Individual Skyline Subspace on User Annotated Tags

by

Jiaxing Liang

B.Sc., Simon Fraser University, 2013

B.Eng., Zhejiang University, 2013

A Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Summer 2015

APPROVAL

Name: Jiaxing Liang

Degree: Master of Science

Title of Thesis: Individual Skyline Subspace on User Annotated Tags

Examining Committee: Dr. Petra Berenbrink,
Associate Professor, Chair

Dr. Jian Pei,
Professor, Senior Supervisor

Dr. Martin Ester,
Professor, Supervisor

Dr. Binay Bhattacharya,
Professor, Internal Examiner

Date Approved: June 10th, 2015

Abstract

Skyline computation is important in applications that involve multi-criteria decision making. In this thesis, we consider the following problem: given a query point, find the subspaces where the query point is in the subspace skyline. Although efficient algorithms for subspace skyline computation were developed in many existing studies, finding skyline subspaces for one certain query point is still an open problem. We develop an algorithm based on bottom-up set enumeration to compute the skyline subspace efficiently. We formulate the problem of identifying the uniqueness of a given vertex to skyline subspace queries problem on graph and proposed effective pruning methods to tackle this problem. We further conduct experiments on both real world datasets and synthetic datasets to verify the efficiency of our methods.

Keywords: Skyline; subspace; set enumeration; graph;

To my parents.

"If you spend too much time thinking about a thing, you'll never get it done."

— Bruce Lee, *(1940 - 1973)*

Acknowledgments

I would like to express my deepest gratitude to my senior supervisor, Dr. Jian Pei, for his great patience, warm encouragement and continuous support throughout my Master's studies. His wisdom and passion in research has influenced and inspired me a lot, which gave me confidence and interest in accomplishing this thesis.

I would like to thank Dr. Martin Ester for being my supervisor and giving me helpful suggestions on my thesis. I also thank Dr. Binay Bhattacharya and Dr. Petra Berenbrink for serving in my examining committee.

I am also very grateful to my lab mates for their kind help. A special thank goes to Dr. Huaizhong Lin, Dr. Aihua Wu, Dr. Fuyuan Cao, Dr. Kui Yu, Dr. Dongwan Choi, Guanting Tang, Xiao Meng, Juhua Hu, Chuancong Gao, Yu Yang, Xiangbo Mao, Xuefei Li, Lumin Zhang, Xiang Wang, Xiaoning Xu, Yu Tao, Lin Liu, Li Xiong, Beier Lu, Zicong Cun, and Xiaojian Wang.

Moreover, my sincerest gratitude goes to my parents for their endless love and support through all these years.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, we first introduce the basic idea of skyline subspace problem and several interesting applications that motivate the problem, which will be studied in this thesis. Then, we will summarize the major contributions and describe the structure of the thesis.

1.1 Motivations

The skyline operator is an important research topic for multi-criteria decision making applications.

One classical example of skyline queries is searching for hotels that are cheap and close to the beach [4]. We assume that each hotel has two attributes: the price and the distance from the hotel to the beach. Suppose there is Hotel A and Hotel B and the price of Hotel A is lower than the price of Hotel B, and the distance from Hotel A to the beach is also shorter than the distance from Hotel B to beach. Then Hotel A dominates Hotel B. We call those hotels that are not dominated by others in terms of price and distance to the beach skyline hotels. There are many recent studies on efficient methods for skyline computation, subspace skyline analysis and skyline computation in different scenarios. We will review the related work on Chapter 2.

However, all the previous studies are about the skyline computation. The questions about computing the subspaces of a query point with respect to skyline remain open.

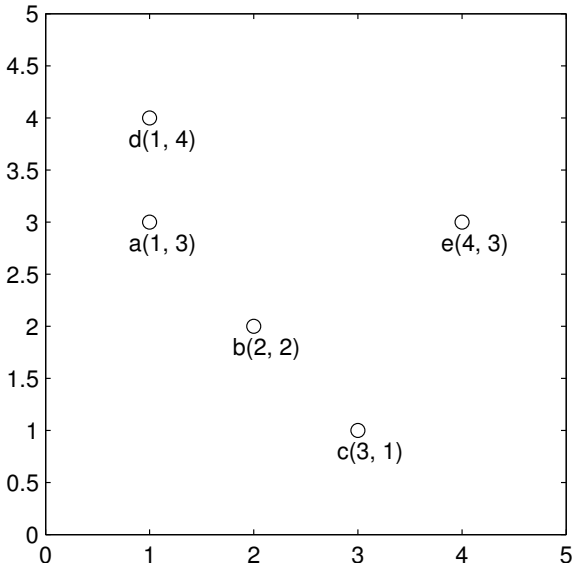

Figure 1.1: An example of data points on full space (X, Y)

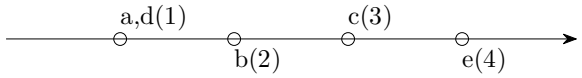

Figure 1.2: The projection of data points on subspace X

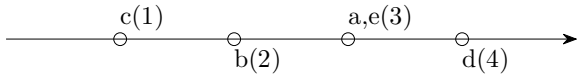

Figure 1.3: The projection of data points on subspace Y

Example 1.1. *Consider a set of* 5 *data points in 2-d space* (X, Y) *as shown in Figure 1.1. The points a, b and c are skyline points in space* (X, Y) *since each of them is not dominated by any other points. The definitions of* skyline *and* domination *are shown in 3.1 in Chapter 3.*

In Figure 1.2 and Figure 1.3, we also plot the projections of the data points on dimensions X and Y, respectively. In our thesis, we are only interested in the non-trivial subspaces that are non-empty. In Example 1.1, subspace X and subspace Y are two non-empty subspaces that we are interested in. In subspace X, the projections of the point a and d have the

same value. Both of them are subspace skyline points of subspace X. In subspace Y, the projection of point c is also a skyline point.

From Figure 1.1, we can see that point a, point b and point c are skyline points in the full space (X, Y). However, there are differences among them if we look at them in different projections. Point a is a subspace skyline point in subspace X. Point c is a subspace skyline point in subspace Y. Although b is also skyline point in the full space (X, Y), it is not a skyline point in projection X or projection Y.

Taking the value 1 on subspace Y is sufficient for c to be a skyline point in the full space (X, Y). No other points are able to dominate c because c is the only point with the minimal value 1 on subspace Y. While taking the value 1 on subspace X is not sufficient for a to be a skyline point in full space, because d also has the same minimal value 1 on subspace X. Point b does not have minimal value on subspace X or subspace Y.

Point d and e are still subtly different although neither of them is a skyline object in full space (X, Y). With the same minimal value 1 as point a on subspace X, point d is a skyline point in subspace X. Point e is not a skyline point in full space (X, Y) or any subspaces of (X, Y).

In our thesis, we are interested in finding all the minimal subspace projections such that a query point is a skyline point on those projections. We call these projections *skyline subspaces* of the query point.

Why are we interested about the skyline subspaces of the query points? The information of skyline subspace helps us understand the data better. In Section 6.1, we analyze a real dataset of DBLP citation network. We take the distances from all authors to a conference as a dimension. Consider all conferences as the full space, we want to compute the subspaces on which the query author point is a skyline point. By computing those subspaces, we are able to know what distinguishes an author from its peers in terms of research topics and academic connections of that author. For example, Dr. Stephan Schulz publishes papers or connects with other authors who publish papers in the computing science conferences *ENTCS*, *J. Symb. Log.*, *FLAIRS* and *ACM Trans. Graph.*. Dr. Stephan Schulz is a skyline point in terms of connections and publications in these four conferences and we claim that he has a good reputations in computer science. We are interested in the subsets of these conferences that make him distinguished with others. After computing the skyline subspaces of Dr. Stephan Schulz in the DBLP citation network, we find the distances from Stephan to conferences *ENTCS* and *ACM Trans. Graph.* are 2 and 1 respectively, which

makes him a skyline point among all the researchers. In Section 6.2, we analyze the Yelp academic dataset. Given a query business spot, taking the Euclidean distances between the business spots and the business categories (restaurant, coffee shop, etc.) as criteria, we are interested in finding the sets of business categories that makes the business spot special.

Computing the skyline points of all the subspaces [18, 22] is costly when users are only interested in finding subspaces of a certain the query points. In this thesis, we are focusing on skyline subspaces of one query point but not the whole dataset. The skyline subspace queries is also known as *skyline membership queries* mentioned in [18] but the problem of answering *skyline membership queries* efficiently had not been solved.

1.2 Contributions

The original skyline operator problems have been studied in [4, 6] and subspace skyline problems have been studied in [18, 22]. In this thesis, our goal is to find the subspaces where the query point is in the subspace skylines. By tackling this problem, we make the following contributions.

- We develop an algorithm framework to answer the *skyline subspace query*, finding the subspaces where a query point is in the subspace skyline. We present a bottom-up algorithm based on set enumeration and dominant candidate sets intersection to solve the problem.
- We apply skyline subspace query on two specific applications: Computing skyline subspaces on graph and computing the skyline subspaces on Euclidean space. We develop effective pruning algorithms to reduce the unnecessary computation.
- A performance study using both synthetic and real data sets is conducted to evaluate our approach. For the skyline subspace problem on the graph, we run our algorithm on DBLP citation network to test its efficiency and scalability. For the spatial skyline subspace problem, we run our algorithm on the YELP academic dataset.

1.3 Organization of the Thesis

The rest of the thesis is organized as follows. In Chapter 2, we review the related work of skyline queries. We then formulate our *skyline subspace* problem in Chapter 3. In

Chapter 4, we propose the basic framework of our algorithm and the pruning method on *skyline subspace queries on graph*. In Chapter 5, we present our algorithm of *spatial skyline subspace queries*. We report our experimental results in Chapter 6, and conclude the thesis in Chapter 7.

Chapter 2

Related Work

Our problem of *skyline subspace query* is mainly related to the existing work on general skyline queries, subspace skyline computation and skyline queries with specific constraints which are reviewed in Section 2.1, Section 2.2 and Section 2.3, respectively.

2.1 General Skyline Queries

The general skyline problem is to find all the points that are not dominated by any other points. There is a number of studies on skyline in Data Mining area. The problem of finding the maxima (skyline) of a set of vectors was first investigated in [13] where an $O(n \log^{d-2} n)$ algorithm for dimensionality $d \geq 4$ and an $O(n \log n)$ time algorithm for dimensionality $d = 2, 3$ are proposed. The algorithm in [13] is based on the divide and conquer principle. To integrate the skyline operator into database, Borzsony et al. [4] proposed the Block-nested-loops Algorithm (BNL) and Divide and Conquer Algorithm (DC) to compute the skyline queries. BNL essentially maintains a window of incomparable objects and compare an object in the database with the objects in the window in each iteration. It outputs the skyline at the end of all the iterations. DC divides the dataset into several partitions and each partition can fit in memory. The skylines in all partitions are computed individually in main memory, and then merged to produce the final skyline objects. The sort-first-skyline (SFS) [6] algorithm also maintains a window of object, which is similar to the BNL algorithm. In addition to that, it sorts the input data first so that it can guarantee the objects in the window can be output immediately as skyline points, which makes the algorithm more efficient. Kossmann et al. [12] studied the relationship between nearest

neighbours and skyline points and developed the NN algorithm to compute skyline using R-tree [3, 10] to index the data points. Different from original skyline query problem which needs to scan the whole database to output all the skyline points at the very end, the progressive skyline problem is to progressively return the skyline points as they are identified. Tan et al. [20] proposed the Bitmap algorithm and Index algorithm to tackle this problem. The Bitmap algorithm exploits a bitmap structure to identify whether a point is a skyline point. The Index algorithm transfers the multi-dimensional objects into 1-dimensional space and stores the objects in a B+-tree structure. To explore the progressive skyline problem further, Papadias et al. [16, 17] developed the bound-and-branch skyline (BBS) algorithm which takes the advantage of R-tree [3, 10] to search for the nearest neighborhood. These works are about finding the skyline points efficiently in databases.

2.2 Subspace Skyline Computation

The research of subspace skyline problem is to study the relationship between the property of skyline and subspace. The major research problem of this field is that given a set of n-dimensional points, we want to compute all the skyline points in all the subspaces of the full space.

For the subspace skyline problem, Pei et al. [18] proposed the *Skyey* algorithm based on the property of decisive subspaces to compute the skyline points for every subspace. The algorithm not only outputs the skyline points in every subspace, but it also returns the *skyline groups*. The *skyline groups* contain the skyline objects sharing the same values on all dimensions in the corresponding subspaces. The *Skyey* algorithm takes the advantage of sharing sorted order of the objects among different subspaces to make the computation efficient. Yuan et al. [22] developed the *Top-Down Skyline Algorithm* to compute the skyline in every subspace. They also developed a novel data structure *skylist* to stores the skyline objects in different subspaces in a compact way. Both of their algorithms are in the top-down manner.

Most of time, people are interested in computing the skyline of one particular subspace instead of all subspaces. To tackle the problem of computing skyline in one particular subspace, Tao et al. [21] proposed the SUBSKY algorithm using a single B-tree. They applied a transformation on the multi-dimensional data to 1-dimensional value to enable several effective pruning heuristics. The subspace skyline queries on high dimensional data

was studied in [11]. To study the subspace skyline queries on high dimensional data, Jin et al. [11] proposed novel notions of *maximal partial-dominating space*, *maximal partial-dominated space and the maximal equality space* between pairs of skyline objects in the full space. In our thesis, we are focusing on skyline subspace of one query point but not the whole dataset. Different from previous works of subspace skyline. Our work is to find the subspaces in terms of the query point. We also extend our work on the graph setting and spatial setting.

2.3 Skyline Query in Specific Scenarios

In regular skyline problem, the n-dimensional values of the data points is known. However, in many scenarios, the values of the data points is unknown at the beginning and somehow it is costly to compute the actual values of all the data points. Finding efficient ways to compute the skyline points in those scenarios are also interesting problems to study. In this section, we will introduce several papers that study the skyline query problem in different scenarios.

To explore the relationship Euclidean geometry and skyline, the spatial skyline problem is studied in [19]: Given the two sets P of data points and Q of query points, the spatial skyline of P with respect to Q is the set of those points in P whose distances to every point in Q are not dominated by any other point of P. Sharifzadeh et al. [19] proved two important theorems of spatial skyline problem: Any point $p \in P$ which is inside the convex hull of Q is a skyline point. Any point whose Voronoi cell intersects with boundaries of convex hull of the query points is a skyline point. They proposed the algorithm B^2S^2 and VS^2 to compute the spatial skyline based on the geometry properties of convex hull and Voronoi diagram. B^2S^2 algorithm is based on Branch and Bound algorithm which stores the points in R-tree. VS^2 stores the points in Voronoi diagram and takes the advantage of Voronoi diagram to compute the nearest neighbours efficiently.

Road network skyline problem is studied in [8]: Given a road network modeled as a graph $G = (E, V)$ and a set of query points Q, the road network skyline query of V with respect to Q is the set of those points in V whose distances to every point in Q are not dominated by any other point of V. Deng et al. [8] proposed Collaborative Expansion Algorithm (CE), Euclidean Distance Constraint Algorithm (EDC) and Lower-Bound Constraint Algorithm (LBC) to solve the problem. CE is a straight forward method which is based on Dijkstra

Algorithm without taking the geometry information of the vertices into account. The EDC algorithm is based on the A* algorithm which takes the advantage of the property that the spatial distance between two vertices is less than the road distance between them. The LBC algorithm decreases the skyline point candidate size by introducing the concept of *path distance lower bound*. Both the EDC and the LBC algorithm utilize Euclidean distance as the lower bound of shortest path distance in road network to perform pruning.

Both the spatial skyline problem and the road network skyline problem are the particular types of metric skyline problem. Metric distance satisfies the triangle inequality: $dist(x, z) \leq dist(x, y) + dist(y, z)$. Metric distance is the general case of spatial Euclidean distance and road network distance. Chen et al. [5] illustrate a triangle-based pruning mechanism to answer metric skyline queries through a metric index with M-tree. To go further on the problem of skyline computation on metric space, Fuhry et al. [9] proved that the distance between a metric space skyline point and q is bounded by $2r_q + d(q, NN(q))$ where r_q is the radius of the enclosing ball for the set of query points Q in terms of the query point q. Using this property, they proposed the N^2RS algorithm, which only searches for the skyline points in a certain range. The B^2MS^2 algorithm based on a generic index tree was also proposed in [9].

In the road network skyline problem, Deng et al. [8] takes the advantage of geometry spatial information. To solve the skyline problem on pure graphs without any spatial information, Zou et al. [23] proposed the SSP Query algorithm to tackle this problem. They first introduced the concept of *1Hop Shortest Path Tree* to index the tree in the database in order to compute the shortest-distance queries efficiently. They also introduced the SSP pruning method to prune the unnecessary skyline point candidates. After getting the skyline point candidates, they performed the BNL algorithm [4] to find the skyline points.

2.4 Reverse Dynamic Skyline Query

All of the previous works are focusing on finding the skyline points themselves. Papadias et al. [16] introduced the concept of *dynamic skyline query*: Given a query point q and a data set P, a *dynamic skyline query* according to q returns all data points in P that are not dynamically dominated. A point $p_1 \in P$ dynamically dominates $p_2 \in P$ with regard to the query point q if for all $i, |q^i - p_1^i| \leq |q^i - p_2^i|$ and for at least one $j, |q^j - p_1^j| < |q^j - p_2^j|$.

q^i represents the value in i^{th} dimension of q. Dellis et al. [7] introduced an opposite version of this problem *reverse skyline query*: Given a query point q and a data set P, a reverse skyline query according to q returns all data points $p_1 \in P$ where q is in the dynamic skyline of p_1. They proposed BBRS algorithm, which is an improved customization of the original BBS [16] algorithm to tackle this problem.

In our thesis, we study the reverse version of the problem in [21]: Given a set of data points and a query point, we want to compute all the subspaces where the query point is not dominated by any other point. We also extend our work to find the skyline subspaces of a query point on the graph and Euclidean space.

Chapter 3

Problem Definition

The traditional skyline query problem is to find the skyline points which are not dominated by any other points. In this thesis, we consider a variant of this problem. We consider a set of objects S in an n-dimensional space and a query object u in this space. The object u may be a skyline point in some subspaces. We call these subspaces the *skyline subspaces*. The skyline subspaces queries are to find those *skyline subspaces*. Different from the work by Tao et al. [21]: given a *subspace* as a query, determine the sets skyline points in that subspace, the problem we want to solve is in a reverse way: given a query *point*, determine the subspaces where the query point is in the subspace skyline. In this chapter, we will introduce the general skyline subspace queries and its applications in two different settings: skyline queries on graph and spatial skyline queries.

3.1 General Skyline Subspace Queries

Definition 3.1 (Skyline)**.** *For objects $u, v \in S$, u dominates v if and only if for all i, $(1 \leq i \leq n)$, $u.i \leq v.i$ and there exists a j $(1 \leq j \leq n)$ such that $u.j < v.j$. Object u is a* skyline object *if u is not dominated by any other objects in S.*

Definition 3.2 (Skyline Subspace)**.** *Subspace $\mathcal{B}$ is a (non-trivial) $|\mathcal{B}|$-dimensional subspace of $\mathcal{D}$ if $\mathcal{B} \subseteq \mathcal{D}(\mathcal{B} \neq \emptyset)$. For an object u in space $\mathcal{D}$, the* projection *of u in subspace $\mathcal{B}$, denoted by $u_{\mathcal{B}}$, is a $|\mathcal{B}|$-tuple $(u.i_1, \ldots, u.i_{|\mathcal{B}|})$, where $i_1, \ldots, i_{|\mathcal{B}|} \in \mathcal{B}$, $u.i$ is the value of u on i. If $u_{\mathcal{B}}$ is not dominated by any $w_{\mathcal{B}}$ in subspace $\mathcal{B}$ where $w \in S$, then $\mathcal{B}$ is a* skyline subspace *for $u_{\mathcal{B}}$.*

Definition 3.3 (Minimal Skyline Subspace)**.** *A* skyline subspace $\mathcal{B}$ *is a* minimal skyline subspace *for object* u *if and only if there is no such a* skyline subspace $\mathcal{C}$ *for object* u *that* $\mathcal{C} \subset \mathcal{B}$.

Definition 3.4 (Skyline Subspace Queries)**.** *Given a set of objects* S *in an* n*-dimensional space* $\mathcal{D}$ *and a query object* $q = (q.1, \ldots, q.n)$ *in space* $\mathcal{D}$*. The skyline subspaces query is to find all the* minimal skyline subspaces *for query object* q.

points	A	B	C
x	4	2	3
y	3	3	3
z	2	4	1

Table 3.1: A set of objects as our running example

In Table 3.1, if the query point is y, the minimal skyline subspaces of y are (A, B) and (B, C). In subspace (A, B), we have $x_{(A,B)} = (4, 2), y_{(A,B)} = (3, 3), z_{(A,B)} = (2, 4)$ and $y_{(A,B)}$ is not dominated by $x_{(A,B)}$ or $z_{(A,B)}$. In subspace (B, C), $x_{(B,C)} = (2, 3), y_{(B,C)} = (3, 3), z_{(B,C)} = (4, 1)$ and $y_{(B,C)}$ is not dominated by $x_{(B,C)}$ or $z_{(B,C)}$. Therefore, (A, B) and (B, C) are both skyline subspace of y. In subspace (A, C), we have $y_{(A,C)} = (3, 3), z_{(A,C)} = (2, 1)$ and $y_{(A,C)}$ is dominated by $z_{(A,C)}$. Therefore, (A, C) is not a skyline subspace. (A, B, C) is a skyline subspace but it is not a minimal skyline subspace because it is a proper superset of one of the skyline subspace (A, B).

3.2 Skyline Subspace Queries on Graph

In this section, we introduce the skyline subspace queries on the graph setting. First, we will introduce a kind of graph such that each vertex in the graph contains several labels, denoted by a *labeled graph*. We are interested in the distances from all vertices and all labels. The set of distances from a vertex to all labels is represented by the *labeled distance vector* of that vertex. The following shows the definitions of *labeled graph* and *labeled distance vector*.

Definition 3.5 (Labeled Graph)**.** *A labeled graph is defined as a undirected and unweighted graph* $G = (V, E, L)$*, with each vertex* $v \in V$ *contains a set of labels* $L_v \subseteq \mathcal{L}$*, where* $\mathcal{L}$ *is the universal set of labels.*

Definition 3.6 (Labeled Distance Vector on Graph in d-hop)**.** *Given a labeled graph G with n labels and a hop number d, the Labeled Distance Vector of a vertex v is $LV_v = (dist_1, dist_2, \ldots, dist_n)$, where n is the number of labels, $dist_i$ is the distance from vertex v to the closest vertex that contains label i. If a label j is not reachable by the vertex v in d hops, then $dist_j = \infty$.*

We simply denote the value of *labeled distance vector* of a vertex v on dimension D by $v.D$ in our thesis.

Definition 3.7 (Skyline Subspace Queries on Graph)**.** *Given a labeled graph $G = (V, E, L)$, a query vertex q and a hop number d, the skyline subspaces query on graph is to find all the* minimal skyline subspaces *of the query vertex q with respect to the* labeled distance vectors *of all the vertices on graph G in d hops.*

Experts	Skills
u	Accounting
y	Bioinformatics
w	C++, Bioinformatics

Table 3.2: An example of skill sets on LinkedIn profile

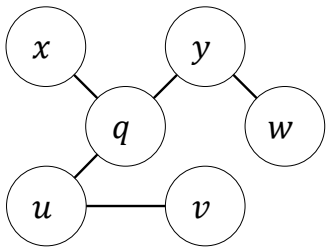

Figure 3.1: An example of LinkedIn network

Consider an example on LinkedIn to illustrate the skyline subspace queries on graph. In Figure 3.1, a graph is represented by the LinkedIn connection network. Table 3.2 shows the skills of each person of the LinkedIn network which can be treated as vertices with labels. Both of them together represent a *labeled graph*. In Table 3.2, it shows that vertex u has a skill of Accounting, vertex w has skills of Bioinformatics and C++ and vertex y has a skill of Bioinformatics. In this example, we want to compute the subspace skyline queries in 3 hops.

Distances	A	B	C
u	0	2	3
v	1	3	∞
w	3	0	0
x	2	2	3
y	2	0	1
q	1	1	2

Table 3.3: Distance between each person and each skill

In the header row of Table 3.3, A, B and C stand for Accounting, Bio-informatics and C++ respectively. The table shows the distances between the people and the skills. For example, the distance between v and skill B is 3 because y is the closest vertex to v that contains label B and the distance between v and y is 3. Since C++ is not reachable by v in 3 hops, the distance between v and C is ∞. Each row of Table 3.3 represents a *labeled distance vector* of a vertex. In this example, if the query point is q, the minimal skyline subspaces of q are (A, B) and (A, C) because there is no such a vertex $w \in S$ that $w_{(A,B)}$ dominates $q_{(A,B)}$ or $w_{(A,C)}$ dominates $q_{(A,C)}$.

3.3 Spatial Skyline Subspace Queries

In this section, we will introduce spatial skyline subspace queries. Given a set of points in a 2-dimensional Euclidean space. Some of the points contain some labels. The distance between a point and a label is the Euclidean distance between that point and the closest point with that label. Our goal is to compute the skyline subspace queries on this spatial setting.

Definition 3.8 (Spatial Labeled Distance Vector in range r)**.** *Given a set of points P, the Labeled Distance Vector of a point v is $LV_v = (dist_1, dist_2, \ldots, dist_n)$, where n is the number of labels, $dist_i$ is the Euclidean distance from vertex v to the closest vertex that contains label i. If a label j is not reachable by the point v in range r, then $dist_j = \infty$.*

We simply denote the value of *labeled distance vector* of a point v on dimension l by $v.l$ in our thesis.

Definition 3.9 (Spatial Skyline Subspace Queries)**.** *Given a set of points P, a query point q and range r, the skyline subspaces query on Euclidean space is to find all the* minimal

skyline subspaces *with respect to the* spatial labeled distance vectors in range r *of every point in* P.

Spots	Categories
u	Asian Food
y	Breakfast
w	Cafes, Breakfast

Table 3.4: An example of spots with categories

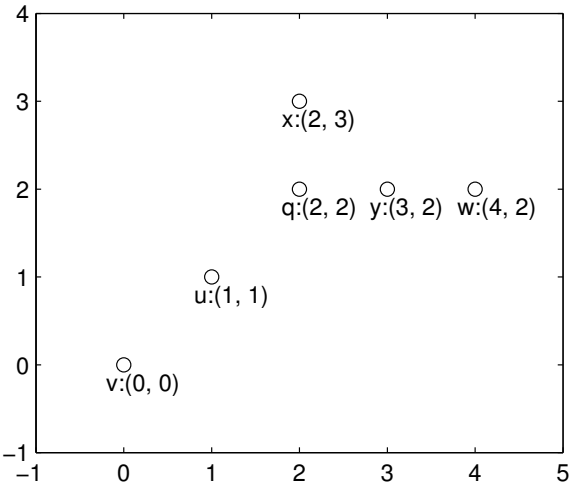

Figure 3.2: An example of spatial locations of spots

Table 3.4 shows the business spots with categories. Figure 3.2 shows the geometric locations of all spots. We consider each spot a point in a 2-dimensional Euclidean space and the categories the spot contains as its labels.

Distances	A	B	C
u	0	$\sqrt{5}$	$\sqrt{10}$
v	$\sqrt{2}$	$\sqrt{13}$	$\sqrt{18}$
w	$\sqrt{10}$	0	0
x	$\sqrt{5}$	$\sqrt{2}$	$\sqrt{5}$
y	$\sqrt{5}$	0	1
q	$\sqrt{2}$	1	2

Table 3.5: Distance between each spot and each category

In the header row of Table 3.5, A, B and C stand for Asian Food, Breakfast and Cafes, respectively. The table shows the distances between each spot and each categories. In this example, if q is the query point, then the minimal skyline subspaces are (A, B) and (A, C).

Chapter 4

A Pruning-based Method on Graphs

In this chapter, we will introduce the algorithms to compute the skyline subspace queries. One naive way to solve this problem is to compute the labeled distance vectors of all the vertices first and enumerate all subspaces to check whether the query vertex is a subspace skyline in those subspaces using the existing skyline computation algorithms. Given a label graph $G = (V, E, L)$, the time complexity of this method is $O((|V| + |E|)|L| + 2^{|L|}|L||V|^2)$. $O((|V|+|E|)|L|)$ is the complexity to traverse the whole graph to get *labeled distance vectors* of all vertices. $O(2^{|L|})$ is the complexity of enumerating every subspace of a $|L|$-dimensional space. $O(|L||V|)$ is the time complexity of checking whether the query is a skyline point in a certain subspace. We compare each pair of points to check whether one point dominates the other one. This brute force method is very time consuming. In order to make the algorithm more efficient, we propose a bottom-up set enumeration algorithm and manage to avoid some unnecessary computation by applying some pruning techniques in our method.

4.1 BFS Label Collecting

We collect the d-hop labels using Breadth-First-Search and get the *labeled distance vector* of the query vertex. The idea is that we start with the query vertex and traverse the graph in a breadth first manner. If we visit a vertex with a new label that we have not visited before, we update the corresponding entry of that label in the *labeled distance vector* to the distance from the query vertex. The Breadth First Search process will end if all reachable

Symbol	Interpretation
q	query vertex q
LV_v	labeled distance vector of vertex v representing the distances between v and each label.
$\mathcal{B}$	subspace $\mathcal{B}$
SDS_u	strictly dominating subspace of u
EQ_u	equivalence subspace of u
$CAND_{\mathcal{B}}$	dominating candidate set of subspace $\mathcal{B}$
(u, dom)	element in dominating candidate set. $(u, dom) \in CAND_{\mathcal{B}}$ means that u dominates query vertex q in subspace $\mathcal{B}$.
(u, eq)	element in dominating candidate set. $(u, eq) \in CAND_{\mathcal{B}}$ means that u dominates query vertex q in subspace $\mathcal{B}$.

Table 4.1: Symbols used in the pruning-based method on graph

vertices in up to d hops have been visited. Then we will get the *labeled distance vector* of the query vertex when the BFS label collecting ends. Algorithm 1 shows the process of label collecting. The time complexity is $O(|V| + |E|)$.

4.2 Dominating Candidate Sets

By collecting the label in d hops from the query vertex, we build the *labeled distance vector* of our query vertex. To avoid computing the labeled distance vectors of some unnecessary vertices, we define a concept of *dominating candidate set* to store the candidate vertices that dominate the query vertex q in certain subspaces. Given a subspace $\mathcal{B}$, the *dominating candidate set* of $\mathcal{B}$ contains the vertices that dominates or equals the query vertex on subspace $\mathcal{B}$.

Definition 4.1 (Dominating Candidate Set)**.** *Given a subspace $\mathcal{B}$, the dominating candidate set of that subspace is the set of vertices that dominate the query vertex q or equal to query vertex q on every dimension in subspace $\mathcal{B}$, denoted by $CAND_{\mathcal{B}}$ with respect to q.*

We define the elements of the dominating candidate set, (u, dom) and (u, eq) in the following way: $(u, dom) \in CAND_{\mathcal{B}}$ if vertex u dominates query vertex q in subspace $\mathcal{B}$, and $(u, eq) \in CAND_{\mathcal{B}}$ if vertex u equals query vertex q in subspace $\mathcal{B}$.

The reason we put all vertices equal to the query vertex to the candidate set is that if a vertex equals to the query vertex in a subspace $\mathcal{B}$ then that vertex may dominate the query

Algorithm 1 Label Collecting

Input: A graph $G = (V, E)$, a list of label sets $F = \{L_v | v \in V\}$, the label sets of all vertices, a query vertex q, the number of hops d;
Output: The labeled distance vector LV_q of the query vertex q;

```
1: push (q, 0) to Q
2: while Q is not empty do
3:    (v, dis) = de-queue Q
4:    if dis = d then
5:       continue
6:    end if
7:    for all not visited neighbour u of v do
8:       push (u, dis + 1) to Q
9:       for all label l in L_u do
10:         if (l, *) not in LV_q then
11:            add (l, dis + 1) to LV_q
12:         end if
13:      end for
14:   end for
15: end while
```

vertex in some supersets of subspace $\mathcal{B}$.

If $CAND_{\mathcal{B}} = \emptyset$ or every vertex in $CAND_{\mathcal{B}}$ is equal to the query vertex q in subspace $\mathcal{B}$, then q is a skyline point in subspace $\mathcal{B}$. Therefore, we can determine whether a subspace $\mathcal{B}$ is a *skyline subspace* by checking whether the dominating candidate set of $\mathcal{B}$ is empty or whether all elements in the *dominating candidate set* of $\mathcal{B}$ are all equal to the query vertex in subspace B. In other words, given a subspace $\mathcal{B}$, if there does not exist a vertex u, such that $(u, dom) \in CAND_{\mathcal{B}}$, then $\mathcal{B}$ is a skyline subspace of query vertex q. To understand the concept of *dominating candidate set* better, we will show a running example in Section 4.2.2.

4.2.1 Dominating Candidate Set of 1-dimensional subspace

We will introduce an algorithm to compute the *dominating candidate set* of 1-dimensional subspace. We will also introduce the concepts of *Strictly Dominating Subspace* and *Equivalence Subspace* which help us prune the some of the vertices from the dominating candidate sets.

Definition 4.2 (Strictly Dominating Subspace)**.** *Given a vertex u, the* strictly dominating subspace $\mathcal{B}$ *for u, SDS_u, is the subspace that consists of all the dimensions l such that*

$u.l < q.l$, where q is the query vertex (with respect to q).

Definition 4.3 (Equivalence Subspace)**.** *Given a vertex u, the* equivalence subspace $\mathcal{B}$ *for u, EQS_u, is the subspace that consists of all the dimensions l such that $u.l = q.l$, where q is the query vertex (with respect to q).*

In this algorithm, we start from the vertices with labels and initialize the values of the corresponding dimensions of those vertices to 0. In the next step, we push all the neighbours of those vertices into a queue and update their labeled distance vectors. The updating procedure is in a breadth first manner. For every dimension l, the procedure of updating the labeled distance vectors of vertices ends when the distance from the current vertex that we are visiting to the label l is greater than $q.l$, where $q.l$ is the distance from the query vertex q to the label l. The time complexity of the algorithm is $O((|V|+|E|)|L|)$.

Algorithm 2 Dominating Candidate Set on 1-Dimensional Subspace

Input: A graph $G = (V, E)$ and the labeled distance vector LV_q of the query vertex q;
Output: Dominating candidate Set $CAND$ in all one dimension subspace in LV_q, EQS and SDS;

```
1: for all vertex v contains label l do
2:    for all (l, dist) in LV_q do
3:       push (v, 0) to Q
4:    end for
5: end for
6: while Q ≠ ∅ do
7:    for all (l, dist) in LV_q do
8:       (v, dist_{v,l}) = de-queue Q
9:       if dist_{v,l} = dist then
10:         add (u, equal) to CAND_l
11:         add l to EQS_u
12:         continue
13:      end if
14:      add (u, dom) to CAND_l
15:      add l to SDS_u
16:      for all not visited neighbour u of v do
17:         push (u, dist_{v,l} + 1) to Q
18:      end for
19:   end for
20: end while
```

4.2.2 Running Example of Computing Dominating Candidate Set

In this subsection, we will give an example of how the algorithm of finding *dominating candidate set on* 1*-dimensional subspace* works. We will also show how the *strictly dominating subspace* and the *equivalence subspace* of each vertex are built.

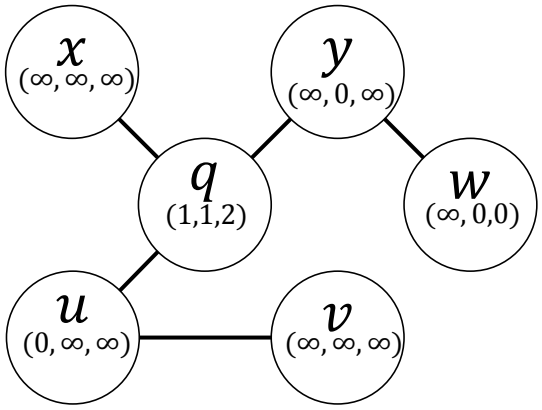

Figure 4.1: Label distance vector after the first iteration

	SDS	EQS
u	A	∅
v	∅	∅
w	BC	∅
x	∅	∅
y	B	∅

Table 4.2: SDS and EQS of each vertex after the first iteration

Subspaces	Dominating candidate
A	(u, dom)
B	$(w, dom), (y, dom)$
C	(w, dom)

Table 4.3: Dominating candidate set of each 1-dimensional subspace after the first iteration

Consider the LinkedIn network represented by Table 3.2 and Figure 3.1 as our running example. Again, we take q as the query vertex. The labeled distance vectors of all vertices are originally initialized as $(\infty, \ldots, \infty)$. As shown in Figure 4.1, we start from the vertices with labels and mark the corresponding entries of the labeled distance vectors of those vertices as 0. We add the label A to SDS_u because u dominates the query vertex q in dimension A. Table 4.2 shows the SDS and EQS of all the vertices after the first iteration. After the first iteration, since u dominates q in the 1-dimensional subspace A, we add (u, dom) to the dominating candidate set $CAND_A$ as shown in Table 4.3.

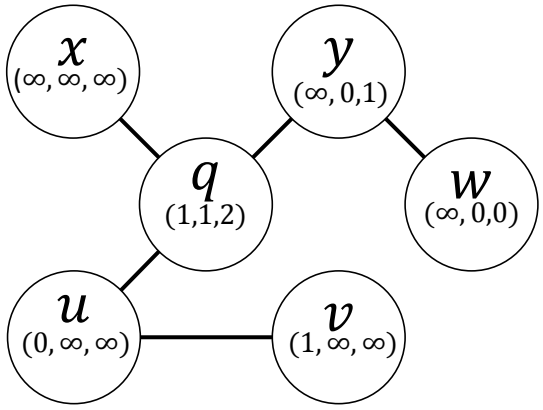

Figure 4.2: Label distance vector after second iteration

	SDS	EQS
u	A	∅
v	∅	A
w	BC	∅
x	∅	∅
y	BC	∅

Table 4.4: SDS and EQS of each vertex after the second iteration

Subspaces	Dominating candidate
A	$(u, dom), (v, eq)$
B	$(w, dom), (y, dom)$
C	$(w, dom), (y, dom)$

Table 4.5: Dominating candidate set of each 1-dimensional subspace after the second iteration

Then, we explore the graph in a breadth first manner. On the second iteration, as shown in Figure 4.2, we visit the neighbours of the vertices that were visited in the first iteration and update the corresponding entries of their labeled distance vectors. Table 4.4 shows that subspace A is added to EQS_v because on the second iteration we update the distance between v and label A to 1 which is equal to the distance between q and label A. For the same reason, we add (v, eq) to $CAND_A$ as shown in Table 4.5. It means that in 1-dimensional subspace A, v is equal to q and it is still possible for v to dominate q.

After two iterations, the process of building the dominating candidate sets of all 1-dimensional subspace ends. The *strictly dominating subspaces* and the *equivalence subspaces* of all the vertices on graph are built. In Table 4.6, we collect the 3-hop labels by Breadth-First-Search and get the label vectors. By this point, if the value of label l in the labeled distance vector of vertex v is ∞, then it means the distance between label l and vertex v is longer than the distance between the label l and the query vertex q.

Distances	A	B	C
u	0	∞	∞
v	1	∞	∞
w	∞	0	0
x	∞	∞	∞
y	∞	0	1
q	1	1	2

Table 4.6: Distance between each person and each skill in 2-hop

Although some information is still missing (equal to ∞) in Table 4.6, we are still able to get the minimal skyline subspaces of q: (A, B) and (A, C) from the Table 4.6.

4.3 Dominating Candidate Pruning

In this section, we will introduce a way to prune the unnecessary vertices using the *strictly dominating subspace*, SDS, and the *equivalence subspace*, EQ of the vertices.

Lemma 4.3.1. *Given a vertex u, if there exists a vertex v, such that $(SDS_u \cup EQS_u \subseteq SDS_v \cup EQS_v) \wedge (SDS_u \subseteq SDS_v)$, and $(u, dom) \in CAND_{\mathcal{B}}$, then for any subspace $\mathcal{B}$, we have $(v, dom) \in CAND_{\mathcal{B}}$.*

Proof. We prove this lemma by contradiction. Suppose there exists a subspace $\mathcal{B}$, such that $(u, dom) \in CAND_{\mathcal{B}}$, but $(v, dom) \notin CAND_{\mathcal{B}}$. Then $(u, dom) \in CAND_{\mathcal{B}}$ implies that vertex u dominates query vertex q in subspace $\mathcal{B}$. $(v, dom) \notin CAND_{\mathcal{B}}$ implies that v does not dominate query vertex q in subspace $\mathcal{B}$. There are two possible cases that v does not dominate q.

Case 1: The values of all dimensions in subspace $\mathcal{B}$ of v are all equal to the values of q. Since u dominates q in subspace $\mathcal{B}$, in one of the dimensions of $\mathcal{B}$, the value of u is less than q , say dimension c. By the definition of *strictly dominating subspace*, we have $c \in SDS_u$. We have $c \notin SDS_v$, since v equals to q in subspace $\mathcal{B}$. Therefore, $SDS_u \subseteq SDS_v$ and $SDS_u \not\subseteq SDS_v$, a contradiction.

Case 2: In one of the dimensions of subspace $\mathcal{B}$, say dimension e, vertex v has a greater value than query vertex q. By the definitions of *strictly dominating subspace* and *equivalence subspace*, we have $e \notin SDS_v \cup EQS_v$. Since u dominates q in subspace $\mathcal{B}$ and $e \in \mathcal{B}$, the value of u in dimension e is less than or equal to the value of q in dimension e. Thus, $e \in SDS_u$ or $e \in EQS_u$. Thus $e \in SDS_u \cup EQS_u$. Therefore, we have $SDS_u \cup EQS_u \not\subseteq SDS_v \cup EQS_v$, a contradiction. □

Subspace $\mathcal{B}$ is a skyline subspace (with respect to q) if and only if $CAND_{\mathcal{B}}$ does not contain any vertices that dominate q in subspace $\mathcal{B}$. As long as $CAND_{\mathcal{B}}$ contains one vertex, say (w, dom) that dominates q, $\mathcal{B}$ is not a skyline subspace. We determine whether a subspace $\mathcal{B}$ is skyline subspace by checking whether there exists such a (w, dom) that $(w, dom) \in CAND_{\mathcal{B}}$. By Lemma 4.3.1, we notice that (u, dom) always comes with the vertex (v, dom). Thus, given a subspace $\mathcal{B}$, pruning u from $CAND_{\mathcal{B}}$ does not affect the result of deciding whether the subspace $\mathcal{B}$ is a skyline subspace.

Given a point u, if we are able to find such a point v that $(SDS_u \cup EQS_u \subseteq SDS_v \cup EQS_v) \wedge (SDS_u \subseteq SDS_v)$, by Lemma 4.3.1, u can be pruned from the *dominating candidate*

sets that u belongs to.

How to find such a vertex v? One way to find such a v is to enumerate all the vertices in the graph G to check whether the statement $(SDS_u \cup EQS_u \subseteq SDS_v \cup EQS_v) \wedge (SDS_u \subseteq SDS_v)$ is true or not. It takes $O(|V|)$ time to determine whether a vertex u can be pruned from its *dominating candidate sets*.

In our thesis, we heuristically search for the vertex v from the neighbours of the query vertex q. The intuition is that usually the neighbours of query vertex q will have *strictly dominating subspaces* with more dimensions, comparing to the vertices that are not the neighbours of query vertex. The reason is that for any label D, every vertex w (except for q itself) on the shortest path from the query vertex q to the label D has $w.D < q.D$. In other words, the *strictly dominating subspaces* of the vertices on the shortest path must contain dimension D. Heuristically, a neighbour vertex of the query vertex q is on multiple shortest paths from the query vertex q to the labels.

For example, in Figure 4.3, one of the neighbours w is on the shortest paths from query vertex q to label D, label E, and label F. Thus, the neighbours of the query vertex q tend to have *strictly dominating subspaces* with multiple dimensions. In Figure 4.3, we have $SDS_u = \{A, B\}, SDS_x = \{C\}, SDS_w = \{D, E, F\}$. By Lemma 4.3.1, the vertices enclosed by the solid line can be pruned by vertex u, the vertices enclosed by the dot line can be pruned by vertex x, and the vertices enclosed by the dash line can be pruned by vertex w. We develop a 1-hop pruning algorithm. In Algorithm 3, S contains all query vertex's neighbours. If both SDS_u and $SDS_u \cup EQS_u$ are subsets of SDS_v and $SDS_v \cup EQS_v$ ($v \in s$), respectively, then we prune u from the *dominating candidate sets*. The running time is $O((|V| * |d|)|L|)$ ($|d|$ is the degree of the query node q).

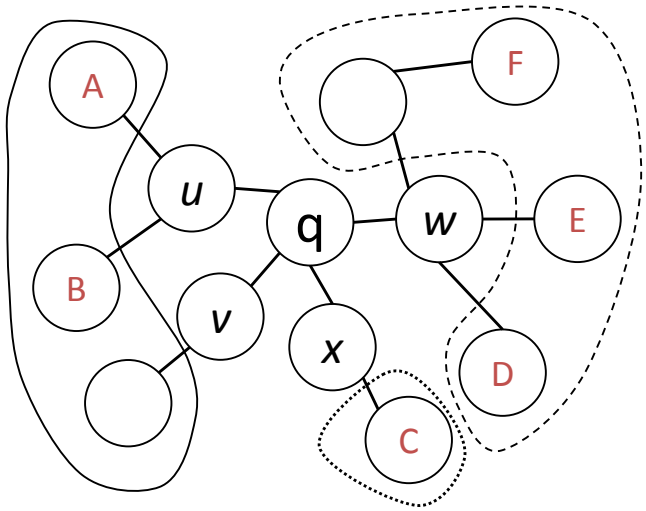

Figure 4.3: Example of neighbours of query vertex q

Algorithm 3 1-hop Pruning

Input: Strictly Dominating Subspace SDS, Equivalence Subspace EQS, Dominating candidate $CAND$, query vertex q, graph $G = (V, E)$;

Output: Pruned Dominating candidate $CAND$;

```
1: S = neighbours of query point q
2: for all (l, dist) in LV_q do
3:    for all u in CAND_l do
4:       for all v in S do
5:          if u ≠ v ∧ SDS_u ∪ EQS_u ⊆ SDS_v ∪ EQS_v ∧ SDS_u ⊆ SDS_v then
6:             delete u from CAND_l
7:          end if
8:       end for
9:    end for
10: end for
```

Table 4.7 shows SDS and EQS of all vertices on the graph from Figure 3.1. We can prune the some of the vertices from the *dominating candidate set* according this table. By Lemma 4.3.1, vertex v can be pruned by u and vertex w can be pruned by y.

Dim	SDS	EQS	$SDS \cup EQS$
u	A	$\emptyset$	A
v	$\emptyset$	A	A
w	BC	$\emptyset$	BC
x	$\emptyset$	$\emptyset$	$\emptyset$
y	BC	$\emptyset$	BC

Table 4.7: The SDS and EQS of each vertex

Distances	A	B	C	Pruned By
u	0	∞	∞	
v	1	∞	∞	u
w	∞	0	0	y
x	∞	∞	∞	u
y	∞	0	1	
q	1	1	2	

Table 4.8: Label distance vector after 1-hop pruning

Subspaces	Dominating candidate
A	(u, dom)
B	(y, dom)
C	(y, dom)

Table 4.9: Pruned dominating candidate set of 1-dimensional subspace

Table 4.8 shows that v and x are pruned by u. Also, y is pruned by w. Table 4.8 shows the vertices to be pruned and their labeled distance vectors. Although w seems to be better than y (because w dominates y), vertex w is pruned by vertex y because both vertices y and vertex w have the same SDS and EQS and y is a 1-hop neighbour of the query vertex q. Table 4.9 shows the dominating candidate sets of the 1-dimensional subspaces after the 1-hop neighbour pruning.

4.3.1 Dominating Candidate Sets for Multi-dimensional Subspaces

We generate the dominating candidate set on multi-dimensional subspaces by intersecting the dominating candidate sets of subspaces with lower dimensionality. The $CAND$ set

intersection operation is computed in the following way: Given subspaces $\mathcal{A}$ and $\mathcal{B}$,

$$\begin{aligned} W &= \{(u, dom) | \forall u, (u, dom) \in CAND_{\mathcal{A}} \wedge (u, eq) \in CAND_{\mathcal{B}}\} \\ X &= \{(u, dom) | \forall u, (u, eq) \in CAND_{\mathcal{A}} \wedge (u, dom) \in CAND_{\mathcal{B}}\} \\ Y &= \{(u, dom) | \forall u, (u, dom) \in CAND_{\mathcal{A}} \wedge (u, dom) \in CAND_{\mathcal{B}}\} \\ Z &= \{(u, eq) | \forall u, (u, eq) \in CAND_{\mathcal{A}} \wedge (u, eq) \in CAND_{\mathcal{B}}\} \\ CAND_{\mathcal{A} \cup \mathcal{B}} &= W \cup X \cup Y \cup Z \end{aligned}$$

For example, we consider two dominating candidate sets $CAND_{\mathcal{A}} = \{(u, dom)\}$ and $CAND_{\mathcal{B}} = \{(u, eq)\}$. We want to compute the CAND intersection of these two dominating candidate sets $CAND_{\mathcal{A} \cup \mathcal{B}}$. Both $CAND_{\mathcal{A}}$ and $CAND_{\mathcal{B}}$ contain the element u. In this example, vertex u dominates query vertex q in subspace $\mathcal{A}$ and vertex u is equal to query vertex q in subspace $\mathcal{B}$. Therefore, vertex u dominates query vertex q in the subspace $\mathcal{A} \cup \mathcal{B}$ because u is less than q in at least one dimension of $\mathcal{A} \cup \mathcal{B}$ (u dominates q in $\mathcal{A}$).

4.3.2 Bottom-up Subspace Eumeration

In this section, we introduce a bottom-up algorithm to compute the dominating candidate sets from 1-dimensional subspaces to n-dimensional subspaces. This algorithm is inspired by the Apriori algorithm for mining association rules [2]. A similar bottom-up algorithm has also been used in enumerating different subspaces in subspace clustering problem [1].

Property 4.1. *If in a k-dimensional subspace $\mathcal{B}$, $CAND_{\mathcal{B}}$ contains a set of vertices S, then in any (k-1)-dimensional projection $\mathcal{C}$ of $\mathcal{B}$, $CAND_{\mathcal{C}}$ also contains the set of vertices S.*

Proof. If a vertex v is in $CAND_{\mathcal{B}}$, where space $\mathcal{B}$ is a k-dimensional subspace, then the vertex v is not greater than query vertex q in any dimension of space $\mathcal{B}$. For any (k-1)-dimensional projection $\mathcal{C}$ of B, the vertex v is also is not greater than query vertex q in any dimension of projection $\mathcal{C}$. Therefore, in projection $\mathcal{C}$, $CAND_{\mathcal{C}}$ contains the vertex v. □

The algorithm computes the dominating candidate sets from one dimensional subspaces to n-dimensional subspaces. We already introduce an algorithm to compute the dominating candidate sets on all 1-dimensional subspaces. Let's say that L_k is the set of k-dimensional subspaces on which the dominating candidate sets are non-empty. Having all the dominating candidate sets on all $(k-1)$-dimensional subspaces, we are able

to construct the non-empty dominating candidate sets on k-dimensional subspaces $L_k = \{\mathcal{A} \cup \{b\} \,|\mathcal{A} \in L_{k-1} \wedge b \in \bigcup L_{k-1} \wedge b \notin \mathcal{A}\}$.

Algorithm 4 Subspace Eumeration

Input: Dominating candidate $CAND$ of all one dimension subspaces in LV_q;

Output: A set of subspaces SUB where query vertex q is a skyline point;

1: **for all** $(l, dist)$ in LV_q **do**
2: if $\exists u, (u, dom) \in CAND_l$, add l to SUB, else add l to L_1
3: **end for**
4: $k = 2$
5: **while** $L_{k-1} \neq \emptyset$ **do**
6: $L_k = \{\mathcal{A} \cup \{b\} \,|\mathcal{A} \in L_{k-1} \wedge b \in \bigcup L_{k-1} \wedge b \notin \mathcal{A}\}$
7: **for all** $\mathcal{C}$ in L_k **do**
8: **for all** (k-1)-projection $\mathcal{S}$ of $\mathcal{C}$ do **do**
9: **if** $\mathcal{S} \notin L_{k-1}$ **then**
10: delete $\mathcal{C}$ from L_k
11: **end if**
12: **end for**
13: **end for**
14: **for all** $\mathcal{C}$ in L_k **do**
15: Let $\mathcal{A}$ and $\mathcal{B}$ be two subspace of $\mathcal{C}$
16: $CAND_{\mathcal{C}} = CAND_{\mathcal{A}} \cap CAND_{\mathcal{B}}$
17: **if** $CAND_{\mathcal{C}}$ is empty **then**
18: add $\mathcal{C}$ to SUB
19: remove $\mathcal{C}$ from L_k
20: **end if**
21: **end for**
22: $k = k + 1$
23: **end while**
24: **return** SUB

We still take the LinkedIn network in Table 3.2 and Figure 3.1 as our running example. By running the *set enumeration* algorithm we get the *dominating candidate sets* in all subspaces in Table 4.10.

Subspaces	Dominating Candidates	Generated by
A	(u, dom)	
B	(w, dom)	
C	(w, dom)	
AB	$\emptyset$	$A \cap B$
AC	$\emptyset$	$A \cap C$
BC	(w, dom)	$B \cap C$

Table 4.10: Dominating candidate sets of all subspaces

As shown in Table 4.10, $CAND_{BC}$ is computed by $CAND_B \cap CAND_C$, $CAND_{AB}$ is computed by $CAND_A \cap CAND_B$ and $CAND_{AC}$ is computed by $CAND_A \cap CAND_C$. In this example, $CAND_{BC} = CAND_B \cap CAND_C = \{(w, dom)\}$, $CAND_{AB} = \emptyset$ and $CAND_{AC} = \emptyset$. The query vertex q is a skyline point on subspaces AB and AC because their dominating candidate sets are all empty.

Chapter 5

Spatial Skyline Subspace

In this chapter, we will introduce the algorithms to compute the spatial skyline subspace queries in range r. In the computation of the spatial skyline subspace queries, we use the same set enumeration framework as shown in Chapter 4 with different pruning techniques. We introduce a method to prune some of the *dominating candidates* based on the geometric property of the spatial skyline subspace query problem.

5.1 Label Collecting in Range r

We index the data points in an R-tree. In an R-tree we can get all the points in a certain rectangle in $O(M \log_M n)$ time [10], where M is the maximum number of entries of each bounding box and n is the total number of points. By indexing the data points in an R-tree, if we want to access the neighbouring points of a certain query point q, we can query the square box with its center on point q in the R-tree without accessing the whole set of data points. We compute the *labeled distance vector* of the query point q by retrieving the labels of the points in the circle with center $(p.x, p.y)$ and radius r. To retrieve the points in the circle, we make a rectangle query on the R-tree with the lower-left corner point $(p.x - r, p.y - r)$ and the upper-right corner point $(p.x + r, p.y + r)$, and check whether those points with labels are in range r of the query point q. Then we collect the distances to those labels from q as the labeled distance vector of q.

Consider the example in Figure 3.2 and Table 3.4 in Chapter 3, we compute the spatial skyline subspace query of q in range $r = 2$. We first make a query on R-tree to get all the points in rectangle $(0, 0)$, $(4, 4)$. Then we have points x, y, w, u in range $r = 2$ of the query

point q. The distances from q to u, y, w are $\sqrt{2}$, 1, 2, respectively. Therefore, the labeled distance vector of query point q is $(\sqrt{2}, 1, 2)$.

5.2 Dominating Candidates in Spatial Subspace Skyline

In this section, we will show an algorithm to compute the *CAND* of all 1-dimensional subspaces.

Algorithm 5 Dominating Candidates

Input: R-tree R that indexes the spatial points and labeled distance vector LV_q of query point q;

Output: Dominating Candidates Set *CAND* of all 1-dimensional subspaces, *SDS* and *EQS* of all points;

```
1:  for all (l, dist) in LV_q do
2:    for all point p contains label l do
3:      rec = R.query(p.x − dist, p.y − dist, p.x + dist, p.y + dist)
4:      for all point u in rec do
5:        d = distance(u, p)
6:        if d < dist then
7:          add (u, dom) to CAND_l
8:          add l to SDS_u
9:        end if
10:       if d == dist then
11:         add (u, eq) to CAND_l
12:         add l to EQS_u
13:       end if
14:     end for
15:   end for
16: end for
```

In Algorithm 5, we construct the labeled distance vectors of other points from every point with labels. For each label l, we enumerate all the points that contain the label l. For each point p with label l, we retrieve all the points in range $q.l$ of p. For each point u in range $q.l$ of p ($q.l$ is the distance from q to label l), we assign the distance between p and

u to $u.l$. The running time of the algorithm is $O(M \log n)$. M is the number of points in query rectangle of the point q.

We still consider the Figure 3.2 and Table 3.4 as the running example. We start from the point u which contains the label A and get all the points in range $q.A = \sqrt{2}$ of u. Then we have point v in range $\sqrt{2}$ of u and let $v.A$ be the distance from u to v which is $\sqrt{2}$. Thus, we put (v, eq) to $CAND_A$, because $v.A = q.A$. The *labeled distance vectors* of all the points are shown in Table 5.1. The dominating candidate sets of all the 1-dimensional subspaces are shown in Table 5.2. The *strictly dominating subspaces* and the *equivalence subspaces* of all the points are shown in Table 5.3.

Distances	A	B	C
u	0	∞	∞
v	$\sqrt{2}$	∞	∞
w	∞	0	0
x	∞	∞	∞
y	∞	0	1
q	$\sqrt{2}$	1	2

Table 5.1: Labeled distance vector of each point

Subspaces	Dominating candidate
A	$(u, dom), (v, eq)$
B	$(w, dom), (y, dom)$
C	$(w, dom), (y, dom)$

Table 5.2: Dominating candidate set of each 1-dimensional subspace

	SDS	EQS
u	A	$\emptyset$
v	$\emptyset$	A
w	BC	$\emptyset$
x	$\emptyset$	$\emptyset$
y	BC	$\emptyset$

Table 5.3: SDS and EQS of each point

5.3 Same Region Pruning

After getting the list of dominating candidate sets and the *SDS* and *EQS* of all the points, we can prune some of the candidate points of the dominating candidate sets. By Lemma 4.3.1, we can see that if two points u and v have the same SDS_u and SDS_v, and they have the same EQS_u and EQS_v, then one of them can be pruned by the other.

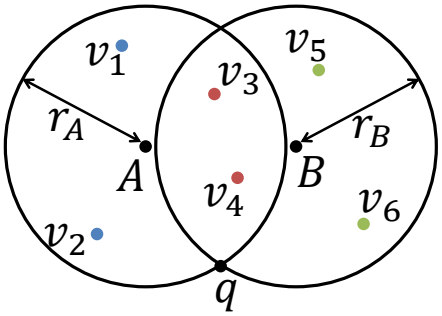

Figure 5.1: We can pick one candidate from each region (with same color) and eliminate the others.

We will show an example of how the spatial points can be pruned from the dominating candidate set. In Figure 5.1, point q is the query point, point A is a point with label A and point B is a point with label B. r_A is the distance between the query point q and the label A, and r_B is the distance between the query point q and the label B. The blue points (v_1 and v_2) dominate the query point q in the 1-dimensional subspace A. The green points (v_5 and v_6) dominate the query point q in subspace B. The red points (v_3 and v_4) dominate the query point q in subspace (A, B). In this example, the points with the same colors,

(v_1, v_2), (v_3, v_4), (v_5, v_6) have the same *strictly dominating subspace SDS* and the same *equivalence subspace EQS*. Therefore, we can keep one of the points from each region as a representative and prune the others from their *dominating candidate sets*. We will show that the number of regions is bounded by the square of the number the of vertices with labels.

Property 5.1. *n circle can only divide the 2-dimensional space into at most $n(n-1)+2$ different regions.*

Proof. Proof by induction.
Basic: If $n = 1$, then one circle divide the 2-dimensional space into 2 different regions.
Inductive step: Assume that the statement holds for k, i.e., k circles can only divide the space into at most $k(k-1)+2$ different regions. The $(k+1)$th circle can only intersect with at most k circle with $2k$ intersection points. Then the $(k+1)$th can only be divided into $2k$ segments and each segment divides the region it is located in into two. Therefore, $k+1$ circles can only divide the space into at most $k(k-1)+2+2k = k(k+1)+2$ different regions.
Since both the basis and the inductive step has been performed, by mathematical induction, the statement holds for all natural number n. □

By Property 5.1, we know that the total number of regions is not greater than the square of total number of vertices containing label.

In our algorithm, we determine whether two points u, v are in the same region by checking whether SDS_u and SDS_v are the same.

Algorithm 6 Same Region Pruning

Input: Dominating Candidates $CAND$, query vertex q, graph $G = (V, E)$;
Output: Pruned Dominating Candidates $CAND$;

```
1: for all (l, dist) in LV_q do
2:   for all u in CAND_l do
3:     if u is in the same region as other points then
4:       delete u from CAND_l
5:     end if
6:   end for
7: end for
```

By applying the Algorithm 6, we prune some of the elements in the *dominating candidate set* in all 1-dimensional subspace. Then we will use the same set-enumeration method of skyline subspace computation from Chapter 4 to compute the *dominating candidate sets* of all the subspaces in order to answer the *spatial skyline subspace queries*. The running time of the algorithm is $O(|L||V|)$.

Chapter 6

Experiments

In this chapter, we will evaluate our approach by reporting the experimental results with respect to the running time on computing the subspace skyline on both real data sets and synthetic data sets. We compare the running time of the algorithms with and without using pruning method. Both of the algorithms compute the subspace skyline using *dominating candidate sets* enumeration framework. The only difference between these two algorithms we compare is whether the pruning method is applied.

We implement our algorithms using C++. We use Microsoft Visual Studio 2010 to compile our C++ programs. Experiments are conducted on a PC with an Intel Core(TM) i7-3779 3.40GHz CPU, 16GB main memory and a 900G hard disk, running the Microsoft Windows 7 Enterprise Edition operating system.

6.1 Experiments of Skyline Subspace Query on Graph

In this section, we will introduce the empirical studies on skyline subspace queries on graphs. In all of the experiments of this section, we uniformly choose 100 nodes as the query nodes and compute the average running time of those queries. Using the model of Kronecker graph provided by [14], we generate graphs of different sizes. According to [14], a Kronecker graph has several real world network properties: heavy tails for the in-degree and out-degree distributions; heavy tails for the eigenvalues and eigenvectors; small diameters; and the "Densification Power Law" (DPL).

We use the MATLAB code from *graph500* [1] to generate graphs with scale from 15 to 20 which corresponds to the number of vertices from 2^{15} to 2^{20}. All the graphs with different sizes are generated from the initial matrix:

$$\begin{bmatrix} 0.3 & 0.24 \\ 0.24 & 0.22 \end{bmatrix} \tag{6.1}$$

with edge factor 2 which is the expected average degree of the vertices. The label is generated in power law distribution. We use the Pareto function from a python package numpy [2] to generate the label information of the vertices. We randomly generate 1000 different labels for these graphs in power law distribution. Our algorithms run skyline subspace queries in 2-hops neighbourhood. Figure 6.1 shows that the speed-up factor of algorithm with pruning is increasing with increasing number of labels. Neither method is linearly scalable with respect to the number of the labels.

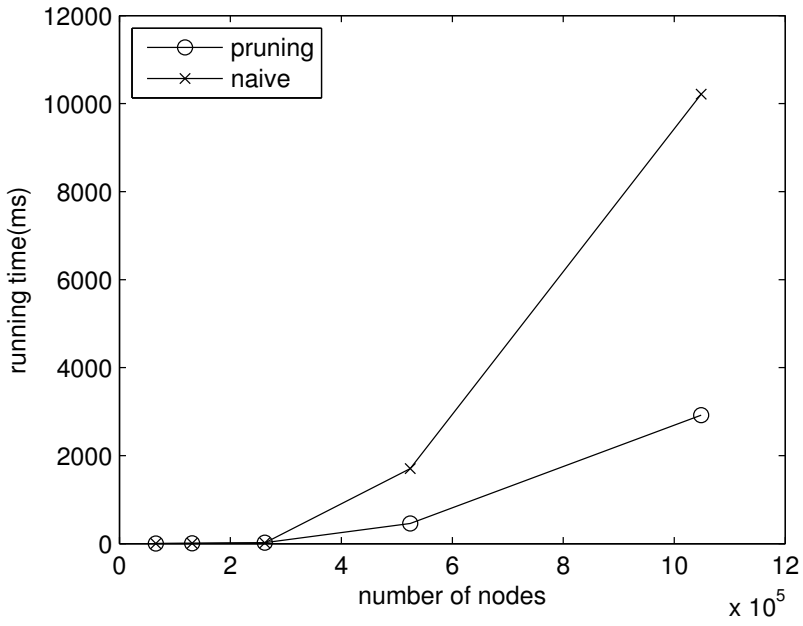

Figure 6.1: Kronecker graphs with 1000 different labels

We also do some empirical studies on real world datasets. We download a dataset of

[1] http://www.graph500.org/specifications#sec-3_3

[2] http://docs.scipy.org/doc/numpy/reference/generated/numpy.random.pareto.html

datasets	Facebook Network	DBLP Network
number of nodes	4039	906505
number of edges	88234	1656732
number of different labels	20	1000
average degree	21.8455	1.8276

Table 6.1: Dataset statistics

Facebook network from *Stanford Network Analysis Project* [3]. In Figure 6.2, we show the running time of our algorithms in 3-hops neighbourhood with different numbers of unique labels in total. The labels are assigned to the vertices in uniform distribution. In the Facebook network dataset, the running time of the algorithm with pruning is much less than the running time of the algorithm without pruning. We note that neither method is linearly scalable with respect to the size of the graphs.

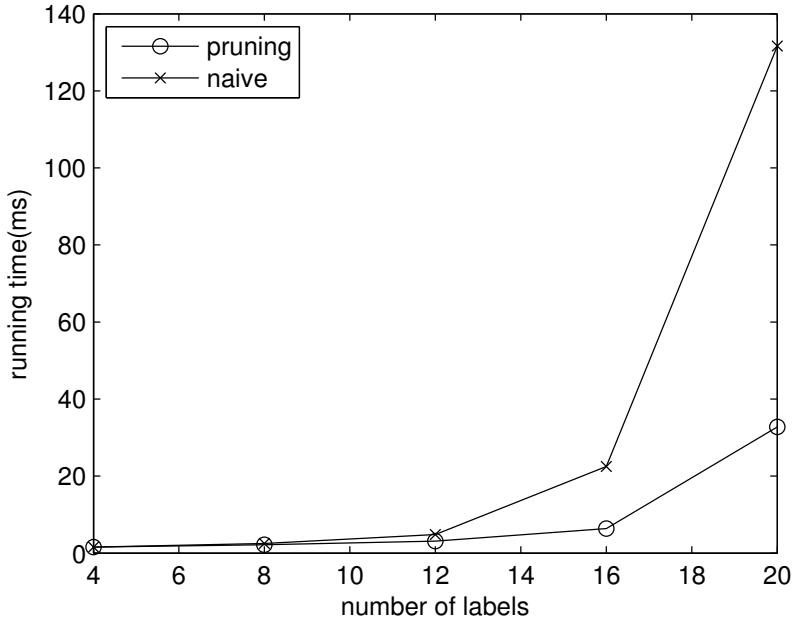

Figure 6.2: Running time of the algorithms on facebook network

Figure 6.3 shows the distribution of the degrees in the Facebook network dataset. The degrees of most of the vertices are less than 200 while some of the vertices are with degrees

[3] http://snap.stanford.edu/data/egonets-Facebook.html

greater than 800.

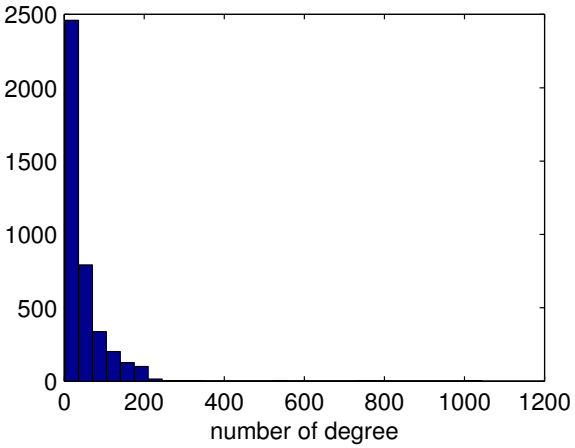

Figure 6.3: Facebook network degree distribution

Figure 6.4 shows the running time of the queries with different numbers of hops on Facebook network with 20 unique labels. We conduct this experiment by using different hop numbers d in the input. The speed-up factor of the pruning-based algorithm is increasing with the increasing number of hops.

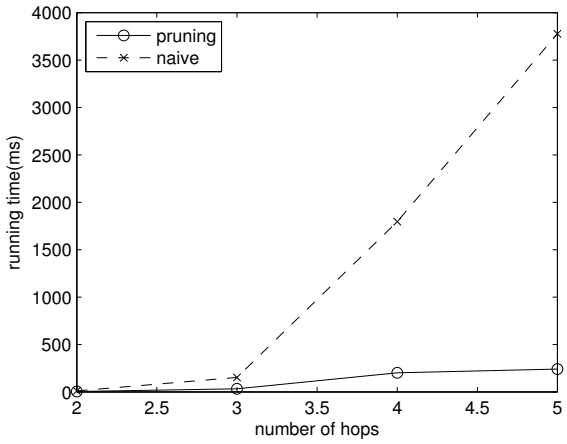

Figure 6.4: Running time of the queries different numbers of hops

Figure 6.5 shows the running time of the queries with different label densities on Facebook dataset. We use the number of labels each node contains in average in x-axis. It shows that it takes a long time to answer the queries when the label density is 1. The running

time of both end (too dense or too sparse) are relatively smaller. The reason is that if the label density is too sparse then the query node tends to have a distance vector with lower dimensionality. If the label density is too dense then the distance vector of the query node tends to have a number of dimensions with values 0. It also shows that our pruning method is more suitable for the labeled graph with sparse label distribution.

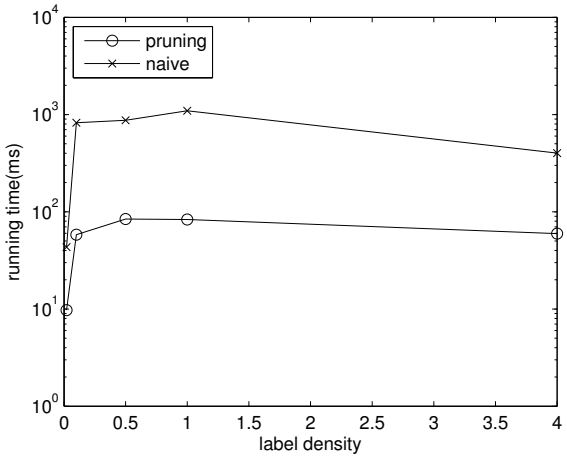

Figure 6.5: Running time of different label densities

We also use the DBLP dataset from *arnetminer* [4] to evaluate the performance of our algorithms. The DBLP dataset provides us the author list of each publication. We build a citation network based on these author lists in the following way. We add an edge between author X and author Y if X is one of the top ten co-authors of Y and Y is also one of the top ten co-authors of X. We choose the top one thousand most frequent conferences as the labels of the vertices. If an author publishes more than five papers in a conference, then we add the corresponding label of that conference to that author. Figure 6.6 shows that the pruning-based algorithm is about 10 times faster than the algorithm without pruning on DBLP network. We note that neither algorithm is linearly scalable with respect to number of conferences.

[4] http://arnetminer.org/billboard/citation

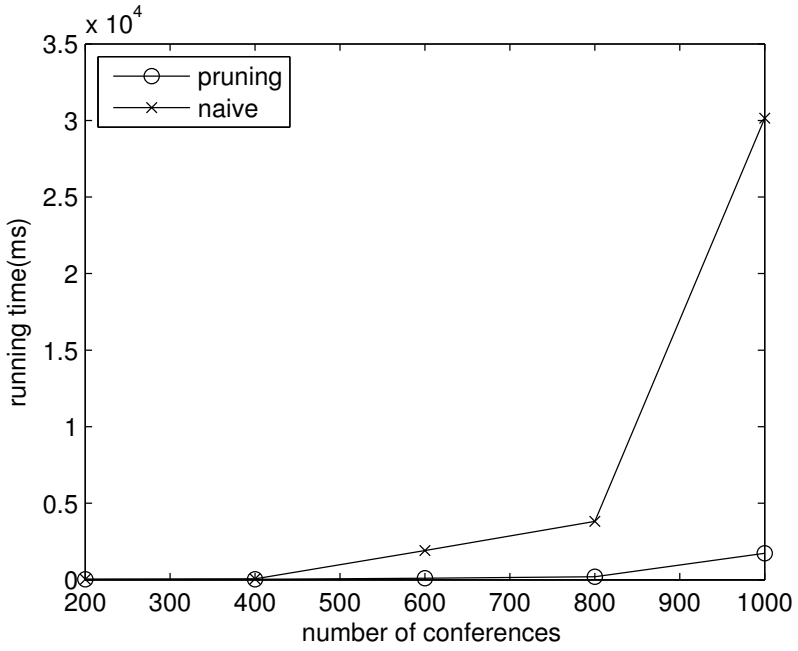

Figure 6.6: Running time of the algorithms on DBLP network

Figure 6.7 shows the distribution of the degrees of the DBLP network. Comparing to the facebook network, the variance of the degrees of the vertices in the DBLP network is small.

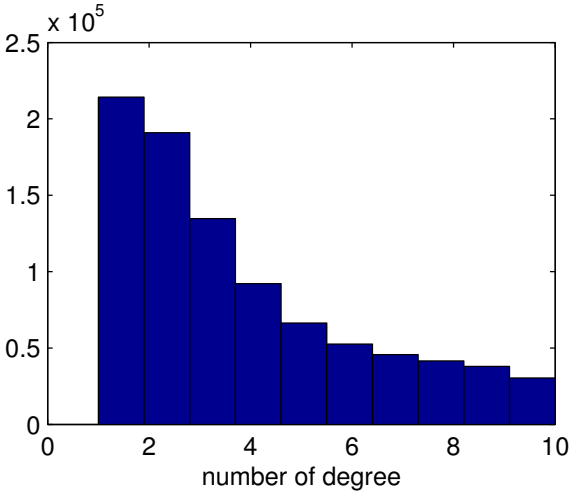

Figure 6.7: DBLP network degree distribution

6.2 Experiments of Spatial Skyline Subspace Query

In this section, we will introduce the empirical studies on spatial skyline subspace queries. We use the Yelp academic dataset [5] to evaluate our algorithms. Yelp academic dataset provides 13490 different business spots. Each business spot consists of its longitude and latitude information. We use the longitude and latitude information of the business spot as its spatial information. The business spot consists of a set of category information, neighbourhood information and university information. We use that information as the labels of the business spot. Each business spot also contains a star rating.

In Figure 6.8, we choose the top 20 most popular categories as the labels of the business spots and we compare the running time of the algorithms in the sense of different radii. The running time of the prune-based algorithm is about 2 times faster than the algorithm without pruning.

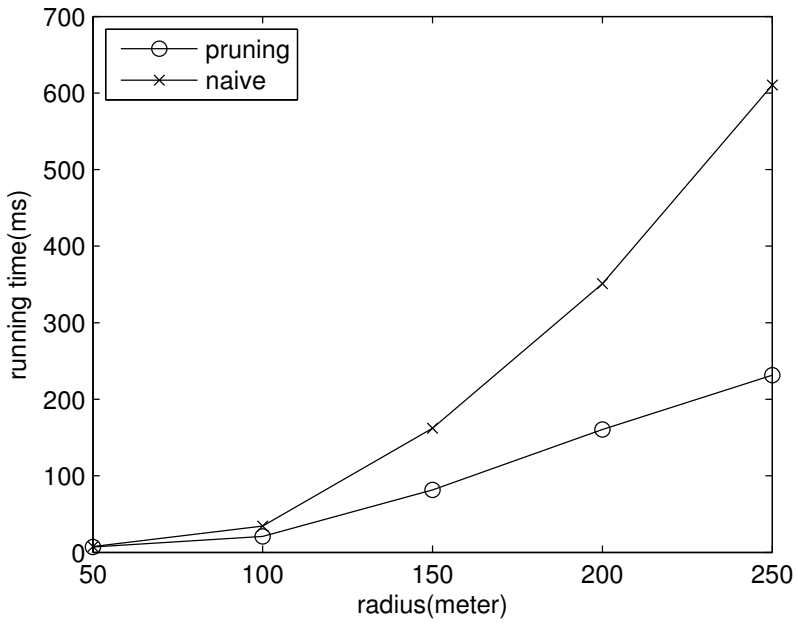

Figure 6.8: Yelp Data Set with Top 20 most popular categories

In Figure 6.9, we choose the top 300 most popular categories as the labels of the business spots. We assign a category to a business spot as a label if the business has the highest star rating among all the business with that category. We note that both algorithms are

[5] https://www.yelp.ca/academic_dataset

linearly scalable with respect to radius. Table 6.2 shows the average numbers of business spots in terms of radius in yelp dataset.

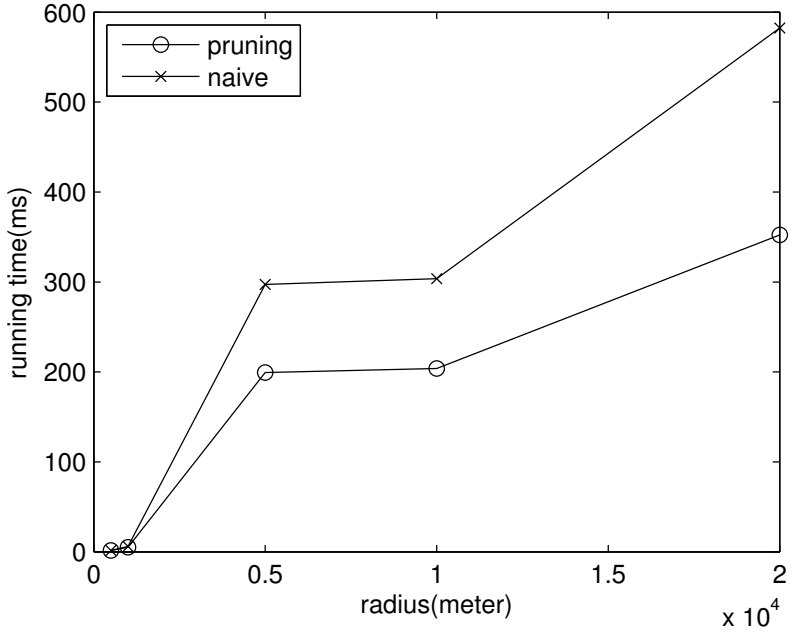

Figure 6.9: Yelp Data Set with 300 different labels

We also evaluate the running time of the algorithm on data with the radius fixed but the number of labels is different. In this thesis, we conduct the empirical study on the spatial skyline queries with radius of 10000 meters. We first get a random permutation of the labels. Second, we run our programs on the first 100 labels, the first 200 labels, ..., etc. Figure 6.10 shows the running time per query in average. The running time of the pruning-based algorithm does not have significant better performance than the algorithm without pruning.

Figure 6.11 shows the running time of spatial synthetic data with 20 different labels. Both the spatial points and the labels of the points are generated randomly in uniform distribution. The pruning-based algorithm is 7 times faster than the naive algorithm when the radius is 40000 meters.

In summary, the experimental result shows that our algorithms compute the skyline subspace efficiently and the pruning methods improve the running time of the programs effectively.

Radius (meters)	Number of Points
50	11
100	26
150	42
200	59
250	76
500	159
1000	298
5000	597
10000	657
20000	721

Table 6.2: Number of points in certain radii in average

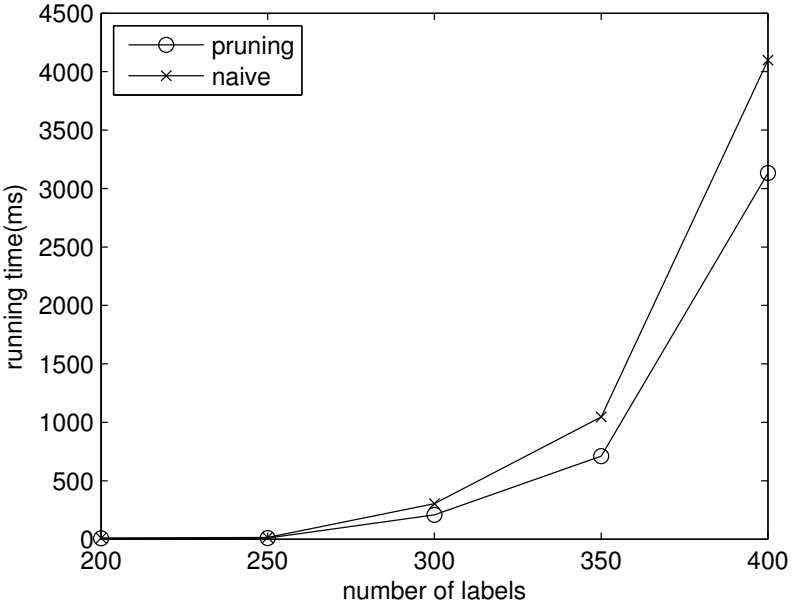

Figure 6.10: Yelp dataset in 10000 meters neighbourhood

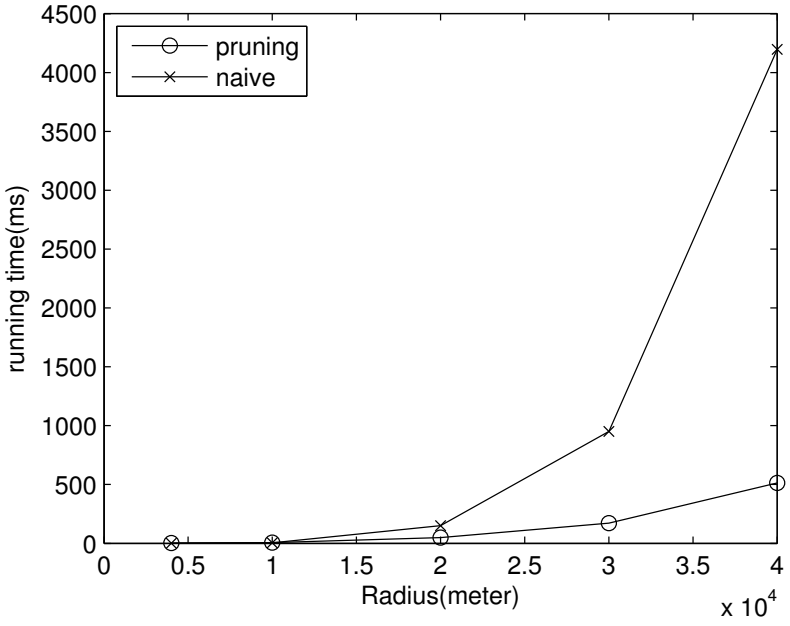

Figure 6.11: Spatial synthetic dataset

Chapter 7

Conclusions

The skyline subspace problem is originally motivated by the problem of what distinguishes one from its peers in social networks [15]. In this thesis, we formulate the social network into a graph with labels and consider the distances between a person and the labels as the factors that distinguish the person from its peers. We propose a bottom-up algorithm to answer *skyline subspace query* which is based on set enumeration and dominating candidate sets intersection. To tackle the problems of skyline subspaces on graph and the skyline subspaces on Euclidean space, we develop effective pruning methods to reduce the search space. We do empirical studies using both synthetic and real datasets to evaluate our approach. We generate the synthetic graph based on the Kronecker graph model and the real world datasets are from DBLP and YELP. The experimental results verify the efficiency of our algorithms.

As for future work, we can consider the following directions.

- *Using top-down set enumeration.* Our algorithm is based on bottom-up set enumeration. In bottom-up manner, we take the advantage of the property that if a target skyline subspace is found then we do not need to search for the subspaces that contain this subspace. For top-down approach, one of the advantage we can take is that if the query point is strictly dominated by some points in a certain subspace $\mathcal{A}$, then we do not need to check the subsets of the subspace $\mathcal{A}$ because we know that the query point cannot be a skyline point in those subspaces.

- *Further pruning method development in Euclidean space.* There are many properties in euclidean space. In [19], Sharifzadeh et al. took the advantage of the property

of convex hull to reduce the size of skyline candidates. They also used the Voronoi diagram structure to index the graph. For future work, we can index the spatial points using Voronoi diagram instead of R-tree and apply pruning methods based on some geometry properties such as the property of convex hull.

- *Skyline subspace algorithm on other applications.* Finding the skyline subspaces on road networks and metric space are still an open problem. This topic is an interesting problem to be studied in the future.

Bibliography

[1] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. *Automatic subspace clustering of high dimensional data for data mining applications*, volume 27. ACM, 1998. 28

[2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, A. I. Verkamo, et al. Fast discovery of association rules. *Advances in knowledge discovery and data mining*, 12(1):307–328, 1996. 28

[3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. *The R*-tree: an efficient and robust access method for points and rectangles*, volume 19. ACM, 1990. 7

[4] S. Borzsony, D. Kossmann, and K. Stocker. The skyline operator. In *Data Engineering, 2001. Proceedings. 17th International Conference on*, pages 421–430. IEEE, 2001. 1, 4, 6, 9

[5] L. Chen and X. Lian. Dynamic skyline queries in metric spaces. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, pages 333–343. ACM, 2008. 9

[6] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE*, volume 3, pages 717–719, 2003. 4, 6

[7] E. Dellis and B. Seeger. Efficient computation of reverse skyline queries. In *Proceedings of the 33rd international conference on Very large data bases*, pages 291–302. VLDB Endowment, 2007. 10

[8] K. Deng, X. Zhou, and H. T. Shen. Multi-source skyline query processing in road networks. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 796–805. IEEE, 2007. 8, 9

[9] D. Fuhry, R. Jin, and D. Zhang. Efficient skyline computation in metric space. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, pages 1042–1051. ACM, 2009. 9

[10] A. Guttman. *R-trees: a dynamic index structure for spatial searching*, volume 14. ACM, 1984. 7, 31

[11] W. Jin, A. Tung, M. Ester, and J. Han. On efficient processing of subspace skyline queries on high dimensional data. In *Scientific and Statistical Database Management, 2007. SSBDM'07. 19th International Conference on*, pages 12–12. IEEE, 2007. 8

[12] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 275–286. VLDB Endowment, 2002. 6

[13] H.-T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *Journal of the ACM (JACM)*, 22(4):469–476, 1975. 6

[14] J. Leskovec, D. Chakrabarti, J. Kleinberg, and C. Faloutsos. Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. In *Knowledge Discovery in Databases: PKDD 2005*, pages 133–145. Springer, 2005. 37

[15] Y.-C. Lo, J.-Y. Li, M.-Y. Yeh, S.-D. Lin, and J. Pei. What distinguish one from its peers in social networks? *Data Mining and Knowledge Discovery*, 27(3):396–420, 2013. 47

[16] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 467–478. ACM, 2003. 7, 9, 10

[17] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Transactions on Database Systems (TODS)*, 30(1):41–82, 2005. 7

[18] J. Pei, W. Jin, M. Ester, and Y. Tao. Catching the best views of skyline: A semantic approach based on decisive subspaces. In *Proceedings of the 31st international conference on Very large data bases*, pages 253–264. VLDB Endowment, 2005. 4, 7

[19] M. Sharifzadeh and C. Shahabi. The spatial skyline queries. In *Proceedings of the 32nd international conference on Very large data bases*, pages 751–762. VLDB Endowment, 2006. 8, 47

[20] K.-L. Tan, P.-K. Eng, B. C. Ooi, et al. Efficient progressive skyline computation. In *VLDB*, volume 1, pages 301–310, 2001. 7

[21] Y. Tao, X. Xiao, and J. Pei. Subsky: Efficient computation of skylines in subspaces. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 65–65. IEEE, 2006. 7, 10, 11

[22] Y. Yuan, X. Lin, Q. Liu, W. Wang, J. X. Yu, and Q. Zhang. Efficient computation of the skyline cube. In *Proceedings of the 31st international conference on Very large data bases*, pages 241–252. VLDB Endowment, 2005. 4, 7

[23] L. Zou, L. Chen, M. T. Özsu, and D. Zhao. Dynamic skyline queries in large graphs. In *Database Systems for Advanced Applications*, pages 62–78. Springer, 2010. 9