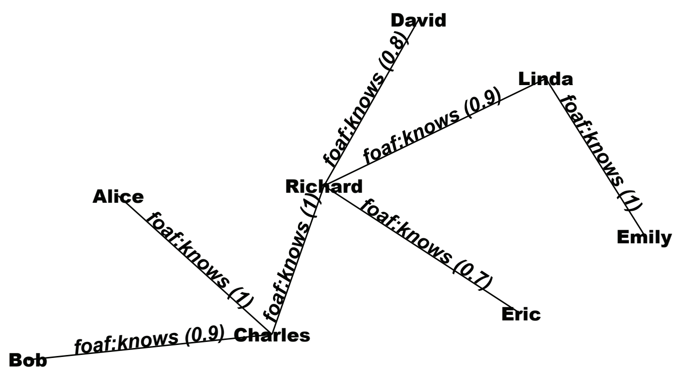


Figure 2.3: An example of a probabilistic social graph.

2.1.5 Chapter Organization

The rest of this chapter is organized as follows. The research problem is formally defined in Section 2.2. In Section 2.3 we focus on the methods for constructing the k-backbone graphs. We explain how to explore d-local graphs and random walk in Section 2.4. The basics of Markov logic networks are introduced in Section 2.5 and the inference method in LinkProbe is discussed in Section 2.6. We illustrate the error analysis in Section 2.7. The experimental validation of our system is presented in Section 2.8. Section 2.9 surveys related works. Section 2.10 concludes the chapter with future work.

2.2 Problem Formulation

2.2.1 Probabilistic Social Graphs

Definition 2.1 A *Probabilistic Social Graph* (PSG) is a social graph where each edge is associated with a probabilistic value to reflect the likelihood that a specified relation (e.g., friendship) exists among the two linked people.

Probabilistic social graphs can be stored in a database, denoted as $\mathcal{DB}$. $\mathcal{DB}$ may have the following schema: $< from, to, prob >$. Specifically, each pair of $from$ and to contains the two IDs of the vertices (i.e., involved people) associated to an edge (i.e., a friendship between two people). In addition, $prob$ denotes the weight of an edge, reflecting the probability of a friendship.

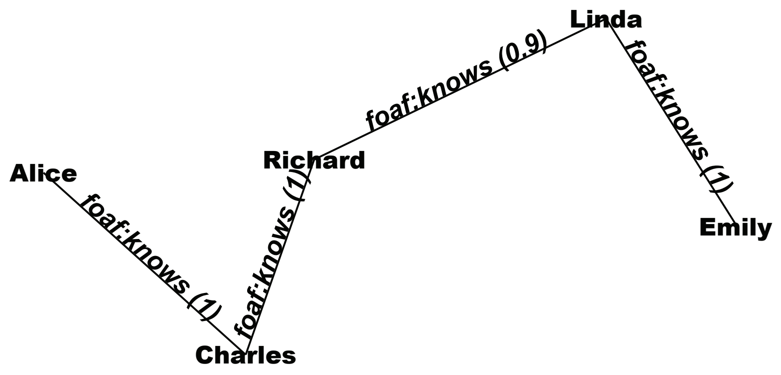


Figure 2.4: 1-backbone.

Throughout this chapter, without loss of generality, we consider such relation as a mutual friendship between people. For example, a record $< 11, 32, 0.8 >$ means that the person with ID 11 knows another person with ID 32 with a probability of 0.8.

2.2.2 Link Prediction

Definition 2.2 Given a probabilistic social graph G, the purpose of *link prediction* is to predict the probability that a specific link exists between two nodes in G. Take Figure 2.3 as an example. $Query(Bob, Emily, knows)$ may be launched to investigate how well Bob knows Emily, i.e., from the probabilistic view, to estimate the likelihood that a potential friendship edge exists from Bob to Emily.

2.3 The k-backbone Graphs

Definition 3.1 The *weighted degree* (WD) of a vertex u in a probabilistic social graph G, denoted as $WD^G(u)$, is defined to be the sum of the weights of all the edges incident to u. In other words, $WD^G(u)$ can be calculated as $WD^G(u) = \sum_{e \in \mathrm{E}(u)} W(e)$, where $\mathrm{E}(u)$ denotes the set of edges incident to u and $W(e)$ means the associated weight of edge e in G.

Algorithm 4: Retrieval of vertices in k-backbone graphs

```
Input: A probabilistic social graph $G$ stored in database $\mathcal{DB}$
Output: All the vertices in the $k$-backbone graph, output as $B$
4.1 $B = \emptyset$ ;
4.2 foreach $< from, to, prob >$ in $\mathcal{DB}$ do
4.3     $\mathcal{DB} = \mathcal{DB} \bigcup < to, from, prob >$ ;
        /* We assume that relations are undirected here */
4.4 foreach each vertex $v_i$ in $\mathcal{DB}$ do
4.5     Compute the weighted degree of $v_i$, $WD^G(v_i)$ ;
4.6     Compute the weighted out-degree of $v_i$, $WD^G{}_{out}(v_i)$, by accessing the $\mathcal{DB}$ using the
        $B^+$-tree index with $< friend\_s, friend\_t >$ as the composite key ;
4.7     if $WD^G(v_i) \geq k$ then
4.8         $B = B \bigcup v_i$ ;
```

Definition 3.2 A *k-backbone graph* of a probabilistic social graph G is a subgraph of G which can be acquired by deleting from G all the vertices with the weighted degree less than k and all the associated edges.

Figures 4 and 5 demonstrate examples of 1-backbone and 2-backbone graphs. The detailed steps of k-backbone graph construction are formalized in Algorithm 4. For each vertex v_i, we add v_i to the k-backbone graph if $WD^G(v_i)$ is no less than k.

2.4 Random Walk

In the previous section, we illustrate how to explore the skeleton of a given social graph by discovering its k-backbones. In this section, we elaborate on how to focus our exploration on a relatively small part of the graph by retrieving d-local graphs and then employing a MCMC-based

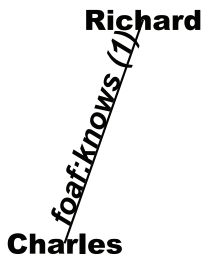


Figure 2.5: 2-backbone.

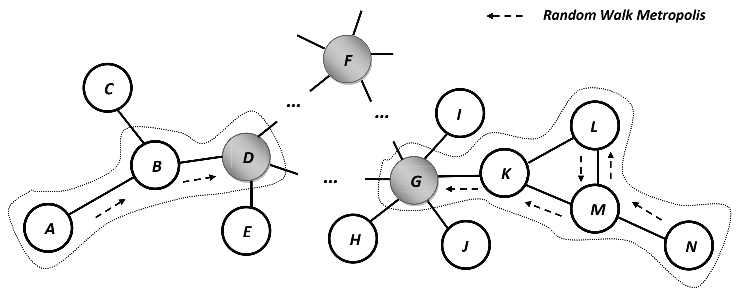


Figure 2.6: An illustration of the random walk Metropolis on a social graph.

sampler, the Random Walk Metropolis (RWM), in order to connect d-local graphs with the derived k-backbone graph.

2.4.1 d-local Graphs

Definition A *d-local graph* of a node X in a probabilistic social graph G is a subgraph of G which can be acquired by deleting from G all the neighboring nodes connected with an edge with a probability less than d and all the associated edges. In a social graph, a vertex in the *d-local graph* of a node X means that a person knows X with a probability of no less than d.

2.4.2 Markov Chain Monte Carlo

A Markov chain is a stochastic process which consists of possible states of random variables. It can be denoted as a sequence states of X_1, X_2, X_3, ..., X_n, which satisfy

$$p(X_{n+1} = x|X_n = x_n, X_{n-1} = x_{n-1}, ..., X_1 = x_1) = \\ p(X_{n+1} = x|X_n = x_n)$$

where $p(x|y)$ is the transition probability from state y to state x. Markov Chain Monte Carlo (MCMC) is a technique to generate samples from the state space by simulating a Markov chain [69, 3]. The formed Markov chain is constructed in such a way that the chain spends more time in the regions with higher importance, i.e., the stationary distribution of the Markov chain is the same as the target distribution. That is, the Markov chain can converge to the target distribution (the posterior) as its equilibrium distribution. From the perspective of Monte Carlo sampling, as the number of samples are sufficiently large, all the samples can become the fair samples from the posterior. Consequently, we are able to approximate the sophisticated target posterior based on deliberately constructing a Markov Chain of all the Monte Carlo samples.

2.4.3 Connecting Local Graphs with the Backbone Graphs

The Random Walk Metropolis (RWM) sampler is one of the most used MCMC-based samplers, which generates a sequence of random walks utilizing a proposal density and decides on whether to reject the proposed moves by employing the rejection sampling.

The random walk Metropolis algorithm simulates a Markov chain in which each state X_{t+1} only depends on the immediately previous state X_t. A new sample X' is proposed depending on the current state X_t. X' is accepted as the next state X_{t+1} with the probability of an acceptance rate α, which can be formalized as Equation 2.1. $P(X)$ is the probability of state X.

$$\alpha = P(X_{t+1} = X'|X_t) = \min\left\{1, \frac{P(X')}{P(X_t)}\right\} \tag{2.1}$$

$$P(u \longrightarrow w) = \min\left\{1, \frac{WD^G(w)}{WD^G(u)}\right\} \tag{2.2}$$

In our research problem, let u be an arbitrary vertex in a probabilistic graph G and $\mathrm{V}(u)$ denote the set of all the vertices adjacent to u in G. w is a random sample from $\mathrm{V}(u)$. If we use $WD^G(i)$ to represent the weighted degree of vertex i in G, Equation 2.1 can be rewritten as Equation 2.2, where $P(u \longrightarrow w)$ represents the probability that random walk moves from u to w. In addition, at each trial, the proposed state w for the current state u is uniformly drawn from $\mathrm{V}(u)$. By doing this, our random walk Metropolis algorithm starts from an initial vertex (usually a person whose friend relation is of interest), then moves along edges in G, and terminates once it reach a node in a k-backbone graph. Therefore, by launching the random walk Metropolis sampler, we obtain a Markov chain of random samples of nodes in G.

Here we illustrate our random walk Metropolis (RWM) algorithm by taking the graph in Figure 2.6 as an example. In Figure 2.6, grey nodes indicate the vertices in the k-backbone and dashed arrows represent a random walk by the RWM. Without loss of generality, assume that all the edges hold a weight of 1. Suppose we aim to investigate the probability that A knows N given the topology specified in Figure 2.6. We run the random walk Metropolis for A and N, respectively, to identify the set of their respective local closely related vertices, denoted as $local(A)$ and $local(N)$. At first, the RWM initiates a Markov chain with A as its starting point by adding A into $local(A)$. Afterwards, the RWM proposes a new state by sampling uniformly over all the vertices adjacent to A. Because B is the only vertex directly connected to A, B will be suggested as the next state of the Markov chain. Then RWM accepts B with a probability of $min\{1, \frac{WD^G(B)}{WD^G(A)}\} = min\{1, \frac{3}{1}\} = 1$. Thus we insert B into $local(A)$. Subsequently, the RWM verifies that the termination condition is met, i.e., the RWM terminates if the current state is a k-backbone vertex. Because B is not a k-backbone vertex, the RWM continues and a new proposal state for B will be selected from $\{A, C, D\}$. Supposing D is chosen, the RWM accepts D with a probability of $min\{1, \frac{WD^G(D}{WD^G(B)}\} = min\{1, \frac{4}{3}\} = 1$. As a result, D is added into $local(A)$. Since D is a k-backbone vertex, the RWM for A terminates with D as the ending point. Similarly, we run the RWM for N and the resulting Markov chain is $local(N) = \{N, M, L, M, K, G\}$

The detailed steps of the RWM sampling method are summarized in Algorithm 5. In Algorithm 5, we initiate a Markov chain with u as the starting point. Afterwards, the RWM proposes a new state (vertex) to move to by sampling uniformly over all the adjacent vertices. Then, the RWM accepts the proposed state with a probability of the ratio of the weighted degree. After each move, the RWM verifies that the termination condition is met. The resulting Markov chain terminates when it moves to a vertex which belongs to the k-backbone graph.

By launching the RWM sampler from a particular vertex, for instance U, we can obtain a chain of vertices (nodes) locally related to U. As far as the prediction of a specific link is concerned, for example, $Query(X, Y, knows)$, LinkProbe launches two independent runs of the RWM sampler for X and Y in order to connect their respective d-local graphs with the k-backbone graph.

Algorithm 5: The RWM sampling method (k, u, G)

```
    Output: A Markov chain of vertices in $G$: $X = X_0, X_1, ..., X_i, ...$
5.1 $X_0 = u$ ;
5.2 $L = \emptyset$ ;
5.3 $L = L \bigcup X_0$ ;
5.4 for $i$ = 0 to $max\_tries$ do
5.5     if $X_i$ is a node in the $k$-backbone graph then
5.6         Return ;
5.7     else
5.8         Obtain a random node $X'_{i+1}$ from the adjacent list of $X_i$, $\mathrm{V}(X_i)$ ;
5.9         Generate a random number between 0 and 1, $jitter$ ;
5.10        if $jitter \leq \min\left\{1, \frac{WD^G(X'_{i+1})}{WD^G(X_i)}\right\}$ then
5.11            $X_{i+1} = X'_{i+1}$ ;
5.12        else
5.13            $X_{i+1} = Xi$ ;
5.14        $L = L \bigcup X_{i+1}$ ;
5.15        $i++$ ;
```

2.4.4 Properties of the RWM Sampler

- The RWM sampler explores a probabilistic social graph locally by favoring the people who are more “influential" to the query results. First, the resulting Markov chain is ergodic so that each vertex has a certain probability to be visited. Second, the constructed Markov chain satisfies the detailed balance property. Therefore, a local vertex with a higher probabilistic degree holds a higher probability to be selected by the RWM sampler.

- The RWM algorithm is efficient in reaching the k-backbone graphs.The average length of all shortest paths between any two vertices is logarithmically small. On the other hand, the RWM algorithm tends to generate vertices with a higher probabilistic degree. Therefore, the RWM sampler can efficiently reach a k-backbone node which has a probabilistic degree of at least k.

2.5 Markov Logic Networks

2.5.1 Markov Networks

As one of the probabilistic graphical models, Markov networks (also called Markov random fields) can represent the full joint probability distribution D over a set of random variables $X = X_1, ...X_n$, as shown in Equation 2.3. In Equation 2.3, Φ_k and $x_{\{k\}}$ represent the potential function and the state of the k^{th} clique, respectively. Furthermore, Z is the normalizing constant and can be computed using Equation 2.4. Each variable can be represented as a node in the Markov network and the correlation among variables are reflected by the edges between nodes.

$$\mathrm{D} = P(X = x) = \frac{1}{Z} \prod_k \Phi_k(x_{\{k\}}) \tag{2.3}$$

$$Z = \sum_{x \in X} \prod_k \Phi_k(x_{\{k\}}) \tag{2.4}$$

Based on the full joint distribution D, we are able to answer any conditional probability query by summing the corresponding entries in D. To be specific, if we use $E = E_1, ..., E_m$ to denote the set of *evidence variables* and $Y = Y_1, ..., Y_k$ to represent the set of *query variables*, a conditional probability query of Y given the fact $E = e$ can be calculated as

$$\sum_H P(Y, H | E = e), \tag{2.5}$$

where $H = X - Y - E$ is a set of *hidden variables*.

2.5.2 Markov Blankets

In a Markov network, the *Markov Blanket* (MB) for node X, denoted as $MB(X)$, is the set of all the neighboring (i.e., directly connected) nodes of node X. The Markov blanket for an arbitrary node X maintains the following property: node X is conditionally independent of all other nodes in the network, given the Markov blanket for node X. This property can be represented using

Equation 2.6. In Equation 2.6, $MB(X)$ represents the Markov blanket for node X and $\overline{MB(X)}$ denotes all the nodes in the network which do not belong to the Markov blanket for node X.

$$P(X|MB(X), \overline{MB(X)}) = P(X|MB(X)) \tag{2.6}$$

2.5.3 Markov Logic Networks

Markov logic networks (MLN) are a powerful modeling tool which can provide the full expressiveness of probabilistic graphical models and first-order logic in a unifying representation. An MLN can be represented using a set of pairs (F_i, w_i), where F_i is a formula (a rule) in first-order logic and w_i is a real value representing the weight of the corresponding formula. The weight indicates how strong a formula is. The main idea behind MLN is that a possible world that violates a formula is not impossible but less probable.

The inference in an MLN is based on the inference in its resulting grounded Markov network. In other words, an MLN needs to be converted to a Markov network for inference by the means of creating a node for each grounded predicate and generating an edge between two nodes if the corresponding two grounded predicates appear in the same rule grounding.

In such a Markov network derived from an MLN, each node can be treated as a binary variable and each assignment of truth values to all the binary variables constitutes a possible world pw. The probability distribution of possible worlds (PW) in a ground Markov network can be defined as

$$P(PW = pw) = \frac{1}{Z} \exp(\sum_{i=1}^{F} w_i n_i(pw)), \tag{2.7}$$

where F is the number of formulas in the MLN, $n_i(pw)$ is the number of true groundings of F_i in the possible world pw, w_i is the weight of F_i, and Z is the normalizing constant.

2.5.4 Inference in Markov Logic Networks

As mentioned in the previous subsection, inference in an MLN is based on the inference in its derived Markov network. However, in many cases, generating an exact full joint distribution in a Markov network is *NP-Hard* and computationally intractable. In the worst case, the complexity of the algorithm in the case of n random variables is $\mathcal{O}(n2^n)$, requiring exponential time and space to construct the corresponding Markov network representation. Therefore, several approximate algorithms were developed to overcome this restriction.

MC-SAT

MC-SAT [25, 26] is a state-of-the-art approach to conduct approximate inference over Markov logic networks by combining *slice sampling* with satisfiability testing. Specifically, MC-SAT employs WalkSAT[2] as a procedure to sample from each slice, i.e., to sample a new state given the auxiliary variables. Furthermore, detailed balance among all the generated samples is preserved in MC-SAT so that inference calculations can be made only on the generated samples. MC-SAT has been proven to be orders of magnitude faster than standard MCMC methods, such as Gibbs sampling and simulated tempering, and is applicable to any model that can be expressed in Markov logic.

WalkSAT

Here we briefly introduce the WalkSAT solver, which is used in MC-SAT as a procedure to find a satisfying solution. *Satisfiability* is the problem of finding an assignment of truth values to all the variables that can satisfy all formulas in a knowledge base. Because *Satisfiability* is an NP-complete problem, WalkSAT serves as one of the most efficient approaches to *satisfiability* problems based on stochastic local search. Starting from a random initial state, WalkSAT repeatedly changes (flips) the truth value of a binary variable in a random unsatisfied formula. Specifically, with probability p, WalkSAT chooses the variable that maximizes the number of satisfied formula, and with probability $1 - p$, WalkSAT selects a random variable. WalkSAT has been proven to be able to solve *satisfiability* problems with thousands of random variables in a fraction of a second. The complete algorithm of a WalkSAT solver is formalized in Algorithm 6.

[2]Strictly speaking, a variant of WalkSAT, namely SampleSAT, is used in MC-SAT.

Algorithm 6: WalkSAT($\mathbb{C}$, max_tries)

```
Input: A set of logic formulas, $\mathbb{C}$, and the maximum number of tries, $max\_tries$
Output: An assignment of all binary predicates satisfying each formula in $\mathbb{C}$
6.1 Randomly assign TRUE/FALSE to each binary predicate.
6.2 $k = 0$
6.3 for $k \leq max\_tries$ do
6.4     Randomly select a unsatisfied formula, $F_i$, in $\mathbb{C}$
6.5     With probability $p$, select a binary predicate in $F_i$ and flip (change) the value of that
        predicate.
6.6     if all the formulas in $\mathbb{C}$ are satisfied then
6.7         Return the current assignment of predicates as the solution.
6.8     $k++$
```

2.6 Inference

2.6.1 MC-SAT$^+$

The inference in LinkProbe is based on the MC-SAT$^+$ method, which is a probabilistic extension of the well-known MC-SAT algorithm. In this section, we elaborate on how LinkProbe employs MC-SAT$^+$ to conduct the inference.

Notice that a possible world (a ground Markov network) is comprised of an assignment of truth values to all the nodes in an inference graph. Therefore, in order to instantiate a sample (a valid possible world), we need to assign truth values to all the involved nodes. Specifically, in each sampling cycle, MC-SAT$^+$ employs the WalkSAT solver and a biased sampling method to find values for *hidden nodes* and *probabilistic evidence nodes*, respectively. On the contrary, throughout all the sampling cycles, MC-SAT$^+$ keeps the values of *deterministic evidence nodes* fixed.

Searching for Values for Hidden Nodes

MC-SAT$^+$ launches the WalkSAT solver to find the truth values for all hidden nodes. Starting from a random initial state, WalkSAT repeatedly flips the truth value of a hidden node (can be treated as a binary random variable) in a random unsatisfied rule grounding. At each iteration in a sampling cycle of MC-SAT$^+$, WalkSAT selects a hidden variable v in a randomly chosen unsatisfied rule grounding to flip the value of v. The selection criterion is as follows. With probability p, WalkSAT chooses the hidden node that can minimize the number of unsatisfied rule groundings. With probability $1 - p$, WalkSAT chooses a random hidden node in the referred rule grounding.

Generating Values for Probabilistic Evidence Nodes

We utilize inverse transform sampling to randomly generate values for probabilistic evidence triples based on their probability information specified in $\mathcal{DB}$. Inverse transform sampling is a basic method to draw random samples from any probability distribution given its cumulative distribution function [37]. Suppose we have a record $< from, to, prob >$ in $\mathcal{DB}$. When we need to assign values to the node representing this record, we randomly generate its value as TRUE with a probability of $prob$ and as FALSE with a probability of $1 - prob$. Afterwards, we keep this randomly generated value unchanged during each sampling cycle of MC-SAT$^+$. The complete assignment operation of all the probabilistic evidence triples can be done by calling the procedure $biased\_sample()$.

Determining Values for Deterministic Evidence Nodes

For *deterministic evidence nodes*, i.e., a record $< from, to, prob >$ where $prob = 1$, we set their values as TRUE and keep such values unchanged during the entire inference procedure.

Algorithm Description of MC-SAT$^+$

The complete algorithm of the MC-SAT$^+$ method is formalized in Algorithm 7. In Algorithm 7, each sample $x^{(i)}$ consists of *deterministic evidence nodes*, ${x_{de}}^{(i)}$, *probabilistic evidence nodes*, ${x_{pe}}^{(i)}$, and *hidden nodes*, ${x_h}^{(i)}$, i.e., $x^{(i)} = {x_{de}}^{(i)} \cup {x_{pe}}^{(i)} \cup {x_h}^{(i)}$. In particular, since we keep the truth values of the *deterministic evidence nodes* unchanged during the entire sampling procedure, we rewrite ${x_{de}}^{(i)}$ as x_{de}. An illustration of the sampling in MC-SAT$^+$ is shown in Figure 2.7. As demonstrated in Figure 2.7, similar to MC-SAT [25, 26], MC-SAT$^+$ utilizes WalkSAT

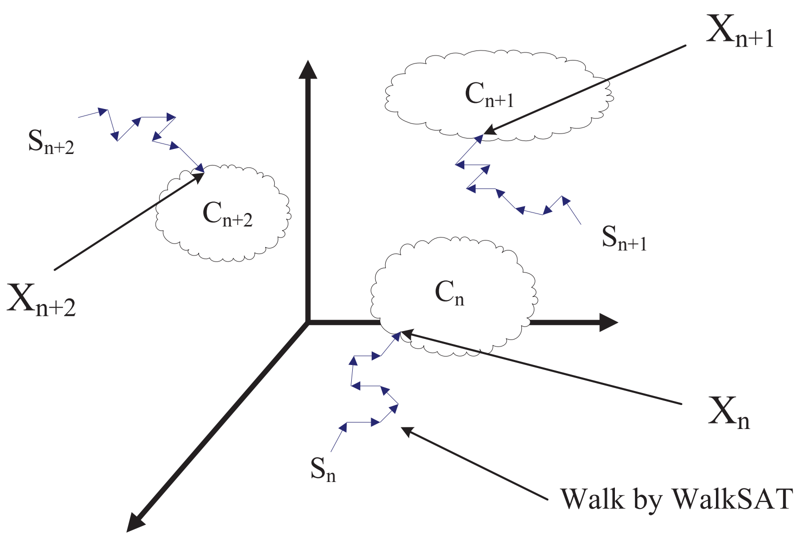


Figure 2.7: An illustration of the sampling procedure in the MC-SAT$^+$ method.

as a procedure to identify each valid solution X_n to each slice (constraint set) C_n from a random assignment S_n. The weight w_j for formula f_j can be approximated as $w_j = \log \frac{p_j}{1-p_j}$, where p_j is the probability that formula f_j holds. If p_j is 1, then f_j is a hard formula.

2.6.2 Probabilistic Inference in LinkProbe

Here we elaborate on the four steps of the inference procedure of LinkProbe for each submitted link query.

Deriving the k-backbone Graph

As discussed in Section 2.3, we retrieve the k-backbone graph for a given social network G.

Retrieving the d-local Graphs

As discussed in Section 2.4, we retrieve the d-local graphs for a link query.

Algorithm 7: The MC-SAT$^+$ Inference Method

```
Input: The $k$-depth inference subgraph for pQuery($P$, $DB$)
Output: We report $\frac{counter}{num\_samples}$ as the probability that the triple $\{s, p, o\}$ is true
7.1  Assign truth values to $x_{de}$ ;
7.2  ${x_{pe}}^{(0)} = biased\_sample()$ ;
     /* Generate the truth values of the probabilistic evidence
        node                                                        */
7.3  ${x_h}^{(0)} = WalkSAT(hard\_formulas)$ ;
     /* Find the truth values for the hidden nodes by applying
        WalkSAT with all hard rules as the constraints              */
7.4  counter = 0 ;
7.5  for $i = 1$ to $num\_samples$ do
7.6      ${x_{pe}}^{(i)} = biased\_sample()$ ;
7.7      $C = \emptyset$ ;
7.8      for All $f_j \in$ formulas satisfied by $x^{(i-1)}$ do
7.9          With probability $1 - e^{-w_j}$, add $f_j$ to C ;
             /* $w_j$ is the weight of $f_j$                          */
         /* Slice sampling                                          */
7.10     ${x_h}^{(i)} = WalkSAT(C, max\_tries)$ ;
         /* Search for the valid truth values for hidden nodes by
            calling WalkSAT with C as the constraints.
            Consequently, we obtain $x^{(i)}$ as $x_{de} \cup {x_{pe}}^{(i)} \cup {x_h}^{(i)}$    */
7.11     if The value of the query node in $x^{(i)}$ is TRUE then
7.12         counter ++ ;
             /* $counter_i$ represents the counter for $q_i$. Here we count
                the frequency of each grounded VP predicate being
                true in the samples                                 */
```

Connecting the d-local Graphs with the k-backbone Graph

For a link query, in particular $Query(X, Y, knows)$, we issue two runs of the RWM sampler from X and Y, respectively. LinkProbe collects all the discovered nodes during the RWM sampling.

Inferencing over Inference Subgraph

For a link query, $Query(X, Y, knows)$, its inference subgraph is comprised of the k-backbone graph, the respective d-local graphs of X and Y, and all the discovered nodes during the RWM

Algorithm 8: Inference Method in LinkProbe (d, k, X, Y)

```
    Output: The probability $p$ that $X$ holds a certain relationship with $Y$, for example,
            knows(X,Y).
8.1 Retrieve all the vertices in the $k$-backbone graph of $G$ as $K$
8.2 Identify the set of the vertices in the $d$-local graph of $X$ as $\mathbb{X}$.
8.3 Identify the set of the vertices in the $d$-local graph of $Y$ as $\mathbb{Y}$.
8.4 Generate a set of vertices $\mathbb{X}'$ by launching a RWM sampler from vertex $X$.
8.5 Generate a set of vertices $\mathbb{Y}'$ by launching a RWM sampler from vertex $Y$.
8.6 Construct the inference subgraph $I = K \bigcup \mathbb{X} \bigcup \mathbb{X}' \bigcup \mathbb{Y} \bigcup \mathbb{Y}'$
8.7 $p$ = MC-SAT$^+$ ($I$, $G$)
```

sampling. As for each query, LinkProbe applies MC-SAT$^+$ over its inference subgraph instead of on the original graph.

2.6.3 Algorithm Description

Algorithm 8 shows how LinkProbe builds the inference subgraph for a given link query and then conducts joint inference over it. First, LinkProbe retrieves the k-backbone graph. Afterwards, LinkProbe identifies the d-local graphs. Next, LinkProbe launches random walk Metropolis samplers to connect the d-local graphs with the k-backbone graph in order to form the final inference subgraph. Subsequently, LinkProbe conducts MC-SAT$^+$ on the inference subgraph.

2.7 Error Analysis

$$\left\{\begin{array}{ll} knows(X,Z) \wedge knows(Z,Y) & \longrightarrow knows(X,Y) \\ knows(X,T) \wedge knows(T,Y) & \longrightarrow knows(X,Y) \\ knows(X,U) \wedge knows(U,Y) & \longrightarrow knows(X,Y) \\ knows(X,V) \wedge knows(V,Y) & \longrightarrow knows(X,Y) \\ knows(X,W) \wedge knows(W,Y) & \longrightarrow knows(X,Y) \\ ... & \longrightarrow ... \\ ... & \longrightarrow ... \end{array}\right. \tag{2.8}$$

$$\begin{cases} knows(X,Z) \wedge knows(Z,Y) & \longrightarrow knows(X,Y) \\ knows(X,T) \wedge knows(T,Y) & \longrightarrow knows(X,Y) \end{cases} \tag{2.9}$$

In this section, we elaborate on the error induced by inferencing over the inference subgraph rather than over the original graph. We use $error$ to describe the error between our proposed inference graph-based MLN implementation, represented as $LinkProbe()$, and the original graph based MLN implementation, represented as $Naive()$, with the same number of samples drawn.

Without loss of generality, suppose we adopt the transitive property of friendship for inference. For simplicity of presentation, we focus the MLN inference on the Markov blanket of the grounding Markov network (the Markov blanket of a node contains all the variables that shield the node from the rest of the network). Our objective is to predict the acquaintance between two people, X and Y. In the naive "original graph" implementation, the constraint set for a certain slice can be represented as Equation 2.8, where X, Y, Z, T, U, V and W denote different people. If we have 10^6 people in the social graph, then the total number of formula groundings in Equation 2.8 could be very close to 10^6, which will lead to extreme inefficiency in terms of inference time and memory usage.

We aim to compare the inference result generated by the entire graph implementation $Naive()$ with that returned by the inference subgraph implementation $LinkProbe()$, with respect to the value of $knows(X,Y)$. Suppose X, Y, Z and T are the only nodes in the inference subgraph while others are not. Then, we can obtain Equation 2.9, which is only comprised of the nodes appearing in the inference subgraphs. For the referred slice during MC-SAT$^+$ inference, $Naive()$ implementation employs the constraint set Equation 2.8 while $LinkProbe()$ implementation utilizes the constraint set Equation 2.9. In addition, we use $\mathcal{N}$ to denote the set of the formula groundings in Equation 2.8 and $\mathcal{L}$ to represent the set of the formula groundings in Equation 2.9. Each predicate in Equation 2.8 and Equation 2.9 has three candidate values TRUE, FALSE and UNKNOWN.

2.7.1 Error Analysis

Case 1

All the left sides of the formula groundings in $\mathcal{N}$ are FALSE. In this case all the formula groundings in $\mathcal{N}$ are trivially satisfied. According to the LazySAT implementation of WalkSAT, any formula grounding trivially satisfied by evidence can be removed without affecting the final solution found by WalkSAT. Therefore, it is safe for us to delete all the formula groundings in $\mathcal{N} - \mathcal{L}$. Therefore, in this case Equation 2.8 and Equation 2.9 are equivalent with respect to the run of WalkSAT. As a result, $Naive()$ and $LinkProbe()$ will yield the same value of $knows(X, Y)$.

Case 2

At least one left side of the formula groundings in $\mathcal{N}$ is TRUE or UNKNOWN.

- Case 2-A: TRUE or UNKNOWN only appears in the left side(s) of the formula groundings in $\mathcal{L}$. In this case the left sides of all the formula groundings in $\mathcal{N} - \mathcal{L}$ are FALSE. As a result, it is safe to delete all the formula groundings in $\mathcal{N} - \mathcal{L}$ because they are all trivially satisfied. $Naive()$ and $LinkProbe()$ will return the same value of $knows(X, Y)$ in this case.

- Case 2-B: TRUE or UNKNOWN only appears in the left side(s) of the formula groundings in $\mathcal{N} - \mathcal{L}$. In this case all the left sides of all the formula groundings in $\mathcal{L}$ are FALSE. There could be an error between $Naive()$ and $LinkProbe()$ because some formula groundings in $\mathcal{N} - \mathcal{L}$ that need to be satisfied are dropped and all the formula groundings in $\mathcal{L}$ are trivially satisfied. However, because any person who knows X or Y with a probability no less than d on the probabilistic social graph G should be included in the inference subgraph (according to the definition of d-local graph), the probability that case 2-B occurs is less than $(1 - d)^{(n+m)}$, where n and m are the numbers of friends on the inference subgraph. If we take d as 1, then case 2-B never happens.

- Case 2-C: TRUE or UNKNOWN appears in the left side(s) of the formula groundings in $\mathcal{L}$ and in $\mathcal{N} - \mathcal{L}$.

- Case 2-C-1: At least one left side of the formula groundings in $\mathcal{L}$ is TRUE. In this case, because the formula groundings in $\mathcal{L}$ are able to determine the value of $knows(X, Y)$ as TRUE, it is safe to remove all the formula groundings in $\mathcal{N} - \mathcal{L}$ from $\mathcal{N}$. $Naive()$ and $LinkProbe()$ will return the same value of $knows(X, Y)$ as TRUE. Since $\mathcal{L}$ is comprised of predicates involving the most globally influencing people and most locally connected people, this case happens with a very high probability.
- Case 2-C-2: All left sides of the formula groundings in $\mathcal{L}$ are FALSE or UNKNOWN. In this case, the removal of all the formula groundings in $\mathcal{N} - \mathcal{L}$ may lead to a WalkSAT solution with a different value of $knows(X, Y)$, resulting in an error between $Naive()$ and $LinkProbe()$ for the current slice sampling cycle. However, since the predicates in $\mathcal{L}$ describe the relationship between the persons of interest and their most locally related people and the most globally influencing people, each predicate in $\mathcal{L}$ is expected to be TRUE with a very high probability. Therefore, case 2-C-2 occurs only with a very low probability.

By reducing Equation 2.8 (corresponds to the entire graph) to Equation 2.9 (corresponds to the inference subgraph), LinkProbe decreases the number of formula groundings which need to be taken into account in each sampling cycle by several orders of magnitude, which leads to much higher inference efficiency and scalability without significant sacrifice in accuracy compared to the naive MLN implementation.

2.7.2 Error Upper Bound

During MC-SAT$^+$, LinkProbe first draws samples based on slice sampling, then calculates the frequency of TRUE and FALSE over all the samples to report the final inference result. Suppose N samples are drawn in total, then each incorrect count will induce the inference error of $\frac{1}{N}$. The error between $LinkProbe()$ and $Naive()$ can be described using Equation 2.10. In Equation 2.10, $P(k)$ is the probability that k incorrect counts occur during the inference. Because the incorrect count during the MC-SAT inference follows the Poisson Binomial distribution, the probability of having exactly k incorrect counts during N samples, $P(k)$, can be calculated as Equation 2.11, where p_i is the probability that case 2-C-2 occurs in the i^{th} sampling cycle, F_k is the set of all subsets of k integers that can be selected from {1,2,3,...,N}, and S^C is the complement of S. Because the mean of the Poisson Binomial distribution is $\sum_{i=1}^{N} p_i$, the error between $LinkProbe()$ and $Naive()$ can be bounded as shown in Equation 2.12. Notice that in Equation 2.12, N is very large and $\forall i, p_i$ is very small.

If we assume that the probability that case 2-C-2 occurs in each sampling cycle is identical, say P, then the incorrect count follows the Binomial distribution $B(N, P)$. In this case, Equation 2.10 can be rewritten as Equation 2.13.

$$error \leq \sum_{k=1}^{N} \frac{1}{N} \times k \times P(k) \tag{2.10}$$

$$P(k) = \sum_{S \in F_k} \times \prod_{i \in S} p_i \times \prod_{j \in S^C} (1 - p_j) \tag{2.11}$$

$$error \leq \frac{1}{N} \times \sum_{i=1}^{N} p_i \tag{2.12}$$

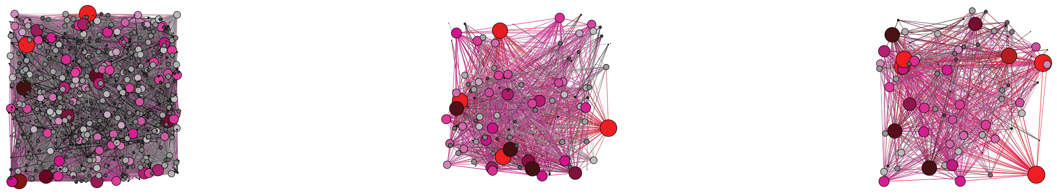
$$error \leq \sum_{k=1}^{N} \frac{1}{N} \times k \times \binom{N}{k} \times P^k \times (1 - P)^{N-k} \tag{2.13}$$

2.8 Experimental Validation

In order to validate the superiority of LinkProbe, we implemented LinkProbe and investigated its performance with two real-world data sets, LiveJournal (LJ) [50] and High Energy Physics (HEP) [49]. Table 2.1 shows the basic statistics of the above two data sets. We used the relation transitive property as the inference rule and the probability that the rule is true was approximated as the fraction of closed triangles. Each result on inference accuracy and efficiency is obtained by averaging 100 randomly generated link queries against the referred data set.

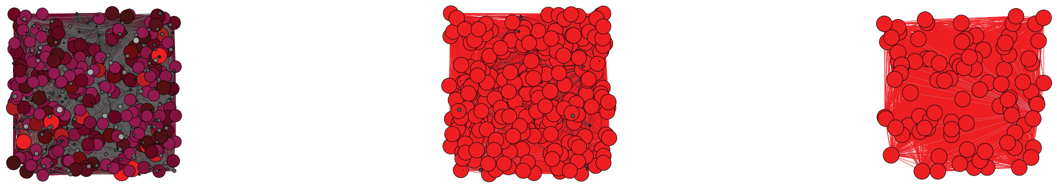
Table 2.1: Data sets used in the experiments.

Name	**Description**	**Number of nodes**	**Number of edges**	**Average clustering coefficient**	**Fraction of closed triangles**
LJ	social network	4,847,571	68,993,773	0.3123	28.820%
HEP	collaboration network	12,008	237,010	0.6115	65.950%



(a) $k = 1500$. (b) $k = 3000$. (c) $k = 4500$.

Figure 2.8: The k-backbone graphs of LJ.



(a) $k = 200$. (b) $k = 400$. (c) $k = 600$.

Figure 2.9: The k-backbone graphs of HEP.

2.8.1 k-backbone Graphs

We retrieved 1500-backbone, 3000-backbone and 4500-backbone graphs for LJ and 200-backbone, 400-backbone and 600-backbone graphs for HEP. Table 2.2 lists the number of people involved in the above backbone graphs. Their respective social graphs are shown in Figure 2.8 and Figure 2.9, respectively. As illustrated in Figures 2.8 and 2.9, a higher k value leads to a sparser social graph, filtering out more nodes (more people) from the original data set.

Table 2.2: Number of people in the corresponding k-backbone graphs.

	LJ, k=1500	**LJ, k=3000**	**LJ, k=4500**	**HEP, k=200**	**HEP, k=400**	**HEP, k=600**
Number of people	652	124	46	425	252	95

2.8.2 Memory Consumption

LinkProbe is capable of inferencing over large-scale social graphs. Table 2.3 shows the approximate memory requirement of LinkProbe with data sets and k values specified. In addition, Table 2.4 illustrates the approximate memory consumption by the MLN naive implementation.

Table 2.3: Memory consumption by LinkProbe.

	LJ, k=1500	**LJ, k=3000**	**LJ, k=4500**	**HEP, k=200**	**HEP, k=400**	**HEP, k=600**
Required memory (GB)	3.4638	0.3638	0.28813	2.51	0.485	0.1206

2.8.3 Inference Accuracy

Definition *The Absolute Error* (AE): *The Absolute Error* is defined to be the difference between the predicted probability and the true probability regarding the existence of a particular link (i.e., a link query).

Definition *The Mean Absolute Error* (MAE): *The Mean Absolute Error* is defined to be the average value of the *the absolute error* over the involved link queries.

Table 2.4: Memory consumption by the MLN naive implementation.

	LJ	**HEP**
Required memory (GB)	$4.76 * 10^{11}$	$1.28 * 10^7$

Social Networks

In this subsection, we investigate the performance of LinkProbe on the LiveJournal (LJ) data set in terms of inference accuracy. We retrieved the 1500-backbone graph, 3000-backbone graph and 4500-backbone graph of the LiveJournal data set. i.e., we set k as 1500, 3000, and 4500, respectively. We increased the number of samples drawn in LinkProbe to investigate its impact on inference accuracy. Figure 2.10 shows the results. In Figure 2.10, when we gradually raised the number of samples, the mean absolute error dropped accordingly. The reason is that when we draw more samples using MC-SAT$^+$, the distribution of samples (possible worlds) provided an approximation more close to the reality. On the other hand, when we adopted a larger k value, the corresponding mean absolute error increased because the inference subgraph filtered out more nodes and contained a smaller fraction of the original graph.

Collaboration Networks

In this subsection, we investigate the performance of LinkProbe on the High Energy Physics (HEP) data set in terms of inference accuracy. We varied k from 200 to 400 to 600 and gradually increased the number of samples drawn in the inference to study their impact on inference accuracy. The experimental results are shown in Figure 2.11. In Figure 2.11, with the enlargement of the number of samples drawn, the mean absolute error is reduced accordingly. On the other hand, when we took a larger k value, the mean absolute error rose.

2.8.4 Inference Efficiency

Here we measure the end-to-end execution time (running time) counted from the time of query submission to the time returning the final inference results.

Social Networks

In this subsection, we investigate the performance of LinkProbe on the LiveJournal (LJ) data set in terms of inference efficiency. We varied k from 1500 to 3000 to 4500 and changed the

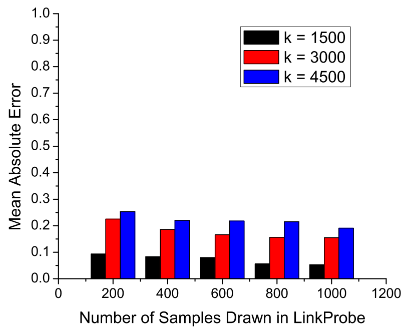


Figure 2.10: The impact of number of samples on inference accuracy for LJ.

number of samples. Figure 2.12 shows the results. In Figure 2.12, when we gradually increased the number of samples, the mean running time extended accordingly. The reason is that LinkProbe requires more time to come up with samples using MC-SAT^{+}. With a larger k value, the mean running time dropped because a smaller fraction of the original graph was included in the inference subgraph.

Collaboration Networks

In this subsection, we investigate the performance of LinkProbe on the High Energy Physics (HEP) data set in terms of inference accuracy. We increased k from 200 to 400 to 600 and varied the number of samples. The results are shown in Figure 2.13. In Figure 2.13, with the enlargement of the number of samples drawn in the inference, the mean running time extended. On the contrary, when we raised the k value, the mean running time dropped accordingly.

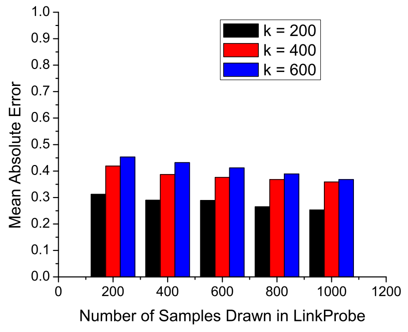


Figure 2.11: The impact of number of samples on inference accuracy for HEP.

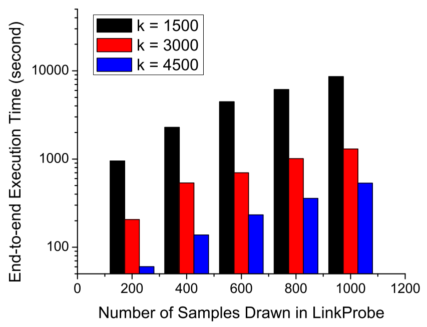


Figure 2.12: The impact of number of samples on inference efficiency for LJ.

2.9 Related Work

2.9.1 Sampling in Probabilistic Databases

Sampling-based approaches [37, 88, 11, 89] are proposed for managing incomplete and uncertain data in probabilistic databases [2, 4, 5, 13, 19]. The idea is simple and intuitive: we construct random samples while observing the prior statistical knowledge and constraints about the data. Thus, each sample is one possible realization (possible world) in the space of uncertainty, and the entire set of samples reveals the distribution of the uncertain data which we want to model. Queries and inferences are then conducted against this distribution. MCDB [37], for example, allows a user to define arbitrary variable generation functions that embody the database uncertainty. MCDB employs these functions to pseudorandomly generate realized values for the uncertain attributes, and evaluates queries over the realized values. However, compared to the statistical relational learning

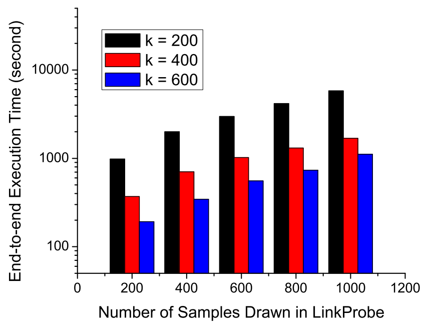


Figure 2.13: The impact of number of samples on inference efficiency for HEP.

approaches, most of the above works are only focused on simpler probabilistic models for query processing.

2.9.2 Statistical Relational Learning

Techniques of utilizing statistical relational learning have attracted more and more interests from researchers due to the rapidly increasing applications on large-scale uncertain data (i.e., online data produced by users), for example, natural language processing, entity resolution, information extraction and social network analysis. Statistical relational learning is an emerging area that combines statistics, artificial intelligence and machine learning for addressing uncertainty with complex inherent structural constraints. In statistical relational learning, uncertainty is dealt with using statistical methods. Structural constraints can be represented using first-order logic. The inference in statistical relational learning is typically driven by using probabilistic graphical models, e.g., *Bayesian networks* and *Markov networks*. *Bayesian networks* represent a joint distribution of random variables as a directed acyclic graph. Its nodes are the random variables while the edges correspond to direct influence from one node to another. The BayesStore project [82] is one recent work based on *Bayesian networks* for probabilistic inference. Compared to *Bayesian networks*, *Markov networks* represent a joint probability distribution of random variables as an undirected graph, where the nodes represent the variables and the edges correspond to the direct probabilistic interaction between neighboring variables. Oliveira and Gomes [63] employed Markov logic networks for web reasoning over partially correct rules. However, one key issue of applying Markov Logic Networks in practice is its low efficiency, especially in domains involving many objects, due to the combinatorics. Some approaches were proposed to alleviate this problem. For example, Singla and Domingos [75] proposed a lazy implementation of Markov logic networks, named as LazySAT. Mihalkova and Richardson [59] presented a meta-inference algorithm that can speed up the inference by avoiding redundant computation. Shavlik and Natarajan [74] provided a preprocessing technique to reduce the effective size of inference networks. However, all the above approaches experimented only with very small-scale data sets, involving only a very limited number of objects. Our work aims at this challenge: how to apply statistical relational reasoning in large-scale problems, i.e., in a social graph which involves millions of people?

2.9.3 Link Prediction on Social Networks

As one of the core tasks in social networks, link prediction has been extensively studied. Liben-Nowell et al. [55] formalized the link prediction problem and developed approaches to link prediction based on measures for analyzing the proximity of nodes in a network. Unfortunately, they considered only the features that are based on the link structure of the network itself. Leroy et al. [48] introduced a framework to predict the structure of a social network when the network itself is totally missing while some other information regarding the nodes is available. They first generated a bootstrap probabilistic graph using any available feature and then applied the link prediction algorithms in [55] to the probabilistic graph. Backstrom et al. [7] proposed a supervised random walks based method for link prediction that performs a PageRank-like random walk on a social network so that it is more likely to visit the nodes to which new links will be created in the future. What distinguishes our effort from existing works in this trend is that our approach treats link prediction as a knowledge-based inference problem and utilizes Markov Logic Networks to capture complex and structural relation and interaction among social entities. Our system can be considered as an attempt of enabling statistical relational learning over large scale data for link prediction.

2.10 Conclusion and Future Work

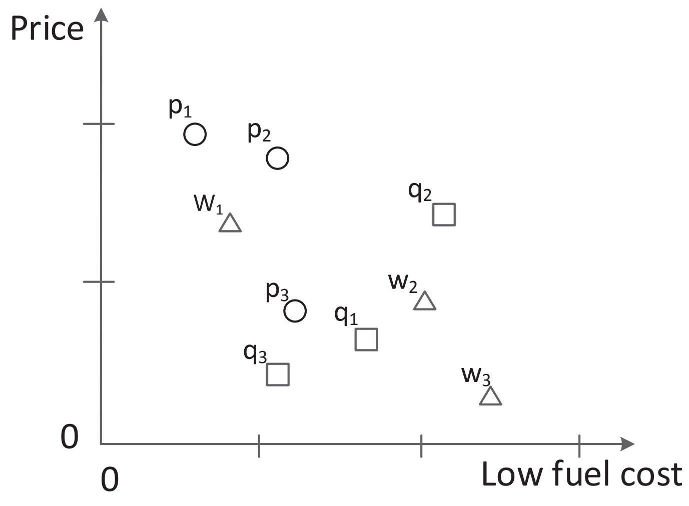
LinkProbe is a novel prototype to predict link existence in large-scale social networks based on Markov Logic Networks. It aims to handle soft inference rules which are common in reality and was proven to scale well towards large social networks. Compared to the naive MLN implementation, LinkProbe decreases the number of formula groundings by several orders of magnitude, leading to a much higher inference efficiency and scalability without significant sacrifice in accuracy. Our extensive experiments with realistic data sets showed that LinkProbe manages to provide a tunable balance between MLN inference efficiency and inference accuracy.

Chapter 3

An Efficient MapReduce Framework for Probabilistic Skylines over Uncertain Data

3.1 Introduction

Recently, skyline queries [16] [9] [42] [47] have been widely used in various systems including multi-criteria decision making, search punning, and personalized recommendation systems. Given more than one criterion, the skyline queries prune the search space of a large collection of multi-dimensional objects to a small set by returning objects that are not dominated by, or to say, superior to, others. Due to the rapid increase in the amount of uncertain data, the past decade also witnesses an extensive study on uncertain database management in a variety of applications in the real-world, such as probabilistic ranking queries [35], probabilistic k-NN query [53], and probabilistic top-k query [52]. As an extension of the traditional skyline queries, the probabilistic skyline queries were proposed in [68] to cope with uncertain datasets. The probabilistic skyline queries



Obj.	Ins.	Prob.
h1	(10, 4)	0.5
	(10, 2)	0.2
	(10, 3)	0.3
h2	(20, 4)	0.3
	(20, 5)	0.4
	(20, 5)	0.3
h3	(15, 4)	0.6
	(20, 5)	0.2
	(15, 3)	0.2

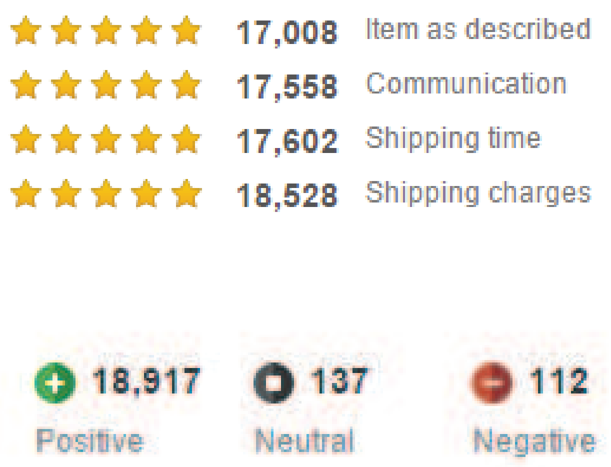
Figure 3.1: Example query 1 (uncertainty from different sources)

find the probability of each object to be in the skyline set, where each object is a set of instances (tuples) and associated with a probabilistic distribution. The exact p-skyline queries return a set of uncertain objects whose skyline probabilities are not less than p and their corresponding exact skyline probabilities. Because the computation of such probabilistic skylines on uncertain data is algorithmically expensive, a few solutions [68] [6] [8] [39] have been proposed. However, there is a lack of study in the literatures on how to take full advantage of MapReduce, a popular parallel programming model, to boost the probabilistic skylines computation. In this paper, we focus on how to leverage the MapReduce model to process the exact p-skyline queries on uncertain data for the high efficiency and scalability in a high dimensional space.

3.1.1 Our Motivating Problem

The exact p-skyline queries on uncertain data can be used in numerous application scenarios in the real world. The following are two examples of the p-skyline queries on uncertain data.

Example1: *Alice is looking for a car by searching multiple agent website with low price and low-fuel cost. Different travel agent websites (e.g., Cars.com, Edmunds.com) may return different results. In Figure 3.1, the $y-$dimension captures the price of a car, and the $x-$dimension represents*



★★★★★ 17,008 Item as described
★★★★★ 17,558 Communication
★★★★★ 17,602 Shipping time
★★★★★ 18,528 Shipping charges

18,917 Positive
137 Neutral
112 Negative

Obj.	Ins.	Prob.
r1	(4, 5)	0.5
	(4, 4)	0.3
	(4, 1)	0.2
r2	(3, 4)	0.6
	(3, 2)	0.2
	(3, 1)	0.2

Figure 3.2: Example query 2 (uncertainty from different voters)

the car's estimated fuel cost every year. A car dominates another when it has lower price and lower annul fuel cost. Eric inquires the car from different websites. An exact p-skyline query returns (1) all the cars whose skyline probability is at least p and (2) the exact skyline probability of each returned car.

In the above example, each car can be treated as an object while each instance of this object corresponds to the car's information from different website, associated with a probability (confidence value) reflecting the importance of the website.

Example2: *Eric is inquiring a rice cooker at Ebay.com. All rice cookers are listed by different Merchandising shops. Every rice cooker has three evaluation metrics: (a). price; (b). The shipping time; (c). shipping charges. For the same rice cooker (The same model), the evaluations shown may vary much. An exact p-skyline query returns (1) all the rice cookers where the probability that there does not exist another restaurant with a lower price, shipping time and shipping charges is at least p and (2) the exact skyline probability for each returned rice cooker.*

In the above example, each rice cooker can be treated as an object while each instance of this object corresponds to different merchandise shop, associated with a probability (confidence value) reflecting the usefulness of the merchandise shop.

3.1.2 Our Goal and Faced Challenges

Parallel computation has drawn a lot of interests recently [1] [66] [87] [18] and MapReduce [20] was introduced as an easy-to-use parallel computing paradigm with high scalability and reliability. One advantage of the MapReduce model is that users only need to write the Map and Reduce functions without much concern about the details of the parallelization. Today MapReduce has been incorporated into various data processing systems [57] [90] [80] [62]. However, there is little work on how to take full advantage of the MapReduce model to make the probabilistic skyline computation scale up to high volume data.

In this paper, we aim to use the MapReduce model for the efficient and scalable processing of the exact p-skyline queries on the high dimensional uncertain data. However, this task is not trivial. The following are the major challenges:

- How to avoid unnecessary dominance tests based on the characteristics of data distribution to reduce the computation cost?
- How to divide the workload among all machines to maximize the parallelism?
- How to re-use the intermediate results to boost the efficiency of skyline computation?

3.1.3 Our Contributions

To the best of our knowledge, our work is the first to investigate how to evaluate exact p-skyline queries on uncertain data based on the MapReduce parallel computing paradigm for high efficiency and scalability.

- In order to avoid unnecessary dominance tests, we apply three pruning rules to our uncertain data for preprocessing to reduce the computation overhead. The experimental results show the superiority of our solutions over the other approaches.
- For achieving the maximized parallelism, we proposed a two-phase MapReduce framework and both the angle-based partitioning and the grid-based partitioning are used in our proposed framework.
- We propose a random pivot points-based approach in order to re-use the intermediate result to further boost the efficiency of skyline computation. In this approach, we distribute pivot points by using random sampling for the high dimensional data.
- Our experimental results with both realistic and synthetic datasets verify the efficiency and scalability of our proposed framework.

3.1.4 Organization of this paper

The rest of this paper is organized as follows. Section 2 surveys related work. The probabilistic skyline query and relevant techniques utilized in our solutions are formally defined in Section 3. Our advanced two phase MapReduce-based solution is illustrated in Section 4 and 5. Section 6 introduces how to optimize the local probabilistic skyline query in MapReduce. The experimental validation of our design is presented in Section 7. Section 8 concludes the paper.

3.2 Related Work

During the past decade, skyline queries [9] have been extensively studied and applied in various decision making applications. Varieties of methods [77, 64, 60, 17, 92, 47, 42] are proposed to efficiently solve this problem. Compared to the traditional skyline, the probabilistic skyline is a popular and powerful paradigm for extracting decisive information from uncertain databases. The problem of evaluating probabilistic skyline query is firstly raised by Pei *et al.* [68]. They also developed bottom-up and top-down methods using R-tree to efficiently prune the search space. Atallah *et al.* [6] proposed a sub-quadratic complexity algorithm, relying on space partitioning techniques combined with the dominance counting algorithm [58]. A probabilistic skyline solution using Z-tree is presented in [39], which find as many incomparable groups of instances as possible. However, all the above mentioned approaches are all serial algorithms, which are inevitable involved in performance issues like memory and efficiency, especially when input data is persistently growing. Our study reduces the overhead of serial algorithm running on only single machine by offering a parallel computing strategy.

A wide collection of parallel or distributed techniques are incorporated to deal with skyline or probabilistic skyline query. Grid-based partitioning method is popularly adopted by many studies [1, 84, 87, 85], since it is able to divide the original dataset into smaller subsets. Similarly, Angle-based separation over multi-dimensional space for skyline query was firstly investigated in [81], and attracted a variety of methods [40, 91].To the best of our knowledge, this paper is the first to apply the angle-based techniques to the probabilistic skyline issue.

In the recent years, the MapReduce framework [20] has been incorporated into probabilistic skyline computation [24, 67, 91, 61]. Ding *et al.* [24] proposed probabilistic skyline query in MapReduce (PSMR), a MapReduce strategy using filter-refine two phases approach. However, they assume that every tuple in the distributed uncertain database is attached with a probability, which could be regraded as a bivariate distribution, rather than multivariate distribution in state-of-the-art studies [68, 6, 39, 8]. Park *et al.* [67] also concentrated on probabilistic skyline queries in MapReduce framework, and follow the same data assumption with research mainstream. They initially indexed the data space by using a quadtree [70], which is created by randomly selecting samples from the dataset. Then, unqualified objects are filtered out when processing local probabilistic skyline, and global probabilistic skyline is evaluated afterward. There are two main approached, PS-QPF-MR and PS-BRF-MR [67], introduced in their study. PS-QPF-MR assumes that instances in one object must be integrated together, which is not applicable in real case, since instances of one object can be easily collected from multiple machines separately. In our study, we assume that instances of one object are distributed across individual nodes in the distributed environment, and emphasize the performance comparisons against PS-BRF-MR.

3.3 Preliminaries

3.3.1 Probabilistic Skyline over Uncertain Data

Given a data space with d dimensions $(D_1, \ldots, D_d)$ and a dataset, a skyline point is a data point that is not dominated by any other points. A point p dominating a point q is denoted as $p \prec p$ if $\forall k \in [1, d], p.D_k \leq q.D_k$; conversely, $p \not\prec p$ represents that p is not able to dominate q. Skyline query returns all skyline points meeting this condition. Besides, the domination relationship between points can be extended to group relationships. For example, $p \prec D$ denotes that p dominates all points in D. Similarly, $D \prec p$ represents that all points in D dominates p.

Let $OS = \{O_1, O_2, \ldots, O_n\}$ denotes an uncertain object set, and its cardinality is n. We assume that an certain object has several instances, each of which is a d-dimensional attribute vector and has a probability to exist. Formally, O_i is an instance set $O_i = \{p_1, p_2, \ldots, p_m\}$, where every element is an d-dimensional instances following discrete probability density distribution (PDF) of O_i. Any instance p's occurrence probability $Pr(p)$ only depends on the object's PDF, i.e., every object is independent of each other. For any object O, we assume that $\sum_{p \in O} Pr(p) = 1$. This is a realistic assumption adopted in many literatures such as [68] [8] [39] for analyzing uncertain data.

Here we introduce the definition of probabilistic skylines, which are quite different from traditional skylines. The goal of probabilistic skyline is to obtain the skyline probability $SKYProb(O_i)$ of one object O_i, i.e., the likelihood that O_i becomes a skyline object. Given two objects U and V, the probability that one instance p in V is dominated by U is:

$$Pr(U \prec p) = \sum_{q \in U,\ q \prec p} Pr(q) \tag{3.1}$$

The probability that p is a skyline instance with respect to U is equal to the probability that p is not dominate by U, which can be represented by:

$$Pr(U \not\prec p) = 1 - \sum_{q \in U,\ q \prec p} Pr(q) \tag{3.2}$$

Then we obtain the probability that p is not dominated for all objects in OS except V, which is regarded as the likelihood of p to be a global skyline instance. As objects are independent of each other, $SKYProb(p)$ ($p \in V$) can be computed as follows:

Symbol	Meaning
OS	object set
$SKYProb(.)$	the skyline probability of an instance or an object.
p	an instance
U, V	an object
$Pr(p)$	probability of an instance
t	probability threshold t in $t-$skyline set
$TSKY$	object set with skyline probability larger than t
V^A	an angle partition
P	pivot point set

Table 3.1: Symbolic notations.

$$SKYProb(p) = \prod_{U \in OS,\, U \neq V} (1 - \sum_{q \in U,\, q \prec p} Pr(q)) \tag{3.3}$$

Finally, for object O_v, we are able to obtain the probability that O_v is a skyline object using:

$$SKYProb(V) = \sum_{p \in V} Pr(p) SKYProb(p) \tag{3.4}$$

Given a threshold t, a *p-skyline query*, denoted as $TSKY(OS)$, retrieves all the objects whose $SKYProb(o)$ is larger than p.

$$TSKY(OS) = \{o \in OS | SKYProb(o) > p\} \tag{3.5}$$

Take figure 3.1 as an example. We assume that all instances of an object are assigned the same weight. The instance skyline probabilities are firstly computed based on Equation 3.3. For p_1, no instance dominates it, so $SkyProb(p_1) = 1$; for p_2, $SkyProb(p_2) = 1 \times \frac{2}{3} \times \frac{2}{3} = \frac{4}{9}$, since w_1 and q_3 dominates it; as q_3 dominates p_3, $SkyProb(p_3) = \frac{2}{3}$. Then $SkyProb(P) = 1 * 0.5 + \frac{4}{9} * 0.2 + \frac{2}{3} * 0.3 = 0.789$. Similarly, $SkyProb(Q) = 0.833$, $SkyProb(W) = 0.844$. Given the threshold $t = 0.8$, SKYProb returns the objects set consisting of Q and W.

3.3.2 MapReduce Framework

A typical MapReduce framework consists of two user-defined functions: map and reduce. For every record in the input data sets, the map function will partition it into a sorted set of intermediate results. Worker nodes redistribute data based on the output keys (outputted from the map function), such that all data belonging to one key is able to be accessed locally in the reduce step. This is called the shuffle step. The reduce function fetches the data, from individual partition. Provided by the map function, Reduce process produces the final output data. The map and reduce function could be formally defined: $map(k_1, v_1) \rightarrow list(k_2, v_2)$ and $reduce(k_2, list(v_2)) \rightarrow list(k_3, v_3)$.

The MapReduce infrastructure automatically renders the data split, storage and replication without manual intervention.

3.3.3 Angular Partitioning

Angle-based space partitioning scheme [81] transforms data point from Cartesian coordinate to hyper-spherical coordinate. In the first map phase, we use angle-based partitioning strategy to map data points into N partitions. The main advantage of angular transformation is that this strategy being able to filter data points as many is possible since the left bottom corner data points dominate most other data points in an angle.

Given a d-dimensional data point $p = [p_1, p_2, \ldots, p_d]$, the hyperspherical coordinates of p include a radial coordinate r and $d-1$ angular coordinates $\varphi_1, \varphi_2, \ldots, \varphi_{d-1}$ [81], denoted by $\{r, \varphi_1, \varphi_2, \ldots, \varphi_{d-1}\}$, where r is the distance to the origin, and φ_i represents an angular coordinate $0 \leq \varphi_i \leq \frac{\pi}{2}$. Similarly, an angle-based space partition is represented by $V^A = \{\Phi_1^A, \Phi_2^A, \ldots, \Phi_{d-1}^A\}$, where Φ_i^A is an angle range $[\varphi_i^j, \varphi_i^k]$ in the i^{th} dimension.

3.4 The Naive Approaches

In this section, we introduce a baseline approach of using MapReduce framework to process probabilistic skyline queries. The most straightforward solution is to partition objects in a pair-wise way. Then the dominance check runs between every pair of two objects (see Equation 3.2). After that, each instance p's skyline probability $SKYProb(p)$ (see Equation 3.3) is obtained by the product of output from previous results. Finally we merge all instances' intermediate skyline probability into the final object skyline probability in Equation 3.4. In this section, we elaborate on an intuitive solution to probabilistic skyline queries in a three-phase MapReduce framework.

Assume that the number of objects is m, and the number of instances is n. In the first phase, we create pair-wise comparisons between any two objects. Every pair of two distinct objects is assigned one key, and the number of pairs is C_m^2. The replication of o_i is performed $m-1$ times in order to let o_i forms pairs with all other objects. The map stage collects all objects and assigns a key considering every pair, and transmits all pairs to the reduce phase. The reduce function fetches every object pair (one of C_m^2 pairs), and two block nested loop comparisons are performed. Say the objects are U and V. For every q in V, $Pr(U \not\prec q)$ is computed (Equation 3.2). Reversely, we compute $Pr(V \not\prec p)$for every p in U. The results from all reducers are written into HDFS.

In the second phase, the target is to obtain instance skyline probability. we read the output from the first phase and use the instance ID key as the partitioning key at the end of Map phase. Then every reducer groups all intermediate results for each instance and do an product of them (Equation 3.3). The output from the second phase reducers is outputted into n (n is the number of instances of all objects) files in HDFS.

Similarly, in the third phase, output generated by the second phase is read by map function and every object ID is assigned as a new partitioning key for reduce phase. In the Reduce function, We sum all instances' skyline probability for one object (Equation 3.4) as the final result, that is, the final object skyline probability. Then $t-$skyline set is obtained easily by filtering out object with skyline probability larger than t.

It is obvious that many operations in the second and third phases are similar and therefore this approach suffers much from high I/O cost. One possible way to alleviate the I/O cost is that we combine the second and third phases. In the second phase, we let the object ID is assigned as the partitioning key for the reduce phase. In the reduce phase, all intermediate skyline results (from Equation 3.2) are grouped for each instance. Then we compute instance skyline probabilities using Equation 3.3. Afterwards, we calculate the skyline probabilities for objects.

There are several restrictions of the aforementioned naive approaches. First, how to partition objects evenly is a challenging problem, because the number of the instances in different objects might vary significantly. Second, we don't know the distribution of instances of one object in advance. If all the instance of one object always appear in the right-corner of coordinate axis, and the object is certain to be a non-skyline object. A desired approach should prune object as early as possible. Third, the duplication overhead heavily degrade the performance, since objects are compared in a pair-wise way, which leads to a very high I/O cost.

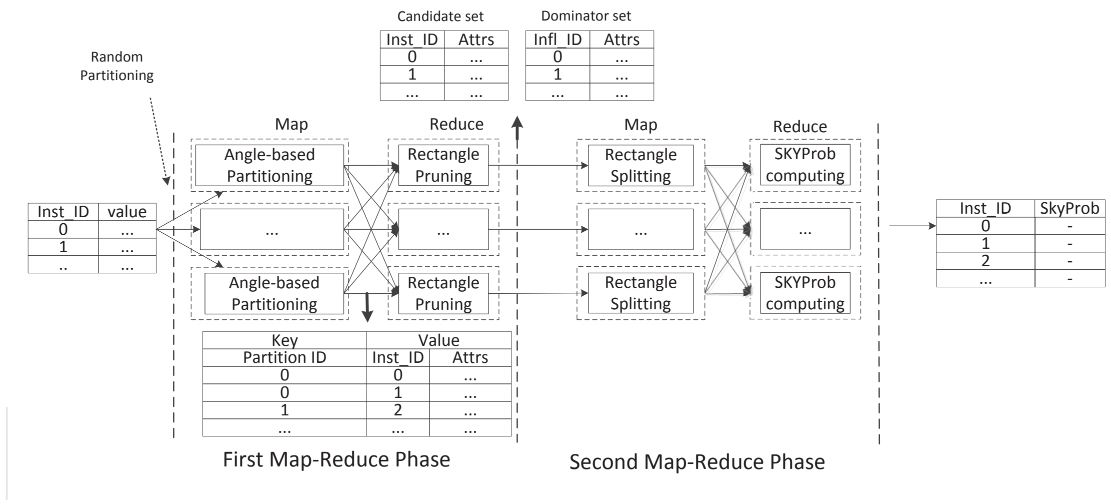


Figure 3.3: An overview of parallel skyline processing using MapReduce.

3.5 Hybrid MapReduce Framework

3.5.1 Overview of Our Approach

To overcome the aforementioned shortcomings, we propose the two-phase MapReduce algorithms, namely *MR-SkyProb*, as shown in Figure 3.3. *MR-SkyProb* contains two MapReduce phases. In the first phase, it contains two steps:

(1) **filtering out unqualified objects:** After instances of objects are mapped to N reducers through angle-based partition scheme, unqualified objects (i.e., objects that is not returned in query results) are filtered out based on proposed pruning rules.

(2) **collecting candidates objects:** Objects which are not filtered out in this phase come to be candidate objects, the set of which is named as O_c. Instances dominating candidate objects are collected, and its set is named by I_p.

In the second MapReduce phase, all candidate objects are distributed to separate machines and object skyline probability is computed locally. Afterwards, $t-$skyline object set is derived.

3.5.2 Pruning Rules

Filtering objects which will not be returned in the query result set is an efficient way to reduce computation complexity. In this section, we investigate three pruning rules to filter out unqualified objects.

Given an object $U = \{p_1, p_2, \ldots, p_m\}$, an d-dimensional region $R(U)$ can be created to include all instances of U. The minimum corner U_{min} of $R(U)$ is represented by a $d-$dimensional virtual point $(\min_{p_i \in U} p_i[1], \min_{p_i \in U} p_i[2], \ldots, \min_{p_i \in U} p_i[d])$, where p_i is an instance in U. Similarly, we use U_{max} to denote the maximum corner point of U, that is, $(\max_{p_i \in U} p_i[1], \max_{p_i \in U} p_i[2], \ldots, \max_{p_i \in U} p_i[d])$

Lemma 3.5.1 *Given $U, V \in O$,* if *$U_{max} \prec V_{min}$, then $SKYProb(V) = 0$, $V \notin TSKY(OS)$.*

Lemma 3.5.2 *Given $U \in O$, an instance $p \notin U$,* if *$U_{max} \prec p$, $SKYProb(p) = 0$.*

Lemma 3.5.3 *Given $U \in O$, $U = \{p_1, p_2, \ldots, p_m\}$, $|U| = m$, U is an unqualified object and can be filtered out from the result if $1 - \sum_{i=1}^{n} Pr(p_i)(SKYProb^{+}(p_i) - 1) < p$. Here we assume one partition V^A has n instances out of m ($n <= m$) for U, containing instances $\{p_1, p_2, \ldots, p_n\}$. We let $SKYProb^{+}(p_i)$ represent the local skyline probability of instance p_i computed in a particular partition.*

3.5.3 The First MapReduce Phase

Here we discuss the details of the first MapReduce phase.

Angular Partitioning

Algorithm 9 displays the working procedure of the first MapReduce phase. The map phase projects data points (instances) to N machines based on angular partition scheme. The defined map function determines every instance's hyper-spherical coordinate, and maps it to the designated

Algorithm 9: First Phase Procedure

```
8.1  Algorithm map()
         Input: object set $OS$, angle partition array $a$
8.2      foreach object $u \in OS$ do
8.3          foreach instance $p \in u$ do
                 /* $S^3$ Decide the angle partition which have $p$      */
8.4              $a_j \leftarrow S^3(p)$ ;
                 /* send request to Distributed Memory Server       */
8.5              request(p, u.id) $\rightsquigarrow$ maintain $MAX(u)$ and $MIN(u)$ ;
8.6              output(j, p) ;

8.1  Algorithm Reduce()
         Input: instance Array $Arr_i$, object Maximum and Minimum corner point Array
                $Arr_{max}$, $Arr_{min}$,
8.2      $O_c \leftarrow \emptyset$ ;
8.3      $I_n \leftarrow \emptyset$ ;
         /* Pruning Rule 1                                          */
8.4      foreach object $u \in Arr_{min}$ do
8.5          $O_c \leftarrow u$ ;
8.6          foreach object $v \in Arr_{max}$ && $v \prec u$ do
8.7              $O_c.remove(u)$ ;
8.8              break ;
         /* Pruning Rule 2                                          */
8.9      foreach object $U \in O_c$ do
8.10         foreach instance $p \in U$ do
8.11             $I_n \leftarrow p$ ;
8.12             foreach object $v \in Arr_{max}$ && $v \prec p$ do
8.13                 $I_n.remove(p)$ ;
8.14                 break ;
         /* Compute the local Probabilistic Skyline (Pruning Rule
            3)                                                      */
8.15     $O_c$ = localProbSkyline($O_c$) ;
8.16     return $O_c$, $I_n$;
```

reducer, which is representative of a hyper-spherical space partition. Another crucial job must be done in the map phase is to collect U_{min} and U_{max} given any object U. Since data is distributed across separate machines, it can not aggregate instances iteratively to find one object's maximum or minimum point. To resolve this issue, we build an independent server to compute the maximum and minimum of every object across separate map processes. The data structure maintained in the server follows key-value pattern, where key is the object ID, and value consists of two virtual points representing maximum and minimum corner of this object. When the map function iterates to an instance of object U, it initiates a request to the server maintaining corner coordinates of U. After the map phase completes, the maximum and minimum points of every object are acquired in the server, and will be utilized in the reduce phase. This technique is denoted as uncertainty-aware allocation.

At the beginning of the reduce phase, maximum and minimum corner points of object U is retrieved if some instance of U is transmitted in this partition. Objects are firstly examined if they are able to be pruned under the Lemma 3.5.1 and Lemma 3.5.2. The procedure works as follows. A list L_{min} contains all U_{min} of all objects in this partition; similarly, a list L_{max} contains all U_{max} in this partition. Using SFS [17], an approach to speed up traditional skyline query by presorting instances, data points with the sum of all dimensions of each point is presorted in ascending order. Our algorithm begins with retrieving the starting element from L_{max}, and compares it against all minimum corner points in L_{min} to check the dominance relationship. Any object U whose U_{min} is dominated by any U_{max} is pruned, as it can not be in the $t-$skyline set. Remained objects which can not be pruned out from L_{min} are collected. Based on Lemma 3.5.2, for every instance in these remained objects, we check if an instance p is dominated by some objects' U_{max}. If the dominance relationship occurs ($U_{max} \prec p$), $SKYProb(p)$ is marked to 0.

Filter Optimization by Pre-computation using Pivot Points

Next, we propose an efficient pruning strategy to further filter unqualified objects. Recall that Lemma 3.5.3 introduces a filtering strategy by computing the partial skyline probability $SKYProb^{+}(U)$ of object U. In order to obtain $SKYProb^{+}(U)$, $SKYProb^{+}(p)$ of every instance p from U contained in this partition must be gained. Therefore, the challenge comes from how to efficiently collect the probability of every instance p being a skyline instance. The straightforward method

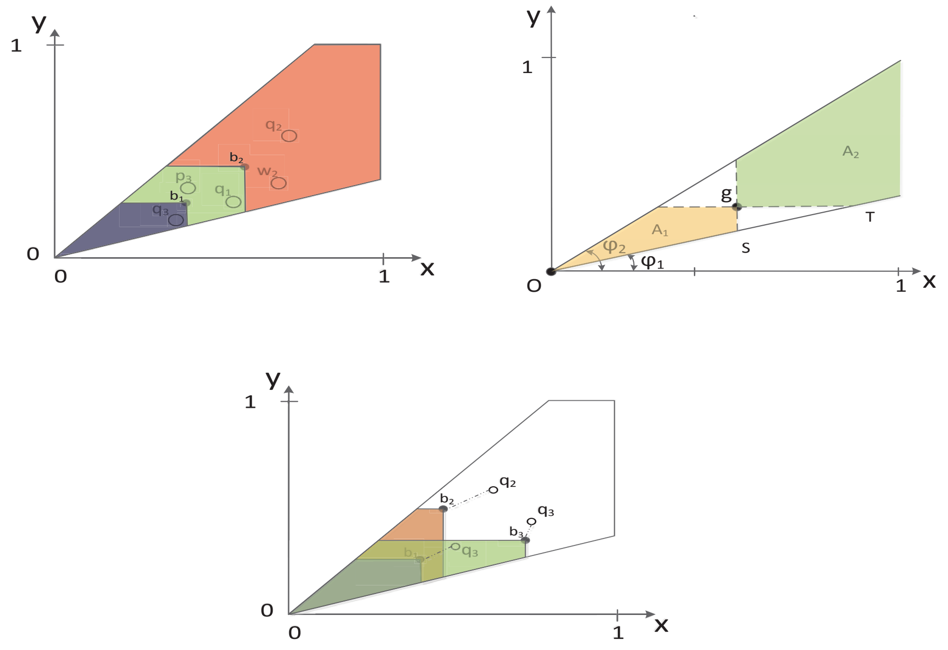


Figure 3.4: The visual exhibition of Optimization Capability. (a) A example shows how pivot points speed up processing. (b) Optimization Capability derivation of one pivot point. (c) Optimization Capability of multiple pivot point.

of computing $SKYProb^{+}(p)$ is to iterate every instance in this partition, and compare the dominance relationship between p and other local instances in this partition. Obviously this approach performs inefficiently.

Given a rectangle area containing instances, one interesting observation is that data points close to maximum boundary of the area are often easily dominated by data points close to minimum boundary. The intuitive thought is to associate data points close together into a group, and information is collected in every group before actual probabilistic skyline processing is launched. Pivot point set P is defined as a set of d-dimensional data points whose cardinality is m, $P = \{b_i : 1 \leq i \leq m\}$. A_i is denoted as the d-dimensional hyper-rectangle, the minimum corner of which is the origin, and the maximum corner of which is b_i. Let each area A_i consists of an n-dimensional vector $V_i = [p_{r1}, p_{r2}, \ldots, p_{rn}]$, where n is the number of objects in this partition (reducer), and p_{ri} gathers the sum of instance (from) existing probability in A_i. After built, these vectors will be able to repeatedly contribute to the skyline probability computation. If some point is dominated by b_i, it is dominated by all instances in A_i. It is able to speed up processing by not comparing instance domination relationship for all instances in A_i, but applying the pre-computed information directly.

```
Algorithm 10: Local Skyline Computation
Input: object set $OS$
Output: candidate object $O_c$
10.1  $O_c \leftarrow \emptyset$ ;
10.2  build pivot point set $P$ from $OS$ ;
10.3  foreach pivot point $p_i \in P$ do
10.4      construct $V_i$ from instances in $A_i$ ;
10.5  foreach object $u \in OS$ do
10.6      foreach instance $p \in u$ do
10.7          $p_i \leftarrow$ the most dominating pivot point from $P$ ;
10.8          $V_p \leftarrow V_i$ ;
10.9          $S_A \leftarrow$ instances partially dominates $p$ ;
10.10         foreach instance $q \in S_A$ do
10.11             if $q \prec p$ then
10.12                 $V_p \leftarrow q$ ;
10.13         SkyProb$^+$(p) $= \prod_{w \in V_p}(1 - w)$
10.14     SkyProb$^+$(U) $= \sum_{p \in U} Pr(p)$SKYProb$^+(p)$;
10.15     if SkyProb$^+$(U) $\geq p$ then
10.16         $O_c \leftarrow U$ ;
10.17 return $O_c$;
```

Take Figure 3.4(a) as an example, following the hotel data introduced in Section **??**. Assume that the angle of partition V^A is between ϕ_1 and ϕ_2. Two virtual data points b_1, b_2 separate V^A into three areas, denoted by 3 colors, blue, green and red in order. It is observed that q_2 is located at red area and q_2 is always dominated by any instance in blue and green areas since b_2 dominates q_2, and q_2 is also affected by partial instances in red area (w_2 in Figure 3.4(a)). According to this observation, we are able to precompute the sum of existing instance probabilities in blue and green areas, and use the intermediate result for computing SKYProb$^+(q_2)$.

The placement and choice of P will be discussed in the later section. After P is deployed well, the detailed computing procedure is as follows. Recall that A_i consists of an n-dimensional vector $V_i = [p_{r1}, p_{r2}, \ldots, p_{rn}]$, where n is the number of objects in this partition. Let V_i is initialized to 0. Assume an instance p from object O_w is in A_i. p's instance probability is added to p_{rw} in V_i. Similarly, every instance in A_i is iterated and V_i is maintained.

After maintenance of V_i ($1 \leq i \leq |P|$) is completed, every instance q in the partition is iterated to compute $SkyProb^{+}(q)$. Assume an instance q lies in area A_m. It is necessary to locate a pivot point b_i which overwhelmingly dominates q, which will incur the most domination power. In our implementation, we let A_i with the most instances as area having the most domination power. Since we already have information for each area A_i, the pivot point b_j domination q with the largest instance number is extracted from pivot point candidates. That is, A_j fully dominates q. In addition, we look for area set S_A which partially dominates q: S_A = $\{A_i | \exists p \in A_i, p \prec q, i \neq j\}$. An aggregating vector $[p_{r1}, p_{r2}, \ldots, p_{rn}]$ is maintained for comparing domination interrelation in S_A. If any instance in S_A dominate q, the instance's existing probability is added to the vector. After that, aggregated by the vector V_j retrieved before, the product of every element in the object vector is the final $SkyProb^{+}(q)$. Given object U, we are able to compute $SkyProb^{+}(U)$ easily. If $SkyProb^{+}(U)$ is smaller than t, we safely prune U out according to Lemma 3.5.3.

Definition 1 *Probability skyline object candidates set O_c is an object set $O_c = \{U \in OS | SKYProb^{+}(U)$ $\geq t\}$, where OS is the global object set.*

Definition 2 *Influential instance set I_p is an instance set $I_p = \{p \in I_f | \exists p^{'} \in C_o, p \prec p^{'}\}$, where I_f is the instance set, which .*

Another job during the reduce phase is to return the probability skyline object candidates and affecting instance, both of which is required in the next MapReduce phase. Definition 1 and 2 give the definition of O_c and I_p. During the reduce phase, C_o is collected when comparing maximum and minimum corner points, and objects filtered by pivot point are removed from C_o. Line 4-10 of Algorithm 9 and Line 19-21 of Algorithm 10 describe the process. I_p is obtained when applying Lemma2, and we illustrates the procedure in Line 11-18 of Algorithm 9. After that, it moves forward to the second MapReduce phase.

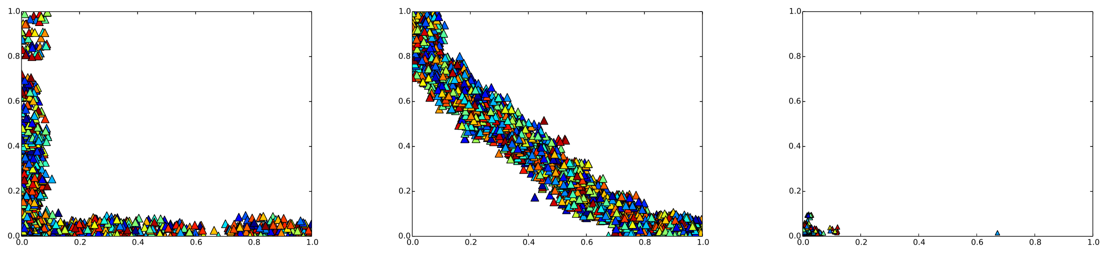


Figure 3.5: The visualization of data shape after the first MapReduce Phase based on the original input data distribution. (a) Independent distribution. (b) Anti-correlated distribution. (c) Correlated distribution.

3.5.4 The Second MapReduce Phase

Naive approach and an study on data pattern

After the first MapReduce phase is completed, the skyline candidate object set O_c and Influential instance set I_p is collected. The next step is to compute the real instance skyline probability. The most straightforward parallel algorithm is to broadcast I_p to all machines, and distribute objects in O_c evenly to machines and compute locally. This baseline method will be evaluated in the later section.

Before illustrating our optimized algorithm, we study the distribution shape of remained candidate objects after the first MapReduce phase, since the remained data decides the final step of merging. An experiment is conducted to visually investigate the data shape. Under the two-dimensional space, objects are distributed between [0, 1] in domain of each dimension. Three types of object distribution (independent distribution, correlated distribution, and anti-correlated distribution) are tested respectively. The cardinality of objects is 5000. Figure 3.5 shows the result. It is observed fewer data is left in the second MapReduce phase, and data from right top corner is all filtered out in independent and Anti-correlated data.

Optimization by Grid-based Partitioning

Inspired by Figure 3.5, we propose an approach to efficiently evaluate global skyline probabilities of remained candidate objects. Recall that we have two remained sets of data, object set C_o

and instance set I_p. Each candidate object in C_o may be dominated by some instances from I_p. In order to let parallel partitions execute probabilistic skyline query, we propose a dynamic mapping strategy to allocate data to target machine.

We firstly introduce how to distribute C_o. We let instances in C_o is sorted by ascending value in one dimension, x axis in default. Given the number of reducers n_r, instances are distributed to n_r equal-sized buckets along x values. As a result, one bucket of instances constitute a grid. Take Figure 3.6 as an example. S_3 is a grid denoted by green color, and contains a quarter number of all instances.

Besides C_o, instances from I_p is disseminated to the target reducer. Given one grid collecting C_o data, the instances affecting candidate instances in the reducer lie in a region from the origin to maximum boundary of the grid. In Figure 3.6, M_3 is the maximum boundary point of grid S_3. Then instances in the area from the origin to M_3 are sent to the target reducer. We repeat the operation given every grid of C_o data.

After data is mapped to the target reducer, in the reduce phase, the skyline probability computation starts with the iteration of every instance from C_o. Each instance computes its skyline probability by comparing against instances from I_f. After that, instance skyline probability for all instances is outputted to Distributed File System.

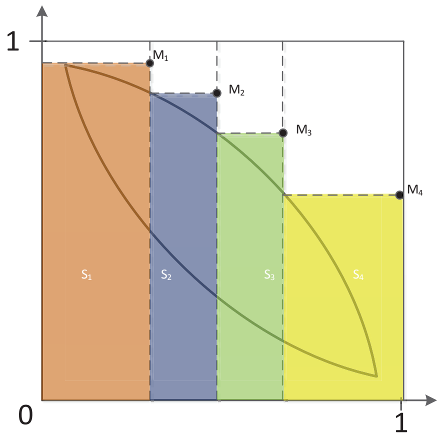


Figure 3.6: The visual exhibition of Grid-based partitioning.

After every instance in C_o obtains its instance skyline probability, the second phase is completed. object skyline probability is grouped by iterating all instance skyline probability by Equation 3.4. $t-$skyline object set is easily obtained by retrieving objects U whose skyline probability SkyProb(U) is larger than or equal to t.

3.6 PIVOT POINTS-BASED OPTIMIZATION

Next we elaborate on the optimization capacity of pivot points and study how to select and place pivot points.

3.6.1 Optimization Capability of One Pivot Point

In order to visualize the capability power, assuming that data points are uniformly distributed, we initially study a simple case when only one pivot point exists in 2-dimensional space. Higher dimensional cases can be easily extended. Given a 2-dimensional angle, the pivot point divides the angle area into two parts denoted as A_1 and A_2. A_1 lies in the left bottom to g, and A_2 lies in the right top to g. Figure 3.4(b) depicts the scenario. The optimization capability could be represented by how many data points in A_2 is able to repeatedly use A_1 directly without comparing one by one, since intermediate sum of instance probability in A_1 has been computed in advance. Therefore, given a pivot point g, the optimization capability of g can be represented as:

$$OC(g) = \sum_{p \in A_2} S_{A_1}(p_x, p_y, \phi_1, \phi_2) \tag{3.6}$$

where S_{A_1} represents the volume of A_1, p_x and p_y denote p's coordinates, as depicted in Figure 3.4(b). Given that the pivot point g is fixed, an instance p is able to repeatedly use A_1's pre-computed results if p is in A_2.

S_{A_1}, the area of dominance region to g, is defined by $0 \leq x \leq g_x$, and $xtan(\phi_1) \leq y \leq min(xtan(\phi_2), y_g)$. Therefore, S_{A_1} is represented by

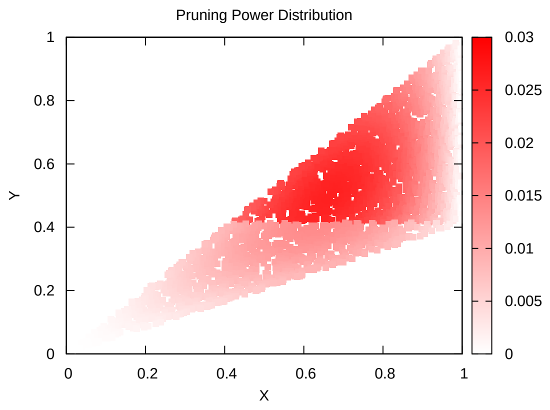
$$S_{A_1} = \int_{0}^{x_g} \int_{xtan(\phi_1)}^{min(xtan(\phi_2),y_g)} dydx \tag{3.7}$$

In order to find the optimal placement of pivot point g, we propose a strategy applying random selection. One arbitrary pivot point g located ϕ_1 and ϕ_2 is randomly selected. S_{A_1} of g is easily obtained from Equation 3.7 given the location of g. To compute the optimization capability $OC(g)$, we sample quantities of random points evenly between ϕ_1 and ϕ_2, and compute the sum of optimization capability of all sampled points. Because the points are sampled uniformly in the partition, the number of sampled points falling in A_2 is in accord with the area of A_2. Then $OC(g)$ is regarded as S_{A_1} and the product of number of sampled points falling in A_2. Figure 3.7(a) shows the graph of optimization capability density for pivot point located at angle range of $[\pi/8, \pi/4]$. The darker areas indicate a higher optimization capability. It is found that the pivot points in $(0.4 \leq x \leq 0.6, 0.4 \leq y \leq 0.6)$ area have the highest optimization capability.

3.6.2 Optimization Capability of Multiple Pivot Points

In this subsection, we investigate the impact of multiple pivot points on the optimization capability. As introduced in the previous sections, pivot point set P is defined as a set of d-dimensional data points. Under the multiple pivot points environment, every instance must nominate one pivot point to utilize precomputed information, and therefore have multiple options. Take Figure 3.4(c) as an example. Three pivot points b_1, b_2, b_3, and three instances q_1, q_2, and q_3 are in the partition. Both b_1 and b_2 dominates q_2. Then either b_1 or b_2 can be q_2's pivot point. However, given multiple pivot points, we let each instance x's optimization capability is determined by the largest dominance region of a pivot points. In the above example, q_2 selects b_2 as the pivot point since S_{b_2} is larger than S_{b_1} and produces more optimization capability. Then, given multiple pivot points scenario, the optimization capability of an instance p is represented by A_i which dominates p and has the largest area.

It is obvious that the Optimization Capability of P, OC(P) is dependent on the cardinality of P and the placement of P in the partition. To find the optimal OC(P), we randomly compose



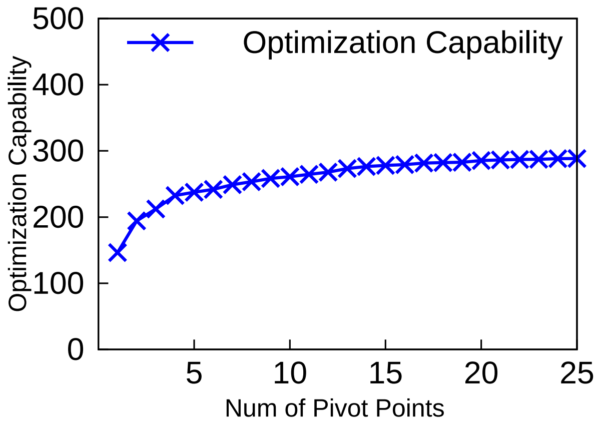


Figure 3.7: A synthetic experiment to show Optimization capability. (a) Optimization capability given one random pivot point. (b) Optimization capability of multiple pivot points.

the number of $|P|$, and randomly generate combinations of P, and assign P with a combination having the largest OC(P). Let D is a angle partition area. Pivot point number $|P|$ is randomly chosen from 1 to fixed value m. Given the number of pivot points $|P|$, We generate a number of random combination of pivot points in arbitrary fashion. For a fixed combination of P, we compute the Optimization Capability of every instances in D based on the rule introduced above. We sum all of them as the optimization capability of this combination. Then P is designated as the combination with the largest optimization Capability. The above introduces the algorithms to obtain the optimal combination of P. If the number of instances in D is too high, it is highly inefficient to compute OC(p) for every instance p. Instead, we randomly selected p from instance pool in D.

Parameters	Default Values
d (dimensions)	4
t (No. of machines)	10
$\|O\|$ (object number)	2^{21}
$\|P\|$ (No. of pivot points in P)	10
$numAngle$ (No. of Angles in the first Phase)	27
$numPartition$ (No. of partitions in the second Phase)	5

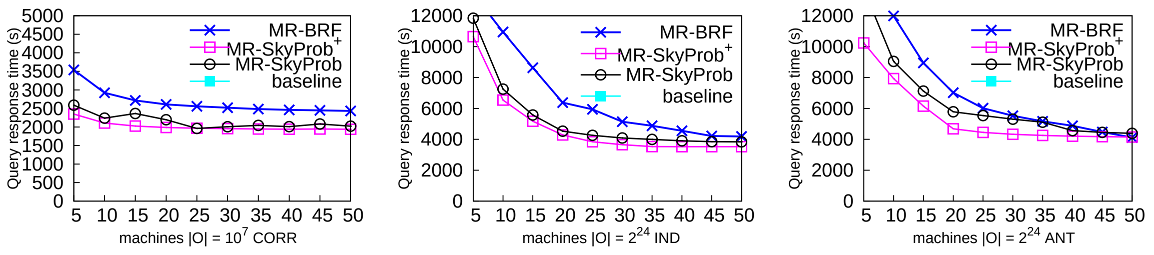
Table 3.2: Experimental parameter values.

To evaluate the effects of the number of pivot points in optimal capability, we experiment the optimal capability in the 2-dimensional space. We partition the 2-dimensional space into into 4 partitions evenly, and optimization capability is evaluated in each partition follows . The optimization capability computation of each partition $part$ follows the above approach. Given the number of P m, after the combination of P in every partition $part$, OC(part) is obtained, the OC(m) is regarded as the mean of OC(part). Figure 3.4(a) shows the OC trend with $|P|$ from 1 to 25. It is found that, OC increases when $|P|$ is elevated, but the slope is gradually dropped.

3.7 Experimental Validation

In this section, we present an experimental evaluation of the proposed algorithm MR-SkyProb and MR-SkyProb$^+$ against other approaches on several synthetic and real-world large datasets.

3.7.1 Experimental Setup



(a) Correlated (b) Independent (c) Anti-Correlated

Figure 3.8: Vary No. of Machines

We implemented MR-SkyProb using java on top of Apache Hadoop 2.2.0. The experiments were conducted on a 50-node shared-nothing cluster. Each node was equipped with one Intel Xeon 2.4 GHz processor, 2 GBytes of memory, running 64bit Ubuntu Linux Server 14.04LTS with Linux 3.13 kernel. All nodes were connected by 1 GigaBit Ethernet network.

Real dataset: TripAdvisor Dataset[1] is employed in our experimentation. Consisting of reviews and ratings for hotels from different authors, it has 7 attributes (e.g. room rating, location rating, and etc) in total. The dataset includes 9129 hotels and around 700K reviews, which can be regarded as 9129 objects and 700k instances.

Synthetic dataset: A group of synthetic datasets is generated to conduct extensive evaluation. The dimensionality of datasets is varied from 2 to 64. Objects are distributed between [0, 1] in domain of each dimension. We build three varieties of object distribution (independent distribution, correlated distribution, and anti-correlated distribution), which are typically used to evaluate the performance of probabilistic skyline algorithms [68] [68]. The default cardinality of objects is 2^{21}. For each object, instances are created under a rectangle region, the edge size of which follows a normal distribution in range [0, 0.02] with an expectation of 0.1 and a standard deviation of 0.025. The number of instances in each object follows uniform distribution in range [1, 200]. The sizes of resulting synthetic datasets are varied from 88MB to 86GB depending on the number of object ($|O|$), the number of dimensions (d) and the number of instances of each object (l).

We denote our proposed algorithms as MR-SkyProb and MR-SkyProb$^+$, which is an extension of MR-SkyProb with second MapReduce phase optimization. PSMR [24], and PS-BRF-MR [67] are also evaluated. Our assumption is based on the fact that the instances are disordered along several machines. PS-QPF-MR [67] is not applicable in this situation since it requires that the instances from one object are placed in one machine prior to the query process. In addition, we have one baseline method, that is a MR-SkyProb approach without the first MapReduce phase filtering.

[1]http://times.cs.uiuc.edu/ wang296/Data/LARA/TripAdvisor/

3.7.2 The effect of Number of machines

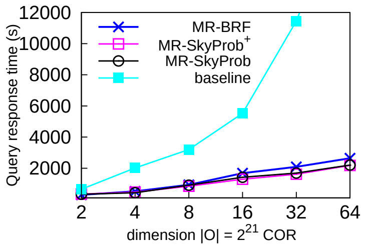
Firstly, we investigate the effect of Number of machines. In order to explore the scalability of the proposed methods, the number of objects is set to 2^{24} in this experiment set. In Figure 3.8, since the baseline approach performs too slowly (the measured time is always larger than 5,000s in Correlated data, 12,000s in Independent and Anti-Correlated data), we can't observe the baseline trend. With the number of machines growing, the query time of all approaches decreased overally. It is observed that MR-SkyProb$^+$ always outperforms against the other three approaches, especially in Correlated and Independent distribution. There is no large difference of query efficiency between MR-SkyProb and MR-SkyProb$^+$ when data shape is Correlated or independent, but the obvious gap can be found easily in anti Correlated test.

3.7.3 Varying dimensions

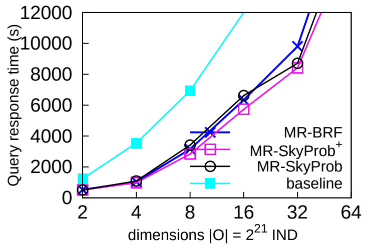
In this set, we investigate the effect of data dimensionality. The number of dimension is varied from 2 to 64 under the three different distributions. The overall trend is that the query response time is increased when the number of dimension is growing. Figure 3.9(a) shows that MR-SkyProb$^+$ always outperforms against the baseline approach and MR-BRF under the Correlated Distribution dataset. When the number of dimensions keeps growing, the gap between MR-SkyProb and MR-BRF increases. Baseline is always the slowest, and MR-SkyProb$^+$ is always quicker than MR-SkyProb in different distributions. In Figure 3.9(c), MR-BRF outperforms against the two MR-SkyProb approaches when the number of dimension is below 16, but MR-SkyProb$^+$ become quicker when number of dimensions grows after 32.

3.7.4 Varying Cardinality

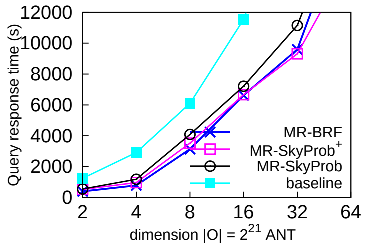
In the third set of experiment, we evaluated the performance of baseline and MR-Skyline upon different cardinality. In Figure 3.10, it is observed that the elapsed query time is elevated when number of objects increases, and baseline is extremely slowest compare to the other two methods. The gap between the baseline and MR-SkyProb enlarges with the increments of object



(a) Correlated



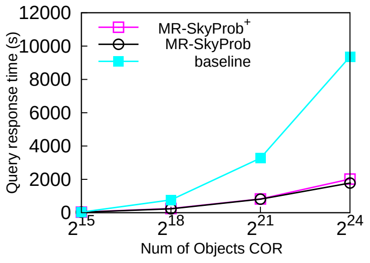
(b) Independent



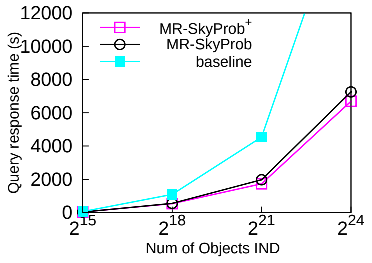
(c) Anti-Correlated

Figure 3.9: Vary Dimension.

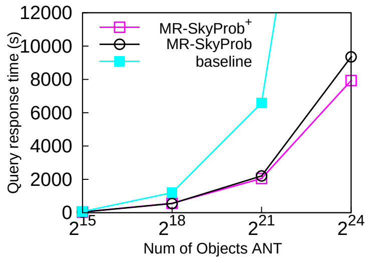
cardinality in Figure 3.10(a). Similar results occur for data following Anti-correlated distribution and Independent distribution. One interesting observation is that MR-SkyProb$^+$ is slower than MR-SkyProb when the cardinality is low (2^{15} and 2^{18}). The reason is that the second phase does not perform much work, because large amount of data is filtered during the first MapReduce phase and fewer objects are left during the second MapReduce phase.



(a) Correlated



(b) Independent

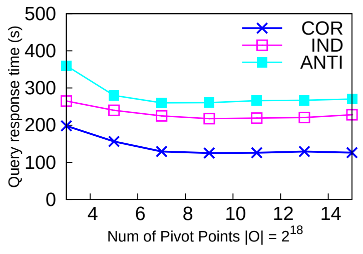


(c) Anti-Correlated

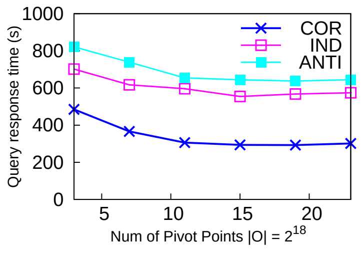
Figure 3.10: Vary Cardinality.

3.7.5 The Effects of Pivot Point

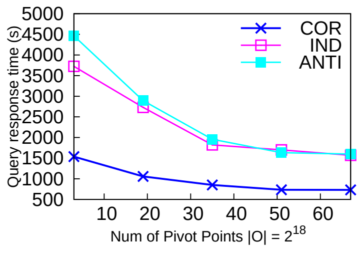
In this experiment set, we study the the effects of Pivot-Points-Based Optimization power, by adjusting the number of pivot point to observe the performance change. We measure the performance of data of three shape distributions under 4, 8 and 16 dimensions. Dark blue line, red line and light blue line represent the correlated distribution, independent distribution and ANTI correlated distribution respectively. Figure 3.11(a) shows that the red line decreases until 9 pivot points, and rises afterward. Dark Blue line meets the optimized concave point when number of pivot point is around 11. In Figure 3.11(b), three lines' pit points are all at around 15 for 8-dimensional data. The optimized pivot point number increases when the dimension rises after 15. In Figure 3.11(c), all three lines keep dropping when the number of dimensions grows. The reason behind this is that data with 16 dimensions needs large number of pivot point to achieve the optimized placement for ease of querying.



(a) Pivot Point Num Variation Dim4



(b) Pivot Point Num Variation Dim8

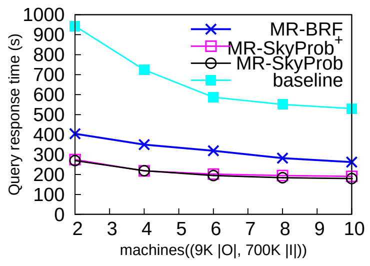


(c) Pivot Point Num Variation Dim16

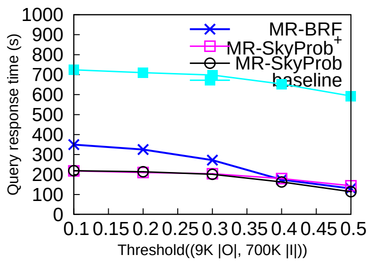
Figure 3.11: Pivot Point Experiment.

3.7.6 Real Dataset Evaluation

In this experiment set, we evaluate the performance of TripAdvisor dataset under two settings. Figure 3.12(a) shows the elapsed time trend due to the changes in the number of machines. It is observed that the query response time dropped when the number of machines is increasing. Baseline always performs the slowest, and MR-SkyProb and MR-SkyProb$^+$ are quicker than MR-BRF. One interesting observation is that the gap between MR-BRF and MR-SkyProb is decreasing. The reason behind is that |O| of the dataset is not large enough, and dimension number is low. Figure 3.12(b) depicts the performance comparisons due to changes of probability threshold t_p. The query response time of all methods is decreasing when probability threshold is increasing, since the filter component is able to filter more objects in the first step when the probability threshold is increasing. The interesting observation is that MR-BRF outperforms against the others when probability threshold is large. The reason behind these results are the way the dataset is distributed.



(a) Various number of machines



(b) Various threshold

Figure 3.12: Real Data Experiment.

Bibliography

[1] Foto N. Afrati, Paraschos Koutris, Dan Suciu, and Jeffrey D. Ullman. Parallel skyline queries. In *ICDT*, pages 274–284, 2012.

[2] Parag Agrawal, Omar Benjelloun, Anish Das Sarma, Chris Hayworth, Shubha Nabar, Tomoe Sugihara, and Jennifer Widom. Trio: A System for Data, Uncertainty, and Lineage. In *VLDB*, pages 1151–1154, 2006.

[3] Christophe Andrieu, Nando de Freitas, Arnaud Doucet, and Michael I. Jordan. An Introduction to MCMC for Machine Learning. *Machine Learning*, 50(1-2):5–43, 2003.

[4] Periklis Andritsos, Ariel Fuxman, and Renee J. Miller. Clean Answers over Dirty Databases: A Probabilistic Approach. In *ICDE*, page 30, 2006.

[5] Lyublena Antova, Christoph Koch, and Dan Olteanu. Query Language Support for Incomplete Information in the MayBMS System. In *VLDB*, pages 1422–1425, 2007.

[6] Mikhail J Atallah and Yinian Qi. Computing all skyline probabilities for uncertain data. In *Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 279–287. ACM, 2009.

[7] Lars Backstrom and Jure Leskovec. Supervised random walks: predicting and recommending links in social networks. In *WSDM*, pages 635–644, 2011.

[8] Christian Böhm, Frank Fiedler, Annahita Oswald, Claudia Plant, and Bianca Wackersreuther. Probabilistic skyline queries. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 651–660. ACM, 2009.

[9] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The Skyline Operator. In *ICDE*, pages 421–430, 2001.

[10] Haiquan Chen, Wei-Shinn Ku, Min-Te Sun, and Roger Zimmermann. The partial sequenced route query with traveling rules in road networks. *GeoInformatica*, 15(3):541–569, 2011.

[11] Haiquan Chen, Wei-Shinn Ku, Haixun Wang, and Min-Te Sun. Leveraging Spatio-temporal Redundancy for RFID Data Cleansing. In *SIGMOD Conference*, pages 51–62, 2010.

[12] Zaiben Chen, Heng Tao Shen, Xiaofang Zhou, and Jeffrey Xu Yu. Monitoring path nearest neighbor in road networks. In *SIGMOD Conference*, pages 591–602, 2009.

[13] Reynold Cheng, Sarvjeet Singh, and Sunil Prabhakar. U-DBMS: A Database System for Managing Constantly-evolving Data. In *VLDB*, pages 1271–1274, 2005.

[14] Zhiyuan Cheng, James Caverlee, and Kyumin Lee. You are where you tweet: a content-based approach to geo-locating twitter users. In *CIKM*, pages 759–768, 2010.

[15] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. Friendship and mobility: user movement in location-based social networks. In *KDD*, pages 1082–1090, 2011.

[16] Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang. Skyline with presorting. In *ICDE*, pages 717–719, 2003.

[17] Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang. Skyline with Presorting. In *ICDE*, pages 717–719, 2003.

[18] Adan Cosgaya-Lozano, Andrew Rau-Chaplin, and Norbert Zeh. Parallel Computation of Skyline Queries. In *HPCS*, page 12, 2007.

[19] Nilesh Dalvi and Dan Suciu. Efficient Query Evaluation on Probabilistic Databases. *The VLDB Journal*, 16(4):523–544, 2007.

[20] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[21] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, pages 1–38, 1977.

[22] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[23] Li Ding, Lina Zhou, Timothy W. Finin, and Anupam Joshi. How the Semantic Web is Being Used: An Analysis of FOAF Documents. In *HICSS*, 2005.

[24] LinLin Ding, Guoren Wang, Junchang Xin, and Ye Yuan. Efficient probabilistic skyline query processing in mapreduce. In *IEEE International Congress on Big Data, BigData Congress 2013, June 27 2013-July 2, 2013*, pages 203–210, 2013.

[25] Pedro Domingos. Markov logic: a unifying language for knowledge and information management. In *CIKM*, page 519, 2008.

[26] Pedro Domingos, Daniel Lowd, Stanley Kok, Hoifung Poon, Matthew Richardson, and Parag Singla. Just Add Weights: Markov Logic for the Semantic Web. In *URSW*, pages 1–25, 2008.

[27] Tobias Emrich, Hans-Peter Kriegel, Nikos Mamoulis, Johannes Niedermayer, Matthias Renz, and Andreas Züfle. Reverse-nearest neighbor queries on uncertain moving object trajectories. In *DASFAA*, pages 92–107, 2014.

[28] Chris Fraley and Adrian E. Raftery. How many clusters? Which clustering method? Answers via model-based cluster analysis. *Comput. J.*, 41(8):578–588, 1998.

[29] Chris Fraley and Adrian E Raftery. Model-based clustering, discriminant analysis, and density estimation. *Journal of the American Statistical Association*, 97(458):611–631, 2002.

[30] Irene Gargantini. An effective way to represent quadtrees. *Commun. ACM*, 25(12):905–910, 1982.

[31] Jennifer Golbeck and Matthew Rothstein. Linking Social Networks on the Web with FOAF: A Semantic Web Case Study. In *AAAI*, pages 1138–1143, 2008.

[32] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD Conference*, pages 47–57, 1984.

[33] Haibo Hu, Dik Lun Lee, and Victor C. S. Lee. Distance indexing on road networks. In *VLDB*, pages 894–905, 2006.

[34] Haibo Hu, Dik Lun Lee, and Jianliang Xu. Fast nearest neighbor search on road networks. In *EDBT*, pages 186–203, 2006.

[35] Ming Hua, Jian Pei, Wenjie Zhang, and Xuemin Lin. Ranking queries on uncertain data: a probabilistic threshold approach. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 673–686, 2008.

[36] Xuegang Huang, Christian S. Jensen, and Simonas Saltenis. The Islands Approach to Nearest Neighbor Querying in Spatial Networks. In *SSTD*, pages 73–90, 2005.

[37] Ravi Jampani, Fei Xu, Mingxi Wu, Luis Leopoldo Perez, Christopher Jermaine, and Peter J. Haas. MCDB: A Monte Carlo Approach to Managing Uncertain Data. In *SIGMOD*, pages 687–700, 2008.

[38] Christian S. Jensen, Jan Kolárvr, Torben Bach Pedersen, and Igor Timko. Nearest neighbor queries in road networks. In *GIS*, pages 1–8, 2003.

[39] Dongwon Kim, Hyeonseung Im, and Sungwoo Park. Computing exact skyline probabilities for uncertain databases. 2011.

[40] Henning Köhler, Jing Yang, and Xiaofang Zhou. Efficient parallel skyline processing using hyperplane projections. In *SIGMOD Conference*, pages 85–96, 2011.

[41] Mohammad R. Kolahdouzan and Cyrus Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *VLDB*, pages 840–851, 2004.

[42] Donald Kossmann, Frank Ramsak, and Steffen Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *VLDB*, pages 275–286, 2002.

[43] Wei-Shinn Ku, Roger Zimmermann, Haojun Wang, and Chi-Ngai Wan. Adaptive nearest neighbor queries in travel time networks. In *GIS*, pages 210–219, 2005.

[44] Ugur Kuter and Jennifer Golbeck. Using probabilistic confidence models for trust inference in web-based social networks. *ACM Trans. Internet Techn.*, 10(2), 2010.

[45] Ken C. K. Lee, Wang-Chien Lee, and Baihua Zheng. Fast object search on road networks. In *EDBT*, pages 1018–1029, 2009.

[46] Ken C. K. Lee, Wang-Chien Lee, Baihua Zheng, and Yuan Tian. ROAD: A new spatial object search framework for road networks. *IEEE Trans. Knowl. Data Eng.*, 24(3):547–560, 2012.

[47] Ken C. K. Lee, Baihua Zheng, Huajing Li, and Wang-Chien Lee. Approaching the Skyline in Z Order. In *VLDB*, pages 279–290, 2007.

[48] Vincent Leroy, Berkant Barla Cambazoglu, and Francesco Bonchi. Cold start link prediction. In *KDD*, pages 393–402, 2010.

[49] Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *TKDD*, 1(1), 2007.

[50] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[51] Feifei Li, Dihan Cheng, Marios Hadjieleftheriou, George Kollios, and Shang-Hua Teng. On Trip Planning Queries in Spatial Databases. In *SSTD*, pages 273–290, 2005.

[52] Feifei Li, Ke Yi, and Wangchao Le. Top-k queries on temporal data. *VLDB J.*, 19(5):715–733, 2010.

[53] Jiajia Li, Botao Wang, and Guoren Wang. Efficient probabilistic reverse k-nearest neighbors query processing on uncertain data. In *Database Systems for Advanced Applications, 18th International Conference, DASFAA 2013, Wuhan, China, April 22-25, 2013. Proceedings, Part I*, pages 456–471, 2013.

[54] Rui Li, Shengjie Wang, Hongbo Deng, Rui Wang, and Kevin Chen-Chuan Chang. Towards social user profiling: unified and discriminative influence model for inferring home locations. In *KDD*, pages 1023–1031, 2012.

[55] David Liben-Nowell and Jon M. Kleinberg. The link prediction problem for social networks. In *CIKM*, pages 556–559, 2003.

[56] Ching-Yung Lin, Nan Cao, Shixia Liu, Spiros Papadimitriou, Jimeng Sun, and Xifeng Yan. SmallBlue: Social Network Analysis for Expertise Search and Collective Intelligence. In *ICDE*, pages 1483–1486, 2009.

[57] Wei Lu, Yanyan Shen, Su Chen, and Beng Chin Ooi. Efficient Processing of k Nearest Neighbor Joins using MapReduce. *PVLDB*, 5(10):1016–1027, 2012.

[58] Kurt Mehlhorn. Data structures and algorithms 3: Multi-dimensional searching and computational geometry, volume 3 of eatcs monographs on theoretical computer science, 1984.

[59] Lilyana Mihalkova and Matthew Richardson. Speeding up inference in statistical relational learning by clustering similar query literals. In *ILP*, pages 110–122, 2009.

[60] Michael D. Morse, Jignesh M. Patel, and H. V. Jagadish. Efficient Skyline Computation over Low-Cardinality Domains. In *VLDB*, pages 267–278, 2007.

[61] Kasper Mullesgaard, Jens Laurits Pedersen, Hua Lu, and Yongluan Zhou. Efficient skyline computation in mapreduce. In *17th International Conference on Extending Database Technology (EDBT)*, pages 37–48, 2014.

[62] Alper Okcan and Mirek Riedewald. Processing theta-joins using MapReduce. In *SIGMOD Conference*, pages 949–960, 2011.

[63] Pedro Oliveira and Paulo Gomes. Instance-based probabilistic reasoning in the semantic web. In *WWW*, pages 1067–1068, 2009.

[64] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82, 2005.

[65] Dimitris Papadias, Jun Zhang, Nikos Mamoulis, and Yufei Tao. Query Processing in Spatial Network Databases. In *VLDB*, pages 802–813, 2003.

[66] Sungwoo Park, Taekyung Kim, Jonghyun Park, Jinha Kim, and Hyeonseung Im. Parallel Skyline Computation on Multicore Architectures. In *ICDE*, pages 760–771, 2009.

[67] Yoonjae Park, Jun-Ki Min, and Kyuseok Shim. Processing of probabilistic skyline queries using mapreduce. *Proceedings of the VLDB Endowment*, 8(12):1406–1417, 2015.

[68] Jian Pei, Bin Jiang, Xuemin Lin, and Yidong Yuan. Probabilistic skylines on uncertain data. In *Proceedings of the 33rd international conference on Very large data bases*, pages 15–26. VLDB Endowment, 2007.

[69] Sheldon M. Ross. *Introduction to Probability Models, Ninth Edition*. Academic Press, 2006.

[70] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, 16(2):187–260, 1984.

[71] Hanan Samet, Jagan Sankaranarayanan, and Houman Alborzi. Scalable network distance browsing in spatial databases. In *SIGMOD Conference*, pages 43–54, 2008.

[72] Jagan Sankaranarayanan, Houman Alborzi, and Hanan Samet. Efficient query processing on spatial networks. In *GIS*, pages 200–209, 2005.

[73] Gideon Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978.

[74] Jude W. Shavlik and Sriraam Natarajan. Speeding Up Inference in Markov Logic Networks by Preprocessing to Reduce the Size of the Resulting Grounded Network. In *IJCAI*, pages 1951–1956, 2009.

[75] Parag Singla and Pedro Domingos. Memory-efficient inference in relational domains. In *AAAI*, pages 488–493, 2006.

[76] Xiaodan Song, Ching-Yung Lin, Belle L. Tseng, and Ming-Ting Sun. Modeling and predicting personal information dissemination behavior. In *KDD*, pages 479–488, 2005.

[77] Kian-Lee Tan, Pin-Kwang Eng, and Beng Chin Ooi. Efficient Progressive Skyline Computation. In *VLDB*, pages 301–310, 2001.

[78] Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. Social influence analysis in large-scale networks. In *KDD*, pages 807–816, 2009.

[79] Liang Tang, Haiquan Chen, Wei-Shinn Ku, and Min-Te Sun. Parameterized spatial query processing based on social probabilistic clustering. In *ACM SIGSPATIAL GIS*, pages 410–413, 2014.

[80] Rares Vernica, Michael J. Carey, and Chen Li. Efficient parallel set-similarity joins using MapReduce. In *SIGMOD Conference*, pages 495–506, 2010.

[81] Akrivi Vlachou, Christos Doulkeridis, and Yannis Kotidis. Angle-based space partitioning for efficient parallel skyline computation. In *SIGMOD Conference*, pages 227–238, 2008.

[82] Daisy Zhe Wang, Eirinaios Michelakis, Minos N. Garofalakis, and Joseph M. Hellerstein. Bayesstore: managing large, uncertain data repositories with probabilistic graphical models. *PVLDB*, 1(1):340–351, 2008.

[83] Jue Wang and Pedro Domingos. Hybrid Markov Logic Networks. In *AAAI*, pages 1106–1111, 2008.

[84] Shiyuan Wang, Beng Chin Ooi, Anthony K. H. Tung, and Lizhen Xu. Efficient Skyline Query Processing on Peer-to-Peer Networks. In *ICDE*, pages 1126–1135, 2007.

[85] Xiaowei Wang and Yan Jia. Grid-based probabilistic skyline retrieval on distributed uncertain data. In *Database Systems for Adanced Applications*, pages 538–547. Springer, 2011.

[86] Zhen Wen and Ching-Yung Lin. How accurately can one's interests be inferred from friends. In *WWW*, pages 1203–1204, 2010.

[87] Ping Wu, Caijie Zhang, Ying Feng, Ben Y. Zhao, Divyakant Agrawal, and Amr El Abbadi. Parallelizing Skyline Queries for Scalable Distribution. In *EDBT*, pages 112–130, 2006.

[88] Junyi Xie, Jun Yang, Yuguo Chen, Haixun Wang, and Philip S. Yu. A Sampling-Based Approach to Information Recovery. In *ICDE*, pages 476–485, 2008.

[89] Mohan Yang, Haixun Wang, Haiquan Chen, and Wei-Shinn Ku. Querying uncertain data with aggregate constraints. In *SIGMOD Conference*, pages 817–828, 2011.

[90] Chi Zhang, Feifei Li, and Jeffrey Jestes. Efficient parallel kNN joins for large data in MapReduce. In *EDBT*, pages 38–49, 2012.

[91] Ji Zhang, Xunfei Jiang, Wei-Shinn Ku, and Xiao Qin. Efficient parallel skyline evaluation using mapreduce.

[92] Shiming Zhang, Nikos Mamoulis, and David W. Cheung. Scalable skyline computation using object-based space partitioning. In *SIGMOD Conference*, pages 483–494, 2009.

[93] Jun Zhu, Ni Lao, and Eric P. Xing. Grafting-light: fast, incremental feature selection and structure learning of markov random fields. In *KDD*, pages 303–312, 2010.

[94] Jun Zhu, Zaiqing Nie, Xiaojiang Liu, Bo Zhang, and Ji-Rong Wen. Statsnowball: a statistical approach to extracting entity relationships. In *WWW*, pages 101–110, 2009.