Space Adaptation Techniques for Preference Oriented Skyline Processing

by

Sriram Rathinavelu

A Thesis Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

Approved November 2014 by the
Graduate Supervisory Committee:

Kasim Candan, Chair
Hasan Davulcu
Mohamed Sarwat

ARIZONA STATE UNIVERSITY

December 2014

ABSTRACT

Skyline queries are a well established technique used in multi-criteria decision applications. There is a recent interest among the research community to efficiently compute skylines but the problem of presenting the skyline that takes into account the preferences of the user is still open. Each user has varying interests towards each attribute and hence “one size fits all” methodology might not satisfy all the users. True user satisfaction can be obtained only when the skyline is tailored specifically for the user based on his preferences.

This research investigates the problem of preference aware skyline processing which consists of inferring the preferences of the user and computing a skyline specific to that user, taking into account his preferences. This research proposes a transformation that transforms the data from a given space to a user preferential space where each attribute represents a preference of the user. This study proposes two techniques *Preferential Skyline Processing* and *Latent Skyline Processing* to efficiently compute preference aware skylines in the user preferential space. Finally, through extensive experiments and performance analysis the correctness of the recommendations and the algorithm’s ability to outperform the naive ones are confirmed.

DEDICATION

To my Dad.

ACKNOWLEDGEMENTS

I take this opportunity to extend my heartfelt gratitude to my advisor, Dr. K. Selçuk Candan. I am very thankful for his insightful guidance and inspiring thoughts which has totally transformed my problem solving attitude and made my research possible. I am certain the skills, expertise and wisdom he imparted on me will help my future endeavours. I am grateful to my committee members Dr. Hasan Davulcu and Dr. Mohamed Sarwat for being part of my committee and their valuable guidance for shaping my thesis. I would like to thank the advisors from school of computing, informatics and decision systems engineering and International Students and Scholars Office for helping me with all my paperwork needed to complete my degree. I am grateful to developers of Euclidean Vector Algebra (EVA). EVA is a symbolic calculator to work with Clifford numbers. I would like to thank my lab mates Mijung Kim, Jung Hyun Kim, Parth Nagarkar, Mithila Nagendra, Shengyu Huang, Xiaolan Wang, Xilung Chen, Xinsheng Li, Sicong Liu and Yash Garg for their thoughtful discussions and advice. I would like to thank my friends Ilam Anna, Senthil, Akshaya and Tharanee for their continued support and cooperation. I am also thankful to Mithun for helping me analyze the results of the thesis related to basketball. Finally, I would like to thank my family for their continued support and cooperation without whom this thesis wouldn't have been possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Motivation

In the current era of information explosion, the users are often challenged by two obstacles that hinder their ability to quickly perceive data. The first challenge is the "problem of plenty" where the users are overloaded with information, part of which is inferior compared to the rest. The second challenge is that the data can be often too general and it does not necessarily describe concepts that best represents users' needs. In recent times, The *Skyline Operator* introduced by Börzsönyi *et al.* (2001) has emerged as an important summarization technique for multi-dimensional high cardinal data sets so much so, it has sparked a growing interest in the area of efficient skyline query processing among the database research community[1] indicated by works of Kossmann *et al.* (2002); Papadias *et al.* (2003); Wu *et al.* (2006). Multi-criteria optimization described in Ehrgott (2005) describes solution to a decision problem as the ability to distinguish between good and bad objects among a set of objects on the assumption of existence of certain criteria, according to which the quality of the objects are ascertained. Considering each input datum as a tuple in a database, the skyline can be defined as a technique to filter out a subset of interesting input tuples from a potentially large set of data. In particular, the result of a skyline query consists of those input tuples for which there is no input tuple having better or equal values in all the attributes and a better value in at least one attribute. For example, in order to attend a conference at a foreign country, a traveler might be interested in the hotels

[1]The work of Börzsönyi *et al.* (2001) has received more than 1500 citations in Google scholar.

that are cheapest and closest to the conference venue. It might so happen that there could be few cheap hotels very near to the venue making the decision making easier or the venue could be a downtown city where in the hotels are considerable more expensive. In the latter case, as the two goals are contradictory, making up a decision can be tough specially when there are too many options. In such cases, the skyline computation can help us out by filtering only those hotels that are best trade-offs between the two attributes, price and distance. Skyline query processing eases the problem of decision making thus helping us overcome the first challenge.

However in few cases the data themselves do not best describe the users' requirements and the second challenge remains unsolved. This challenge manifests in two ways and both the problems are described in this section. Consider an example where an NBA team manager wants to recruit a new player to play in Point Guard (PG) position. There are different attributes to players like Points per Game (PPG), Assists per game (APG), Rebounds per game (RPG) and blocks per game (BPG) amongst others to gauge them. However, there are certain attributes which are more important and characterizes a point guards' game more than others. Identifying that characteristic might not be straightforward and even if identified the skyline on the NBA player statistics might not help the manager in deciding the point guard to recruit. In this case the data do not describe the users' preferences and makes the decision making more challenging. The problem is due to the presence of a gap between the existing data and the data that describe the users' needs referred to as the *semantic gap*. There are numerous other fields where semantic gap exists. Consider a retail store selling laptops to its customers. Different customers have different needs and different laptops have different purposes. The skyline computation on the entire data set might always present choices which are irrelevant to a particular user. Another area where this problem persist is in Information Retrieval – Consider the following setup, where there are text documents in a TF-IDF space [Salton and McGill (1986)]. TF-IDF is a vector space model where text documents (in general any objects) can be represented as vectors in an algebraic space. Different users can also be represented in the same TF-IDF space using keywords that are relevant to the user. Keywords interesting to the user can be obtained from the articles user had authored or liked. The problem of finding the preference aware skyline of interesting documents to a particular user is yet to be solved.

The second problem is to find the skyline of interesting documents to a group of users. Recommendations to a group of people have also been recognized as an important problem and have gained attention recently [Jameson and Smyth (2007)]. Different users can have different interests and there is a new problem where it is needed to find the skyline of interesting documents not just to a particular user but to the entire group. Another example of this problem could the case where the same NBA manager is now trying to recruit a player who can play in both Point Guard (PG) and shooting guard (SG) position. The manager does not need the skyline of just point guard players, but he needs skyline of players who can play in both point guard and shooting guard positions and this is a much harder problem to solve. The solution proposed to solve this problem can in general be extended to compute skylines in latent spaces, where the characteristics (preferences) are not independent, but to preserve the semantics, the transformation has to treat those preferences as independent concepts. More about latent skyline processing is discussed in 1.2. Motivated by the above observations, this thesis proposes *Preferential and Latent skyline Query Processing techniques* to enable users to quickly and effectively obtain information. In the literature of skyline query processing, the closest to our work is Dynamic Skylines [Papadias *et al.* (2003); Sharifzadeh and Shahabi (2006); Deng *et al.* (2007)] where the query is not executed in the original data space but in a dynamic space that is query independent, however this does not take into account neither the preferences for the user nor the group.

1.2 Latent Skylines

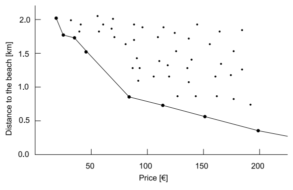
Section 1.1 describes that the solution to a decision problem as the ability to distinguish between good and bad objects among a set of objects, on the assumption of existence of certain criteria, according to which the quality of the objects is ascertained. Section 1.1 also introduced semantic gap and the need to transform points from given data space to a preferential space. When preferential skylines are computed for a single user, the preferences are obtained as an alternative orthonormal space using Singular Value Decomposition. This has many computational advantages as the data can be moved from the given domain space to the user preferential space by rotation, which is a simple linear transformation. However, there are instances where the preferences (or characteristics) are not orthogonal to each other and yet these characteristics are the criteria according to which the quality of the objects should be measured. The preferences or characteristics can be visualized as vectors in the given space and there is a need to evaluate data points not on the basis of existing dimensions (attributes) but based on these characteristic vectors.

The straightforward way to solve this problem is to project the data points on to the characteristic vectors. Simple dot product semantics can be used to obtain the membership of each data point with the characteristic vectors. This would again lead to a simple linear transformation, however there is a significant downside to this approach. Recall that each of these characteristic vectors represent semantics that are significant to the user. When the points are projected on to the characteristic vectors, it so happens that, since the characteristic vectors aren't independent of each other (possibly correlated), a data point that lie perfectly on a characteristic vector might have a membership on the other characteristic vector. This would mean that the characteristic vectors can never truly be differentiated independently of each other. Consider another scenario, where a NBA manager is trying to recruit a player who can play both as a Point Guard (PG) and a Shooting Guard (SG). These are different positions and requires different skill and techniques, however it is fairly common in basketball for a player to play in two positions. The latent skill required to excel in a particular position (PG/SG) can be obtained through Singular Value Decomposition on statistics of players who play the best in those positions. The latent semantic obtained from SVD will give the characteristics of players who play in a position. One can use intuition and predict that there would be correlation between the characteristic vectors of these two positions. Hence any player who is perfectly aligned with the characteristic vector will have membership scores on other vector too. This would semantically mean, that these are the characteristics (definitions) of PG and SG and though a player is aligned with the definition of a particular position (say PG), on transformation, the new space would indicate him as a SG player as well. Projections does not retain the semantics of the space and hence a transformation that retains the semantics of the space is needed. A point perfectly aligned with a characteristic vector should hold membership only on to that characteristic vector upon transformation. This motivates to propose new methods of transformation and this research introduces a novel transformation technique called *Stretching*.

Stretching tries to pull apart the space between the characteristic vectors so that the characteristic vectors become mutually independent of each other while preserving the semantics of the space. On completion of stretching, the skyline is computed in the stretched space and is referred as the latent skyline. The latent skyline suggests the set of possibilities to consider given the latent preferences.

1.3 Related Work

The skyline of a dataset is defined as those points which are not dominated by any other point in the data set. A point dominates another point in the dataset if it is as good or better in all attributes and better in at least one attribute. For example, consider a dataset of hotels having two attributes: price and distance. It is quite natural to prefer hotels that are cheaper and closer to our location. Thus hotels that are cheaper and closer will dominate our selection over costlier hotels which are far off. Figure 1.1a adopted from Börzsönyi *et al.* (2001) shows the skyline of cheap hotels near the beach in Nassau, Bahamas



(a) Skyline of hotels in Nassau, Bahamas

(b) A view of night sky in Manhattan

Figure 1.1: Representation Of Skylines

The task of finding the skyline of a set of data points where first attempted by Kung *et al.* (1975) in 1975 under the name of maximum vector problem. Kung et al proposed a solution for maximum vector problem based on Divide and Conquer algorithm. Another solution was proposed by Stojmenovic and Miyakawa (1988) involving parallel computation. However, these algorithms were applicable only in situations where the data set fits into the main memory.

Börzsönyi *et al.* (2001) was the first to investigate maximal vector problem in context of databases and coined the term skyline. The term skyline is uses because of its graphical representation. He suggests the skyline of Manhattan can be computed as the set of buildings which are high and close to river Hudson. In other words every building seen in Figure 1.1b is a part of skyline of Manhattan. Chomicki *et al.* (2013) suggested a formal model for expressing Pareto dominance.

Let $\mathcal{A} = \{A_1...A_d\}$ be a finite set of attributes (a relation schema) where d is the number of dimensions. Every attribute $A_i \epsilon \mathcal{A}$ is associated with an infinite domain $\mathcal{D}_{A_i}$. We work with the universe of tuples $\mathcal{U} = \prod_{A_i \epsilon \mathcal{A}} \mathcal{D}_{A_i}$ Given a tuple $t \epsilon \mathcal{U}$, the value of its attribute A_i is denoted by $t[A_i]$. The Pareto dominance or just dominance relation $\succ$ of tuple t over tuple s is defined as

$$t \succ s \equiv \bigvee_{A_i \epsilon \mathcal{A}} t[A_i] >_{A_i} s[A_i] \wedge \bigwedge_{A_i \epsilon \mathcal{A}} t[A_i] \geq_{A_i} s[A_i] \tag{1.1}$$

Since the introduction of skyline operator, numerous algorithms has been suggested for efficient computation of skyline queries. These algorithms can be classified into two categories depending on whether the data is preprocessed or not. The first category discussed in section 1.3.1 is called *sequential algorithms* because they have to scan all of the input to compute skyline. The second category discussed in section 1.3.2 called *Indexing algorithms* includes all index based methods which uses an indexing mechanism to prune points which are not guaranteed to be skylines.

1.3.1 Sequential Algorithms

Börzsönyi *et al.* (2001) proposed two algorithms for processing skyline queries in large databases. *Divide-and-conquer (D&C)* and *Block Nested Loop (BNL).* D&C approach divides the dataset into several partitions so that each partition fits into main memory. The partial skyline of each partition is obtained using any skyline algorithm and the final complete skyline is obtained by merging all the partial skylines. BNL algorithm scans the entire data set and keeps a list of candidate points in the main memory. The list initially contains only the first data point and for every subsequent data point p, there are three cases:

1. If p is dominated by any point in the list, it is discarded and is not part of a skyline.

2. If p dominates any point in the list, it is inserted, and all of the points dominated by p are dropped.

3. If p is neither dominated nor dominates any point in the list, it is inserted into the list

Chomicki *et al.* (2003) proposed a variation of BNL algorithm called *Sort First Skyline* (SFS). SFS first sorts the dataset based on a monotone function. Candidates are inserted into the list in the order of their scores and sorting ensures that a point p dominating p' is encountered before p', hence unlike BNL any point in the list of candidate points is a skyline. Godfrey *et al.* (2005) proposed *LESS* (Linear Elimination Sort for Skyline) algorithm which is a combination of SFS and BNL. The external sort routine used by SFS is integrated into LESS and the sort routine has two major changes.

1. It uses an elimination filter window in pass zero of the external sort routine to eliminate records quickly.

2. It combines the final pass of the external sort with the skyline-filter pass.

The Elimination Filter window of LESS is similar to candidates list in BNL and the skyline-filter pass used in the last pass of the external sort filters skyline like SFS.

Typically LESS saves a pass by combining the last merge pass of the external sort with the Skyline-filter pass.

Bartolini *et al.* (2008) proposed a new sequential scanning algorithm called *SaLSa* (Sort and Limit Skyline algorithm). Salsa is based on the observation, that for a suitably chosen sorting functions, it is possible to compute skyline without looking at all of the data.

1.3.2 Index Based Skyline Algorithms

Tan *et al.* (2001) proposed a technique which uses bitmaps to decide if a point is in skyline. A data point $\mathcal{P} = (p_1, p_2 ... p_d)$, where d is the number of dimensions, is mapped to a m bit vector, where m is the total number of distinct values over all dimensions. The skylines of a data set are computed by just using bit wise operations like 'AND' and 'OR'. For more information on the bitmap based algorithm please refer to Tan *et al.* (2001). In the very same paper, Tan *et al.* (2001) proposed another technique called *index* to compute skylines. It employs an index mechanism where a d dimensional space is mapped into d single dimensional lists where each list is implemented as a B+ tree. The idea behind this algorithm is, if the minimum value among all dimensions in a tuple is larger than the maximum value among all dimensions in another tuple, then the first tuple dominates the second tuple. Since the structure is sorted based on the maximum value, there is no need to examine all the records to filter the skylines thus saving on I/O.

Kossmann *et al.* (2002) proposed an algorithm called *Nearest Neighbors* (NN) that can compute skylines in a progressive manner. NN uses R-Trees to split the domain spaces to regions recursively and compute the NN in certain regions while pruning the remaining. The algorithm can ignore regions which are not guaranteed to contain any skylines. Each of the NN is a part of the skyline and thus the algorithm progressively computes the entire skyline by finding NN on all the regions that can't be pruned.

Branch and Bound Skyline (BBS) is an algorithm proposed by Papadias *et al.* (2005) that improves on NN algorithm. Some of the regions investigated by NN are redundant meaning they return the skyline points found by previous queries. BBS alleviates this problem and is I/O optimal (i.e.) it performs a single access only to those nodes that may contain skyline points. In this paper, Papadias *et al.* (2005) also suggests different types of skyline queries

1. *Constrained Skyline*: A set of constraints is additionally specified and the algorithm returns only those skyline points that satisfies the constraints.

2. *Ranked Skyline*: A monotone preference function f and a parameter k is additionally specified and the algorithm returns the top k skyline points which maximizes or minimizes the function f.

3. *Dynamic Skyline*: Dynamic Skyline queries specifies m dimensions functions, where each function $f_1, f_2, ...f_m$ takes as parameters coordinates of data points along a subset of d axis, where d is the dimensionality of the data and $m \leq d$. The goal is to find the skyline in the new data space with dimensions defined by $f_1, f_2, ...f_m$.

4. *Group-By Skyline*: This is similar to the group-by clause in traditional RDBMS. Instead of aggregation over group by clause, the goal is to find the skyline for each value in the group by clause.

1.3.3 Other Skyline Algorithms

In addition to the problem of computing skyline over a dataset, the database research community have focused on other interesting problems involving skyline queries.

1. *Subspace Skyline Queries:* Tao *et al.* (2006) investigated the problem of finding skylines in various sub-spaces and proposed a new technique called SUBSKY. SUBSKY converts each d-dimensional point to a 1-D value and indexes it with a B-Tree. The converted value represents the best value of the point among all attributes. The algorithm then proceeds to scan all the points in the sorted order and finds skylines in the subspaces while maintaining the worst value of the point among all attributes, which will eventually increase and when it is greater than the best value previously indexed, the algorithm can terminate and report the skylines. Pei *et al.* (2005) proposed an algorithm called *skyey* that takes a set of objects S in space $\mathcal{D}$ as input, and returns the set of objects that are part of a subspace skyline for each non empty subspace. The algorithm finds the decisive sub-spaces (attributes) in which objects are good. Yuan *et al.* (2005) proposed an algorithm called *Skycube* which precomputes the results of all possible sub space skyline queries for a given dataset in an efficient manner.

2. *Skyline On Distributed Databases:* Given the growing popularity of decentralized databases, the problem of finding skylines over distributed data sources had also been studied. Balke *et al.* (2004) was one of the seminal works in this domain and it assumed the data to be vertically partitioned. The future works like Rocha-Junior *et al.* (2009) also dealt with the case of horizontal data partitioning.

3. *Skyline On Continuous Data Streams:* Lin *et al.* (2005) considered the problem of efficiently computing the skyline against the most recent N elements in a stream of data. (i.e.) computing the skyline for the most recent $n(\forall n \leq N)$ elements.

4. *Skyline On Incomplete Data Sets:* Lofi *et al.* (2013) focused on the problem of computing skylines over incomplete data. This paper presents a hybrid approach combining dynamic crowd sourcing with heuristic prediction techniques to reduce the error in skyline result while also reducing the time taken for computation of result. They propose a heuristic approach for predicting missing data and also a prediction risk along with it. The prediction risk is a measure of error, the prediction can bring to the skyline result. By resorting to crowd sourcing only for data with high prediction risk, they minimize the error and as well as the time taken for the result.

Chapter 2

BACKGROUND AND PRELIMINARIES

2.1 Latent Semantic Indexing

Singular Value Decomposition (SVD) is a well-known matrix factorization technique proposed by Deerwester *et al.* (1990). SVD can be used in scenarios where there is an underlying latent semantic structure in the data that is partially obscured by some noise. SVD attempts to use a statistical technique called *Eigen decomposition* to recover the latent structure. SVD factors a $m \times n$ matrix R into three matrices as following:

$$R = U.S.V^T \tag{2.1}$$

Where, U and V are two orthogonal matrices of size $m \times r$ and $n \times r$ respectively. The column vectors of U, also called the left singular matrix are the eigenvectors of the $m \times m$ square matrix $R.R^T$. The column vectors of V, also called the right singular matrix are the eigenvectors of the $n \times n$ square matrix $R^T.R$. r is the rank of matrix R. S is a diagonal matrix of size $r \times r$ having all singular values of matrix R as its diagonal entries. These singular values are the square roots of the eigenvalues of $R.R^T$ and $R^T.R$. All entries of matrix S are positive and stored in decreasing order of magnitude. It is possible to reduce the $r \times r$ matrix S to have only k largest diagonal values to obtain a matrix S_k, $k < r$. If the matrices U and V are reduced accordingly, then the reconstructed matrix $R_k = U_k.S_k.V_k^T$ is the closest rank-k matrix to R. The U_k and V_k matrices has the following properties. The r column vectors of U form a r dimensional basis in which the m rows of the matrix can be described. Similarly, the r column vectors of V form a r dimensional basis in which the n columns of the matrix can be described. These column vectors reveal associative patters in the data. In fact, the values of the diagonal matrix S, represent the strength of these associations or correlation. The patterns which have significantly higher eigenvalues do represent the latent patterns in the data, while patterns having smaller values are often noise. By ignoring the smaller singular values, SVD escapes the unreliability, ambiguity and redundancy of underlying data.

Deerwester *et al.* (1990) initially proposed SVD for automatic indexing and retrieval of text documents. It was introduced to tackle the problem of *synonymy* and *polysemy* in the domain of information retrieval. Synonymy refers to the condition when multiple words describe the same fact. Polysemy refers to the condition, when a single word can have multiple meanings. In this research, SVD is used to identify the preferences of the user. The problems involved in finding the preferences of a user is analogous to the condition suggested by synonymy. In real world, the reason for user liking a set of products can be modeled as a fact. Sarwar *et al.* (2000) says, a customer who likes recycled letter pad and recycled memo pad indicates the latent feature that he likes recycled office products.

The impact of latent semantic indexing on Recommendation Systems had been studied by Cacheda *et al.* (2011). They compared the prediction accuracy of different model based approaches and memory based approaches and concluded that SVD based methods provided the best results.

2.2 Problem Statement

Consider a d dimensional data space $\mathcal{D}$ and a data set $\mathcal{P}$ on $\mathcal{D}$ with cardinality S. A point P given by $\{p_1, p_2 ... p_d\}$ where p_i, $1 \leq i \leq d$ is the value of P along dimension i. Given the above, the goal of this thesis is twofold.

1. To find the personalized skyline of a user $\mathcal{U}$, given a set of objects of his interest, $\mathcal{P}^{int}$ in the same domain space $\mathcal{D}$.

2. To find the latent skyline, when there are k preferences or latent characteristics. The latent characteristics are specified as vectors given by $\{V_1^{pref}, V_2^{pref}, ..., V_k^{pref}\}$, where each $V_i^{pref}, 1 \leq i \leq k$ indicates a distinct preference or latent characteristic. These V_i^{pref} characteristics are not mutually independent in the domain space $\mathcal{D}$, but semantically each vector suggests a characteristic or a concept that is to be treated independent of the others.

This section begins with definition of dominance relationships and skyline. Later on user preferences, preferential space and preference aware skylines are explained.

2.2.1 Dominance Relations

A point $P \epsilon \mathcal{P}$ is said to dominate another point $Q \epsilon \mathcal{P}$, denoted as $P \succ Q$, if

1. On every dimension, $p_i \geq q_i$, where $1 \leq i \leq d$.

2. On at least one dimension, $p_i > q_i$, where $1 \leq i \leq d$

Without loss of generality, it is assumed that for all dimensions maximum values are preferable.

2.2.2 Definition Of Skyline

The skyline of a data set $\mathcal{P}$ are the set of points $\mathcal{S}$ where $\mathcal{S} \subseteq \mathcal{P}$, such that no point in the set $\mathcal{S}$ is dominated. The set of points in $\mathcal{S}$ are called the Skyline of the dataset $\mathcal{P}$.

2.2.3 Incomparable Points

Consider points $P, Q \epsilon \mathcal{P}$ where neither point dominates the other. Such points are called as incomparable points and are denoted as $P \sim Q$. All the points that makes up a skyline are incomparable points.

As pointed out already in 2.1, SVD is used on items that are of interest to a user, to infer his preferences. The singular vectors given by SVD form an orthonormal preferential space of a user and can be formally defined as follows.

2.2.4 Preferential Space

The preferential space of a user $\mathcal{U}$, denoted by $\mathcal{D}'$ is an orthonormal space of dimensionality k, $k \leq d$. In the experiments conducted as a part of this work, the k eigenvectors are derived from SVD computed over a set $\mathcal{P}^{int}$, which are objects of interest to the user $\mathcal{U}$. The dimensionality of the preferential space, k can be decided based on the eigenvalues from the singular matrix of SVD. It is to be noted that $\mathcal{P}^{int}$ should be present in the same domain $\mathcal{D}$ as $\mathcal{P}$.

Once the preferential space $\mathcal{D}'$ of the user is obtained, the data $\mathcal{P}$ from the existing data space $\mathcal{D}$ is transformed to the user preferential space $\mathcal{D}'$ by just projecting the points $p \epsilon \mathcal{P}$ on to it. Now we are ready to formally define preferential skylines.

2.2.5 Preferential Skylines

The preferential skylines of a user $\mathcal{U}$ for a given data space $\mathcal{D}$ represented by $PS(\mathcal{P}, \mathcal{U})$ can be defined as the skyline of the dataset $\mathcal{P}'$, where $\mathcal{P}'$ is obtained from transforming $\mathcal{P}$ from the domain space $\mathcal{D}$ to the user preferential space $\mathcal{D}'$.

In the definition of latent skylines, the latent characteristics are specified as vectors and not as a preferential space. These vectors, called as characteristic vectors, can be considered as preferential spaces of dimensionality 1. In context of preferential spaces, the preferential space can be of any dimension, k, $k \leq d$. Each of these k dimensions of a preferential space can be considered as a distinct characteristic vector.

2.2.6 Characteristic Vectors

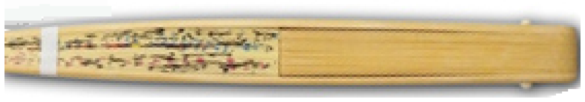
Characteristic vectors, V^{pref} are preferential spaces, $\mathcal{D}'$ of dimensionality 1.

In the experiments conducted as part of this work, the Characteristic vectors are the first eigenvector from the left or right singular matrix of SVD. It is to be noted that the singular vectors of SVD are always unit vectors. The latent skylines involve the problem of finding skylines in an alternate space where its dimensions are given by the characteristic vector. However, to accomplish this in a useful manner, these characteristic vectors must be mutually independent of each other in the alternate space after transformation. This research introduces a novel concept called *stretching* to achieve it. Stretching is introduced in section 2.2.7 and is discussed in detail in section 4.3.4.

2.2.7 Stretching

Given d characteristic vectors, stretching aims to align these d characteristic vectors with d axis, while stretching the space in between them. This nonlinear transformation makes the d characteristic vectors independent of each other.

In linear algebra, a transformation is an orthogonal transformation when the transformation preserves the lengths of vector and the angles between them. Stretching on the other hand preserves the lengths of vector and the semantic relationships of the vector with the characteristic vector while making the space orthonormal. Stretching takes inspiration from the folding fans of the Japanese shown in the Figure 2.1[1]. The ends can be folded on to a single line or stretched 180° between them. The two ends of the fan have one edge hinged on each other and the other edge is free to move about the hinge. Now consider a two dimensional space, with two characteristic vectors V_1^{pref} and V_2^{pref}. Consider these characteristic vectors to be the ends of the fan hinged at origin. Suppose if the ends of the fan are pulled apart until they are aligned with the coordinate axis; this would transform the space in between the characteristic vectors while making them independent.



(a) Folded Japanese fan

(b) Opened Japanese fan

Figure 2.1: Japanese Fans And Stretching

[1]These images were obtained from website http://www.nintendolife.com/news/2010/10/north_america_cools_down_with_club_nintendo_folding_fans

Chapter 3

PREFERENTIAL SKYLINE PROCESSING

3.1 Introduction

As described in 2.2.5, Preferential Skylines are obtained by initially transforming all data to a new orthonormal space called preferential space and finding skylines in this preferential space. The naive way to accomplish this would be to transform all points from the domain space to the preferential space and to find skylines in this new space using any existing skyline algorithm. This thesis, however tries to answer the question, "Is this point dominated by another point in the preferential space?" without transforming the point. If such points are identified, they can be ignored for rest of the computation. This is the idea behind the quad tree based index algorithm that efficiently process preference aware skyline queries.

We begin by stating few assumptions.

- Given a real valued space of d dimensions and for each dimension (attribute) the user can either prefer the maximum or minimum of it.

Figure 3.1 shows an example with six points $\{a, b, c, d, e, f\}$ in a two dimensional space given by axes X and Y. The preferential space for the user is given by X' and Y' in which the six points are transformed as $\{a', b', c', d', e', f'\}$. For ease of explanation, an assumption is made that in the original space, tuples with higher values are considered better along all dimensions. The solid gray and dashed gray lines denote the region of dominance of the points in the original and transformed space respectively. The aim is to find the skyline in the preferential space. Observing the transformation, following observations can be made

1. A point a dominating another point b in original space, may or may not continue the dominance or be dominated in the preferential space. Proof is presented in 3.4.1.

2. A point a incomparable (neither dominates the other) with another point b, may remain incomparable or either one can dominate the other. Proof is presented in 3.4.2

From the above observations, the following inferences can be made about the skylines in the preferential space.

1. Skylines in original space can be dominated in the preferential space, which means they are no longer part of the preferential skyline. As described in Figure 3.1, d which was initially a skyline in the given space, is no longer a skyline in the preferential space.

2. Points which were dominated in the original space, may no longer be dominated in the preferential space and can become skylines. As described in Figure 3.1, a was dominated by c in the given space but becomes a skyline in the preferential space.

In this research, the preferences of the users are captured as eigenvalues and eigenvectors. The semantics of eigenvalues and eigenvectors in context of user preferences is described below.

1. The number of preferences for the user is same as the number of eigenvectors (or eigenvalues).

2. The strength of the user's preference is given by the eigenvalue.

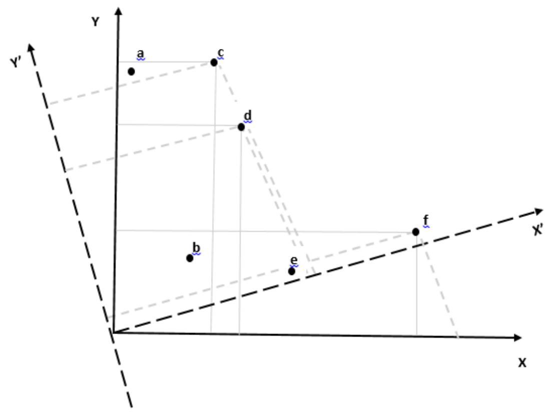


Figure 3.1: An Example For Preferential Skyline Processing

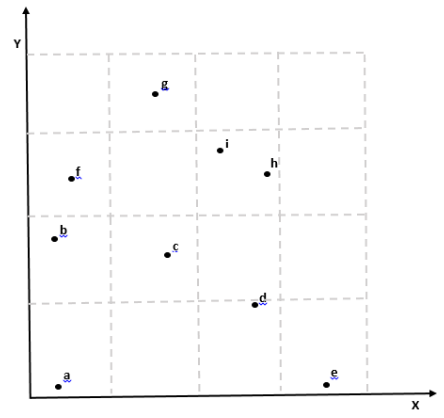
3. Each eigenvector is a linear combination of all the existing attributes (or dimensions). If the coefficient corresponding to an attribute is less than zero, it means the user has dislike towards that attribute and positive coefficient denotes liking towards the attribute.

4. The absolute value of the coefficient in the eigenvector denotes the strength of like or dislike towards that attribute.

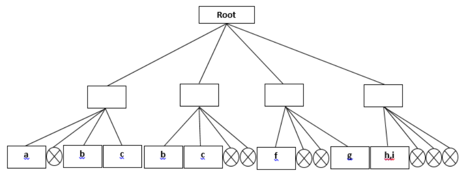
Based on above inferences, it is evident that it is desirable to find the max-skyline in the new space as each dimension in the preferential space indicates the preference of the user. In order to not transform all the points from the given space to the preferential space, there is a need to isolate those points that would be dominated in the preferential space. To achieve this, the points in the given space are indexed using a Quadtree.

3.2 Quadtree

Quadtree is a hierarchical data structure that provides a grid like sub-division of the space while adapting to the distribution of the data. They are based on the principle of recursive decomposition of space to form nodes that contain the points. There are many variants of quadtree based on the principle guiding the decomposition of space. In our research, the Point-Region quadtree is used, where the space is always split at the center of partitions and data are stored in the leaves. The leaves of the tree can be thought of as a bucket as it can store more than one data object. A two dimensional space with points and its corresponding point-region tree is shown in Figure 3.2a and 3.2b respectively.



(a) Sample Two dimensional space



(b) Quadtree for space given in figure 3.2a

Figure 3.2: Quadtree - A $\mathfrak{R}^2$ Example

Consider a d dimensional real valued space, where each dimension can take on a value between 0 and 1. Then (0.5, 0.5 ... 0.5) becomes the point which is at the center of all dimensions. This becomes the root of the tree as well and divides the space into 2^d regions or quadrants. These quadrants, called nodes become the children of the root and represent all the data points within the entire quadrant. These quadrants can be further split into 2^d sub quadrants and so on, hence this technique is a recursive decomposition of space. Finally the leaf nodes of the tree contain the data points in that region. In the implementation used in this research, the tree is made to grow to a specified predetermined depth.

The algorithm for insertion is specified in 1. The following quadtree methods are used for insertion.

1. *function getQuadrant(P,N)* gives the node of the quadtree representing the quadrant in which P is located with respect to the quadtree node N.
2. *add(P, N)* adds the point P to the leaf node N.

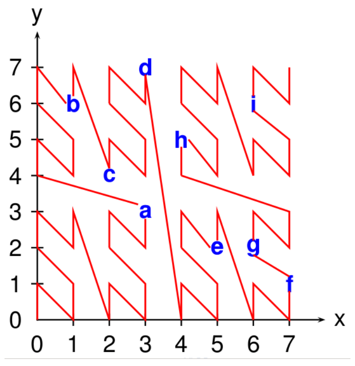
Algorithm 1 Inserting a point in the Quadtree

```
Require: $P$ is a point in a $d$ dimensional space of the form $\{p_1, p_2 ... p_d\}$ where $p_i$ is
  the value of the point $P$ along dimension $i$.
Require: $R$ is the root node of the tree.
  function INSERT($P$)
      Node $N \leftarrow$ GETQUADRANT($P$, $R$)
      while $N$ is not a leaf do
          Node $N \leftarrow$ GETQUADRANT($P$, $N$)
      end while
      ADD($P$, $N$)
  end function
```

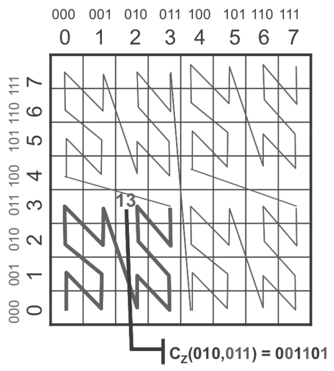
3.2.1 Quadtrees And Z-order

Lee *et al.* (2007) was the first to observe the unique properties of the Z-order curve that matches perfectly with the processing strategies for skyline query processing. Z-order is a fractal[1] that offers a method to map a multidimensional space to one dimensional space. Fractals are recursive in nature and Z-order covers the entire space through repeated application of the pattern *Z*. A Z-order curve on a two dimensional space is shown as a red line in figure 3.3a.

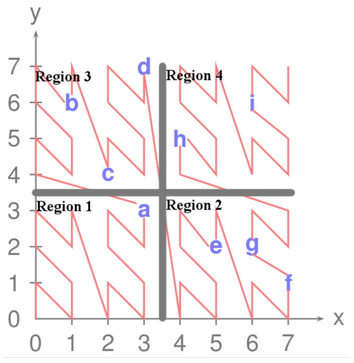
[1]A fractal is composed of similar structures at multiple scales



(a) Z Order Curve



(b) Bit Shuffling



(c) Regions Of Z Order Curve

Figure 3.3: Z Order Curve

The properties of Z-order curves that makes them attractive for skyline processing are

1. *Bit Shuffling:* The bit shuffling or bit interleaving is an extremely efficient implementation to determine the Z-order value of a point. In a d dimensional integer-valued vector space, the z-order value of a point can be determined by interleaving the bits of all the coordinates of that point. This is best illustrated using an example which is visualized in Figure 3.3b[2]. Consider a point $(2, 3)$ in a two-dimensional space. Its Z-order value denoted by $C_z(2, 3)$ is 001101_2 $(= 13_{10})$ and is obtained by shuffling the bits of the inputs, 010_2 $(= 2_{10})$ and 011_2 $(= 3_{10})$.

2. *Monotonic Ordering:* Section 1.3.1 suggested that if a dataset is ordered based on a monotone function, it ensures that a point p dominating p', is encountered before p'. This ordering has several advantages when computing skylines and many algorithms are based on monotonic ordering of data points. Data sorted

[2]This image is adopted from Candan and Sapino (2010)

based on their Z-order values has a monotonic ordering. For example, in the Figure 3.3a[3], h which dominates a and c is encountered before a and c.

3. *Prunable Regions:* The algorithms discussed in 1.3.2 were based on the principle, that by using an index structure, points that are not guaranteed to be skylines can be ignored and this speeds up the computation of skyline. The Z-order curve offers a similar advantage where regions which are not guaranteed to be skylines can be pruned. Consider the Figure 3.3c, any point in "region 4" will dominate all the points in "region 1". Thus if presence of a data point in "region 4" can be ascertained then "region 1" can be eliminated from rest of the computation.

The result of Z-ordering can be described as the order, one would get from a depth first traversal of a quadtree. Gargantini (1982) was the first to takes advantage of Z-ordering to build a quadtree and he proposed the linear quadtree. In our work, all the leaf nodes of the quadtree is assigned a Z-order based on its lowest left vertex and the leaf nodes are sorted by its morton order.

The following observations can be drawn about a quadtree of dimension d and depth t.

3.2.1.1. A d dimensional quadtree of depth t can have up to 2^{dt} leaf nodes.

3.2.1.2. The quadtree can be considered as a d dimensional lattice made up of layers of $d-1$ dimensional sub-lattices along any particular dimension and there are 2^t such sub-lattices along that dimension. Each $d-1$ dimensional sub-lattice is made of $2^{(d-1)\times t}$ leaf nodes.

3.2.1.3. All the Z order values of the leaf nodes in a given sub-lattice along a dimension (say d_1) will have a common pattern. In a d dimensional space, t bits from d

[3]This image is adopted from Liu and Chan (2010)

attributes are interleaved to obtain z ordering which spans $d \times t$ bits as indicated by the possible 2^{dt} leaf nodes. For a given layer among 2^t possible layers, the z value of all leaf nodes in it is computed by interleaving the layer number as the value of the dimension d_1. In other words the t bits of the dimensions d_1 used for bit interleaving would equal the layer number.

3.2.1.4. The layer obtained from interleaving bit 0 represents the lowest $d-1$ dimensional layer along a dimension. Similarly layer obtained from interleaving bit $2^t - 1$ represents the highest $d - 1$ dimensional layer along that dimension.

Given the properties of quadtree, it could be advantageous to identify the leaf nodes that can be pruned. A leaf node can be pruned on the claim that all points in the leaf node is dominated by some point in the preferential space. In a d dimensional space, a leaf node of a quadtree has the following properties.

1. All nodes of the quadtree can be considered as a hypercube with 2^d vertices.

2. For each of the d dimensions, all the vertices can have only two possible values for that dimension. Let those value be $a, a + k$ with $a + k > a$. There are $2^{d/2}$ vertices which has value a and the remaining $2^{d/2}$ vertices has value $a + k$ for that dimension.

3. Each vertex of the leaf node has a bitmap representation, called *vertex signature* obtained by replacing the lower value, a with bit 0 and higher value, $a + k$ with bit 1. The vertex signature would be a number between 0 and $2^d - 1$.

4. Since each vertex of the 2^d vertexes of leaf node are unique, each vertex has a unique vertex signature and takes a value between 0 and $2^d - 1$.

5. There is a vertex, referred to as *Lower Vertex*, which is dominated by all the points in the leaf node in the original space $\mathcal{D}$. The vertex signature of lower vertex would be d-bit zeroes.

6. There is another vertex, referred to as *Upper Vertex* which dominates all the points in the leaf node in the original space $\mathcal{D}$. The vertex signature of upper vertex would be d-bit ones.

The above knowledge on lower vertex and upper vertex does not help in determining if a leaf node is dominated in the preferential space. Hence a technique to find the lower vertex and upper vertex of a leaf node in preferential space is proposed in section 3.2.2

3.2.2 Preferential Lower And Upper Vertex

It is already established that, points are transformed from the given space $\mathcal{D}$ to the preferential space $\mathcal{D}'$ by projecting them on to the preferential space obtained from SVD. In a d dimensional space, the eigenvectors of SVD have d coefficients which can be either positive or negative. Each eigenvector has a bitmap representation referred as *vector signature* obtained by replacing all the positive coefficients with bit 1 and replacing all the negative coefficients with 0. Given the above, the relationship between the vertex signature and vector signature is as follows.

1. For a given leaf node, since each of the 2^d vertices signatures are unique, the vertex which has the highest projection on a eigenvector is the vertex which has the same signature as that of the eigenvector.

2. Similarly, for a given leaf node, the vertex which has the lowest projection on a eigenvector is the vertex, whose signature is the ones' complement form of the eigenvector's signature

The vertex whose signature matches the eigenvector is projected on that vector. The resulting value is the maximum membership that any point in that leaf node can have along the dimension specified by the eigenvector. Similarly, for each eigenvector, the vertex which has the same signature as that of the eigenvector is picked and projected on to that vector. Combining all the projection scores for the d eigenvectors gives the *upper vertex* of the leaf node in the preferential space. Similarly projecting the d vertices whose signature is the ones' complement of the eigenvector, the *lower vertex* of the leaf node in the preferential space is obtained. The computation of the upper and lower vertices of a leaf node in the preferential space simplifies pruning as follows.

1. If a point P' dominates the upper vertex of a leaf node in the preferential space, then P' dominates all the points in the leaf node.

2. If a point P' doesn't dominate the lower vertex of a leaf node in the preferential space, then P' doesn't dominate any point in the leaf node

3. If a point P' doesn't dominate the upper vertex but dominates the lower vertex in the preferential space, then P' dominates few points in the leaf node but not all.

A figure illustrating the preferential lower and upper vertex is given in Figure 3.4. In the Figure 3.4, $\{v1, v2, v3, v4\}$ are the vertexes of a node of a quadtree in the original space given by X and Y axis. The transformed space is indicated by axes X' and Y'. It can be inferred that the signatures of $v4$ and Y' is same. Similarly the signatures of $v3$ and X' is same. Hence the preferential upper vertex indicated by $P1$ is obtained from nodes $v3$ and $v4$. The algorithm for getting lower vertex and upper vertex in preferential space denoted as *preferential lower vertex* and *preferential upper vertex* is given in *Algorithm* 2.

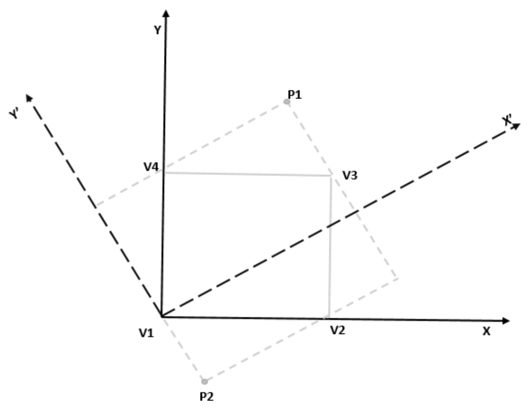


Figure 3.4: Preferential Upper And Lower Vertex

Once the preferential upper and lower vertex of the leaf nodes is established in the preferential space, it is possible to prune away those nodes which are dominated by other nodes in the quadtree. If the upper preferential vertex of a leaf node (say N_1) is dominated by the lower preferential vertex of an another node (say N_2), then any point in node N_2 will dominate all the points in node N_1.

There is however one caveat with the above approach for pruning the nodes of the quadtree. Remember the vertices of a quadtree can have only two possible values for each dimension and are assumed as a and $a + k$, where $k > 0$. This implies the each side of the leaf node is k units. A cube can be constructed knowing the length of the side of the leaf node. As each side of quadtree is k units, in a d dimensional space, the volume of the leaf node would be k^d. When upper preferential vertex and lower preferential vertex is computed, a hypercube is formed in the preferential space (for the same node) as well but the volume of the hypercube formed in the preferential space is many times larger than the volume of the hypercube formed in the original space. Consider two vertices v_1 and v_2 such that the vertex signature of v_1 is ones' complement of v_2. The vertex signature of v_1 is same as the vector signature of some eigenvector (say e_1) that makes up the preferential space with other

Algorithm 2 Finding the preferential lower and upper vertices

```
Require: $N$ is a leaf node of the quadtree
Require: $\mathcal{D}'$ is the orthonormal user preferential space given by SVD
  function getPrefLowerAndUpperVertex($N$, $\mathcal{D}'$)
    $N.prefUpperVertex \leftarrow [\,]$
    $N.prefLowerVertex \leftarrow [\,]$
    $i \leftarrow 0$
    for all eigenvector $V$ in $\mathcal{D}'$ do
      $V_{sign} \leftarrow$ GetSignature($V$)
      $P_{upper} \leftarrow$ GetVertex($N$, $V_{sign}$)
      $P_{lower} \leftarrow$ GetVertex($N$, $2^d - 1 - V_{sign}$)
      $N.prefUpperVertex[i] \leftarrow$ Project($P_{upper}$, $\mathcal{D}'$)
      $N.prefLowerVertex[i] \leftarrow$ Project($P_{lower}$, $\mathcal{D}'$)
      $i \leftarrow i + 1$
    end for
  end function
```

eigenvectors vectors. Hence v_1 will participate towards creation of upper preferential vertex and v_2 will participate towards creation of lower preferential vertex. In a d dimensional space, each eigenvector vector has d coefficients and they can be either positive or negative. Let us assume there are i positive coefficients and $d-i$ negative coefficients. Since e_1 and v_1 have the same vertex signature, the projection of v_1 on e_1 must be $(a+k)\Sigma m_i - (a)\Sigma m_{d-i}$, where Σm_i denotes the sum of all the positive coefficients and Σm_{d_i} denotes the sum of all the negative coefficients. Since vertex signature of v_2 is ones' complement of v_1, the projection of v_2 on e_1 must be $(a)\Sigma m_i - (a+k)\Sigma m_{d-i}$. Subtracting latter from the former, the length of the side of the node in the preferential space is obtained as $k\Sigma m_i + k\Sigma m - d - i$ which is $k \times$

(*sum of coefficients of eigenvector vector*) which can be approximated as km. Thus the volume of hypercube in the preferential space will be $(km)^d$ which is exponentially larger than the volume of the hypercube in the original space.

Though the number of the points in the leaf node doesn't change from the original space to the preferential space, the volume of the hypercube enclosing the volume increases exponentially with dimensions. The additional space in which no point is situated is called the *dead space*. As a consequence, the chance of the lower preferential vertex of a leaf node dominating the upper vertex of a leaf node decreases with increase in dimensionality of the space. To overcome the limitations we suggest two techniques

3.2.2.1. Increase the depth of the quadtree, to reduce the volume of a leaf node of a quadtree.

3.2.2.2. Instead of using the lower preferential vertex of a node as a candidate to prune, find a point within the node which has high chances of pruning other nodes.

3.3 Pruning Strategy

In a d dimensional space, the $k(k \leq d)$ preference vectors are expressed as linear combination of the d attributes that makes the space. Thus each of the d dimensions (attributes) has a membership score towards each of the k preference vectors. For a given attribute, if all of its membership scores with the k preference vectors are positive, then those tuples which has high value for this attribute is likely to have high value in the preferential space as well. Thus by studying the membership scores of the d attributes, it is possible to predict the attributes whose value when high, increase the likelihood of the tuple to be part of the skyline. This attribute among all attributes that have the highest likelihood is called the *Critical Attribute*. The algorithm for finding the critical attribute is given in *Algorithm* 3.

Algorithm 3 Finding the Critical Attribute

```
Require: $d$ is the dimensionality of the original space
  function GetCriticalAttribute($\mathcal{D}'$)
      $critAttr \leftarrow -1$
      $sum \leftarrow 0$
      $maxSum \leftarrow 0$
      for all $i$ from 0 to $d-1$ do
          $sum \leftarrow 0$
          for all eigenvector $V$ in $\mathcal{D}'$ do
              $sum \leftarrow sum + V[i]$
          end for
          if $sum > maxSum$ then
              $maxSum \leftarrow sum$
              $critAttr \leftarrow i$
          end if
      end for
      return $critAttr$
  end function
```

As pointed out in section 3.2.1.4, each dimension has 2^t sub-lattices of quadtree nodes and the sub-lattice which has high values on that dimension can be inferred by bit interleaving value 2^t for that particular dimension. Since it is possible to predict the attribute which is likely to perform well on transformation, it is fair to claim that the tuples which has high values on that attribute (or dimension) is likely to be part of the skyline in the preferential space. Further the top most sub-lattice along the "critical attribute", contains all of the tuples which has high values along that dimension and hence they are the potential candidates of being the skylines in the preferential space. Also the sub-lattices which has low values for that attribute are likely to be dominated by sub lattices which has high values along that dimension. In section 3.2.2, it is pointed out that the likelihood of a quadtree node dominating an other node in preferential space is low because of the dead space. To overcome this problem it is needed to find candidates which have significant dominance region that can prune other nodes.

The pruning strategy adopted in this work is to:

1. Collect all the points from the top most sub-lattice along the "critical attribute" and sort them based on their volume of dominance in the preferential space.

2. Pick the top K points called the "candidate points" from the sorted list and use them to prune the other nodes, by checking the dominance of the point with the preferential upper vertex and preferential lower vertex of the node.

3. If the candidate point dominates the preferential upper vertex, it dominates all the points in that leaf node.

4. If the candidate point doesn't dominate the preferential upper vertex, but dominates the preferential lower vertex, then it may dominate some points in the node.

5. If the candidate point doesn't dominate the preferential lower vertex, then it doesn't dominate any point with the node.

6. The points that are not dominated by the candidate point are collected and added to the sorted list of points already maintained.

7. The list is sorted using the in-built merge sort of java which is optimized for nearly sorted lists. Hence sorting the list twice shouldn't add any significant overhead.

8. The Sort Skyline First algorithm is applied on this sorted list to identify the skylines of the preferential space.

We present the algorithm of preferential skyline processing in *Algorithm* 4

3.4 Proofs:

3.4.1 Proof I:

Let $P = \{p_1, p_2 ... p_d\}$ and $Q = \{q_1, q_2 ... q_d\}$ be two points in a d dimensional space where $P \succ Q$. Let the preferential space $\mathcal{D}'$ be comprised of set of orthonormal eigenvectors $\{V_1^{pref}, V_2^{pref} ... V_k^{pref}\}$, where $k \leq d$. Since $P \succ Q$, $\forall i, 1 \leq i \leq d, p_i \geq q_i$ and $\exists j, 1 \leq j \leq d, p_j > q_j$. In the preferential space P and Q is projected to P' and Q'.

3.4.1.1. *case 1:* $P' \succ Q'$. Let $\mathcal{D}'$ be an one dimensional space with one eigenvector V_1^{pref}. Let $V_1^{pref} = \{v_{11}, v_{12} ... v_{1d}\}$ has all positive coefficients. Since points form the original space are projected to the preferential space, $P' = \{p'_1\}$ can be defined as $P' = \{v_{11} * p_1 + v_{12} * p_2 + ... + v_{1d} * p_d\}$. Similarly $Q' = \{q'_1\}$ can be defined as $Q' = \{v_{11} * q_1 + v_{12} * q_2 + ... + v_{1d} * q_d\}$. Since $\forall i, 1 \leq i \leq d, v_{1i} > 0$ and $P \succ Q$, it is possible to say $v_{11} * p_1 + v_{12} * p_2 + ... + v_{1d} * p_d > v_{11} * q_1 + v_{12} * q_2 + ... + v_{1d} * q_d$ which implies $P' \succ Q'$.

3.4.1.2. *case 2:* $Q' \succ P'$. The proof is similar to 3.4.1.1, except for the assumption V_1^{pref} to have all negative coefficients. In other words, $\forall i, 1 \leq i \leq d, v_{1i} < 0$. Since $P \succ Q$, it is possible to say $v_{11} * p_1 + v_{12} * p_2 + ... + v_{1d} * p_d < v_{11} * q_1 + v_{12} * q_2 + ... + v_{1d} * q_d$ which implies $Q' \succ P'$.

3.4.1.3. *case 3:* $P' \sim Q'$. The proof is combination of 3.4.1.1 and 3.4.1.2. Let $\mathcal{D}'$ be a two dimensional space specified by eigenvectors V_1^{pref} and V_2^{pref}. Let V_1^{pref} has

Algorithm 4 Preferential Skyline Processing

```
Require: $\mathcal{D}'$ is the orthonormal user preferential space given by SVD
Require: $d$ is the dimensionality of the space
Require: $Nodes$ is the list of leaf nodes of the quadtree of depth $t$ sorted by the
  Z-order values of its lower vertex
Require: $C_Z(P)$ is the Z-order value of point $P$
Require: $AddPoints(N, \mathcal{D}', L)$ transforms all the points in a leaf node $N$ and adds
  it to the list L
Require: $AddNode(N, L)$ adds a leaf node $N$ to the list $L$
Require: $DominanceSort(P)$ sorts a set of points based on its volume of dominance
  function PSP($\mathcal{D}'$)
    $critAttr \leftarrow -1$
    $maxValue \leftarrow (1 << t) - 1$
    $mortonMask \leftarrow 0$
    $\mathcal{P}'_{short} \leftarrow$
    $prunableNodes \leftarrow []$
    for all node $N$ in $Nodes$ do
      GetPrefLowerAndUpperVertex($N$, $\mathcal{D}'$)
    end for
    $critAttr \leftarrow GetCriticalAttribute(\mathcal{D}')$
    while $true$ do
      for all $i$ from 0 to $t - 1$ do
        if $maxValue \& i = i$ then
          $mortonMask \leftarrow mortonMask | 1 << (i \times d) + critAttr$
        end if
      end for
```

```
        for all node $N$ in $Nodes$ do
            if $C_Z(N)$ $\&mortonMask == mortonMask$ then
                AddPoints($N$, $\mathcal{D}'$, $\mathcal{P}'_{short}$)
            else if
                thenAddNode($N$, $prunableNodes$)
            end if
        end for
        if Size($\mathcal{P}'_{short}$) $> 0$ then
            break
        end if
        $maxValue \leftarrow maxValue - 1$
    end while
    DominanceSort($\mathcal{P}'_{short}$)
    $\mathcal{P}'_{topK} \leftarrow$ GetTopK($\mathcal{P}'_{short}$, $K$)
    for all point $P$ in $\mathcal{P}'_{topK}$ do
        for all node $N$ in $Nodes$ do
            if $P \succ N.prefUpperVertex$ then
                Prune($N$)
```

all positive coefficients similar to 3.4.1.1 and V_2^{pref} has d-1 negative coefficients and one positive coefficient. P' and Q' can be represented as $P' = \{p'_1, p'_2\}$ and $Q' = \{q'_1, q'_2\}$. From 3.4.1.1, it is possible to infer that $p'_1 > q'_1$. Now, V_2^{pref} can be represented as $\{v_{21}, v_{22}...v_{2d}\}$ and $\exists j, 1 \leq j \leq d, v_{2j} > 0$ and $\forall i, i \neq j, v_{2i} < 0$. p'_2 can be specified as $\{v_{21} * p_1 + v_{22} * p_2 + ... + v_{2d} * p_d\}$ and similarly q'_2 can be specified as $\{v_{21} * q_1 + v_{22} * q_2 + ... + v_{2d} * q_d\}$. Depending on values of P and Q,

```
            **else if** $P' \succ N.prefLowerVertex$ **then**
                PartialyPrune($N$)
                $N.PartialPruner \leftarrow P'$
            **end if**
        **end for**
    **end for**
    **for all** node $N$ in $Nodes$ **do**
        **if** isPruned($N$) **then**
            continue
        **else if** IsPartiallyPruned($N$) **then**
            **for all** point $P$ in $N$ **do**
                $P' \leftarrow$ Project($P$)
                **if** $N.PartialPruner \not\succ P'$ **then**
                    Add(P', $\mathcal{P}'_{short}$)
                **end if**
            **end for**
        **else**
            **for all** point $P$ in $N$ **do**
                $P' \leftarrow$ Project($P$)
                Add($P'$, $\mathcal{P}'_{short}$)
            **end for**
        **end if**
    **end for**
    **return** SFS($\mathcal{P}'_{short}$)
**end function**
```

p'_2 can be greater than or lesser than q'_2. Consider the scenario where $p'_2 < q'_2$, it is possible to say $P' \sim Q'$ since $p'_1 > q'_1$ and $p'_2 < q'_2$

3.4.2 Proof II:

Let $P = \{p_1, p_2 ... p_d\}$ and $Q = \{q_1, q_2 ... q_d\}$ be two points in a d dimensional space where $P \sim Q$. Let the preferential space $\mathcal{D}'$ be comprised of set of orthonormal eigenvectors $\{V_1^{pref}, V_2^{pref} ... V_k^{pref}\}$, where $k \leq d$. Since $P \sim Q$, $\exists i, 1 \leq i \leq d, p_i > q_i$ and $\exists j, 1 \leq j \leq d, p_j < q_j$. In the preferential space P and Q is projected to P' and Q'.

3.4.2.1. *case 1:* $P' \succ Q'$. Let $\mathcal{D}'$ be an one dimensional space with one eigenvector V_1^{pref}. Let $V_1^{pref} = \{v_{11}, v_{12} ... v_{1d}\}$. Let V_1^{pref} has positive coefficients for all the dimensions where $p_i > q_i$ and negative coefficients when $p_i < q_i$. Mathematically, this can be expressed as $\forall i, 1 \leq i \leq d, p_i > q_i \Rightarrow v_{1i} > 0$ and $\forall i, 1 \leq i \leq d, p_i < q_i \Rightarrow v_{1i} < 0$. This implies that on projection of P and Q to $\mathcal{D}'$, P' will always dominate Q'.

3.4.2.2. *case 2:* $Q' \succ P'$. This proof is similar to 3.4.2.1 except for the assumption that V_1^{pref} has negative coefficients for all the dimensions where $p_i > q_i$ and positive coefficients when $p_i < q_i$. Mathematically, this can be expressed as $\forall i, 1 \leq i \leq d, p_i > q_i \Rightarrow v_{1i} < 0$ and $\forall i, 1 \leq i \leq d, p_i > q_i \Rightarrow v_{1i} > 0$. This implies that on projection of P and Q to $\mathcal{D}'$, Q' will always dominate P'.

3.4.2.3. *case 3:* $P' \sim Q'$. This proof is a combination of 3.4.2.1 and 3.4.2.2. Let $\mathcal{D}'$ be a two dimensional space specified by eigenvectors V_1^{pref} and V_2^{pref}. P' and Q' can be represented as $P' = \{p'_1, p'_2\}$ and $Q' = \{q'_1, q'_2\}$. Let V_1^{pref} be similar to 3.4.2.1 in which case $p'_1 > q'_1$. Let V_2^{pref} have all positive coefficients, that is $\forall i, 1 \leq i \leq d, v_{2i} > 0$. p'_2 can be specified as $\{v_{21} * p_1 + v_{22} * p_2 + ... + v_{2d} * p_d\}$

and similarly q'_2 can be specified as $\{v_{21} * q_1 + v_{22} * q_2 + ... + v_{2d} * q_d\}$. Depending on the values of P and Q, p'_2 can be greater than or lesser than q'_2. Consider the scenario where $p'_2 < q'_2$, it implies $P' \sim Q'$ since $p'_1 > q'_1$ and $p'_2 < q'_2$.

Chapter 4

LATENT SKYLINE PROCESSING

4.1 Introduction

Section 1.2 introduced the concept of latent skyline processing. Latent skyline processing is similar to preferential skyline processing, except that the subspace enclosed by the characteristic vectors isn't orthonormal. Since the subspace isn't orthogonal, there is a need to have a transformation that would not only map the data from the original space to the preferential space, but also maintain the semantical independence of the characteristic vectors. The semantical independence of the preferential space is important because each characteristic vector denotes a specific concept which are independent of other characteristic vectors.

If the transformation does not enforce the semantic independence of the characteristic vectors, there could be correlation or anti-correlation between the characteristic vectors. Skyline on such a space would result in vital information loss. The loss is explained in section 4.4 through a transformation in section 4.2. This research introduces a novel transformation technique called *Stretching* in section 4.3 which maintains the semantic independence of the characteristic vectors.

Consider a d dimensional data space $\mathcal{D}$ and a data set $\mathcal{P}$ on $\mathcal{D}$. Consider a subspace of $\mathcal{D}$ formed by $k, k \leq d$ characteristic vectors denoted by $\mathcal{D}_k^{pref}$. These characteristic vectors are denoted as $\{V_1^{pref}, V_2^{pref} ... V_k^{pref}\}$ and together form the sub-space $\mathcal{D}_k^{pref}$. Each of these characteristic vector indicates a specific independent concept in the original space $\mathcal{D}$. When the transformation is complete, the new space, $\mathcal{D}'$ should be a k dimensional orthogonal space. In case, the number of characteristic vectors exceeds the dimensionality of the data space, additional dimensions are introduced in the data space to equal the number of characteristic vectors. Introduction of additional dimensions are trivial as all data would have zero membership towards the new dimension. As a consequence, for rest of the discussion, the number of characteristic vectors k is considered lesser or equal to dimensionality of the data space d.

4.2 Projection

Projection describes a transformation where a point P, $P \epsilon \mathcal{P}$ is projected on to each of the k characteristic vectors and the projection, P makes on the characteristic vector V_k^{pref} is considered as the membership of point P along that dimension. The k projections, P makes on each of the k characteristic vectors would indicate the P' score along the k dimensions in the new k dimensional space. An algorithm for projection is specified in *Algorithm* 5

Algorithm 5 Projecting a point in the subspace

```
Require: $P$ is a point in the $d$ dimensional space $\mathcal{D}$
Require: $\mathcal{D}_k^{pref}$ are the $k$ characteristic vectors: $\{V_1^{pref}, V_2^{pref}, ...V_k^{pref}\}$
  function PROJECT($P$, $mathcal_k^{pref}$)
      $i \leftarrow 0$
      $P' \leftarrow [\,]$
      for all characteristic vector $V_i^{pref}$ in $\mathcal{D}_k^{pref}$ do
          $P'[i] \leftarrow$ PROJECT($P$, $V_i^{pref}$)
          $i \leftarrow i + 1$
      end for
      return $P'$
  end function
```

We consider the point P' to be the point in the new k dimensional space $\mathcal{D}'$. We prove that the above transformation does not maintain the semantic independence of the characteristic vectors in section 4.10.1. The weakness of the transformation through projection is that even if a data point lies completely on a characteristic vector, it still has a non zero membership value with other characteristic vectors in the new space. This is a clear violation of principles of mutual independence which suggests that orthonormal vectors should have no correlations between them.

4.3 Stretching

In a d dimensional orthonormal system, a vector v's membership with the coordinate axis is resolved using the $|v| \cos \theta$ formula where θ is the angle, the vector v makes with the independent dimensions. Further when a vector, v is aligned with a dimension (makes $0°$), the vector has non zero membership only with that dimension. This prevents any correlation between the dimensions. From section 4.2 it is evident that projection doesn't work if the dimensions are not independent. In scenarios, where the sub-space is not orthonormal, two fundamental questions needs to be answered.

1. Given a vector v, find those characteristic vectors with which the vector has non zero membership.

2. Given a vector v and the set of characteristic vectors with which it has non zero membership, find its memberships.

The outline of stretching algorithm is presented in section 4.3.1. The description of the algorithm is given in section 4.3.4 and section 4.9 presents a formal algorithm for calculating latent skylines.

4.3.1 Outline

We begin with the assumption that we have d characteristic vectors in a d dimensional space and later we extend to cover the more general scenario of having k characteristic vectors in a d dimensional space, where $k \leq d$ in section 4.9. Given d characteristic vectors in a d dimensional space, we map each of the characteristic vector to an axis in the original space. Each characteristic vector is mapped to the axis closest to it. Then the subspace is divided into d regions with help of a *Nullification Point* (explained in section 4.3.3), represented by $\hat{c}$ which is a weighted arithmetic mean of the characteristic vectors. Thus the "nullification point", $\hat{c}$ is always within the subspace. Now each of the d characteristic vectors are pulled towards its mapped axis, thus stretching the space between them and making the characteristic vectors orthonormal to each other.

In a d dimensional space, the nullification point divides the subspace into d regions. Each of the d regions are bounded by $d-1$ dimensional hyperplanes and there are d such hyperplanes. This is true for any d dimensional space and can be observed in the original given space as well. For example, a two dimensional space has two one dimensional hyperplane (lines) which we call as X-axis and Y-axis. Similarly a three dimensional space is bounded by three two dimensional hyperplanes namely XY-plane, YZ-plane and ZX-plane. These hyperplane boundaries are formed by characteristic vectors and nullification point. Among d hyperplane boundaries, one boundary is made up only characteristic vectors. It is convenient to visualize the stretching algorithm as stretching of the outer hyperplanes of the regions to the hyperplanes formed by the axis.

4.3.2 Characteristic Vectors Mapping

In section 4.3.1 we suggested that each characteristic vector is associated to the axis closest to it. In some cases, there can be multiple characteristic vectors, for which, a certain axis is closer than the rest. To deal with such conflicts, we propose an iterative algorithm that would map the characteristic vector to an axis. The iterative algorithm is based on the condition, that a characteristic vector is mapped to an axis, if and only if it is the closest unmapped characteristic vector to it. The algorithm used to map characteristic vectors to the axes is given in *Algorithm* 6.

4.3.3 Nullification Point Of Subspace

Consider a d dimensional space with d characteristic vectors. During stretching we pull all of the d vectors to map to the d dimensions. Each of the point in the sub-space could be pulled along any of the d directions. The nullification point of the sub-space divides the sub-space into d regions with each region pulled in a direction. A straightforward method to compute the nullification point might be to take the arithmetic mean of all the characteristic vectors but this could skew the stretch in favor of a characteristic vector. The skewing is explained in detail through two examples in 4.3.3.1 and 4.3.3.2.

4.3.3.1. Consider the following example where we have two characteristic vectors V_1^{pref} and V_2^{pref} in a two dimensional space. Let V_1^{pref} make 10° with X axis and V_2^{pref} make 20° with X axis. In this case V_1^{pref} will be stretched towards X axis over a region spanning 10° while V_2^{pref} will be stretched towards Y axis over a region spanning 70°. If the nullification point is determined by the arithmetic mean of V_1^{pref} and V_2^{pref}, the nullification point $\hat{c}$ will make 15° with X axis. Now if the points between $\hat{c}$ and V_1^{pref}, called region 1 are stretched towards X

Algorithm 6 Characteristic Vector Mapping

```
Require: $\mathcal{D}_k^{pref}$ are the $k$ characteristic vectors: $\{V_1^{pref}, V_2^{pref}, ...V_k^{pref}\}$
Require: The $k$ axis vectors are $\{A_1, A_2...A_k\}$
  function MAP($\mathcal{D}_k^{pref}$)
    $angles \leftarrow [\,][\,]$
    $isAssigned \leftarrow [\,]$
    $i \leftarrow 0$
    for all characteristic vector $V_i^{pref}$ in $\mathcal{D}_k^{pref}$ do
      $j \leftarrow 0$
      $isAssigned[i] \leftarrow$ -1
      for all axis vector $A_j$ in $\{A_1, A_2...A_k\}$ do
        $angles[i][j] \leftarrow$ GETANGLE($V_i^{pref}$, $A_j$)
        $j \leftarrow j + 1$
      end for
      $i \leftarrow i + 1$
    end for
    $done \leftarrow False$
    $minAngle \leftarrow infinity$
    $closestAxis \leftarrow NULL$
    while (!$done$) do
      $i \leftarrow 0$
      for all characteristic vector $V_i^{pref}$ in $\mathcal{D}_k^{pref}$ do
        $j \leftarrow 0$
        for all axis vector $A_j$ in $\{A_1, A_2...A_k\}$ do
          if $angles[i][j] < minAngle$ then
```

```
                $minAngle \leftarrow angles[i][j]$
                $closestAxis \leftarrow j$
            **end if**
            $j \leftarrow j + 1$
        **end for**
        $l \leftarrow 0$
        **for all** characteristic vector $V_l^{pref}$ in $\mathcal{D}_k^{pref}$ **do**
            **if** $angles[l][closestAxis] < minAngle$ **then**
                **if** $isAssigned[l] \neq -1$ **then**
                    **if** $isAssigned[l] \neq closestAxis$ **then**
                        continue
                    **else**
                        $angles[i][closestAxis] \leftarrow infinity$
                        break
                    **end if**
                **end if**
            **end if**
        **end for**
        $isAssigned[i] \leftarrow closestAxis$
        $i \leftarrow i + 1$
    **end for**
    **if** all characteristic vectors are mapped **then**
        done ← true
    **end if**
  **end while**
**end function**
```

axis and points between $\hat{c}$ and V_2^{pref}, called region 2 are stretched towards Y axis, the stretch of region 2 covers larger area ($87.5\% more$) than the stretch of region 1 thus skewing the stretch in favor of points in region 2.

4.3.3.2. Consider a three dimensional space with three characteristic vectors V_1^{pref}, V_2^{pref} and V_3^{pref}. Let $\vec{i}$, $\vec{j}$ and $\vec{k}$ represent the unit vectors along X, Y and Z axis respectively. Let $\hat{c}$ be the arithmetic mean on the three characteristic vectors. Let $V_1^{pref} = 0.9\vec{i} + 0.2\vec{j} + 0.2\vec{k}$, $V_2^{pref} = 0.3\vec{i} + 0.9\vec{j} + 0.3\vec{k}$ and $V_3^{pref} = 0.4\vec{i} + 0.4\vec{j} + 0.8\vec{k}$. V_1^{pref}, V_2^{pref} and V_3^{pref} are stretched towards X, Y and Z axis respectively. When V_1^{pref} and V_2^{pref} are stretched towards X and Y axis, The plane formed by V_1^{pref} and V_2^{pref} coincide with XY plane. This makes up one stretch. Similarly planes formed by vectors V_2^{pref} and V_3^{pref} and vectors V_3^{pref} and V_1^{pref} coincide with YZ and ZX plane respectively making the other two stretches. The data vectors within the sub space will participate in one of the above three stretches depending on its position relative to $\hat{c}$: for instance, if a data vector is present in a volume enclosed by V_1^{pref}, V_2^{pref} and $\hat{c}$, it will be stretched towards XY plane. One can infer that each stretch will span different volume and thus a nullification point based on arithmetic mean will skew the stretches.

When the stretches are skewed, data within one of the regions are stretched considerably more and it takes up more space, there by standing a greater chance of being part of the skyline in the new space. To avoid this bias, we divide the region in proportion with the space available for a region to stretch. Thus in case of 4.3.3.1 region 1's available space to stretch is one-seventh of region 2's available space to stretch. So we formulate the nullification point in such a way that the region 1 is one-seventh of the area of region 2, thus eliminating the bias. The algorithm for computation of nullification point of subspace is given in *Algorithm* 7

4.3.4 Stretching Algorithm

We are ready to formally present the stretching algorithm. We had previously established that, we compute the nullification point of the sub space to divide the subspace into regions. For a d dimensional subspace, we form d regions with each of the d regions enveloped by $d-1$ dimensional hyperplanes. Consider the d characteristic vectors, $\{V_1^{pref}, V_2^{pref}, ..., V_d^{pref}\}$. These d vectors form a d dimensional space. We can replace a characteristic vector by the nullification point vector to form d regions. For example, $\{\hat{c}, V_2^{pref}, ..., V_d^{pref}\}$ and $\{V_1^{pref}, \hat{c}, ..., V_d^{pref}\}$ would be two of the d regions. Each of this region is enveloped by $d-1$ dimensional hyperplanes and there are d such hyperplanes. Thus if $\{\hat{c}, V_2^{pref}, ..., V_d^{pref}\}$ is the region in consideration, then $\{V_2^{pref}, V_3^{pref} ..., V_d^{pref}\}$ and $\{\hat{c}, V_2^{pref}, ..., V_{d-1}^{pref}\}$ would be two of the d such $d-1$ hyperplanes. Of those d hyperplanes, we can observe there would be a hyperplane formed with $d-1$ characteristic vectors called the *outer hyperplane*. We stretch the region by keeping the nullification point $\hat{c}$ fixed, while stretching the outer hyperplane to coincide with the hyperplane formed by those axis vectors which maps to these characteristic vectors. Let $\{A_1, A_2, ...A_d\}$ be the d axis in the original dimensions with an one to one mapping between the characteristic vectors and axis. Let's assume that the characteristic vector V_i^{pref} is mapped to the axis A_i where $1 \leq i \leq d$. Then the hyperplane formed by vectors $\{V_2^{pref}, V_3^{pref}, ..., V_d^{pref}\}$ would be stretched to coincide with the hyperplane formed by the vectors $\{A_2, A_3, ..., A_d\}$. All of the points found in that region would be subjected to this stretch. Consequently the points that are outside the subspace are discarded and not included in the skyline computation.

Algorithm 7 Computation of Nullification Point

```
Require: $\mathcal{D}_d^{pref}$ contains the $d$ characteristic vectors
Require: Array $Angles_d$ contains the angles $d$ regions make with $d$ axis planes
  function COMPUTENULLIFICATIONPOINT($\mathcal{D}_k^{pref}$)
      $angleSum \leftarrow 0$
      $i \leftarrow 0$
      $j \leftarrow 0$
      $temp \leftarrow 0$
      $\hat{c} \leftarrow []$
      for all angle $A$ in array $Angles_d$ do
          $angleSum \leftarrow angleSum + A$
      end for
      while $i < d$ do
          $temp \leftarrow 0$
          while $j < d$ do
              $temp \leftarrow V_j^{pref}[i] * (Angles_d[(j + d - 1)\%d]/angleSum)$
          end while
          $\hat{c}[i] \leftarrow temp$
      end while
      return $\hat{c}$
  end function
```

When a region is stretched, the position of all the vectors in the region change but the nullification point. If we can imagine all the stretches to be concurrent forces acting on the sub-space, then the nullification point can be considered to be in equilibrium under all those forces. It is called the nullification point because all the forces that stretch the space are nullified and the point stays in equilibrium. Since the outer hyperplane coincides with the hyperplane formed by axes after stretching, we can establish that any vectors lying in the outer hyperplane will coincide with the hyperplane formed by the axis. These are the boundary conditions, where nullification point represents area of no stretch and the outer hyperplane represents area of maximum stretch. Every point in the region can be stretched relatively based on the boundary conditions. We answered the question "how much to stretch a point?" but this beckons us with an important question "Where should a point be stretched to?". Stretching can be considered as a rotation operation where the data is rotated along a plane by a certain angle. We know the nullification point is rotated by $0°$ and the outer hyperplane is rotated through an angle θ where θ is the angle between outer hyperplane and axis hyperplane. With these two information we can figure the angle by which any data should be stretched (rotated). We have only one clue to determine the direction of the rotation and that is, if the data lies on the outer hyperplane, it is stretched all the way to the hyperplane formed by axes. So we begin by rotating the data to coincide with the outer plane. We can also consider stretching as an application of non coplanar, concurrent forces on a non rigid body with all the forces intersecting at the nullification point. So it is only natural to rotate the data through the plane formed by the data and the nullification point, and this is guaranteed to intersect the outer plane. The vector as a result of this intersection is termed as *Data at intersection*. We can measure the angles the *data at intersection* makes with the characteristic vectors that forms the hyperplane. We also know that if the original location of the data was on the outer hyperplane, it would have been stretched to the hyperplane formed by axes. We term the vector on the hyperplane formed by axes as *data on axes hyperplane.* Further the angles made by the *Data at intersection* with the characteristic vectors must be relative to the angles made by *data at axes hyperplane* with the axis. In section 4.3, we mentioned that it is imperative to find those characteristic vectors with which a data vector has non zero membership. For the above reason we introduce a new concept called the *critical characteristic vectors.* The critical characteristic vectors of a data vector are the characteristic vectors that forms a hyperplane and has the lowest dimensionality among all the hyperplanes that contains the data vector. For example, in a three dimensional space with three characteristic vectors V_1^{pref}, V_2^{pref} and V_3^{pref}, If a vector v, lies along a characteristic vector V_1^{pref}, then just that characteristic vector (V_1^{pref}) is the critical characteristic vector of the vector v. Similarly if the vector v and two other characteristic vectors, say V_1^{pref} and V_2^{pref} are coplanar, then those two characteristic vectors (V_1^{pref} and V_2^{pref}) become the critical characteristic vectors of the vector v. We claim a given vector must have non zero membership only with its critical vectors.

4.3.4.1. Let θ_1, θ_2 ... θ_k where $k \leq d-1$ be the angles, *data on intersection* makes with the k critical characteristic vectors that makes the outer hyperplane.

4.3.4.2. We can express these angles as θ_1, $m_2 * \theta_1$ ... $m_k * \theta_1$ on the assumption $\theta_1 \leq \theta_i$, where $2 \leq i \leq k$ and $m_2, m_3 ... m_k$ are positive real constants.

4.3.4.3. Let α_1, α_2 ... α_k be the angles, *data on axes hyperplane* makes with k axis. It is to be noted these k axis are mapped to the k characteristic vectors previously.

4.3.4.4. Since the location of the *data on axes hyperplane* should be relative to location of *data on intersection*, 4.3.4.3 can be expressed as α_1, $m_2 * \alpha_1$ ... $m_k * \alpha_1$.

4.3.4.5. Since the sum of direction cosines of a vector is 1, we know $\alpha_1^2 + (m_2 * \alpha_1)^2 + ... + (m_k * \alpha_1)^2$ should be 1.

From 4.3.4.5 we can resolve for α_1 and thus the location of *data on axes hyperplane*. Now we propose the plane of rotation of the data as the plane that contains data and *data on axes hyperplane*. We again find the intersection of the plane of rotation with the outer hyperplane and call this intersection as *data on outer hyperplane*. Given this plane of rotation we know that if data lies on *data on outer hyperplane*, it would coincide with the *data on axis hyperplane*. Thus by measuring the angles between these three vectors we can determine the final position of the data.

β = Angle between data and "data on axes hyperplane"

γ = Angle between "data on outer hyperplane" and "data on axes hyperplane"

$$\delta = ((90 - \beta) \times 90/(90 - \gamma)) - (90 - \beta) \tag{4.1}$$

Thus δ specified in equation 4.1 gives the angle of stretch along the plane of rotation formed by the vectors data and *data on axes hyperplanes*. Now we try to assert the usefulness of the stretching algorithm by trying to answer the question posed at the beginning of the section 4.3 which are crucial to a transformation technique. The questions are repeated here.

1. Given a vector v, find those characteristic vectors with which the vector has non zero membership.

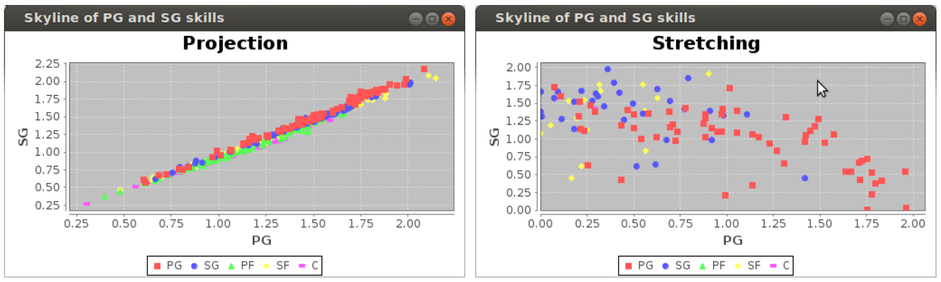
2. Given a vector v and the set of characteristic vectors with which it has non zero membership, find its memberships.

Once the critical characteristic vectors of a data vector are established, its membership when stretched should be relative to its similarity with those critical characteristic vectors in the original space. Thus we can see the stretching algorithm's transformation is consistent with both the requirements.

4.4 Projection Vs Stretch

We saw an example problem involving latent skylines in section 1.2, where a NBA manager is trying to recruit a player who can play in both SG and PG positions. In this section, we use the same example to compare the projection and stretching transformations. The example is re-explained here for convenience. A NBA manager wants to sign a player who can play in both PG and SG positions. His decision making can be eased if he can find the skyline of players where the attributes are their PG rating and SG rating. The manager has all statistics of the players. He needs to find the characteristic vectors of the PG and SG position and transform the existing space into the two dimensional space where one dimension indicates the players' skill (or rating) as a PG player and other position indicates the players' skill (or rating) as a SG player. He can obtain the characteristic vector of PG (and SG) by doing SVD on statistics of players who are classified as legendary PG (and SG) players by NBA. In other words, the skill of these players is the very definition of the PG (and SG) role. Once the characteristic vectors are obtained, he can transform into the latent space either using projection or stretching.

For the sake of the experiment we assume the manager doesn't know the positions of these players. He just knows their statistics and relies on the system to get the best PG and SG players. Figure 4.1 compares the transformation through both projection and stretching and tables 4.1 and 4.2 gives the skyline via both the transformations. Tables 4.1 and 4.2 gives the name of the player and the position he plays. We can see from Figure 4.1a that projection doesn't make the characteristic vectors independent as there is significant correlation and all the players are clumped together. There



(a) Transformation through projection.

(b) Transformation through Stretching.

Figure 4.1: Projection Vs Stretching

is no useful inferences that can be drawn and the skyline in table 4.1 is just three players with very good statistics. However Figure 4.1b treats the two characteristic vectors independent, as one can see players who play in SG position are clumped together Y axis which is the axis SGs' characteristic vector has been stretched to. Similarly players who play as PG are more crowded near X axis which is the axis PGs' characteristic vector has been stretched to. Semantically we can infer

1. The players in the middle of the plot to be a good PG and SG player.
2. The players near X axis are good PG players.
3. The players near Y axis are good SG players.

Thus we can perceive stretching to be a superior transformation to projection.

4.5 Limitations Of Vector Calculus

Vector calculus is the de facto standard for solving problems in Euclidean space. The stretching algorithm also can be implemented using vector calculus but only for two and three dimensions. The above limitation is due to the fact that in vector calculus, cross product in particular does not generalize to higher dimensions. Stretching

Player	*Position*
LeBron James	SF
Kevin Durant	SF
Chris Paul	PG

Table 4.1: Skyline Of Players Through Projection.

Player	*Position*
LeBron James	SF
Jrue Holiday	PG
Kyrie Irving	PG
Kobe Bryant	SG
Russell Westbrook	PG
Monta Ellis	PG
Stephen Curry	PG/SG
Mike Conley	PG
Jason Kidd	PG
Brandon Jennings	PG
Manu Ginobili	SG

Table 4.2: Skyline Of Players Through Stretching.

algorithm involves rotation of a vector on a plane for a particular angle. In order to compute the equation of the plane, we need to find the vector normal to the plane and the common way to find that normal is by using cross product. The cross product can help us find the normal to a plane in three dimensions but fails for higher dimensions, reasons for which is explained below.

Given two linearly independent vectors in $\mathfrak{R}^3$, these two vectors form a plane and the dimension of the space perpendicular to the plane on both sides of the plane is one. We use *right hand rule* to decide one of the opposing directions. In higher dimensions, the problem becomes more complicated since the perpendicular space on both sides of the plane has higher dimensions and a way to consistently choose one of the many possible directions is not apparent. Thus the non applicability of cross product at higher dimensions prevents us from implementing stretching algorithm using vector calculus.

4.6 Introduction To Clifford Algebra

Lehar (2014) describes *Clifford Algebra* a.k.a *Geometric Algebra* as a most extraordinary synergistic confluence of a diverse range of specialized mathematical fields, each with its own methods and formalisms, all of which find a single unified formalism under Clifford Algebra. History of algebra has seen many confounding theories since its Arabic origins around 810 AD. Clifford algebra attempts to incorporate all these theories to achieve a single consistent formalism that generalizes to arbitrary number of dimensions. Lehar (2014) and Diek and Kantowski (1995) lists the historical summary that led to Clifford algebra as follows.

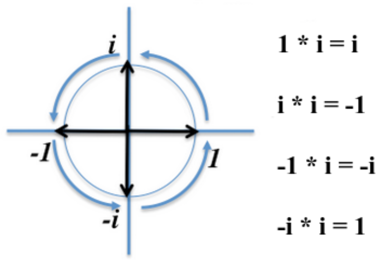


Figure 4.2: Multiplication By i Leads To Rotation By 90°

1. Introduction of complex numbers and the apparent discovery that multiplication of the imaginary component performs a rotation. For example multiplication by i results in a 90° rotation as explained in Figure 4.2[1].

[1]This image is obtained from Lehar (2014)

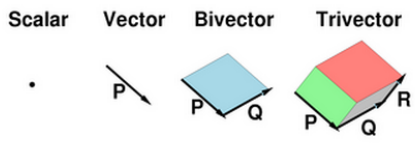
2. Gibbs introduced vectors and introduced dot product and cross product. Dot product has been incorporated into Clifford algebra but cross product as we saw in section 4.5 does not generalize well to arbitrary dimensions.

3. Hamilton proposed the quaternions format where he used three imaginary axis (i j and k) to take advantage of the rotational property of imaginary numbers along with a scalar. He defined a compound number of the form $v = a + bi + cj + dk$ and to this date many applications use quaternions for handling rotation as opposed to standard vector calculus.

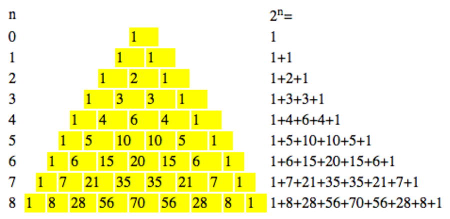
4. Grassman proposed an alternative to the flawed cross product known as exterior product or wedge product. Wedge product is an important component of clifford algebra and is discussed in detail in section 4.6.2.

5. Clifford took all the best ideas from the above works to achieve a consistent formalism that generalizes to arbitrary number of dimensions and is described in this section.

4.6.1 Cliffs And Grades



(a) Grades 0 to 3 (starting from left)



n		2^n=
0	1	1
1	1 1	1+1
2	1 2 1	1+2+1
3	1 3 3 1	1+3+3+1
4	1 4 6 4 1	1+4+6+4+1
5	1 5 10 10 5 1	1+5+10+10+5+1
6	1 6 15 20 15 6 1	1+6+15+20+15+6+1
7	1 7 21 35 35 21 7 1	1+7+21+35+35+21+7+1
8	1 8 28 56 70 56 28 8 1	1+8+28+56+70+56+28+8+1

(b) Clifford subspaces specified by Pascal's Triangle

Figure 4.3: Clifford Subspaces And Grades

Clifford defined a hierarchy of compound number knows as “clif”, or “multivector”, that range from zero dimensions (scalar) to one dimension (vector) to two dimensions (bivector), to three dimensions (trivector), and so on. In Clifford Algebra dimensions are referred to as grades, so a scalar is grade zero, a vector is grade 1, etc. Figure 4.3a[2]. depicts the grades from 0 to 3.

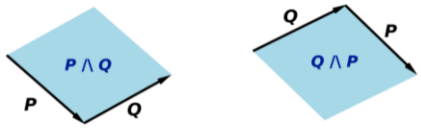
Any component (includes subspace) in a clifford algebra is specified as a cliff. Any scalar quantity is just represented as a scalar of grade 0 and a vector is represented as a vector of grade 1. Any plane is represented as a bivector having grade 2 and so on. In a $\mathfrak{R}^2$ space with basis $\{e_1, e_2\}$ there is a bivector formed by vectors e_1 and e_2 and can be represented as e_{12}. However in $\mathfrak{R}^3$, there are three basis $\{e_1, e_2, e_3\}$ which makes three possible configurations for bivectors possible, namely $\{e_{12}, e_{13}, e_{23}\}$. In fact, $\mathfrak{R}^3$ is composed of a scalar, three vectors (e_1, e_2, e_3), three bivectors (e_{12}, e_{13}, e_{23}) and a trivector (e_{123}). This way we can decompose any space into its lower grades and this decomposition follows the pattern corresponding to Pascal’s triangle. The pascal triangle and breakdown of number of grades for dimensions up to eight is given in Figure 4.3b. We can infer from the Figure 4.3b that a given dimension d would be composed of 2^d grades.

4.6.2 Outer Product Or Wedge Product

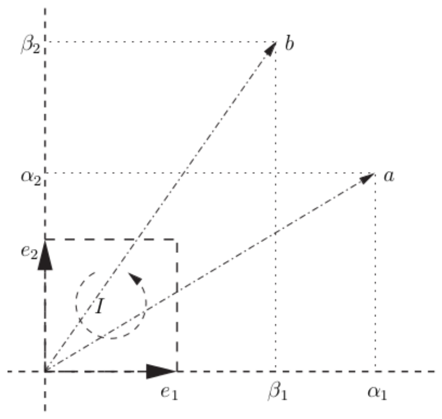
Lehar (2014) defines the wedge product of two vectors as a patch of surface (bivector) whose area is equal to the product of the two vectors, as if one vector was swept along the other, and that surface is within the plane that contains both of the original vectors. Wedge products are anti-commutative and can be called as an oriented area as shown in Figure 4.4a[3]. In Figure 4.4a, P and Q are the two vectors and the

[2]This image is obtained from Lehar (2014)

[3]This image is obtained from Lehar (2014)



(a) Wedge Product Representation



(b) Wedge Product in Euclidean Plane $\mathfrak{R}^2$

Figure 4.4: Wedge Product Representation

shaded area represents their wedge product. Wedge product of two vectors p and q is represented as $p \wedge q$. An important consequence of the above definition is that wedge product of a vector with itself is 0 since there is no area to be swept across.

$$p \wedge p = 0 \tag{4.2}$$

$$p \wedge q = -(q \wedge p) \tag{4.3}$$

Suter (2003) presents an elegant description as how arbitrary grades are decomposed to their basis. Given any n-dimensional vector a, it has to be decomposed to its basis $\{e_1, e_2 ... e_n\}$. In other words any vector is expressed as a linear combination of its orthonormal bases. Bivectors are very much alike and can be expressed as linear combination of basis bivectors.

Consider two vectors $a = (\alpha_1, \alpha_2)$ and $b = (\beta_1, \beta_2)$ in the Euclidean Plane $\mathfrak{R}^2$ depicted in Figure 4.4b[4]. The vectors can be decomposed to their basis as

[4]This image is obtained from Lehar (2014)

$$a = \alpha_1 e_1 + \alpha_2 e_2$$
$$b = \beta_1 e_1 + \beta_2 e_2$$

The outer product of a and b becomes

$$a \wedge b = (\alpha_1 e_1 + \alpha_2 e_2) \wedge (\beta_1 e_1 + \beta_2 e_2)$$
$$a \wedge b = (\alpha_1 e_1 \wedge \beta_1 e1) + (\alpha_1 e_1 \wedge \beta_2 e_2) + (\alpha_2 e_2 \wedge \beta_1 e_1) + (\alpha_2 e_2 \wedge \beta_2 e_2)$$
$$a \wedge b = (\alpha_1 \beta_1 e_1 \wedge e1) + (\alpha_1 \beta_2 e_2 \wedge e_2) + (\alpha_2 \beta_1 e_2 \wedge e_1) + (\alpha_2 \beta_2 e_2 \wedge e_2) \qquad (4.4)$$

Using 4.2 we can simplify 4.4 as

$$a \wedge b = (\alpha_1 \beta_2 e_1 \wedge e_2) + (\alpha_2 \beta_1 e_2 \wedge e_1)$$
$$Let I = e_1 \wedge e_2$$
$$\Rightarrow -I = e_2 \wedge e_1$$
$$\Rightarrow a \wedge b = (\alpha_1 \beta_2 - \alpha_2 \beta_1) I \qquad (4.5)$$

The outer product of two vectors is given in 4.5, which is a bivector and is expressed in terms of basis bivectors, which in euclidean plane $\Re^2$ is $I = e_1 \wedge e_2$. In 3-dimensional space $\Re^3$ made up of basis $\{e_1, e_2, e_3\}$, a bivector can also be expressed in terms of basis bivectors but the expression would be different as there are three basis bivectors namely $\{e_1 \wedge e_2(e_{12}), e_2 \wedge e_3(e_{23}), e_3 \wedge e_1(e_{31})\}$.

Let a and b be two vectors given by 4.6 and 4.7 respectively,

$$a = \alpha_1 e_1 + \alpha_2 e_2 + \alpha_3 e_3 \tag{4.6}$$

$$b = \beta_1 e_1 + \beta_2 e_2 + \beta_3 e_3 \tag{4.7}$$

$$\tag{4.8}$$

The wedge product of a and b is given by 4.9. Please note the basis of the multivector in 4.9 are the bivectors in $\mathfrak{R}^3$

$$a \wedge b = (\alpha_1\beta_2 - \alpha_2\beta_1)e_{12} + (\alpha_1\beta_3 + \alpha_3\beta_1)e_{13} + (\alpha_2\beta_3 - \alpha_3\beta_2)e_{23} \tag{4.9}$$

4.6.3 Geometric Product

Geometric product is a clifford operator obtained from combining dot product and geometric product. Geometric product of vectors a and b is given in equation 4.10. Geometric product of two vectors have important significance in rotations of vectors discussed in section 4.10.

$$ab = a\dot{b} + a \wedge b \tag{4.10}$$

4.6.4 Clifford Reversal

Clifford algebra defines the reverse of a product of vectors as the multivector formed by reversing the order of vectors. For example if $v = \alpha e_1 \wedge e_2$, then reverse of v given by $v^{\sim}$ is $\alpha e_2 \wedge e_1$.

4.6.5 Rotations And Other Formulations

Denker (2003) suggests that rotation can be specified by two parameters

1. Plane of rotation

2. Angle of rotation

These two parameters can be encoded by specifying two vectors. The plane containing the two vectors is the plane of rotation and the angle between the two vectors specify the angle of rotation. In clifford algebra, the term “rotor” is used when describing rotations. Given two vectors a and b to specify the rotation, the rotor of the rotation is given as the geometric product of a and b. The rotor has both a scalar component and a bivector component.

In clifford algebra, the general formula to rotate a vector using a rotor is specified in 4.11

$$v' = r^{\sim} v r \tag{4.11}$$

where

$v =$ given vector

$r =$ rotor

$r^{\sim} =$ clifford reversal of rotor

$v' =$ vector obtained by rotating v using the rotor r

This is the formula to rotate cliffs of any grade in any dimensional space. The full proof for this formula is beyond this text. Please refer Suter (2003) for more details. In this way clifford algebra presents consistent formalism to compute intersection of subspaces of arbitrary dimensions and find angles between them. For an introduction to clifford algebra please refer Denker (2006). For more advanced reading please refer Dorst and Mann (2002).

4.7 Stretch Algorithm

Having introduced the concepts of clifford algebra, we can formally present the core stretching algorithm which is implemented using concepts of clifford Algebra. The core stretching algorithm is given in *Algorithm* 8

Algorithm 8 Simple Stretch Algorithm

```
Require: outerHyperplane is the outer hyperplane of the region in which data is present
  function Stretch(Vector data)
      initPlaneofRotation ← WedgeProduct(nullificationpoint, data)
      dataAtIntersection ← Meet(outerHyperplane, initPlaneOfRotation)
      dataAtAxisHP ← GetDataAtAxis(dataAtIntersection, GetMagnitude(data))
      finalPlaneofRotation ← WedgeProduct(data, dataAtAxisHP)
      dataOnOuterHP ← Meet(finalPlaneofRotation, outerHyperplane)
      angle ← GetStretchAngle(data, dataOnOuterHP, dataAtAxisHP)
      stretchedData ← Rotate(finalPlaneofRotation, angle, data)
      return stretchedData
  end function
```

4.8 Scope for Pruning

In the context of latent skyline processing, we have so far figured an algorithm to stretch all the points that would fall within the space enclosed by the characteristic vectors, but stretching is a costly operation not because of the complex rotations involved but also because of the inherent complexity of each clifford operations. As mentioned in section 4.6, a given dimension d is composed of 2^d grades. So operations in clifford algebra are always exponential in nature. Since we are interested in the latent skyline, we have to use any known skyline algorithm like Sort-Skyline-First to find the skyline amongst all of the stretched data. Our next technique is to figure if a point would be dominated in the stretched space without stretching it.

The only information we have available is whether a given point q, is dominated by some point (say p). We wish to leverage this information and try to understand if p' would continue to dominate q' without stretching q (p' and q' represent stretched p and q respectively). This is the central idea behind our pruning algorithm and we present the basic concepts in the subsequent sections.

4.8.1 Sub-regions

We saw in section 4.3.1 that in a d dimensional space, the nullification point divides the subspace enclosed by the characteristic vectors into d regions. Each of these d regions is close to an $d-1$ dimensional axis hyperplane on to which the outer hyperplane of the region is stretched and superimposed. We can further divide a region into $d-1$ sub-regions based on the axes that is the closest. Given the $d-1$ axis that makes the axis hyperplane, a given data in a region can be classified into one of the $d-1$ sub-regions based on the axes it is closest too. This classification has important implication during stretching, because when we stretch we can now ascertain the dimension along which the datum's value will increase and the dimensions along which the datum's value would decrease.

Let us assume that a point $p = \{x_1, x_2, ...x_k...x_d\}$ in a d dimensional space is in a sub-region k ($k \leq d$). We know that $\{A_1, A_2...A_k...A_d\}$ are the d axis in the original space. Since p is in the sub-region corresponding to axes A_k, we can now say that when p is stretched to $p' = \{x'_1, x'_2, ...x'_k...x'_d\}$ then $x'_k \geq x_k$ and $\forall i, i \neq k, x'_i \leq x_i$.

4.8.2 Pruning Strategy

Let $p = \{p_1, p_2 ... p_d\}$ and $q = \{q_1, q_2 ... q_d\}$ be two points from the same sub-region in a d dimensional space with $p \succ q$. Let $p' = \{p'_1, p'_2 ... p'_k\}$ and $q' = \{q'_1, q'_2 ... q'_k\}$ denote the p and q after stretching respectively. Now after stretching the dimensions along which values would increase, is same for both p and q. However the amount of increase would be different and in fact the amount would depend on the angle made by the point with the nullification point. Let p_{angle} and q_{angle} denote the angle p and q makes with the nullification point respectively. The point with the greater stretch will be point which makes the larger angle with the nullification point.

There are two possible scenarios where q_{angle} could be either lesser or greater than p_{angle}. Consider the case where q_{angle} is greater than p_{angle}. This scenario is henceforth referred as “Pruning I”. In this case, q is stretched more than p. Since both p and q are from the same sub-region, let the axes along which the stretching happens be k for both the points. Since p dominates q and q is stretched more than p, we can infer

$$p_i \geq q_i \forall 1 \leq i \leq d \tag{4.12}$$

$$q_{angle} > p_{angle}$$

$$\Rightarrow q_l - q'_l > x_l - x'_l \forall 1 \leq l \leq d and l \neq k$$

$$\Rightarrow x'_l > q'_l + x_l - q_l$$

Using 4.12

$$x'_l > y'_l + c \text{where } c > 0 \tag{4.13}$$

4.13 indicates that p' dominates q' in all dimensions but dimension k. Along dimension k, q' experiences greater increase than p'. The maximum value q'_k can possibly achieve is equal to magnitude of q which would happen if q is aligned with the characteristic vector. So if we compare the p'_k with magnitude of q and if p'_k is still greater than q, then we can know for sure that p' will dominate q' even without stretching q.

Consider the case where q_{angle} is lesser than p_{angle}. This scenario henceforth referred as "Pruning II". Now p is stretched more than q along dimension k. Along all other dimensions q will have a smaller decrease than p. So along dimensions k

$$\begin{aligned} & p'_k - p_k > q'_k - q_k \\ & \quad \Rightarrow p'_k > q'_k + p_k - q_k \\ & \text{Using 4.12} \\ & \quad p_k \geq q_k \\ & \quad \Rightarrow p'_k > q'_k + c \text{where } c > 0 && (4.14) \\ & && (4.15) \end{aligned}$$

Using 4.14, we can conclude p' dominates q' along dimension k. However for other dimensions we can't be sure if p' would still dominate q'. q will have least decrease when q is aligned with the nullification point, in which case q_{angle} is 0. In all other cases q will experience a decrease along all dimensions but k. So if p' can dominate q then we can be assured that p' will dominate q'.

Thus, with the knowledge of angle the point makes with the nullification point and the tuple which dominates it in a given sub-region, we can check if the point needs to be stretched or not using few dominance checks. This completes our pruning strategy.

Algorithm 9 Orthonormal Space Reduction

```
function OrthonormalSpaceReduction($\mathcal{D}_k^{pref}$)
    $\mathcal{D}' \leftarrow \{\}$
    $\mathcal{D}'[0] \leftarrow \mathcal{D}_k^{pref}[0]$
    $angle \leftarrow 90°$
    $blade \leftarrow$ WedgeProduct($\mathcal{D}_k^{pref}[0]$, $\mathcal{D}_k^{pref}[1]$)
    $\mathcal{D}'[1] \leftarrow$ Rotate($blade$, $angle$, $\mathcal{D}_k^{pref}[0]$)
    for all $\mathcal{D}_k^{pref}[i]$, $i$ ranging from 2 to $k$ do
        $proj \leftarrow$ Project($\mathcal{D}_k^{pref}[i]$, $blade$)
        $\mathcal{D}'[i] \leftarrow$ Rotate($blade$, $angle$ - GetAngle($proj$, $\mathcal{D}_k^{pref}[i]$), $\mathcal{D}_k^{pref}[i]$)
        $blade \leftarrow$ WedgeProduct($blade$, $\mathcal{D}_k^{pref}[i]$)
    end for
    return $\mathcal{D}'$
end function
```

4.9 Final Stretch Algorithm

In this section we combine the earlier stretch algorithm in section 4.7 and the pruning strategy in section 4.8.2 to present the final stretching algorithm. In our previous section we assumed the number of characteristic vectors are same as the dimensionality of the given space. We now present the method to deal with the more general scenario when the number of characteristic vectors k is less than the dimensionality of the space d.

Consider a d dimensional space with k characteristic vectors. Accord to clifford algebra, the k characteristic vectors together form a subspace of grade k. Our aim is to make this subspace orthonormal and project the points from the original space to this orthonormal subspace. We will also project the k characteristic vectors from the original space to this orthonormal space. Thus we have a new k dimensional orthonormal space with k characteristic vectors which can be stretched to the k axis, to observe the latent skyline. The algorithm for orthonormal reduction of space is given in Algorithm 9. The algorithm starts by making the first characteristic vector as the first axis of the orthonormal sub space. It then forms the plane involving first two characteristic vectors and rotates the first characteristic vector over that plane for an angle of $90°$. That makes the second orthonormal axes. For the other characteristic vectors, we follow the following four steps to include them into the orthonormal sub space.

1. We project the characteristic vector on to the orthonormal subspace built so far, to get its projection.

2. We find the plane containing the characteristic vector and its projection. This is the plane of rotation.

3. We rotate the characteristic vector by $90° -$ *angle it makes with its projection* to make it perpendicular.

4. We include the now orthonormal characteristic vector into the orthonormal subspace.

In section 4.8.2 we said if a point p dominates q, then we can use additional checks to determine if q can be pruned without stretching. To make good use of this we need to know points that dominate a given point in a given sub-region. We use skylines in the given space as the candidates to determine if it would dominate a given point. When we project the points from the given dimensions to the orthonormal sub space, we maintain the skylines in the original space. These skylines are stretched before

Algorithm 10 Latent Skyline Processing

```
Require: $\mathcal{P}$ is the set of all the points in the given space
Require: $\mathcal{S}$ is the set of Skyline points in the given space
  function LSP($\mathcal{D}_k^{pref}$)
      $\mathcal{S}' \leftarrow \{\}$
      $\mathcal{P}' \leftarrow \{\}$
      $SubRegionMap \leftarrow [\,][\,]$
      $tempVector \leftarrow []$
      $\mathcal{D}' \leftarrow$ OrthonormalSpaceReduction($\mathcal{D}_k^{pref}$)
      for all Vector $p$ in $\mathcal{P}$ do
          $p \leftarrow$ Project($p$, $\mathcal{D}'$)
      end for
      for all CharacteristicVector $V_i^{pref}$ in $\mathcal{D}_k^{pref}$ do
          $V_i^{pref} \leftarrow$ Project($V_i^{pref}$, $\mathcal{D}'$)
      end for
      $i \leftarrow 0$
      for all Vector $s$ in $\mathcal{S}$ do
          $\mathcal{S}'[i] \leftarrow$ Stretch(Project($s$, $\mathcal{D}'$))
          $i \leftarrow i + 1$
          AddtoSubRegionMap($\mathcal{S}'[i]$)
      end for
      for all Vector $p$ in $\mathcal{P}$ do
          $s \leftarrow$ GetDomFromSubRegionMap($p$)
          if $s! = NULL$ then
              if $s_{angle} > p_{angle}$ then
                  if IsDominating($s'$, $p$) then
```

```
                continue
            end if
        else
            $tempVector \leftarrow p$
            $tempVector[SUBREGION] \leftarrow$ GetMagnitude($p$)
            if IsDominating($s'$, $tempVector$) then
                continue
            end if
        end if
    end if
    AddVector($\mathcal{P}'$, Stretch($p$))
  end for
  return SFS($\mathcal{P}'$)
end function
```

hand and is used to pruned other points.The algorithm for latent semantic processing is present in *Algorithm* 10.

In *Algorithm* 10 once we get the orthonormal sub space, we project all the data and the characteristic vectors into the subspace. Then we stretch all the skylines in the original space and partition the skylines by sub region. Then we process each point not part of a skyline and check for a skyline from the same sub-region that dominates it. If we find a dominating point, we check if the point can be pruned by the skyline; if not, we stretch the point and later use any known algorithm like Sort-First-Skyline to find the latent skyline among all the stretched points. This same algorithm can also be used to shrink the sub space. When the angle between any two characteristic vectors is more than 90°, we have to shrink/contract the space between them to make the characteristic vectors independent.

4.10 Proofs

4.10.1 Proof I:

Let $\mathcal{D}_k^{pref}$ be a two dimensional sub-space made up of two characteristic vectors V_1^{pref} and V_2^{pref}. These two vectors are not orthogonal and hence the angle θ between them is not 90°. Consider a point P lying on V_1^{pref} which means it makes 0° with V_1^{pref} and $\theta°$ with V_2^{pref}. The projection of P on V_1^{pref} indicated by p'_1 is $|P|$ and the projection of p on V_2^{pref} indicated by p'_2 is $|P|\cos\theta$ and if $\theta < 90°$, we can see p'_2 increases with p'_1. Since there is correlation between the two projections, the $\mathcal{D}'$ formed using this transformation does not preserve the independence of the two vectors.

Chapter 5

EXPERIMENTS

In this section, we evaluate the effectiveness and efficiency of both our preference aware skyline processing techniques.

1. Preferential Skyline Processing

2. Latent Skyline Processing

. The proposed preference aware skyline processing techniques were implemented as a Java application and the experiments were run on data stored locally. We tested our methods using both real and synthetic data. We have described our dataset in 5.1. We have focused on two types of experiments. In the first set of experiments explained in 5.2 and 5.3, we attempt to prove that the results generated by our algorithm do take into account the preferences of the user by comparing the results generated with the known ground truth for that preference. Next we prove the efficiency of our approach in 5.4 and 5.5 with respect to various parameters like dimensionality and cardinality of the data set and dimensionality of the user preferences against synthetic and real dataset. All of the experiments where run as a java application on a quad core Intel i5 processor running Ubuntu with 6 GB of RAM.

5.1 Data Sets

Börzsönyi *et al.* (2001) suggested three different kinds of synthetic databases that differ in the way values are generated. We have adopted those same synthetic data sets for our experiments. The three synthetic data sets are

1. *Independent:* In this type of database, all attribute values are generated independently using a uniform distribution.

2. *Correlated:* In this database, points have high values in all dimensions or they have low values in all dimensions.

3. *Anti-Correlated:* In this database, points having high values in one dimension, are bad in one or all of the other dimensions.

In addition to the synthetic data sets we also used real data set. The main challenge regarding the choice of the real data set was to make sure that the preference aware results generated by the algorithm can be verified against a universally acclaimed system. The two real data sets used in our applications are

1. NBA data set (2012-2013) season [1]

2. DBLP data set [2]

.

5.1.1 NBA Data Set

The NBA data set consists of statistics of 470 players who played the 2012-2013 season. For each player, we recorded 8 attributes namely Field Goals per game, Three pointers per game, Free throws per game, rebounds per game, assists per game, steals per game, blocks per game and points per game. In order to compute personalized skylines, we need the data to construct the user preferences. We inferred the users' preferences from the player statistics of the greatest players (legends) in

[1]The data set was obtained from http://espn.go.com

[2]The DBLP data set was downloaded from dblp website http://dblp.uni-trier.de/xml/

NBA history.[3]. There were 76 players classified as legends by the official website of NBA and the career statistics of all these players were obtained. We normalized the records of current and the legendary players to avoid bias.

5.1.2 DBLP Data Set

The dblp data set consists of bibliographic information on major computer science journals and proceedings. In our experiment we used abstracts of 136422 computer science publications on various domains published in different conferences. The abstracts of all these publications were downloaded and a TF-IDF space was built using Apache Lucene. The dimensionality of the data set was reduced to 8 using Singular Value Decomposition on the data set. The SVD was run using Apache Mahout on a hadoop cluster. Further the conferences were grouped into different domains using information from wikipedia[4]. For example, SIGMOD and VLDB were grouped as "Data Management" conferences. This classification is used to infer latent characteristics of different domains like "Operating Systems" and "Data Management".

5.2 Preferential Skyline Processing - Ground Truth Evaluation

We attempted to find the skyline of best players in NBA for each position for the 2012 - 2013 season and compare our results with other sources like ESPN rankings and HoopsWorld[5] rankings. We computed SVD on the statistics of all the legendary players to play in a position to infer the characteristics that are needed in the statistics to become successful in that particular position. Once we inferred the characteristics

[3]The legends to play the game of NBA was obtained from http://www.nba.com/history/legends-index/index.html

[4]http://en.wikipedia.org/wiki/List_of_computer_science_conferences

[5]Hoopsworld shuts down to became hoopshype. source:"http://thedissnba.com/2014/01/16/hoopsworld-shuts-down-founder-launching-new-venture/"

Position	*Player*	*ESPN Rating*	*Hoops World Rating*
PG	Chris Paul	5	1
SG	James Hardeen	2	2
PF	Kevin Love	5	-[a]
SF	Kevin Durant	2	2
C	Joakim Noah	6	4

Table 5.1: Evaluation Of Preferential Skyline Processing: Comparison Of Our Algorithm With ESPN And Hoopsworld Ratings

[a]Kevin Love was injured the season and didn't play all of the games and so hoops world didn't rank him

as a reduced orthonormal space, we found the skyline of the players in the new space. This skyline can semantically be considered as the skyline of players who has the potential to become legends in the future. Since the reduced space inferred from SVD had significant eigenvalue only along the first singular vector, we identified skylines along one dimension. This is analogous to preferential skyline processing in the following manner. Consider a NBA team manager who wants to recruit a new player to his team for a particular position based on the players statistics in the previous season. He further wants the new player to emulate legendary players like *Michael Jordan* and *Magic Johnson.* The preference aware skylines for his needs can be computed by the aforementioned approach.

We tried to verify the correctness of our result by trying to analyze the performance or ranking of that player in the season. In particular we compared with the ESPN ratings and HoopsWorld ratings for the 2012 - 2013 season.

The results are given in table 5.1. The column Position refers to the position of the player in the field. (PG = Point Guard, SG = Shooting Guard, PF = Power Forward, SF = Small forward and C = Center). Since the player suggested by our algorithm is also agreed upon by other ranking systems, the above experiment indicates that the preference aware skyline indeed captures the user selection criteria.

5.3 Latent Skyline Processing - Ground Truth Evaluation

We prove the correctness of our latent skyline processing through the following strategy. We have a data set with different classes of data whose label is known in advance. For each dataset, we split the data into training and testing data set. We have used 80% of available data to train the dataset. We select few classes from the data set and identify the hidden semantics of those classes by doing a SVD on a training set for each class. We stretch the space between the latent semantics and analyze the pattern of the stretched space.

5.3.1 Experiments

The input consists of research papers from the following domains.

1. Data Management
2. Operating Systems
3. Cryptography
4. Geometric Algorithms
5. Computer Hardware
6. Artificial Intelligence

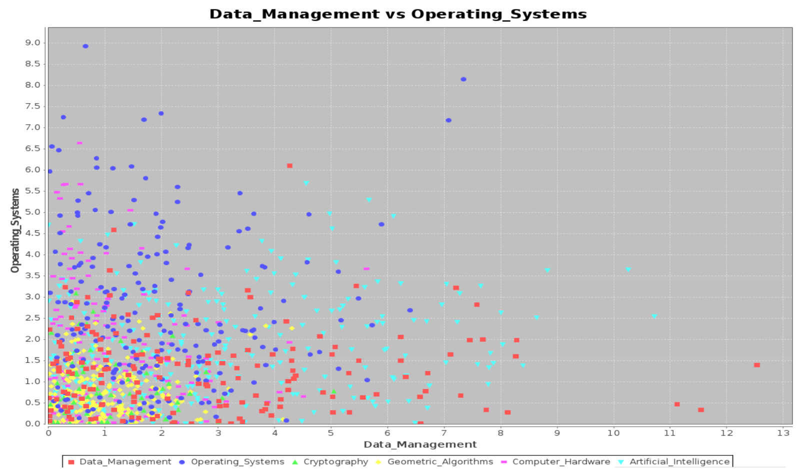
The latent semantics obtained from Data Management and Operating Systems were employed as characteristic vectors. The stretched space is visualized in Figure 5.1a. In the next experiment the latent characteristics related to Data Management and Cryptography were used as characteristic vectors. The stretched space was visualized in Figure 5.1b. Following characteristics can be inferred from the figure.

1. Research papers related to Data Management are aligned closely to the latent concept indicated by Data Management training set.
2. Operating Systems related research papers are aligned closely to the latent concept indicated by Operating Systems training set.
3. Cryptography related research papers are aligned closely to the latent concept indicated by cryptography training set.
4. Artificial Intelligence papers are closely aligned with data management as they have many interdisciplinary topics like TF-IDF, classification and indexing.
5. Operating systems related research papers are closer towards Computer Hardware related research papers.
6. Operating systems related research papers are closer towards Data Management than Cryptography related research papers.

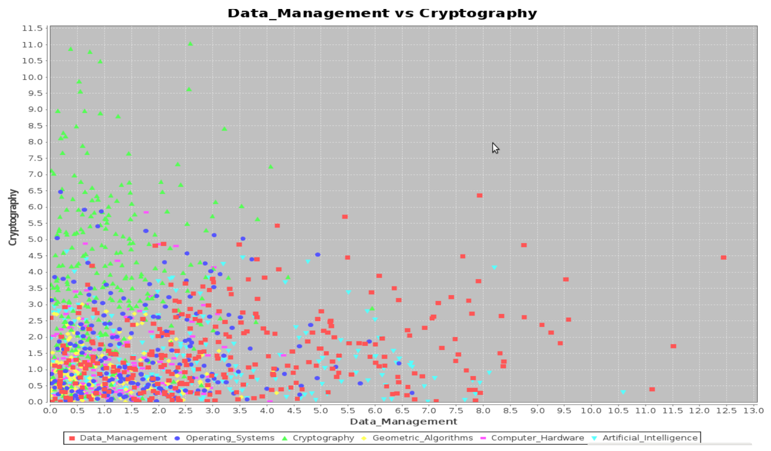
The results produced by the LSP algorithm are in line with the expectations, proving the correctness of the LSP algorithm.

5.4 Efficiency Of Preferential Skyline Processing

In this section, we evaluate the efficiency of the proposed algorithm with respect to the cardinality of the data, dimensionality of the data and dimensionality of the



(a) Database Management And Operating Systems



(b) Database Management And Cryptography

Figure 5.1: Effectiveness Of Latent Skyline Processing - Analyzing The Stretched Space

preferences. We evaluate our algorithm on both synthetic and real data sets. Since there is no existing work that identifies personalized skylines, our baseline method is to transform all points from the domain space to the user preferential space and run SFS on the transformed points. Section 5.4.1 focuses on effect of dimensionality. Section 5.4.2 focuses on effect of cardinality and section 5.4.3 focuses on the effect of the dimensionality of the preferences. In the experiments conducted with synthetic data, the data can be either independent, correlated or anti-correlated. Similarly the data from which the preferences are computed using SVD can either be independent, correlated or anti-correlated. The data from which preferences are inferred for the user is referred as seed. In each experiment, we record

1. the average CPU time for a personalized skyline computation.
2. the percentage of time gained/lost using our algorithm over the naive method.
3. the amount of pruning achieved using our index mechanism.

5.4.1 Effect Of Dimensionality

In order to study the effects of dimensionality, we use the data sets with cardinality 100K and vary the number of dimensions between 2 and 8. In a d dimensional data set, as much as d preferences can be perceived for a user. We have preserved all the d preferences of the user. Section 5.4.3 covers the case where we vary the count of preferences preserved for the user. The graphs indicating the perceived performance gain and pruning is given in Figure 5.2. Since the number of dimensions is increased but not the cardinality, the pruning is expected to drop as the number of dominated points decreases with increase in dimension. In our experiments percentage of points that are pruned is between 50% and 60% in 2 dimensions and drops to a range between 15% to 30% at 8 dimensions and the drop in pruning appears to be linear in time. In

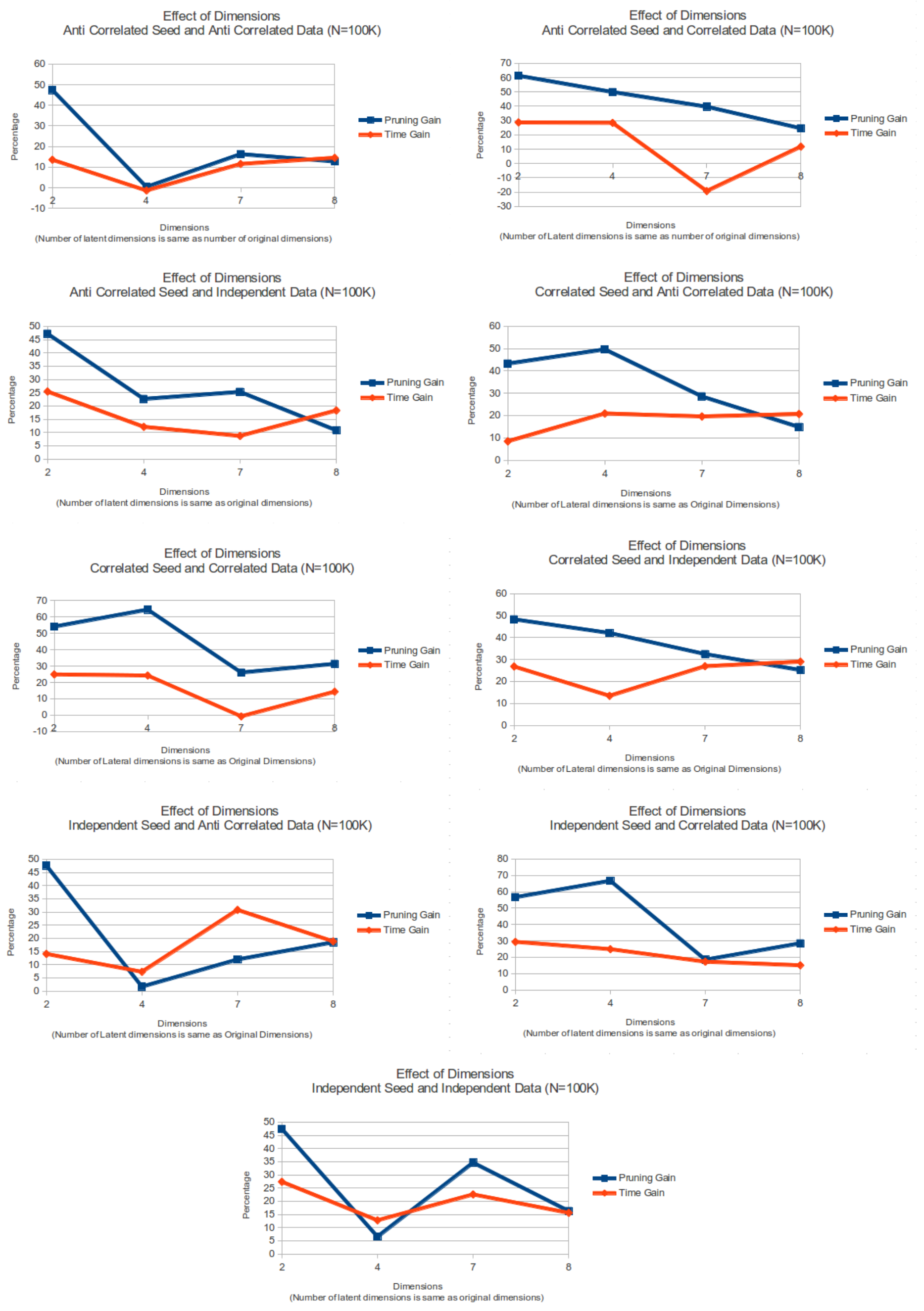


Figure 5.2: Effect Of Dimensionality

high dimensions, even 20% of pruning can result in significant savings as Sort-First-Skyline can become slow in high dimensions. SFS is slow at high dimensions because the worst case complexity of SFS is $\mathcal{O}(nk)$ where n is the number of points and k is the number of skylines. Since number of skylines increases with dimensions, SFS gets slow at high dimensions. Hence marginal reduction in n at high dimensions can produce significant gain. In almost all cases, the percentage of time gained increases with increase in dimensions. In some cases it appears to temporarily drop at low dimensions only to rise at high dimensions.

5.4.2 Effect Of Cardinality

In order to study the effects of cardinality, we use data sets with dimension 4 and vary the cardinality of the data. The different cardinality of the data we used are 25K, 50K, 100K, 500K, and 1M. The graphs indicating the performance gain and percentage of data pruned is given in Figure 5.3. The percentage of data pruned appears to be constant with increase in cardinality for all the experiments. This shows our pruning algorithm is robust and there are no special cases for which it would fail. The percentage of gain in running time is negative at high dimensions as SFS is fast for low cardinality data and the amount of time spent on pruning the data is costlier than SFS at low cardinality. When the cardinality increases, SFS becomes slower and the amount of data pruned starts to positively effect the running time of the PSP algorithm.

5.4.3 Effect Of Preferences dimensionality

As seen in section, we can control the number of user preferences we want to preserve when we transform data to the preferential space. In other words the dimensionality of the preferential space can be controlled. In this section, we used data sets

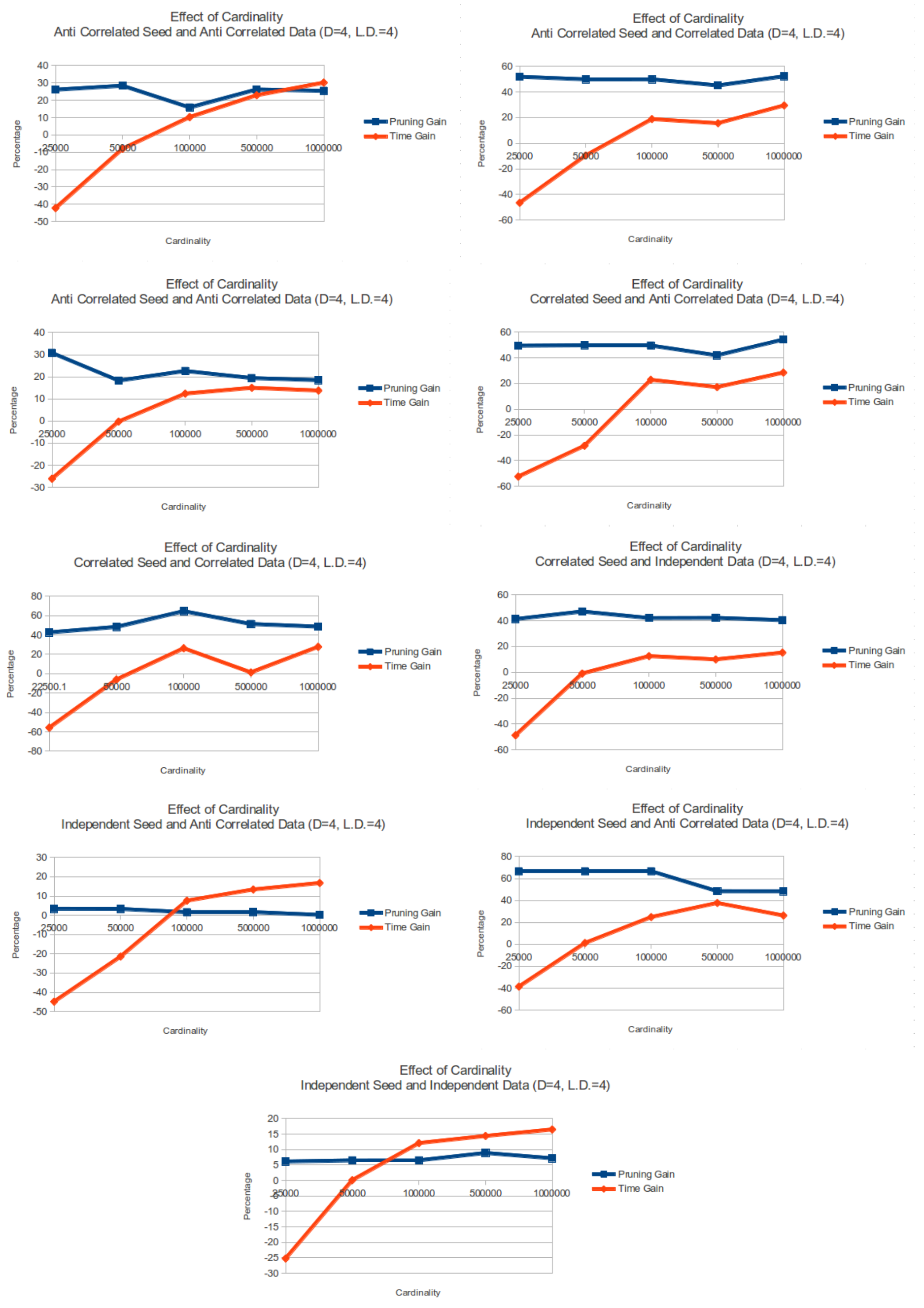


Figure 5.3: Effect Of Cardinality

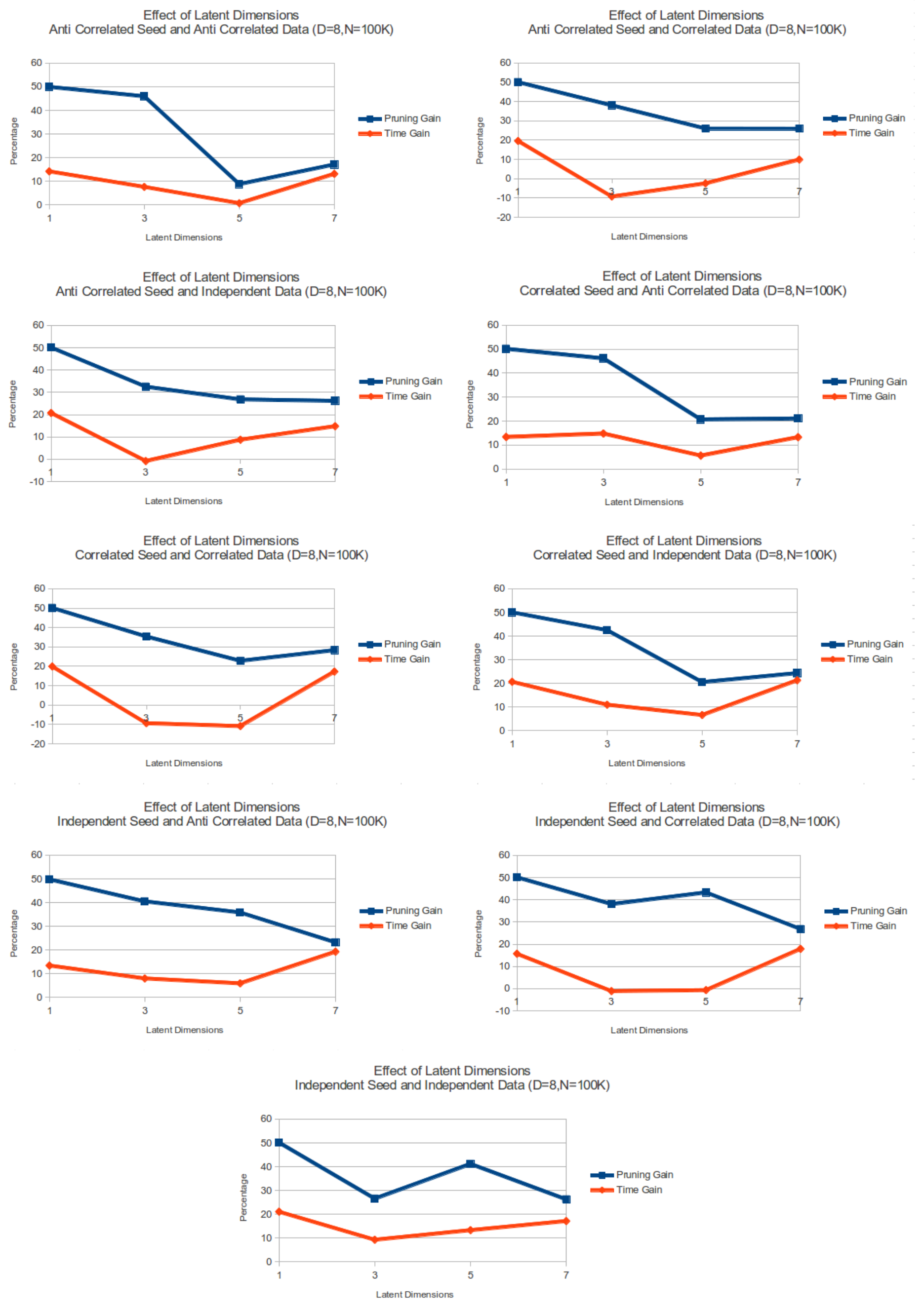


Figure 5.4: Effect Of Latent/Preferences' Dimensionality

with cardinality 100K and dimensionality 8 but we controlled the dimensionality of the preferential space between 1 and 7 and studied the performance of our algorithm. The graphs indicating the performance gain and percentage of data pruned is given in Figure 5.4. The pruning drops when the number of latent dimensions increase for reasons already described in section 5.4.1. When the number of latent dimensions are low, the percentage of time gain is low or negative because SFS is fast in low dimensions and the amount of time we spend on pruning negatively impacts the performance. When the number of latent dimensions is closer to the number of original dimensions then PSP algorithm is about 20% faster than the baseline method in all cases.

5.4.4 Effect of Angle Offset of Preferential Space

In this experiment, the angle that the preferential space makes with the data space is varied and the pruning capabilities and the performance gain of PSP algorithm over baseline method is studied. We used data sets with dimensionality 5 and cardinality 100K. We used preferential spaces of dimensionality 5 as well. The results of the experiment is given in Figure 5.5. In case of correlated data, pruning rate and performance gain is very high when the angle offset is low but both decrease with increase in angle offset. This is expected as correlated data are easier to prune, but as the angle offset increases, the data loses it's correlations and is considerably harder to prune. In case of independent data, angle offset does not influence the pruning rate or performance gain and both stay relatively constant. In case of anti-correlated data, the pruning rate has a gradual increase with angle offset but the performance of the algorithm decreases with increase in angle offset. When the angle offset is low, the data stays anti-correlated even after transformation to the preferential space. Since the number of skylines is high when the data is anti-correlated, the SFS algorithm

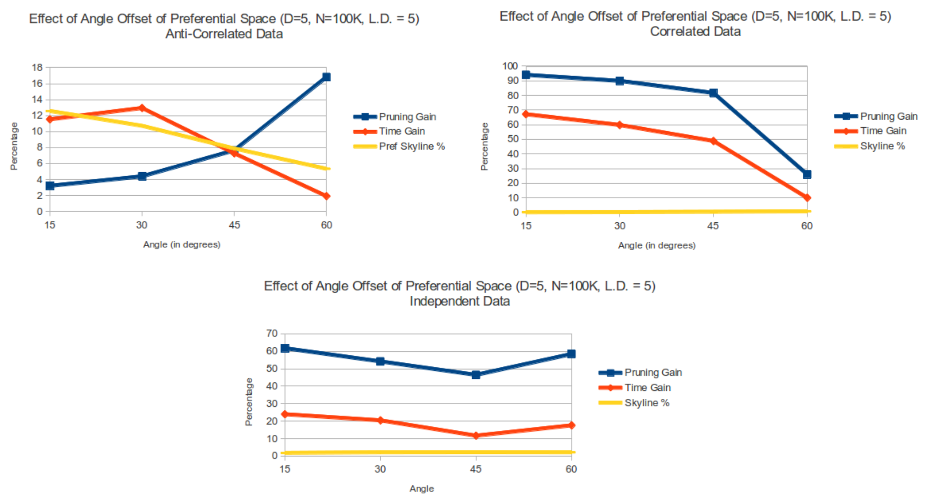


Figure 5.5: Effect Of Angle Offset Of Preferential Space

on complete dataset is very expensive and even a minimal pruning produces significant gains. However when the angle offset is high, the transformed data is not as anti-correlated as before and has lesser number of skylines. In such cases SFS over complete dataset is comparatively faster and the pruning rate of less than 20% does not yield significant gains in performance.

5.5 Efficiency Of Latent Skyline Processing

In this section, we evaluate the efficiency of the proposed algorithm with respect to the cardinality of the data, dimensionality of the data and dimensionality of the latent characteristics. We evaluate our algorithm on both synthetic and real data sets. Since there is no existing work that identifies latent skylines, our baseline method is to stretch all points from the domain space to the latent space and run SFS on the stretched points. Section 5.5.1 focuses on effect of dimensionality. Section 5.5.2 focuses on effect of cardinality and section 5.5.3 focuses on the effect of the dimensionality of the preferences. In the experiments conducted with synthetic data, the

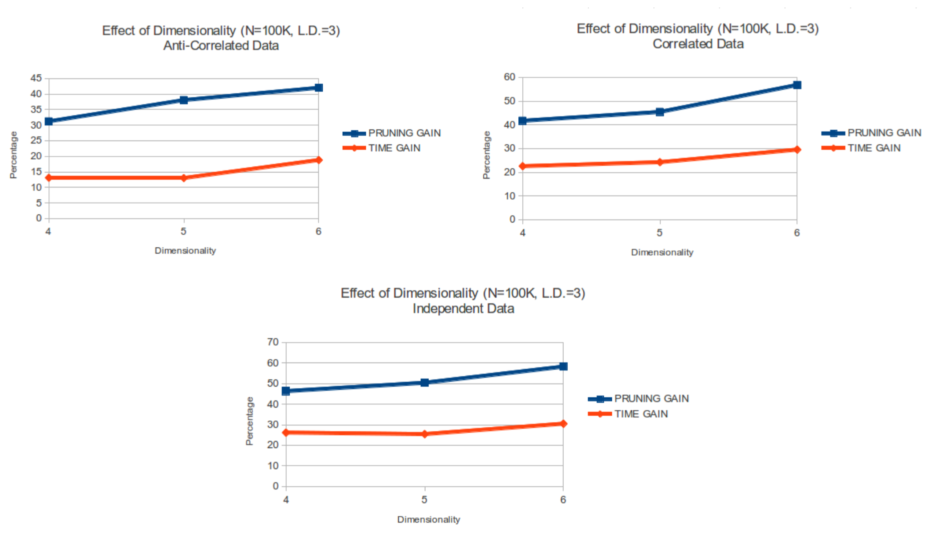


Figure 5.6: Effect Of Dimensionality

data can be either independent, correlated or anti-correlated. In each experiment, we record

1. the average CPU time for a latent skyline computation.

2. the percentage of time gained/lost using our algorithm over the naive method.

3. the amount of pruning achieved using our algorithm.

5.5.1 *Effect Of Dimensionality*

In order to study the effects of dimensionality, we use the data sets with cardinality 100K and vary the number of dimensions between 4 and 6. We maintained the number of latent dimensions as 3. The graphs indicating the performance gain and percentage of data pruned is given in Figure 5.6. In all cases, the pruning rate increases with the dimensionality.

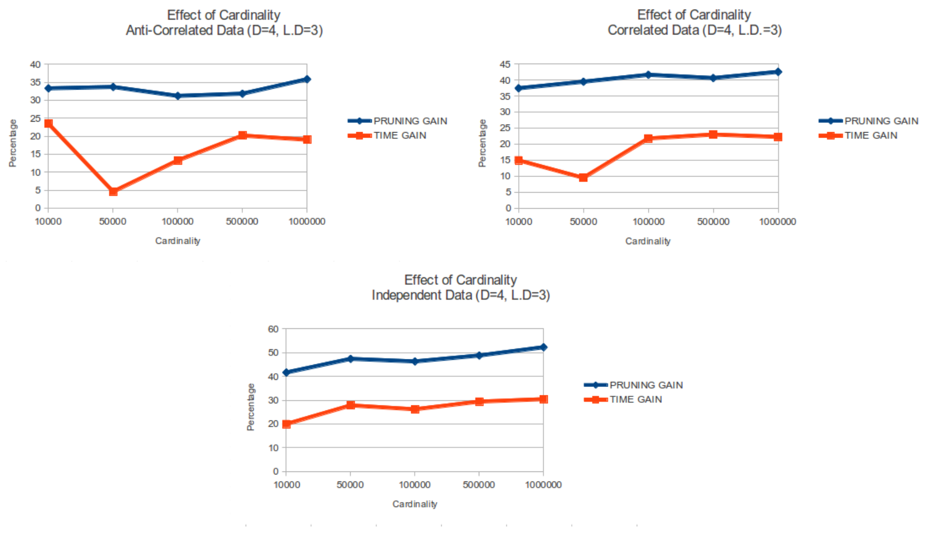


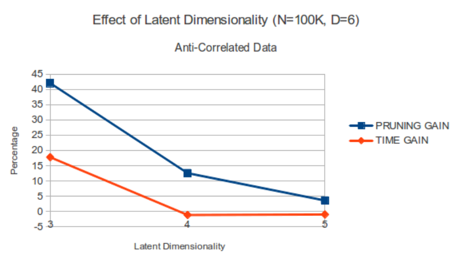
Figure 5.7: Effect Of Cardinality

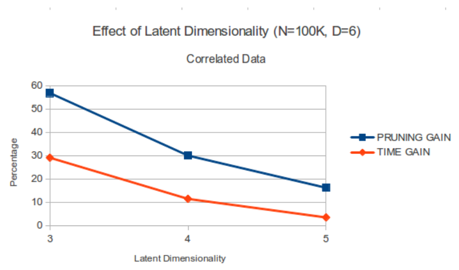
1. In case of correlated and independent data, our algorithm achieves 20% faster running time than the baseline approach for low dimensions and the gain increases up to 30% with increase in dimensionality.

2. In case of anti-correlated data, the algorithm's gain is in between 10% and 20%

5.5.2 *Effect Of Cardinality*

In order to study the effects of cardinality, we use data sets with dimension 4 and vary the cardinality of the data. The different cardinality of the data we used are 10K, 50K, 100K, 500K and 1M. The graphs indicating the performance gain and percentage of data pruned is given in Figure 5.7. In all cases, the percentage of data pruned is fairly constant and is between 35% and 45%. The LSP algorithm performs better than the baseline method in all trials and the percentage of time gained increases from about 15% to 25% with increase in cardinality.





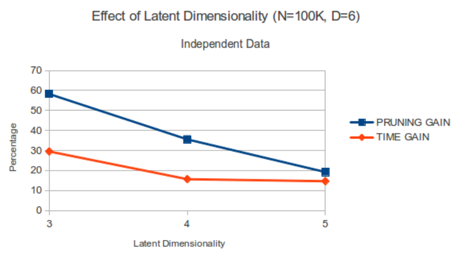


Figure 5.8: Effect Of Latent Dimensionality

5.5.3 Effect Of Preferences dimensionality

The number of latent characteristics on which we determine the skyline becomes the dimensionality of the stretched space and hence its influence in the performance of the algorithm can't be ignored. In order to study the effects of latent dimensionality, we use data sets with dimensionality 6 and cardinality of 100K. The graphs indicating the performance gain and percentage of pruning achieved is given in Figure 5.8. As the number of latent dimensions increase the pruning power and the percentage gain in running time of our algorithm decreases. In case of correlated data, gains in running time decreases from 30% to 5% while in case of independent data, the decrease is from 30% to 15% and in case of anti-correlated data, the decrease is from 20% to no gain. As the number of latent dimensions increase, the amount of data that is pruned by LSP algorithm decrease, resulting in decrease in percentage of time gain.

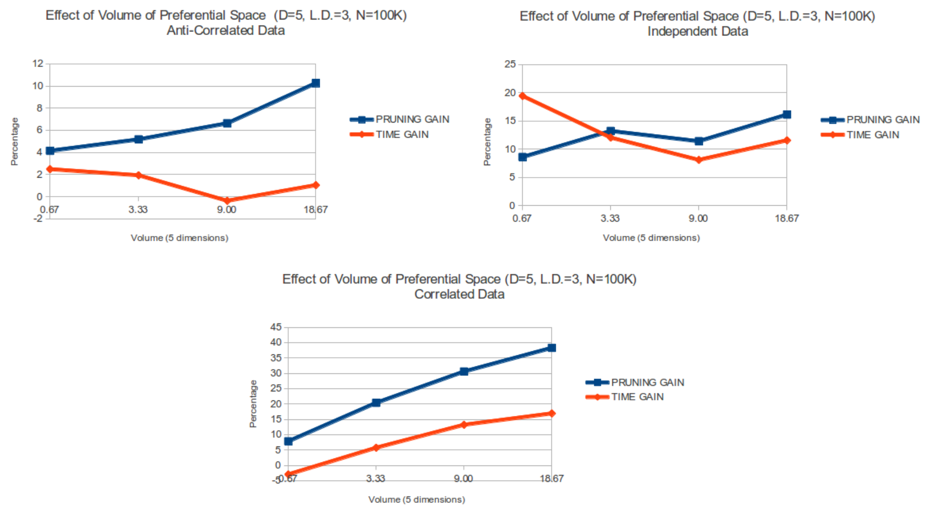


Figure 5.9: Effect Of Preferential Space Volume

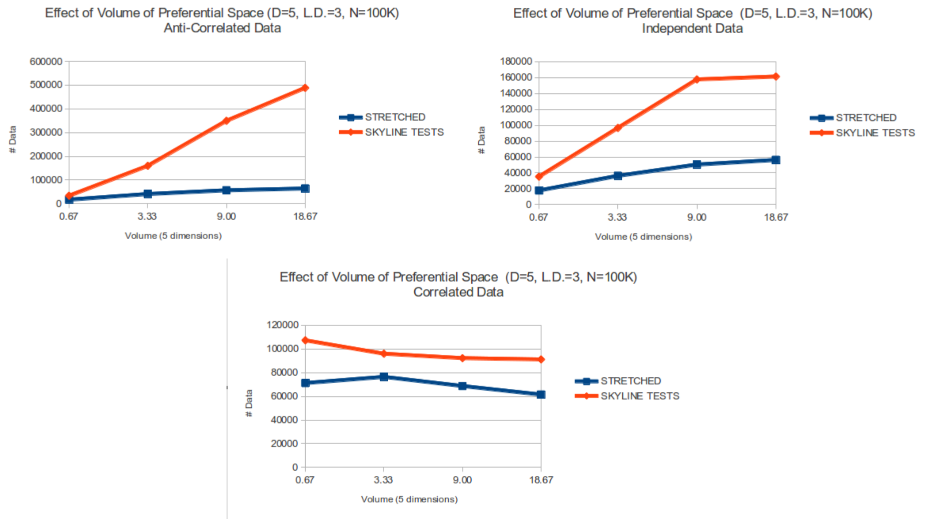


Figure 5.10: Analysis Of Pruning Checks

5.5.4 Effect Of Preferential Space Volume

In this experiment, we control the volume of the preferential space and study the pruning capabilities and performance gain of LSP algorithm over the baseline method. The preferential space can be viewed as a polyhedron connecting a polygonal base to an apex at the origin. Each line that connects the origin to a vertex in the polygonal base is a characteristic vector. Thus the volume of the preferential space can be computed using the formula $(1/3) * b * h$ where b is the base area of the polygon and h is the height of the base from the apex. The results of the experiments is given in Figure 5.9. All the experiments were conducted on data of dimensionality 5 and cardinality 100K. The dimensionality of the latent characteristic space was 3. In case of correlated data, when the volume of the preferential space increases, the pruning rate and the performance gain of LSP algorithm also increases. In case of anti-correlated and independent data, when the volume of the preferential space increases, the pruning rate of LSP algorithm increases but the performance gain of the LSP algorithm decreases. The decrease in performance gain is more pronounced if the data is anti-correlated. This can be attributed to the fact that the number of skylines in the original space is comparatively more when the data is independent or anti-correlated. Since pruning of a point involves finding a skyline point that dominates it, the number of checks needed to prune a data is high when the data is independent or anti-correlated. The number of tests needed to prune a data for different types of data distribution is given in Figure 5.10. When the volume of preferential space increases, the number of tests needed for independent and anti-correlated data is very high and hence the decrease in performance gain.

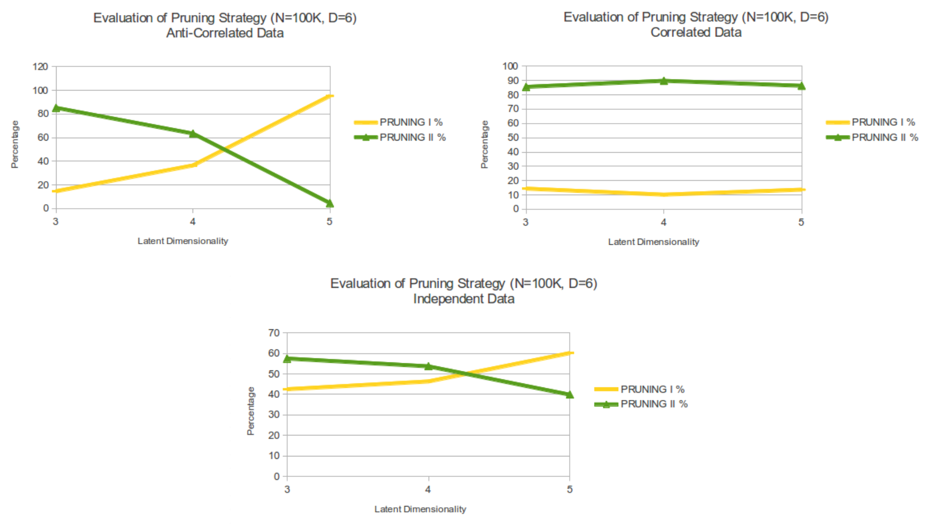


Figure 5.11: Evaluation Of Pruning Strategies

5.5.5 Evaluation of Pruning Strategies

In section 4.8.2 two pruning techniques namely “Pruning I” and “Pruning II” were suggested which would help prune data without stretching them. In this experiment, the latent dimensionality of the space was varied and the contributions of the pruning techniques were observed. Experiments were run on dataset of dimensionality 6 and cardinality 100K. The latent dimensionality of the data was varied from 3 to 5. The results of the experiments is given in Figure 5.11. When both pruning options are possible, the algorithm will try to prune by “Pruning II” before “Pruning I”. The results of the experiments are

1. *Correlated Data:* “Pruning II” is accountable for 80% to 90% of pruning.

2. *Independent Data:* “Pruning II” is dominant when latent dimensionality is low. When latent dimensionality increases, “Pruning I” becomes dominant than “Pruning II”.

3. *Anti-Correlated Data:* "“Pruning II” is dominant when latent dimensionality is low. When latent dimensionality increases, “Pruning I” becomes dominant than “Pruning II”.

Chapter 6

CONCLUSION

In this thesis, we present an efficient way to incorporate preferences into Skyline Query processing systems. This research identifies two core problems users' are faced with when making decisions and proposes algorithms

1. Preferential Skyline Processing

2. Latent Skyline Algorithm

to solve them. Preferential Skyline Processing is used in situations where the users' preference are independent of each other. This research points out that there are situations where the preferences/characteristics may not be independent and proposes Latent Skyline Processing for such situations. This study also introduced a novel transformation technique called stretching to treat preferences independent of each other when dealing with non orthogonal user preferences.

As indicated in the experimental section, the preference framework model introduced in this study was able to prove its claim that it captures the preference of the user and provides preference aware skylines. Through extensive experiments it was shown that the performance of the proposed algorithms significantly outperforms the naive implementations.

REFERENCES

Balke, W.-T., U. Gntzer and J. X. Zheng, "Efficient distributed skylining for web information systems", in "IN EDBT", pp. 256–273 (2004).

Bartolini, I., P. Ciaccia and M. Patella, "Efficient sort-based skyline evaluation", ACM Trans. Database Syst. **33**, 4, 31:1–31:49, URL `http://doi.acm.org/10.1145/1412331.1412343` (2008).

Börzsönyi, S., D. Kossmann and K. Stocker, "The skyline operator", in "Proceedings of the 17th International Conference on Data Engineering", pp. 421–430 (IEEE Computer Society, Washington, DC, USA, 2001), URL `http://dl.acm.org/citation.cfm?id=645484.656550`.

Cacheda, F., V. Carneiro, D. Fernández and V. Formoso, "Comparison of collaborative filtering algorithms: Limitations of current techniques and proposals for scalable, high-performance recommender systems", ACM Trans. Web **5**, 1, 2:1–2:33, URL `http://doi.acm.org/10.1145/1921591.1921593` (2011).

Candan, K. S. and M. L. Sapino, *Data Management for Multimedia Retrieval* (Cambridge University Press, New York, NY, USA, 2010).

Chomicki, J., P. Ciaccia and N. Meneghetti, "Skyline queries, front and back", SIGMOD Rec. **42**, 3, 6–18, URL `http://doi.acm.org/10.1145/2536669.2536671` (2013).

Chomicki, J., P. Godfrey, J. Gryz and D. Liang, "Skyline with presorting.", in "ICDE", edited by U. Dayal, K. Ramamritham and T. M. Vijayaraman, pp. 717–719 (IEEE Computer Society, 2003), URL `http://dblp.uni-trier.de/db/conf/icde/icde2003.html#ChomickiGGL03`.

Deerwester, S., S. T. Dumais, G. W. Furnas, T. K. Landauer and R. Harshman, "Indexing by latent semantic analysis", JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE **41**, 6, 391–407 (1990).

Deng, K., X. Zhou and H. T. Shen, "Multi-source skyline query processing in road networks.", in "ICDE", edited by R. Chirkova, A. Dogac, M. T. zsu and T. K. Sellis, pp. 796–805 (IEEE, 2007), URL `http://dblp.uni-trier.de/db/conf/icde/icde2007.html#DengZS07`.

Denker, J., "How to represent multi-dimensional rotations, including boosts", (2003).

Denker, J., "Introduction to clifford algebra", URL `http://www.av8n.com/physics/clifford-intro.htm` (2006).

Diek, A. and R. Kantowski, "Some clifford algebra history", in "Clifford algebras and spinor structures", pp. 3–12 (Springer, 1995).

Dorst, L. and S. Mann, "Geometric algebra: A computational framework for geometrical applications", (2002).

Ehrgott, M., *Multicriteria Optimization*, vol. 491 of *Lecture Notes in Economics and Mathematical Systems* (Springer Science & Business Media, 2005).

Gargantini, I., "An effective way to represent quadtrees", Commun. ACM **25**, 12, 905–910, URL `http://doi.acm.org/10.1145/358728.358741` (1982).

Godfrey, P., R. Shipley and J. Gryz, "Maximal vector computation in large data sets", in "IN VLDB", pp. 229–240 (2005).

Jameson, A. and B. Smyth, "Recommendation to groups", in "The Adaptive Web: Methods and Strategies of Web Personalization", edited by P. Brusilovsky, A. Kobsa and W. Nejdl, pp. 596–627 (Springer, Berlin, 2007).

Kossmann, D., 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", VLDB '02, pp. 275–286 (VLDB Endowment, 2002), URL `http://dl.acm.org/citation.cfm?id=1287369.1287394`.

Kung, H. T., F. Luccio and F. P. Preparata, "On finding the maxima of a set of vectors", J. ACM **22**, 4, 469–476, URL `http://doi.acm.org/10.1145/321906.321910` (1975).

Lee, K. C. K., B. Zheng, H. Li and W.-C. Lee, "Approaching the skyline in z order", in "Proceedings of the 33rd International Conference on Very Large Data Bases", VLDB '07, pp. 279–290 (VLDB Endowment, 2007), URL `http://dl.acm.org/citation.cfm?id=1325851.1325886`.

Lehar, S., "Clifford algebra: A visual introduction", URL `http://slehar.wordpress.com/2014/03/18/clifford-algebra-a-visual-introduction/` (2014).

Lin, X., Y. Yuan, W. Wang and H. Lu, "Stabbing the sky: Efficient skyline computation over sliding windows", in "Proceedings of the 21st International Conference on Data Engineering", ICDE '05, pp. 502–513 (IEEE Computer Society, Washington, DC, USA, 2005), URL `http://dx.doi.org/10.1109/ICDE.2005.137`.

Liu, B. and C.-Y. Chan, "Zinc: Efficient indexing for skyline computation", Proc. VLDB Endow. **4**, 3, 197–207, URL `http://dx.doi.org/10.14778/1929861.1929866` (2010).

Lofi, C., K. El Maarry and W.-T. Balke, "Skyline queries in crowd-enabled databases", in "Proceedings of the 16th International Conference on Extending Database Technology", EDBT '13, pp. 465–476 (ACM, New York, NY, USA, 2013), URL `http://doi.acm.org/10.1145/2452376.2452431`.

Papadias, D., 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", SIGMOD '03, pp. 467–478 (ACM, New York, NY, USA, 2003), URL `http://doi.acm.org/10.1145/872757.872814`.

Papadias, D., Y. Tao, G. Fu and B. Seeger, "Progressive skyline computation in database systems", ACM Trans. Database Syst. **30**, 1, 41–82, URL `http://doi.acm.org/10.1145/1061318.1061320` (2005).

Pei, J., 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", VLDB '05, pp. 253–264 (VLDB Endowment, 2005), URL `http://dl.acm.org/citation.cfm?id=1083592.1083624`.

Rocha-Junior, J. a. B., A. Vlachou, C. Doulkeridis and K. Nørvåg, "Agids: A grid-based strategy for distributed skyline query processing", in "Proceedings of the 2Nd International Conference on Data Management in Grid and Peer-to-Peer Systems", Globe '09, pp. 12–23 (Springer-Verlag, Berlin, Heidelberg, 2009).

Salton, G. and M. J. McGill, *Introduction to Modern Information Retrieval* (McGraw-Hill, Inc., New York, NY, USA, 1986).

Sarwar, B. M., G. Karypis, J. A. Konstan and J. T. Riedl, "Application of dimensionality reduction in recommender system – a case study", in "IN ACM WEBKDD WORKSHOP", (2000).

Sharifzadeh, M. and C. Shahabi, "The spatial skyline queries", in "Proceedings of the 32Nd International Conference on Very Large Data Bases", VLDB '06, pp. 751–762 (VLDB Endowment, 2006), URL `http://dl.acm.org/citation.cfm?id=1182635.1164192`.

Stojmenovic, I. and M. Miyakawa, "An optimal parallel algorithm for solving the maximal elements problem in the plane.", Parallel Computing **7**, 2, 249–251, URL `http://dblp.uni-trier.de/db/journals/pc/pc7.html#StojmenovicM88` (1988).

Suter, J., "Geometric algebra primer", (2003).

Tan, K.-L., P.-K. Eng and B. C. Ooi, "Efficient progressive skyline computation", in "Proceedings of the 27th International Conference on Very Large Data Bases", VLDB '01, pp. 301–310 (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001), URL `http://dl.acm.org/citation.cfm?id=645927.672217`.

Tao, Y., X. Xiao and J. Pei, "Subsky: Efficient computation of skylines in subspaces.", in "ICDE", edited by L. Liu, A. Reuter, K.-Y. Whang and J. Zhang, p. 65 (IEEE Computer Society, 2006), URL `http://dblp.uni-trier.de/db/conf/icde/icde2006.html#TaoXP06`.

Wu, P., C. Zhang, Y. Feng, B. Y. Zhao, D. Agrawal and A. El Abbadi, "Parallelizing skyline queries for scalable distribution", in "Proceedings of the 10th International Conference on Advances in Database Technology", EDBT'06, pp. 112–130 (Springer-Verlag, Berlin, Heidelberg, 2006), URL `http://dx.doi.org/10.1007/11687238_10`.

Yuan, Y., 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”, VLDB ’05, pp. 241–252 (VLDB Endowment, 2005), URL `http://dl.acm.org/citation.cfm?id=1083592.1083623`.