

Doctor of Philosophy

A Study of Fast Similarity Search Techniques in Metric Spaces

メトリック空間における類似検索の高速化に関する研究



Hisashi Kurasawa

Department of Information and Communication Engineering,
Graduate School of Information Science and Technology,
The University of Tokyo

Supervisor: Prof. Jun Adachi

December 2010

Abstract

The purpose of my research is to reduce the query execution cost of a similarity search in metric spaces. Although a large number of studies have been made on indexing techniques for similarity searches, only a few exploit the data's distribution in a metric space. I developed a new partitioning scheme based on the data distribution that can prune objects more effectively while searching.

Finding similar objects in a large dataset is a fundamental process in various applications, such as record linkage, and the Geographic Information System (GIS). Lowering the query execution cost of a similarity search can speed up these applications. Because metric-space similarity searches can be applied to all types of data whose distance obey metric space postulates such as the triangle inequality, they are very useful for applications that deal with huge amounts of vectors, strings, graphs, sets, and so on.

Similarity search indexes are used for pruning objects dissimilar to a query, and they reduce distance computations and disk accesses. Most indexing schemes use pivots, which are reference objects. They recursively divide a region into subregions by using pivots and construct a tree structure index. They prune subregions while searching by using the triangle inequality. That is, the way of selecting pivots and using them to divide up the space determines the index structure and pruning performance.

The early studies selected pivots by using heuristics. They asserted that good pivots should be outliers of the space, because the distances from such a pivot to each object vary and the pivot easily classifies the objects. They used simple statistical features such as the mean and the variance for selecting pivots. However, their choices ended up being only a little better than random selection. Several selection mechanisms exploit data distributions by using clustering techniques. These methods set the cluster centers as the pivots and divide up the space on the basis of the distances from the pivots. Although they

can effectively classify dense regions, they only work well on particular distribution patterns. A cluster may be separated into multiple regions by a pivot, because the partitioning boundaries are based on the cluster's centers rather than their shapes. My work focused on a problem with the existing indexes wherein they don't consider the partitioning boundary of the pivot and cannot select a good pivot for pruning.

I devised a new pivot selection approach called the "*data distribution approach*", which is based on the data distribution. On the basis of the data distribution approach, I developed two novel methods for pivot partitioning, called Maximal Metric Margin Partitioning (MMMP) and Pivot Capacity Tree (PCTree). These indexing methods successfully reduce the similarity search cost. MMMP first finds the data distribution pattern, especially the boundaries of clusters. Then, it selects a pivot and its partitioning distance on the basis of the shapes of the data clusters. The partitioning boundary is at the maximum distance from the neighbor cluster edges. MMMP deals well with clustered data. On the other hand, PCTree considers the index tree's balance as well as the data distribution. PCTree chooses a pivot on the basis of the balance of the subregions and the estimated effectiveness of the search pruning. As a result, it automatically optimizes the index structure to the data distribution. It also reduces the imbalance of the tree in comparison with MMMP.

The main contribution of this thesis is the data distribution approach for pivot selection. To my best knowledge, this is the first study to exploit the cluster's shapes and tree balancing when making pivot selections.

I conducted several experiments that empirically compared these methods with metric space indexes and they showed the methods to be very efficient. In this thesis, I will introduce the basic techniques of similarity searches and then show my methods and experimental results.

論文概要

本研究の目的は、メトリック空間における類似検索の問い合わせ処理コスト削減である。これまで数多くの類似検索のための索引付け手法が提案されてきたが、いずれの手法も検索対象のオブジェクトの特徴を索引に活かせていなかった。筆者は、検索処理過程における枝刈りを効果的にする、オブジェクトの分布の特徴にもとづいた索引付け処理を研究開発することを目指した。

類似検索は、膨大なオブジェクトの中からクエリに似たものを探す技術である。類似検索は、レコードリンケージや地理情報システムなど、様々なアプリケーションの基盤技術として使われている。メトリック空間（距離空間）における類似検索技術は、三角不等式のような距離の公理を満たすすべてのオブジェクトと距離関数を対象とする。それゆえ、メトリック空間の索引は、ベクトルや文字列、グラフ、集合といったデータを大量に処理するアプリケーションに役立つ技術である。

類似検索索引は、クエリから距離の遠いオブジェクトを枝刈りし、距離計算やディスクアクセスといった検索コストを削減する。ほぼすべての索引付け手法は、Pivotと呼ばれる参照オブジェクトを使っている。索引付け処理では、Pivotで空間を部分空間へ再帰的に分割し、木構造の索引を構築する。この索引は、検索処理中に三角不等式で分割された部分空間を枝刈りするのに使われる。つまり、Pivotの選択手法とPivotを使った空間分割手法は、検索索引の構造と枝刈りの性能を決定すると言える。

先行研究のうち初期の頃のPivot選択手法は経験則にもとづいていた。これらの手法では、Pivotから各オブジェクトまでの距離が分散していればオブジェクトの識別が容易になるとの理由から、オブジェクト空間の端にPivotが位置すると良いと主張していた。そして、平均値や分散値といった単純な統計値をPivot選択に使っていた。しかしながら、このような手法で選ばれたPivotはランダム選択と比べて大きな改善は見られなかった。そこで、クラスタリング技術を使ってデータの分布をPivot選択に活用する手法が提案された。クラスタリング技術を使った手法では、クラスタ

の重心を Pivot として設定し、Pivot からの距離にもとづいて空間を分割している。これらの手法はオブジェクトの密集する空間を効果的に分類できるが、特定のデータの分布にのみ有効である。クラスタの形状ではなくクラスタの重心をもとに分割面を決めているため、クラスタが Pivot によって複数の部分空間に切り離されかねない。筆者は、Pivot によって形成される分割面について考慮が足りないために枝刈りに効果的な Pivot を選べないという、既存手法の問題に注目した。

筆者は“*data distribution-based approach*”と呼ぶデータの分布にもとづいた Pivot 選択の方策を提案した。そしてこの方策にもとづく、2つの新しい Pivot 分割手法、Maximal Metric Margin Partitioning (MMMP) と Pivot Capacity Tree (PCTree) を開発した。MMMP はまず、データの分布傾向のうち特にクラスタの境界を抽出する。そして、クラスタ形状にもとづいて Pivot とその分割距離を決める。MMMP の分割面は、隣り合うクラスタの端からの距離を最大にするように置かれる。MMMP は偏った分布のデータに対して効果的な手法である。一方、PCTree はデータの分布だけでなく、索引木のバランスも考慮した手法である。PCTree では、Pivot によって分割される部分空間のバランスと、Pivot による検索時の枝刈りの効果の、2つを考慮して Pivot を選択する。その結果、PCTree はデータの分布に合わせて索引構造を効果的に変化させている。PCTree は、MMMP の索引木が不均衡になりうる欠点を改善した手法だと言える。これら2つの手法はともに、類似検索の検索コストの削減に役立つ。

本研究の貢献は、Pivot 選択の新しい方策 *data distribution-based approach* を確立したことだ。筆者の知る限りでは、クラスタ形状や索引木のバランスを Pivot 選択に考慮した研究はこれが初めてである。

筆者はこれら2つの提案手法の効果を、既存研究と比較した実験結果を通じて明らかにした。本学位論文では、類似検索の基盤技術について紹介した後に、提案手法とその実験結果について説明する。

Acknowledgement

本研究を行うにあたり、多くの方々にお世話になりました。

指導教員である安達淳教授には、絶え間ない温かいご指導を頂きました。また、様々な知識に触れる機会や、充実した研究環境を与えてくださいました。小さな成果や進捗がでるたびに褒めていただいたり、国際会議でrejectされると14階の学生室までいらして慰めのお言葉をかけていただいたり、常に気遣っていただきました。お陰様で、研究のモチベーションを持ち続けられました。修士課程・博士課程の5年間で安達先生のもとで研究できて、とても良かったです。深く感謝いたします。

高須淳宏教授には、研究における数多くのことをご指導頂き、幅広い知識をご教授頂きました。また、出張のたびにご引率いただき、お陰様で安心して発表することができました。研究について相談事があると高須先生のお部屋に伺って助けていただきました。論文投稿の前には、昼夜を問わず丁寧な添削をしていただきました。ありがとうございます。

深川大路助教には、研究のご指導やコーディングの助言を頂きました。特にコーディングについては、実験評価をするときに非常に助けていただきました。バグの取り方や、人に読んでもらいやすいコードの書き方、高速化のテクニックを教えてくださいました。原稿執筆の際には、評価の表現の書き方などいろいろ丁寧に教えてくださいました。ありがとうございます。

青山友紀教授、森川博之教授には、卒論でのご指導でお世話になりました。卒論のあとも変わらず温かく接してくださいまして、ありがとうございます。毎年MLABフォーラムやOBOG会をととても楽しみにしています。森川先生には博士研究のアドバイザー教員にもなっていただきました。2009年度のMLABフォーラムにて森川先生にいろいろな会社の方をご紹介いただき、就職活動で助かりました。ありがとうございます。

喜連川優教授には、博士研究のアドバイザー教員になっていただきました。いつ

もご多忙の中お時間をさいていただきありがとうございました。提案手法の問題点について非常に鋭いご指摘とともに、温かい励ましのお言葉をいただきました。本博士論文の査読もしていただきます。ありがとうございます。

田浦健次朗准教授には、学部生の頃からいつも温かく接していただき、大変お世話になりました。近山・田浦研究室の飲み会に参加させて頂いたり、個人的にもご飯に誘っていただきました。気兼ねなくお話でき、とても楽しい時間をいつもありがとうございます。修士研究の大規模分散環境での実験や博士研究の索引付け処理の実験で、田浦研の管理する研究環境 InTrigger を使わせていただき、助かりました。ありがとうございます。

浅見徹教授には、本郷でいつも温かく声をかけて頂き、大変お世話になりました。研究室の飲み会にお邪魔させて頂き、ワインなどご馳走になり、とても楽しかったです。浅見・川原研究室の皆様にも大変お世話になりました。

川原圭博講師には、卒論でご指導いただきました。その後も個人的に相談させて頂いたり、バーベキューやお花見などの季節感あるイベントにお誘いいただき、お世話になりました。卒論で研究の楽しさを教えていただけていなかったら、今のようない進路にはきていなかったと思います。遊びに行くたびに温かく接して下さったり、公私ともに多くの助言を頂きました。

猿渡俊介助教には、学部の頃からいろいろとお世話になりました。研究についても私生活の相談も、いつも親身になって接して頂きました。猿さんの常に上を目指す姿勢は刺激になります。ありがとうございます。

卒論のときに所属していた青山・森川研究室の先輩や同期には、博士にいたるまでいろいろとお世話になりました。ありがとうございます。先輩の今泉英明さん、金子晋丈さん、川西直さん、鈴木誠さん、同期の小澤政博さん、天野翔さん、北田亘さん、鈴木理絵子さん、ありがとうございます。

川原さんにお誘いいただいて参加した、照明のワークショップの皆様にもお世話になりました。LIGHTDESIGN INC.の東海林弘靖さん、永島和弘さん、博報堂の田村大さん、宋泰永さん、江口洋平さん、角田仁さん、ありがとうございます。最先端の照明デザインに触れたり、未来の照明について考える機会をもて、普段取り組む研究とは違う世界を体験でき、刺激になりました。

NIIで同じフロアの浅野研究室と山本研究室の皆様には、日々の雑談やスキー旅行など、楽しい時間をありがとうございました。浅野研の森本健一さん、坂巻俊明さん、木村英雄さん、西村康孝さん、山本研の楠戸健一郎さん、榎本尚之さん、顔開さ

ん、巖煉達さん、高橋信行さん、お世話になりました。同じくNIIで同じフロアの総合研究大学院大学の宋剛秀さんにも日頃お世話になりました。

安達研究室の皆様には公私ともに大変お世話になりました。ありがとうございます。事務の久芳藍さん、上野久美子さんには、NIIでの研究生活を支えてくださいました。OBの正田備也助教、Vu Quang Minhさん、上村明さん、相沢純也さん、辻下卓見さん、辰巳正治さん、広畑堅治さん、高橋輝さん、大変お世話になりました。学生のチュイミンさん、木村光樹さん、渡辺健太郎さん、鈴木貴敦さん、那小川さん、ありがとうございます。OGで妻の裕美（旧姓：若木）にも公私ともども助けてもらいました。

最後に、27歳になるまで長い学生生活を支えてくれた家族に感謝を捧げます。

本研究は筆者個人のできたものでなく、学内外を問わず多数の方々のご教示をいただいて成し得たものです。ありがとうございます。

倉沢 央

Contents

1	Introduction	1
1.1	Motivation	2
1.2	Overview of the Thesis	5
2	Related Work	7
2.1	Overview	8
2.2	Metric Spaces and Distance Functions	10
2.3	Search Task	12
2.4	Technical Issues of Similarity Searches	15
2.5	Pruning Techniques using Pivots	16
2.5.1	Pivot Partitioning	16
2.5.2	Pivot Filtering	18
2.6	Similarity Search Indexes	20
2.6.1	Vantage Point Tree (VPT)	22
2.6.2	Spatial Approximation Tree (SAT)	23
2.6.3	List of Clusters (LC)	25
2.6.4	iDistance	27
2.7	Pivot Selection Scheme	29
2.7.1	Simple Statistical Approach	29
2.7.2	Clustering-based Approach	32
2.8	My Pivot Selection Strategy	33
3	MMMP: Margin-based Pivot Selection Scheme	35
3.1	Overview of MMMP and MMMP-Index	36
3.2	MMMP	37
3.2.1	Overview of OPTICS	38
3.2.2	Pivot Selection and Partitioning	41
3.3	MMMP-Index	44
3.3.1	Index Structure	44

3.3.2	Index Construction	44
3.3.3	Range Search	46
3.3.4	Index Construction Cost	46
3.4	Performance Evaluation	50
3.4.1	Outline of Experiments	50
3.4.2	Data Set	50
3.4.3	Synthetic Data	53
3.4.4	Real Data	61
3.5	Summary	63
4	PCTree: Pivot Selection Scheme for Optimizing both Pruning and Balancing	64
4.1	Overview of PCTree	65
4.2	Improvement on the MMMP	66
4.3	PCTree	67
4.3.1	Pivot Capacity	67
4.3.2	Construction	71
4.3.3	Condition for Stopping the Partition	72
4.3.4	Sampling the Pivot Candidates	73
4.3.5	Indexing Cost	74
4.3.6	k-Nearest Neighbor Search and Range Search	75
4.4	Performance Evaluation	81
4.4.1	Outline of Experiments	81
4.4.2	Data Set	81
4.4.3	Synthetic Data	82
4.4.4	Real Data	87
4.5	Summary	95
5	Conclusion	96
	Bibliography	99
	Publication	105

List of Figures

1.1	Similarity Search Index in a Metric Space	3
2.1	Search task	14
2.2	Ball partitioning	17
2.3	Generalized hyper-plane	18
2.4	Pivot filtering	19
2.5	SAT	24
2.6	List of Clusters	25
2.7	iDistance	27
2.8	Pivot Position	30
2.9	Center Object and Corner Object	30
2.10	Partitioning Boundaries	33
3.1	Partitioning by the MMMP	38
3.2	Reachability Distance	39
3.3	Clustering by OPTICS	40
3.4	Pivot Selection in the MMMP	43
3.5	MMMP-Index	45
3.6	Samples for Clustering	53
3.7	Partitioning Performance (8-d, 20 clusters, $\sigma:(0,0.10)$)	54
3.8	Index Performance w.r.t. No. of objects	54
3.9	Index Performance w.r.t. Variance (8-d, 20 clusters and Uniform)	55
3.10	Clustering Result	56
3.11	Index Performance w.r.t. Dimension (20 clusters, $\sigma:(0,0.20)$)	57
3.12	Index Performance w.r.t. Noise (8-d, 20 clusters, $\sigma:(0,0.20)$)	58
3.13	Julia set	59
3.14	Corel Image Features	60
3.15	Photo tag sets, query: “tokyo”	61
3.16	English words	62

4.1	MMMP and PCTree	66
4.2	Labels for measuring the PC	67
4.3	Inside Region and Outside Region	68
4.4	Inside-safe Region, Outside-safe Region, and Boundary Region.	69
4.5	Search Cost w.r.t. Sampling Parameters and Sampling Methods	84
4.6	Index Performance w.r.t. Number of Objects	90
4.7	Index Performance w.r.t. Number of Dimensions	91
4.8	Index Performance w.r.t. the Number of Clusters	92
4.9	Distance densities	93
4.10	Index Performance on Real Datasets	94

List of Tables

2.1	Similarity Search Index	21
3.1	Data Sets	52
4.1	Data Sets	83
4.2	Distance Density Properties of Real Datasets	88

Chapter 1

Introduction

1.1 Motivation

A similarity search is the task of finding objects similar to a query from a large dataset. It is a fundamental process in various applications, such as multimedia information retrieval, record linkage, and the Geographic Information System (GIS). I am interested in the general case where objects and their distance function are in a metric space. This thesis discusses fast similarity search techniques in metric spaces.

Search techniques are useful for dealing with large datasets. There are many search tasks. The most traditional sort is the exact match search. The exact match search is to find objects exactly equal to a query from a large dataset. A binary search algorithm and hash table are well-known exact match search techniques. The similarity search is a more versatile sort of search. It finds not only objects in the dataset equal to the query but also objects that are similar, i.e., close, to it. It can be adapted to all types of data if only their similarities, i.e., distances, are given. Of course, similarity search techniques are more complicated than exact match search techniques. Exact match search techniques cannot be adapted to similarity searches. The criteria for ranking objects depend on similarity search queries. Thus, it is difficult to sort the objects in the dataset beforehand like the binary search algorithm does. Moreover, the hash values of similar objects will be different if one uses a simple hash table. As a result, a hash table is not appropriate either. For these reasons, researchers have worked for several decades to develop faster similarity search techniques.

Similarity searches are used for the following purposes:

- An image retrieval system searches for images similar to a query image. It uses the image features such as shapes, the color histogram, and scenes. For example, face recognition is a popular application [iPh]. It identifies faces of people in a large number of photos and matches faces that are similar to each other or the query image. The image retrieval system is a core technology of image completion [HE07] and 3D reconstruction [ASS⁺09].
- A record linkage system finds entries that refer to the same entry in several databases [HSW07]. The system is designed to compensate for misspelled and incomplete entries. For instance, the pensioner database managed by the Social Insurance Agency of Japan needs to use similarity searches instead of exact match searches because the

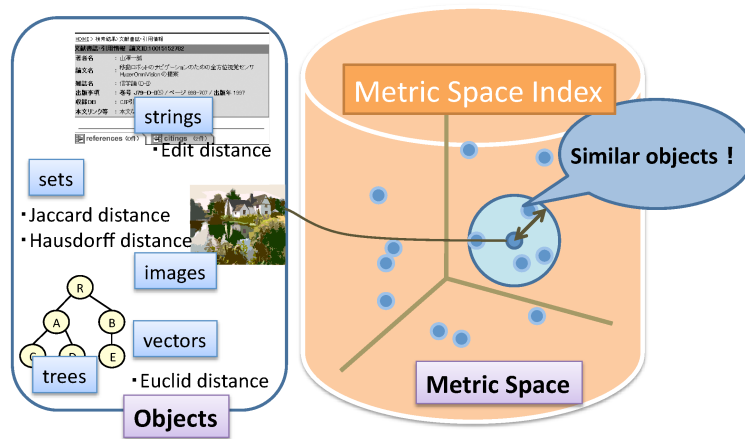


Figure 1.1: Similarity Search Index in a Metric Space

database stores many incomplete entries.

- The Geographic Information System (GIS) manages data linked to locations, and it is the basis of many location-based services. For example, localized search engines use GIS and identify shops near where the user has input a query. In the near future, the growth of sensors and RFID tags may increase the range of purposes for which GIS can be of benefit.

My research focuses on similarity search techniques in a metric space, instead of a specific space such as a vector space. The metric space is an abstraction space for proximity. Figure 1.1 shows the concept of a similarity search index in a metric space. The reasons I selected to study metric spaces are as follows:

- All the fast similarity techniques in a metric space can be used to searching in a more specific space. In contrast, the techniques developed for specific spaces often cannot be extended to the metric spaces or another specific space. For example, although R-tree [Gut84] and its extended indexes [BKSS90; WJ96; KS97; ABHY04] work in a vector space, they cannot be adapted to string searches. Similarly, recent set-similarity join techniques [BMS07; XWL08; XWLS09] are not useful in vector spaces.
- It is difficult to forecast which similarity will be the most useful in the future. I want to develop a novel technique that will be independent of the recent trends in the search field.

The significance of my similarity search techniques is that they make utilizing large amounts of data easier. According to statistics [GCM⁺08; whi10], the amount of information around us is rapidly increasing. For instance, the amount of digital information produced in 2011 is estimated at nearly 1,800 exabytes. We need fast search techniques to handle such large amounts of data. Furthermore, the statistics reveal that a large amount of replicated data exists. In many cases, these replicas are partially copied from original data or are partially edited. Similarity search techniques can deal with such replicas effectively by organizing them based on the similarity. Therefore, I think that the searching on the basis of the similarity of objects will become more important in the future.

With the above-mentioned intentions, I started my study of similarity search techniques three years ago. I still believe that the similarity search is one of momentous advances in computer science and that it will remain a focus of research for years to come.

1.2 Overview of the Thesis

A major part of this thesis tries to develop pivot selection schemes in metric spaces. The purpose is to reduce the execution cost of a similarity search in such spaces. I developed a new partitioning scheme based on the data's distribution that can prune objects while searching more effectively.

My contributions can be summarized as follows:

- I developed a new pivot selection approach called the “*data distribution approach*”. Pivot selection is the key to conducting fast similarity searches.
- On the basis of the data distribution approach, I developed two novel methods for pivot partitioning, called Maximal Metric Margin Partitioning (MMMP) and Pivot Capacity Tree (PCTree).
- I found that the cluster's shapes are useful for selecting a good pivot while developing MMMP.
- I also found that the tree balancing as well as the data distribution should be considered during the study of PCTree.
- I empirically show the efficiency of my methods in experiments by comparing them with several existing indexes using synthetic datasets and real datasets.

The organization of this thesis is as follows. Chapter 2 provides an introduction to related work. A brief introduction to the various similarity search techniques and the existing indexes is provided.

Chapter 3 presents my method for pivot partitioning, called Maximal Metric Margin Partitioning (MMMP). MMMP firstly extracts the data distribution pattern, especially the boundaries of clusters. Then, it selects a pivot and its partitioning distance based on the shapes of the data clusters. The partitioning boundary of the MMMP is at the maximum distance from the neighbor cluster edges. The MMMP can deal with clustered data.

Chapter 4 presents Pivot Capacity Tree (PCTree). PCTree considers the index tree's balance as well as the data distribution. It chooses a pivot on the basis of the balance of the partitioned subregions and the estimated effectiveness of the search pruning. As a result, it automatically optimizes the index structure to the data distribution. PCTree reduces

imbalances in the tree, in comparison with MMMP. These indexing methods reduce the cost of similarity searches.

Chapter 5 contains a summary of the thesis and a discussion of future research directions.

Chapter 2

Related Work

2.1 Overview

Similarity searches have been studied for a few decades. The research field covers distance functions, data structures, indexing techniques, query executions, distributed systems, and so on. The target of my research is indexing techniques. I developed two similarity search indexes. In this chapter, I show the fundamentals and related work on similarity searches.

The existing studies can be divided into two groups: One handles a specific type of objects such as vectors, and the other considers flexible data. I am concerned about the latter and that is why I focused on the metric space approach. The metric space approach takes advantage of a general proximity concept called a “*Metric Space*”. It can handle various objects and distance functions. Moreover, its techniques can be easily adapted to specific spaces. I will briefly describe the metric space concept and the distance functions in Section 2.2.

The similarity search covers several search tasks. Most related studies are on the range search and the k -nearest neighbor search. These search tasks satisfy the need to find objects close to a given query from the dataset. Their applications include, for example, image search engines and spell check systems. By contrast, other studies dealt with search tasks. For instance, similarity join is for finding close object pairs in the dataset. It is used in applications such as record linkage. I will present a review of the search tasks in Section 2.3

The issues of the similarity search are the computational cost and the index size. In Section 2.4, I will clarify these issues and estimate the upper bound of the cost.

After that, I will discuss pruning techniques. Similarity search indexes use the pruning techniques to reduce the number of distance computations. All the indexes use a “*pivot*”, which is a reference object in an index. The pruning techniques using pivots are categorized into pivot partitioning and pivot filtering. Pivot partitioning divides the search space into regions and prunes some of regions during the search. On the other hand, pivot filtering stores distances between a pivot and objects and prunes some of objects during the search. I will show how to prune dissimilar objects by using pivots in Section 2.5

In Section 2.6, I will describe the existing similarity search indexes. Firstly, I will make a comparison of similarity search indexes. The comparison makes clear that the differences among the existing indexes are in the partitioning scheme, index structure, and

pivot selection. After that, I will explain four similarity search indexes: VPT [Yia93], LC [CN05], SAT [Nav02; NR08], and iDistance [JOT⁺03]. I will describe the indexing procedure and the query execution procedure for each index.

In Section 2.7, I will focus on the pivot selection methods. The existing indexes are statistical or clustering-based approaches. The former aims at determining the various distance between the pivot and each object, and the latter aims at choosing pivots based on clusters in the dataset. I will explain their pivot selection procedures.

I show that a pivot and its partitioning distance ought to be selected on the basis of the data distribution. Furthermore, I propose a new approach to pivot selection called the “*data distribution approach*”. I will point out the problem of the existing methods and clarify the position of my research.

2.2 Metric Spaces and Distance Functions

My similarity search index deals with all types of data whose distances obey the metric space postulates. First let us define what a metric space is.

Let $M = (D, d)$ be a Metric space defined for a domain of objects D and a distance function $d : D \times D \mapsto \mathbb{R}$. The following postulates are satisfied in this Metric space [ZADB05].

- $\forall x, y \in D, d(x, y) \geq 0$, (non-negativity)
- $\forall x, y \in D, d(x, y) = d(y, x)$, (symmetry)
- $\forall x, y \in D, x = y \leftrightarrow d(x, y) = 0$, (identity)
- $\forall x, y, z \in D, d(x, z) \leq d(x, y) + d(y, z)$, (triangle inequality)

The distance function d quantifies the similarity between objects in the domain D . There are various distance functions for various data types. Examples of the distance functions are as follows.

The Minkowski distance is defined on n -dimensional vectors as:

$$L_p[(x_1, \dots, x_n), (y_1, \dots, y_n)] = \sqrt[p]{\sum |x_i - y_i|^p}, \quad (2.1)$$

where p is a parameter. This distance has other names depending on the parameter p . That is, the L_1 , L_2 , and L_∞ distances denote the Manhattan distance, the Euclidean distance, and the Chebyshev distance, respectively. The Euclidean distance is used in various applications such as the Geographic Information System (GIS).

The quadratic form distance is defined on correlated n -dimensional vectors as:

$$d_M(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^T \cdot M \cdot (\vec{x} - \vec{y})}, \quad (2.2)$$

where M is an $n \times n$ positive semi-definite matrix. This distance is applied to color histograms of images.

The edit distance is defined on sequences of symbols, such as strings. There are various edit distances. The Levenshtein distance [Lev65] is the most famous of these. For characters c and c' , a string x , and a position i , the following edit operations are used:

- insert c into x at i
- delete c at i from x
- replace c at i in x with c'

The Levenshtein distance between two strings x and y is defined as the minimum number of edit operations needed to transform x into y . This distance is applied to record linkages.

The Jaccard distance is defined on sets as:

$$d(A, B) = 1 - \frac{A \cap B}{A \cup B}. \quad (2.3)$$

It is used in the web mining and recommendation systems.

The Hausdorff distance [HKR93] is defined on sets as:

$$d(A, B) = \max\{\sup_{x \in A} \inf_{y \in B} d(x, y), \sup_{y \in B} \inf_{x \in A} d(x, y)\}, \quad (2.4)$$

where *sup* and *inf* represent the supremum and the infimum, respectively. This distance is used in computer vision.

2.3 Search Task

The similarity search tasks that I will review here are the range query, k -nearest neighbor query, similarity join query, and k -closest pair query. The range query and the k -nearest neighbor query search for objects similar to a query. On the other hand, the similarity join query and the k closest pair query search for object pairs in the dataset.

A range query finds all the objects within a query radius r_q of a query object $q \in D$:

$$R(q, r_q) = \{x \in X | d(q, x) \leq r\}, \quad (2.5)$$

where X is an object set in D . Figure 2.1 (a) shows the idea behind a range query. Notice the query object q does not necessarily exist in the object set X . An example query is to find all stations within 1 kilometer of the university.

The k -nearest neighbor query finds the k closest objects from a query object $q \in D$:

$$kNN(q, k) = A, \quad (2.6)$$

$$A \subseteq X, |A| = k, \quad (2.7)$$

$$\forall x \in A, \forall y \in (X - A), d(q, x) \leq d(q, y). \quad (2.8)$$

Although range queries needs a query radius, the k -nearest neighbor queries simply find the closest objects. k -nearest neighbor queries are useful when we don't have enough knowledge about the object set or its distance distribution. Figure 2.1 (b) shows of the idea behind a 2-nearest neighbor query. An example query is to find the 3 closest stations from the university.

A similarity join query finds all the pairs of objects whose distances are shorter than the query distance r_q :

$$X \subseteq D, Y \subseteq D, r_q \geq 0, \quad (2.9)$$

$$J(X, Y, \mu) = \{(x, y) \in X \times Y : d(x, y) \leq r_q\}, \quad (2.10)$$

where X and Y are object sets. When X and Y are the same, this sort of query is called a “self similarity join” query. Figure 2.1 (c) shows of the idea behind a similarity join query. An example query is to find all the pairs of capital cities and airports that are 20 kilometers apart.

A **k -closest pair query** finds the most similar k pairs of objects:

$$X \subseteq D, Y \subseteq D, k \in R, \quad (2.11)$$

$$kCP(X, Y, k) = A, \quad (2.12)$$

$$A \subseteq X \times Y, |A| = k, \quad (2.13)$$

$$\forall (x, y) \in A, \forall (a, b) \in (X \times Y - A), d(x, y) \leq d(a, b). \quad (2.14)$$

k -closest pair queries resemble similarity join queries. Their difference lies in whether the upper bound distance is certain or not. Thus, k -closest pair queries are a better choice when we have no plan to set the upper bound distance or we need only a fixed number of closest pairs. Figure 2.1 (d) shows the idea behind a 3-closest pair query. An example query is to find the k closest pairs of capital cities and airports.

My research mainly focuses on range and k -nearest neighbor queries.

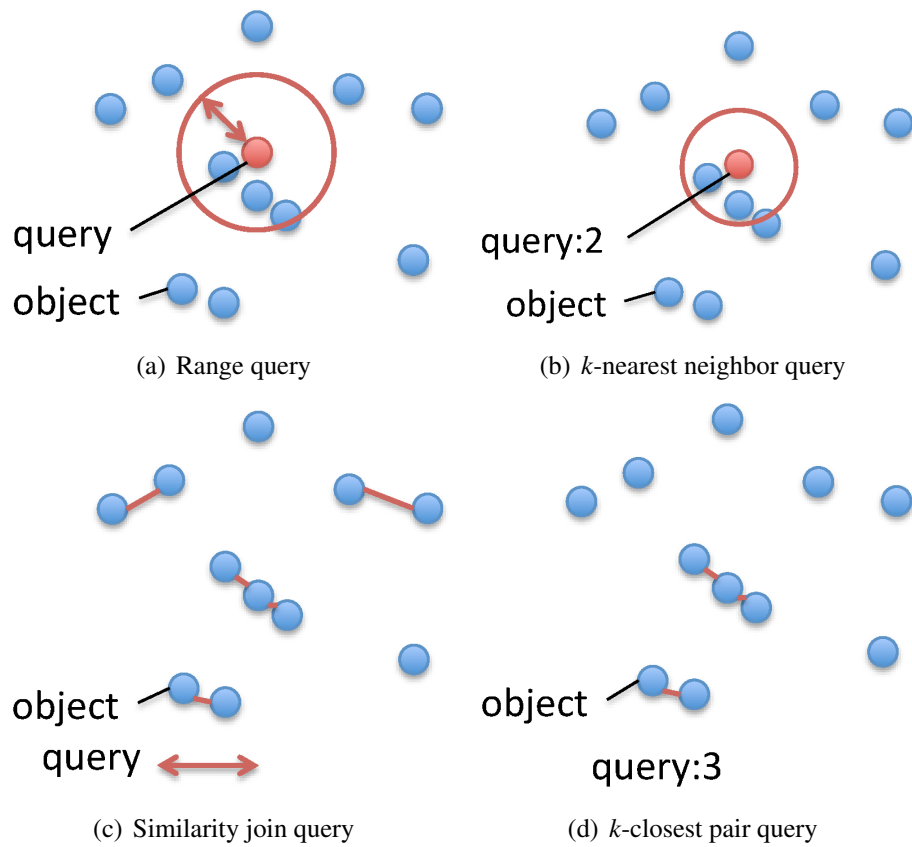


Figure 2.1: Search task

2.4 Technical Issues of Similarity Searches

The technical issues of similarity searches are

- the computational cost, and
- the index size.

The smaller the computational cost is, the shorter the search response time becomes. The smaller the index size is, the smaller the required disk size becomes.

The computational cost consists of the cost of calculating the distance and the number of distance computations. The distance calculation cost depends on the type of object and its distance function, as shown in Section 2.2. For example, the edit distance between strings whose length are m and n requires $O(m \cdot n)$ computations.

Moreover, the number of distance computations depends on the number of objects in the dataset. For instance, the naive range search algorithm, called the linear scan method has the upper bound of the distance computations. It computes all the distance between a query and each object. It requires $O(N)$ distance computations for N objects.

The index size depends on the number of stored distances between objects. The upper bound of the index size is $O(N^2)$ if an index stores all the distances between objects in the dataset

Most similarity search methods aim at reducing the number of distance computations and the index size. The number of distance computations can be reduced by using pre-computed distances and exploiting the triangle inequality. The index size can be reduced by limiting the number of stored distances to $O(N)$. The similarity search methods in metric spaces, however, don't reduce the calculation cost of a distance, because they deal with all types of distances i.e., they don't specify a particular distance function. My research also aims at reducing the number of distance computations and the index size.

2.5 Pruning Techniques using Pivots

Many pruning techniques have been proposed for reducing the number of distance computations. All the proposed techniques use “*Pivot*”, which is a reference object in an index. The pruning techniques are categorized into *Pivot partitioning* and *Pivot filtering*.

2.5.1 Pivot Partitioning

Pivot partitioning divides the search space into regions by using pivots. It prunes some of the regions during the search. It is classified into two types on the basis of the number of required pivots.

Ball partitioning [Uhl91] is a simple partitioning technique, which uses only one pivot and divides a region into two subregions on the basis of the distance from each object to the pivot. For a metric space $M = (D, d)$, suppose a pivot p divides a set S of objects in D into two regions:

$$S_1 = \{o \in S \mid d(o, p) \leq r_p\}, \quad (2.15)$$

$$S_2 = \{o \in S \mid d(o, p) > r_p\}, \quad (2.16)$$

where r_p is the partitioning distance for the pivot p . When searching objects within r_q from a query object q in terms of the metric space M , if the following inequality holds

$$d(q, p) + r_p \leq r_q, \quad (2.17)$$

it is sufficient to check objects in S_1 . Similarly, as shown in Figure 2.2, if the following inequality holds

$$d(q, p) - r_p > r_q, \quad (2.18)$$

it is sufficient to check objects in S_2 . If the two inequalities don't hold, we need to check objects in both S_1 and S_2 . *Multi-way ball partitioning* increases the number of partitions of the Ball partitioning.

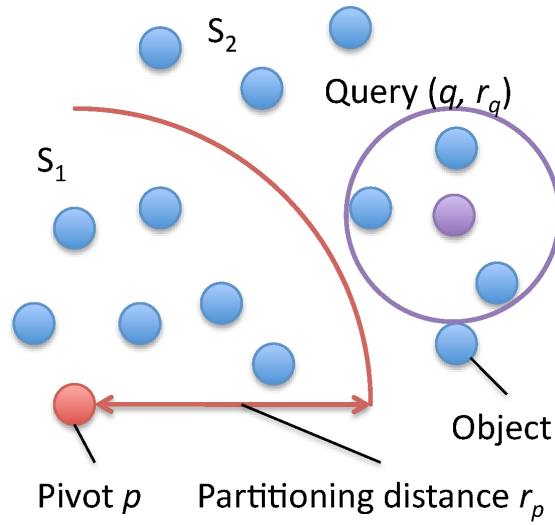


Figure 2.2: Ball partitioning

Generalized hyper-plane partitioning [Uhl91] divides a region into subregions on the basis of the distances to two pivots. For a metric space $M = (D, d)$, suppose two pivots p_1 and p_2 divide a set S of objects in D into two regions:

$$S_1 = \{o \in S \mid d(o, p_1) \leq d(p_2, o)\}, \quad (2.19)$$

$$S_2 = \{o \in S \mid d(o, p_1) > d(p_2, o)\}. \quad (2.20)$$

When searching for objects within r_q from a query object q in a metric space M , if the following inequality holds

$$d(q, p_1) + r_q \leq d(q, p_2) - r_q, \quad (2.21)$$

it is sufficient to check objects in S_1 . Similarly, as shown in Figure 2.2, if the following inequality holds

$$d(q, p_1) + r_q > d(q, p_2) - r_q, \quad (2.22)$$

it is sufficient to check objects in S_2 . If the two inequality don't hold, we need to check objects in both S_1 and S_2 . When the number of pivots exceeds 2, the partitioning is called *Voronoi partitioning*.

Ball partitioning and the multi-way ball partitioning use one pivot and its partitioning

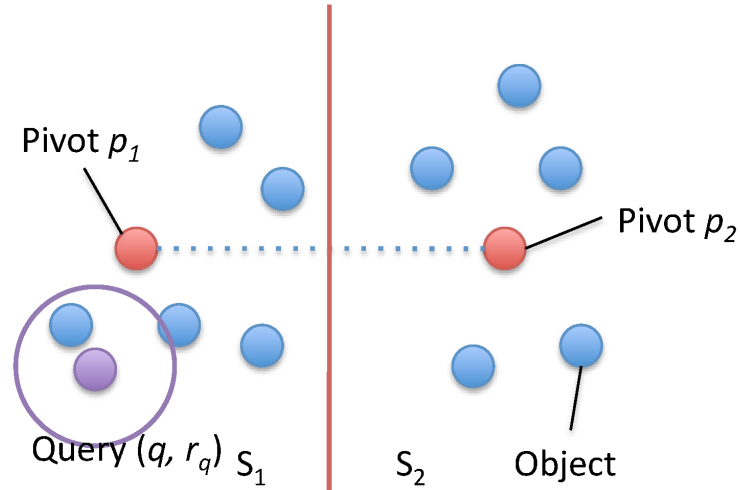


Figure 2.3: Generalized hyper-plane

distance, whereas the generalized hyper-plane partitioning uses two pivots. Compared with ball partitioning, generalized hyper-plane partitioning makes it easier to divide the space because it doesn't need to decide a partitioning distance. However, ball partitioning is better it comes to pruning objects, because it prunes a larger space than the generalized hyper-plane partitioning does.

2.5.2 Pivot Filtering

Pivot filtering stores pre-computed distances between a pivot and objects. It filters out some of these objects during the search.

Like pivot partitioning, it also uses only the triangle inequality. For an object o , a pivot p , a query q , the distance $d(q, o)$ can be constrained:

$$|d(q, p) - d(p, o)| \leq d(q, o) \leq d(q, p) + d(p, o) . \quad (2.23)$$

Figure 2.4 shows the constraint. Pivot filtering stores distances between a pivot and each object and discards dissimilar objects during the search.

Compared with pivot partitioning, pivot filtering needs larger disk space. An index using the pivot partitioning keeps only the pivot and its partitioning distance, whereas an index using pivot filtering stores $O(N)$ distances, where N is the number of objects. Pivot filtering is better than pivot partitioning in terms of pruning. That is, pivot filtering discards

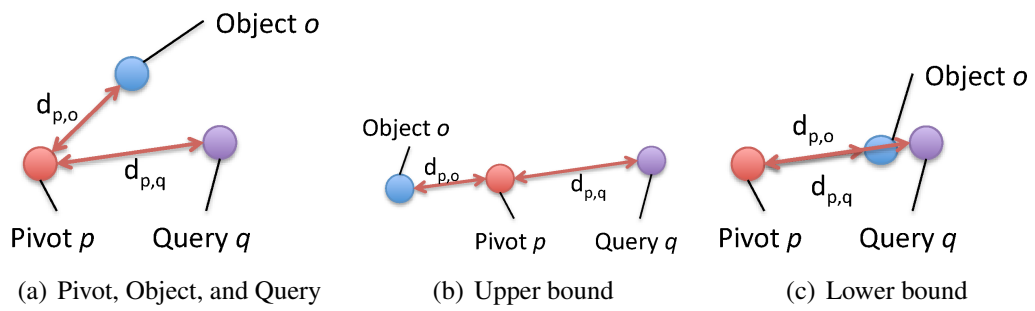


Figure 2.4: Pivot filtering

more dissimilar objects. Thus, many similarity search indexes start by dividing the dataset into regions with pivot partitioning and then use pivot filtering on each region.

2.6 Similarity Search Indexes

Similarity search indexes have been studied over 30 years. The Burkhard-Keller Tree (BKT) is one of the early indexes, developed in 1973 [BK73]. It recursively groups objects by distance and creates a tree index. Its tree has many child nodes. Unfortunately, BKT is not practical for this reason. Pruning techniques (i.e., those in Section 2.5) have been a topic of many subsequent studies [BYCMW94; CMN01; KM83; Vid94; MOV94]. Table 2.1 compares similarity search indexes. I selected these indexes because they are cited in many papers. The main differences among them have to do with their partitioning scheme, index structure, and pivot selection.

All the indexes divide the space with pivots and use ball partitioning, multi-way ball partitioning, generalized hyper-plane partitioning, or Voronoi partitioning. They prune dissimilar objects by using the triangle inequality and placing pivots during the search. Recent studies have relied on these existing partitioning schemes.

The early indexes mainly have a tree structure. Since then structures such as graphs, hashes, and lists have been proposed. Their differences are in the number of nodes at a branch and the balance of the index. A tree has two or three nodes at each branch and is balanced. A graph often has more than 3 nodes at a branch and its tree depth is shallow. A hash has an ordinary tree and an exclusive node. A list has two nodes at each branch and is imbalanced.

The pivot selection schemes of these indexes differ from each other. This means that the pivot selection scheme is the most important feature of an index and determines its performance. Researchers often discuss them and develop new ones. I will review the pivot selection schemes in Section 2.7.

The search performance of an index depends on the data distribution. Thus, performance estimations are not helpful for a user. For example, the vantage point tree (VPT) [Yia93] is good for clustered data, but poor for uniform data. The spatial approximation tree (SAT) [Nav02; NR08] and list of clusters (LC) [CN05] handle various data distributions at the expense of a high search cost for clustered data.

In this section, I will describe these indexes along with iDistance [JOT⁺03] in detail.

Table 2.1: Similarity Search Index

	Paper	Partitioning Scheme	Index Structure	Indexing Cost	Index Size	Search Cost	Pivot Selection
GHT	[Uhl91]	Generalized hyper-plane	Tree	$O(N \cdot \log N)$	$O(N)$	$O(\log N)$	-
VPT	[Yia93]	Ball partitioning	Tree	$O(N \cdot \log N)$	$O(N)$	$O(\log N)$	Maximum Variance
M-tree	[CPZ97]	Generalized hyper-plane	Tree	$O(n \cdot m^2 \cdot \log_m N)$	$O(N + m \cdot m')$	$O(\log N)$	Nearest object
SAT	[Nav02; NR08]	Voronoi partitioning	Graph	$O(\frac{N \log N}{\log \log N})$	$O(N)$	$O(N^{1-O(\frac{1}{\log \log N})})$	-
D-index	[DGSZ03]	Multi-way Ball partitioning	Hash	$O(m \cdot N)$	$O(N)$	$O(m)$	Mean
LC	[CN05]	Ball partitioning	List	$O(\frac{N^2}{m})$	$O(N)$	$O(N)$	Farthest object
iDistance	[JOT ⁺ 03]	Voronoi partitioning	B ⁺ -tree	$O(k \cdot N)$ for Euclid space	$O(N)$	$O(m)$	k-means, BATCH

2.6.1 Vantage Point Tree (VPT)

Vantage point tree (VPT) is a completely balanced tree structure that uses ball partitioning [Yia93].

For an object set S , the VPT algorithm constructs an index structure as follows:

1. If S is empty, finish partitioning.
2. Otherwise,
 - (a) Make a node.
 - (b) Select a pivot p .
 - (c) Set the partitioning distance r_p as the median of the distances between the pivot and objects.
 - (d) Divide S into S_{left} and S_{right} as follows.
 - $S_{\text{left}} = \{o \in S \mid d(o, p) \leq r_p\}$,
 - $S_{\text{right}} = \{o \in S \mid d(o, p) > r_p\}$.
 - (e) For S_{left} and S_{right} , create new nodes and repeat the partition.

The partitioning distance is set so as to equally partitioning the objects in the region. Thus, the two subregions contain the same number of objects. The distance selection results in a balanced binary tree. The height of the tree is $O(\log N)$ where N is the number of objects.

The pivot selection scheme of the VPT is based on a simple statistical feature. The VPT selects as a pivot the object which maximizes the variance of distances among objects. This is because the distances between the pivot and objects are various and the pivot can prune more objects.

A range query of a query object q and a query radius r_q is computed as follows:

1. Start from the root node in the index tree.
2. If the node has no child node, add the objects whose distance from the query q is within r_q to the result set.
3. Read the pivot p and its partitioning distance r_p assigned to the node.

4. Compute the distance from p to q and set it as $d_{p,q}$.
5. If the inequality $d_{p,q} - r_q \leq r_p$ is satisfied, access the child node S_{left} as in step 2.
6. If the inequality $d_{p,q} + r_q > r_p$ is satisfied, access the child node S_{right} as in step 2.

The number of distance computations is $O(\log N)$ when the pruning by the pivots works well. When the pivots often miss pruning dissimilar objects, the algorithm has to traverse many nodes in the tree, and the distance computations increase.

Many variations of the VPT exist. The multi-way vantage point tree (MVP-tree) uses multi-way ball partitioning [BO97]. The excluded middle vantage point forest excludes objects at middle distances by using the multi-way ball partitioning and constructs another index for these excluded objects [Yia99].

2.6.2 Spatial Approximation Tree (SAT)

The spatial approximation tree (SAT) is a graph structure index [Nav02]. Its structure represents Voronoi partitioning of the space. A node in the index is related to a region in the space.

For an object set S , the SAT constructs an index structure as follows:

1. Set the neighbor object set $N(a)$ of an object a to be empty.
2. Set the covering radius set $R(a)$ of a to be 0.
3. For an object $v \in S$:
 - (a) Update $R(a)$ to be $d(v, a)$ if $d(v, a)$ is more than $R(a)$.
 - (b) If the inequality $d(v, a) < d(v, b)$ is satisfied for each object b in $N(a)$, add v to $N(a)$.
 - (c) Make a node $S(b)$ for each object b in $N(a)$.
 - (d) For an object v in S exclusive of $N(a)$:
 - i. Let $c \in N(a)$ be the one minimizing $d(v, c)$.
 - ii. Add $S(c)$ to v .
 - (e) Partition the object set $S(b)$ for each object $b \in N(a)$ as in step 1.

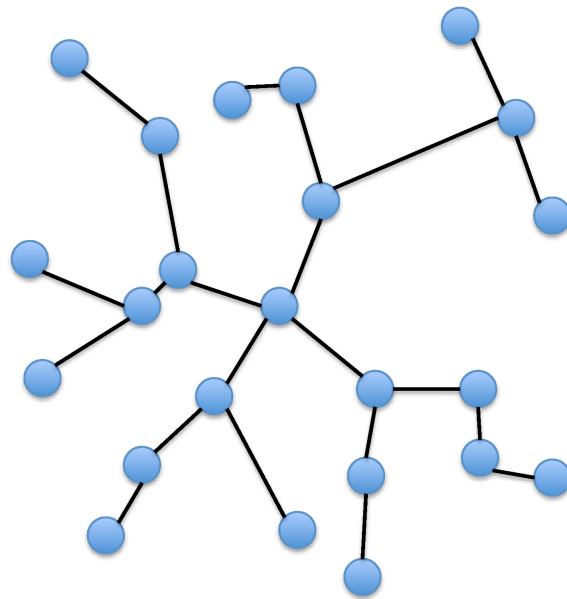


Figure 2.5: SAT

Figure 2.5 shows the index structure of SAT.

A range query of a query object q and a query radius r_q is computed as follows:

1. Start from the root node in the index graph.
2. For the object a in a node $S(a)$ and a distance m , if $d(a, q) \leq R(a) + r_q$ holds,
 - (a) If $d(a, q) \leq r_q$ holds, add a to the result set.
 - (b) Set m to be $\min\{m\} \cup \{d(q, c), c \in N(a)\}$.
 - (c) For each object $b \in N(a)$
 - i. If $d(b, q) \leq m + 2 \cdot r_q$ is satisfied, access the child node $S(b)$ as step 2.

The SAT needs no parameter. It gets rid of the difficulty of tuning parameters and it has a practical advantage. It results in a graph structure. The number of nodes in the graph and the graph's shape depend on the root object that is selected at start of indexing.

The SAT is designed for static datasets. Moreover, a new version for dynamic datasets was recently proposed [NR08].

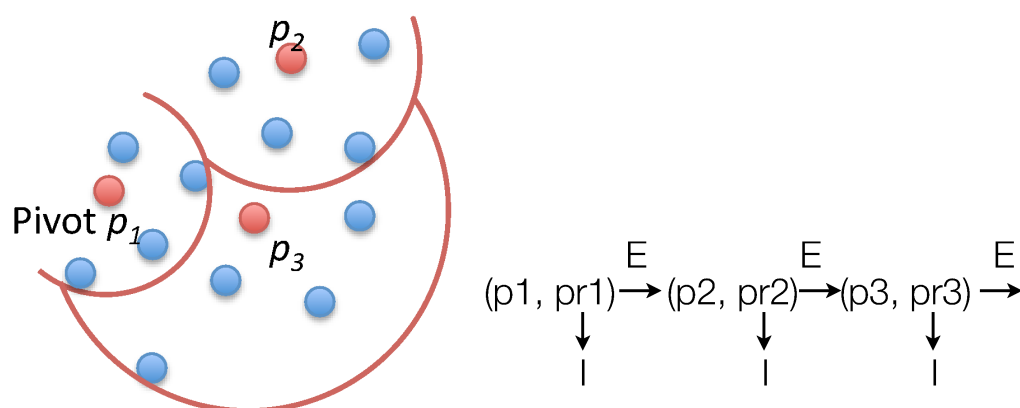


Figure 2.6: List of Clusters

2.6.3 List of Clusters (LC)

List of Clusters (LC) consists of many compact regions [CN05]. It divides a region into a small subregion and an external subregion and repeatedly partitions the external one. The index is a list.

For an object set S , the LC constructs an index structure as follows:

1. If S is empty, return an empty list.
2. Make a node.
3. Select a pivot p from S .
4. Select a radius r_p .
5. Divide the object set S into I and E as follows.
 - $I = \{o \in S - \{p\}, d(p, o) \leq r_p\}$,
 - $E = S - I$.
6. Partition the object set E as in step 1.

Figure 2.6 shows the indexing of LC.

A range query of a query object q and a query radius r_q is computed as follows:

1. Start from the root node in the index list.

2. If the object set in the node is empty, return.
3. If $d(p, q) \leq r_q$ holds, add p to the result set.
4. If $d(p, q) \leq r_p + r_q$ holds,
 - (a) Access I
 - (b) Add the objects whose distance from the query q is within r_q to the result set.
5. If $d(p, q) > r_p - r_q$ holds, access E as in step 2.

The indexing procedure is very simple; thus, its pivot and partitioning distance selection are important factors. The LC has five heuristic schemes for selecting the i th pivot, as follows:

- At random.
- The object closest to p_{i-1} in the remaining set.
- The object farthest from p_{i-1} in the remaining set.
- The object minimizing the sum of distances to previous pivots.
- The object maximizing the sum of distances to previous pivots.

The LC also has suggested two radius selection schemes as follows:

- A fixed radius p_r for all the partitions.
- A fixed number m of objects inside each region and define the radius accordingly.

The experimental evaluation in [CN05] concluded that the pivot selection scheme based on maximizing the sum of distances to previous pivots is better and the radius selection based on a fixed number of objects is better.

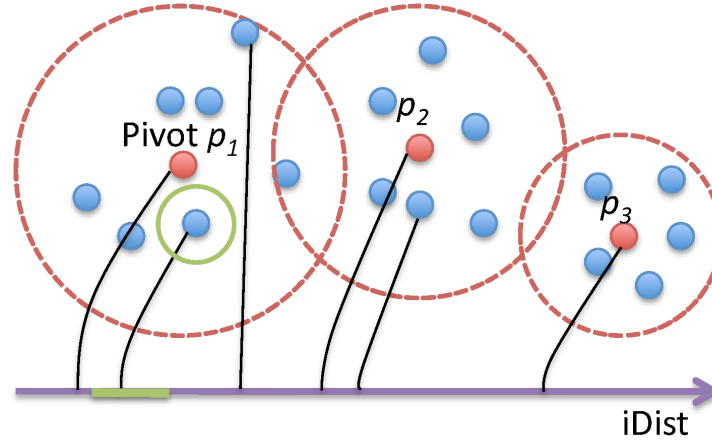


Figure 2.7: iDistance

2.6.4 iDistance

iDistance is designed to manage objects in a metric space with a B⁺-tree [JOT⁺03]. It transforms each object into a 1-dimensional value.

For an object set S , iDistance constructs an index structure as follows:

1. Split the search space into k regions by using k-means clustering.
2. Identify the cluster centers as pivots.
3. Represent an object o in the region R_i ($0 \leq i \leq k$) in a 1-dimensional value as $\text{iDist}(o) = i \cdot c + d(p_i, o)$ where p_i is the pivot of the region R_i and c is constant used to stretch the data ranges.
4. Store the object o in the B⁺-tree with its index key $\text{iDist}(o)$.

Figure 2.7 shows the mapping of iDistance.

A range query of a query object q and a query radius r_q is computed as follows:

1. For each region R_i and its pivot p_i in the regions:
 - (a) Calculate $d_{\min} = i \cdot c + \max \{0, d(p, q) - r_q\}$.
 - (b) Calculate $d_{\max} = i \cdot c + d(p, q) + r_q$.
 - (c) Access the objects whose keys are between d_{\min} and d_{\max} .
 - (d) Add the objects whose distance from the query q is within r_q to the result set.

iDistance divides the search space into the regions by using pivots representing the data clusters. This helps to reduce the number of regions accessed during searching. A recent variant not only improves indexing performance [ZZL⁺08], but also can deal with parallel-distributed queries [NZ06; BNFZ06; DVKV07].

2.7 Pivot Selection Scheme

Most indexes use pivots for pruning dissimilar objects and use the triangle inequality for doing so. The distance density with respect to a pivot influences the pruning performance. For example, if the distances from a pivot to objects are almost equal, pivot partitioning does not work. Many researchers have studied how to choose a good pivot. The proposed selection schemes can be classified into statistical ones and clustering ones.

2.7.1 Simple Statistical Approach

The simple statistical approach aims at determining the various distances between the pivot and each object. This approach uses simple statistics such as the variance [Yia93; VLKJ06], the mean [BNC03; DGSZ03], the sum [CPZ97], and the ratio [PB07]. Several researchers assert that a good pivot should be an outlier [Yia93; BO99]. Another asserts that good pivots are far away from each other [Bri95]. Below, I will introduce the schemes from the earlier research.

VPT [Yia93] selects an outlier object as a pivot. The author of the VPT described an example of it by using a unit square with a uniform probability distribution. He considered pivots at three positions: the square's midpoint p_m , a corner p_c , and the midpoint of an edge p_e , as shown in Figure 2.8. Each pivot partitioning distance is set to be the median of the distances between the pivot and objects. He analyzed the pivots in terms of the boundary length b and the partitioning distance r . The boundary length is proportional in the limit to the probability that no pruning takes place. A longer partitioning distance leads to a lower distance density. He concluded the pivot at square's midpoint is the worst and the pivot at a corner is the best. He also proposed to use the variance for selecting a pivot.

MVP-tree [BO99] improves on the selection scheme of the VPT. The authors of MVP-tree pointed out that a corner object cannot be detected if we don't have the geometry of the data space and the distribution of objects in the data space. They suggested that most of the objects are far away from the corner object. Figure 2.9 shows the distance densities of the center object and the corner object. Hence, they proposed the following heuristics.

1. Choose a random object.

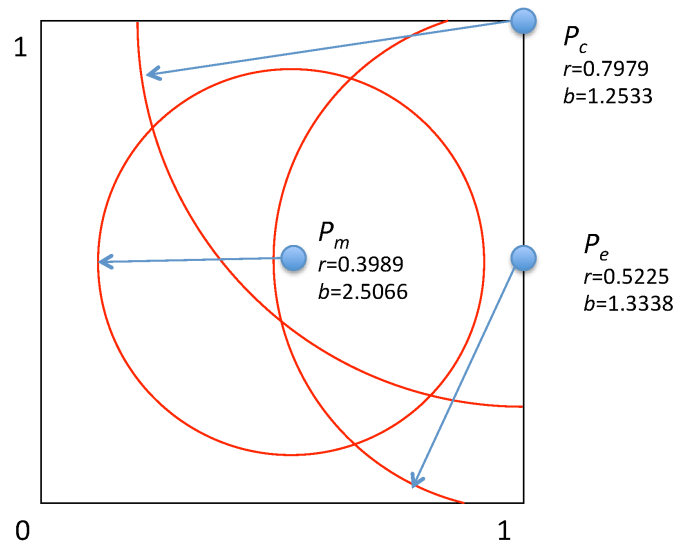


Figure 2.8: Pivot Position

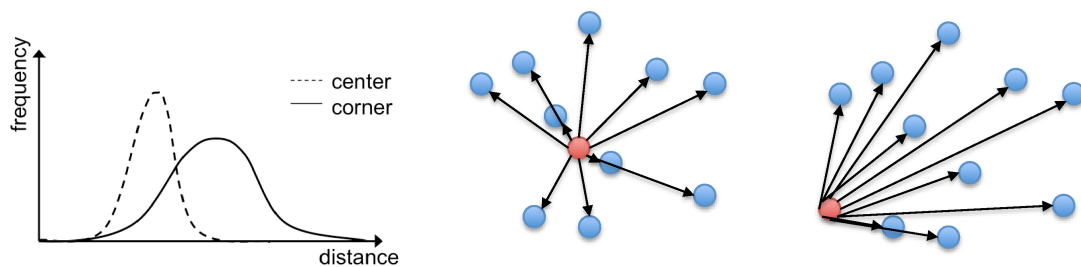


Figure 2.9: Center Object and Corner Object

2. Compute the distances from this object to all the other points.
3. Choose the farthest object as the pivot.

This technique only helps to choose a better pivot than the random selection. It cannot guarantee the best pivot choice, and the performance gain varies between 5% to 10% .

GNAT [Bri95] considers the distances among pivots. The author of GNAT indicated the problem of choosing many pivots is complicated. He proposed the following strategy for selecting m pivots that are far apart.

1. Choose $3 \cdot m$ objects from the dataset.
2. Pick an object at random and select the farthest object from this as the first pivot.

3. Select the farthest object from the first pivot as the second pivot.
4. Select the farthest object from the previous pivots.
5. Repeat until there are m pivots.

D-index [DGSZ03] has a selection scheme using the means [BNC03]. The authors of [BNC03] developed a criterion to compare the efficiencies of two sets of pivots of the same size. This technique is very different from other methods using outliers. Their pivot selection scheme is based on the mean of the distance distribution. They computed the mean for a given pivot set T as:

1. Choose l pairs of objects from the dataset at random.
2. For all pairs of objects, compute their distances in the feature space determined by the pivot set T .
3. Compute μ_{M_T} as the means of these distances.

The method of selecting k pivots is as follows.

1. Choose a set $T = \{p_1\}$ of one object from a sample of m objects in the dataset, such that the pivot p_1 has the maximum μ_{M_T} value.
2. Select a second pivot p_2 from another sample of m objects in the dataset, creating a new set $T = \{p_1, p_2\}$ for fixed p_1 , maximizing μ_{M_T} .
3. Choose a third pivot p_3 from another sample of m objects in the dataset, creating a new set $T = \{p_1, p_2, p_3\}$ for fixed p_1 and p_2 , maximizing μ_{M_T} .
4. Repeat until k pivots have been selected.

Their evaluations show that it is better than random selection or outlier selection.

Sparse Spatial Selection (SSS) [PB07] is a ratio-based pivot selection scheme. It aims at generating a set of pivots well distributed over the whole space. It selects pivots as follows:

1. Set M to be the maximum distance between any pair of objects.

2. Initially add an object to a pivot set T at random.
3. For each object o in the dataset, select o as a new pivot if its distance to every pivot in the current set T is equal or greater than $M \cdot \alpha$, where α is a constant parameter.

The author of [PB07] mentioned that the constant parameter α takes values around 0.4. It keeps the distances among pivots more than $M \cdot \alpha$ and ensures that the pivots are dispersed.

2.7.2 Clustering-based Approach

The clustering-based approach aims at utilizing more elaborate features based on the data distribution. It selects a pivot and its partitioning boundary on the basis of the clusters in the dataset. iDistance is the first such method to do so.

iDistance [JOT⁺03] is a cluster-based partitioning. The authors of [JOT⁺03] pointed out that data in real life are often clustered or correlated. Even when no correlation exists in all dimensions, there are usually subsets of data that are locally correlated. Thus, they aim at selecting a more appropriate pivot by identifying clusters from the space. Their scheme does not depend on a specific clustering method, but they choose BATCH [ZRL96] in [YOTJ01] and k-means in [JOT⁺03]. The partitioning scheme is as mentioned in Section 2.6.4. Their experiments showed that cluster-based partitioning successfully reduces the I/O cost and the computational cost in comparison with traditional partitioning schemes.

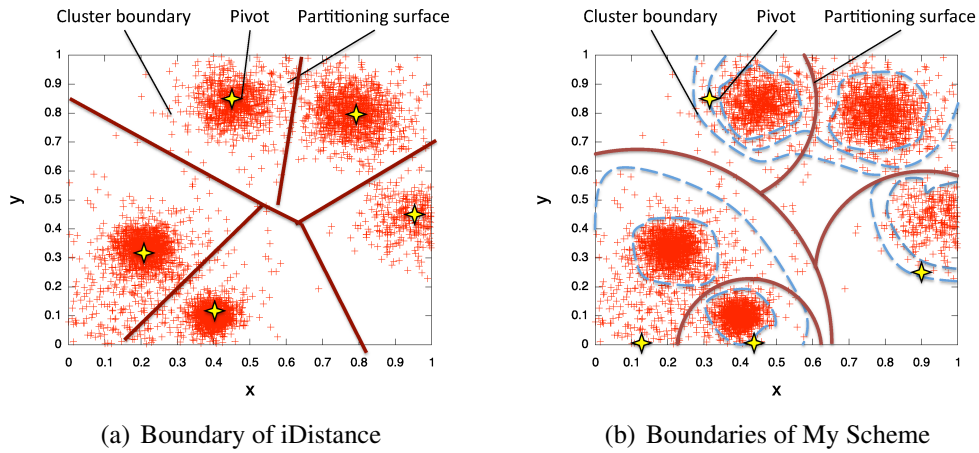


Figure 2.10: Partitioning Boundaries

2.8 My Pivot Selection Strategy

I focused on pivot selection in my research. I think that pivot partitioning can further reduce the search cost by utilizing the data distribution of the objective dataset.

As introduced in Section 2.7, many pivot selection schemes exist. Now, let me clarify the problems of the existing approaches and compare them with my approach. A problem of the simple statistical approach is that it doesn't consider the partitioning boundary of a pivot. Ball partitioning prunes an object if one of Equations (2.17) and (2.18) is satisfied. From these equations, the distance from a query to the partitioning boundary influences the performance of pruning. When we assume that both the queries and objects in the database are drawn from the same probability distribution, a pivot often misses the chance of pruning dissimilar objects if many objects are near the boundary. Although the clustering-based approach selects a better pivot and partitioning boundary, it only works well on particular distribution patterns. A cluster may be separated into multiple regions by a pivot, because its partitioning boundaries are based on cluster centers rather than cluster shapes. It is hard for the clustering-based approach to deal with skewed data clusters. Figure 2.10 shows an example of the partitioning boundaries.

I propose a new pivot selection approach called the “*data distribution approach*”. My approach divides the space on the basis of its distribution pattern, and it especially considers the boundaries of clusters. I developed two partitioning schemes: “*maximal metric margin partitioning (MMMP)*” and “*pivot capacity tree (PCTree)*”. MMMP is margin-based pivot

selection scheme. On the other hand, PCTree takes into consideration tree balancing as well as the data distribution. I will describe the architecture and the experimental evaluations of MMMP and PCTree in Chapters 3 and 4, respectively.

Chapter 3

MMMP:

Margin-based Pivot Selection Scheme

3.1 Overview of MMMP and MMMP-Index

Maximal Metric Margin Partitioning (MMMP) is a partitioning scheme for similarity search indexes. First, MMMP constructs hierarchical clusters with arbitrary shapes by using density-based clustering [ABKS99]. Then, at each branch b in the hierarchical structure, it detects the space boundary that divides the objects in the cluster corresponding to b with the maximal margin as shown in Figure 3.4. During this division, MMMP searches for a pivot object whose partitioning boundary is between clusters and also maximizes the distances to the cluster edges. The boundary is likely to be located in a sparse area between the clusters. The main contribution is that I have developed a pivot selection for Metric spaces that is based on maximal margins and is effective for clustered data. To the best knowledge of the authors, this is the first study to exploit the maximal margins for the pivot selection.

MMMP-Index is an indexing scheme which uses MMMP and pivot filtering. MMMP-Index can prune many objects that are not relevant to a query, and it reduces the query execution cost. My experimental results show that MMMP effectively indexes clustered data and reduces the search cost. For clustered data in a vector space, MMMP-Index reduces the computational cost to less than two thirds that of comparable methods.

The rest of the chapter is organized as follows. In Section 3.2, I introduce the architecture of the MMMP. Section 3.3 describes the indexing scheme of the MMMP-Index, and Section 3.4 discusses the experimental results. I summarize the chapter in Section 3.5.

3.2 MMMP

For a metric space $M = (D, d)$, suppose a pivot p divides a set S of objects in D into two regions:

$$\begin{aligned} S_1 &= \{o \in S \mid d(o, p) \leq r_p\}, \\ S_2 &= \{o \in S \mid d(o, p) > r_p\}, \end{aligned}$$

where r_p is the partitioning distance for the pivot p . When searching for objects within r_q from a query object q in terms of the metric space M , if the inequality

$$d(q, p) + r_q \leq r_p, \quad (3.1)$$

holds, it is sufficient to check for objects in S_1 . Similarly, if the inequality

$$d(q, p) - r_q > r_p, \quad (3.2)$$

holds true, it is sufficient to check objects within S_2 . Let C_1 and C_2 be the largest clusters in S_1 and S_2 , respectively. If the margin between the regions C_1 and C_2 is large, either Equations (3.1) or (3.2) is more likely to hold; i.e., we can prune region S_2 or S_1 . On the other hand, if the boundaries between C_1 and C_2 are close to each other, we need to check the objects within both S_1 and S_2 for a query around the boundaries, even when the centers of C_1 and C_2 are far from each other. This leads us to a pivot selection method based on a large margin criterion.

To find pivots that divide the space into regions with a large margin, the MMMP first makes hierarchical clusters that recursively divide the space into two regions. Currently, MMMP uses OPTICS [ABKS99], which is a density-based clustering method. To reduce the clustering cost, I use OPTICS on a randomly sampled dataset. Then, for each division, MMMP finds a pivot that separates the clusters with a large margin. Figure 3.1 depicts an example of partitioning by MMMP.

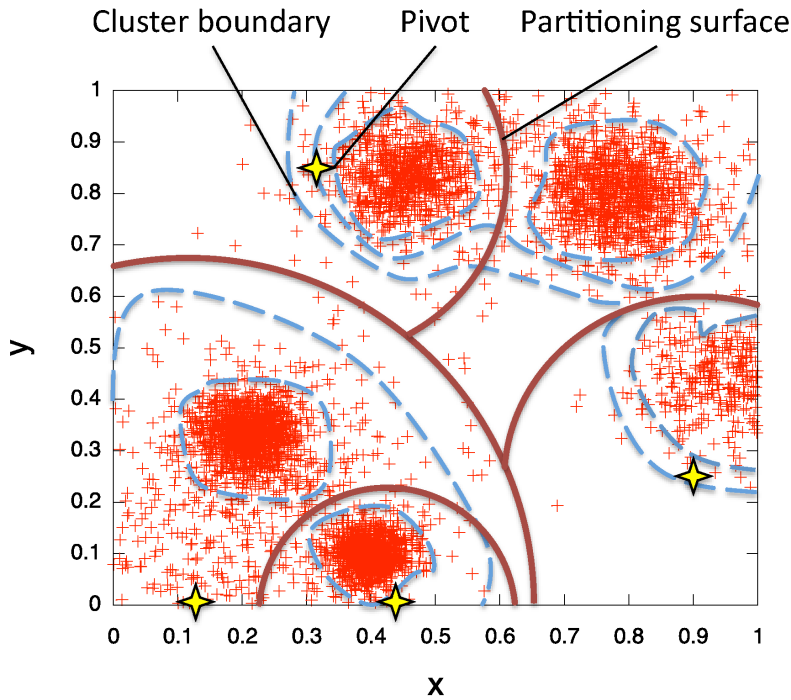


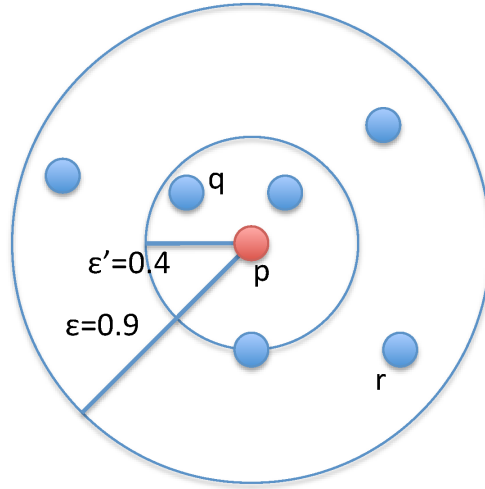
Figure 3.1: Partitioning by the MMMP

3.2.1 Overview of OPTICS

Density-based clustering constructs clusters that are dense regions of objects in the data space. Moreover, it separates outliers from the clusters. OPTICS is a kind of density-based clustering [ABKS99] that constructs hierarchical clusters with arbitrary shapes.

OPTICS calculates two values for each object in the dataset and orders objects according to them. One is the core distance, and the other is the reachability distance. Let $N_\varepsilon(p)$ be the set of objects within the distance ε from p , i.e., $N_\varepsilon(p) \equiv \{o \in S \mid d(o, p) < \varepsilon\}$. For an integer n and an object p , let $d_n(p)$ denote the distance between p and the n th closest object to p . Then, for parameters ε and $MinPts$, the core distance $cd_{\varepsilon, \cdot}(\cdot)$ of an object p is defined as

$$\begin{aligned}
 & cd_{\varepsilon, MinPts}(p) \\
 & \equiv \begin{cases} \text{UNDEFINED} & |N_\varepsilon(p)| < MinPts \\ d_{MinPts}(p) & \text{otherwise} \end{cases} .
 \end{aligned} \tag{3.3}$$



$$\begin{aligned}
 &MinPts=3, \varepsilon=0.9 \\
 &core\text{-}distance(p) = 0.4 \\
 &reachability\text{-}distance(p,q) = \max\{\varepsilon', d(p,q)\} = 0.4 \\
 &reachability\text{-}distance(p,r) = d(p,r)
 \end{aligned}$$

Figure 3.2: Reachability Distance

For parameters ε and $MinPts$, The reachability distance $rd_{\varepsilon, MinPts}(\cdot)$ of p and o is defined as

$$\begin{aligned}
 &rd_{\varepsilon, MinPts}(o, p) \\
 &\equiv \begin{cases} \text{UNDEFINED} & |N_{\varepsilon}(p)| < MinPts \\ \max(cd_{\varepsilon, MinPts}(p), d(o, p)) & \text{otherwise} \end{cases} . \quad (3.4)
 \end{aligned}$$

The reachability distance from the object o_1 to the object o_2 represents the longer distance of the k -nearest neighbor distance of o_1 and the distance between o_1 and o_2 . Note that the object o used in reachability-distance is selected by OPTICS. Figure 3.2 shows an example of the core-distance of p and the reachability-distance.

For a set D of objects, it first makes an ordered list q of objects by performing the following steps:

1. randomly select an object o from D and prepare an ordered list q consisting of o ,
2. repeat the following steps until D becomes empty:
 - (a) choose an object p in D for which the reachability distance from its nearest

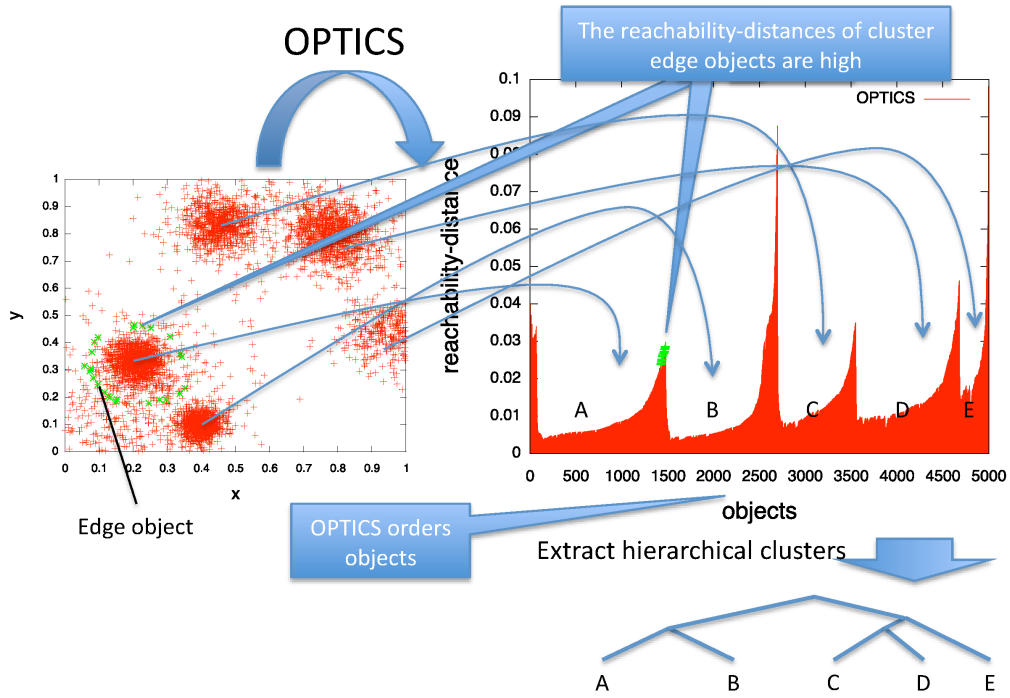


Figure 3.3: Clustering by OPTICS

object in q is the shortest,

(b) move p to the end of q .

As a result, objects in the same cluster tend to form a consecutive portion in the ordered list. Usually, a pair of adjacent objects in the list with a large reachability distance belong to different clusters. OPTICS utilizes this feature to detect the boundary between clusters, Figure 3.3 shows an example of the ordered list with the reachability-distance on the vertical axis. The peaks in the graph correspond to boundaries between clusters.

When the center area of a cluster is denser than the edge area, OPTICS tends to choose an edge object in the cluster first, reach the center area with the shortest path, and then move from the center area to the edge area. Therefore, we can extract objects around the edge of a cluster in the following way. For a cluster C detected by OPTICS, let q_C denote the sub ordered list corresponding to C . Suppose $\text{Order}(o, C)$ denotes the position of an object $o \in C$ in q_C . I define the *edge object set* of C by

$$\text{Edge}(n, C) = \left\{ o \mid \text{Order}(o, C) \leq \frac{n}{100} |C| \right\} \quad (3.5)$$

where n is a parameter. The edge object set represents the set of the objects whose reachability distances are ranked within the top $n\%$ in the cluster.

I chose OPTICS because

- it doesn't require the number of clusters to be given beforehand,
- it constructs hierarchical clusters,
- it can extract arbitrarily shaped clusters without calculating the center object in each cluster, and
- it can efficiently extract objects around the edge of the cluster.

The most important reason for using density-based clustering is to extract arbitrarily shaped clusters using the distances. Thus, other schemes such as k-means and BIRCH are not appropriate for MMMP.

The clustering cost is $O(n^2)$ in the original paper [ABKS99]. Although the clustering cost can be reduced somewhat [BKP06], it is still high. Therefore, I cluster random samples of the data instead of the whole data.

3.2.2 Pivot Selection and Partitioning

MMMP aims to achieve effective search pruning. I focus on the partitioning boundaries and the data distribution patterns to achieve this aim. MMMP maximizes the distances between the partitioning boundary and its closest objects, because the objects can be clearly classified. It divides the space on the basis of the OPTICS's clustering results that effectively separate small dense regions. MMMP tries to find partitioning boundaries that are between pairs of clusters that are at the maximum distances from the cluster edges. However, we cannot directly detect the partitioning boundaries in a metric space, because the partitioning boundaries are created by pivots. Thus, it basically searches for a pivot object that divides the clusters obtained by OPTICS with the largest margin.

MMMP construct hierarchical clusters in the form of a binary tree from the clustering results. Hierarchical clusters are represented with a tree structure whose nodes correspond to subspaces. Although a node can have more than two children, many nodes have just two children in the cluster hierarchy generated by OPTICS. Therefore, I shall first describe the

pivot selection method for the node having two children. For a node v and an object p , let $C_{p,\text{Near}}$ (resp. $C_{p,\text{Far}}$) denote a v 's child cluster that is near (resp. far from) p . The cluster labels are judged on the basis of the distance from p to one object selected randomly from the objects in each cluster. MMMP chooses the following object as a pivot

$$\text{pivot} = \operatorname{argmax}_{p \in S} \left(\min_{o_f \in C_{p,\text{Far}}} d(p, o_f) - \max_{o_n \in C_{p,\text{Near}}} d(p, o_n) \right). \quad (3.6)$$

The first term on the right side of this formula represents the distance to the nearest object in $C_{p,\text{Far}}$, whereas the second term is the distance to the farthest point in $C_{p,\text{Near}}$. Therefore, the right side of Equation (3.6) represents a kind of margin.

When a node has more than two children, MMMP merges the children into two clusters and obtains a candidate pivot by computing Equation (3.6). Then, it chooses the candidate having the largest margin as a pivot.

The reasons why only one object in each cluster is used for distinguishing between $C_{p,\text{Near}}$ and $C_{p,\text{Far}}$ are as follows. One reason is to reduce the computational cost for this operation. The other reason is that the pivot candidates evaluated with the incorrect label clusters have no influence on the pivot selection. These pivot candidates are not relevant to the best pivot in most cases, and would finally be removed by Eq (3.6). Of course, when there is no relevant pivot candidate in the data, this distinction does not work well and the evaluation of a pivot may have a negative value.

The partitioning distance r_p of the pivot p is defined as

$$r_p = \text{Distance}(p) = \frac{1}{2} \left(\min_{o_f \in C_{p,\text{Far}}} d(p, o_f) + \max_{o_n \in C_{p,\text{Near}}} d(p, o_n) \right). \quad (3.7)$$

Figure 3.4 is an overview of the pivot selection method.

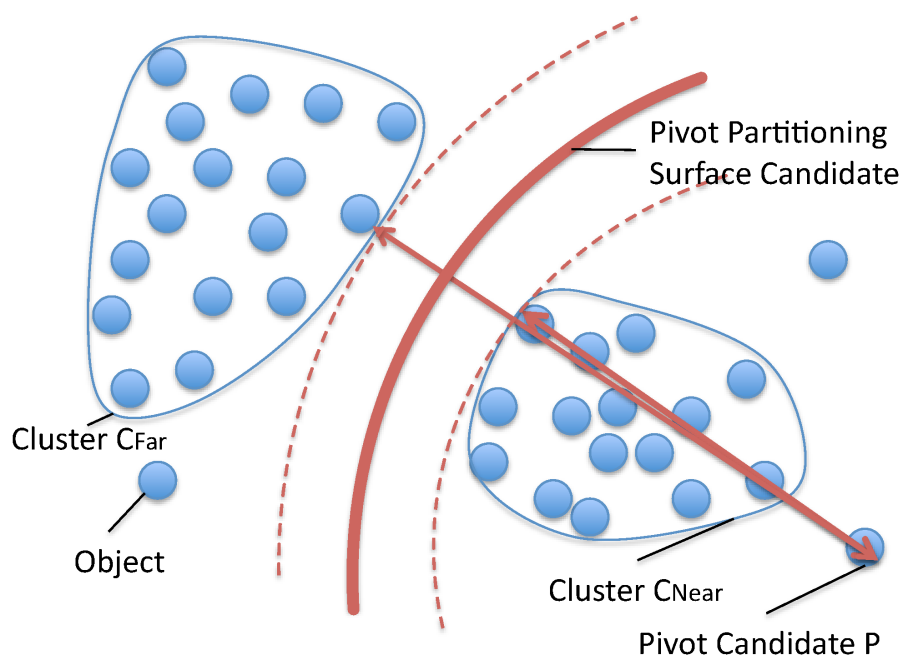


Figure 3.4: Pivot Selection in the MMMP

3.3 MMMP-Index

3.3.1 Index Structure

My indexing scheme, named MMMP-Index, uses MMMP and pivot filtering. As I described in Section 3.2, MMMP is designed to prune regions irrelevant to the query in the clustered data space. However, MMMP does not work well when the number of clusters in the data space is too small or the size of any one cluster is too large. The number of objects managed by a pivot affects the performance of the index. Therefore, I use the MMMP-Index to divide a large cluster into small enough regions for improving the performance by pivots as is done in LC [CN05]. Figure 3.5 shows the structure of the MMMP-Index.

The index is implemented with two B⁺-trees. One B⁺-tree manages two kinds of pivots; the pivots selected by the MMMP and those chosen by compact partitioning. The pivots form a tree structure. Each pivot has its own ID and the IDs of its leaf pivots. Smaller IDs are assigned to earlier selected pivots. The pivots' keys on the B⁺-tree are also based on their IDs. The other B⁺-tree stores objects. The object key is measured by $d(o, p) + ID_p \times c$, where p is the pivot that manages the object and c is a parameter that is sufficiently larger than the distance between objects. With these key definitions, when we search for objects whose distances from the pivot are within a certain range, we can sequentially access the disk and can reduce the page access cost.

In the implementation of the MMMP-Index, I use $\text{Edge}(n, C_{p,\text{Far}})$ (resp. $\text{Edge}(n, C_{p,\text{Near}})$) defined by Equation (3.5) instead of $C_{p,\text{Far}}$ (resp. $C_{p,\text{Near}}$) in Equations (3.6) and (3.7) for reducing the computational cost of the pivot evaluation. Note that the edge object set defined by Equation (3.5) is a subset of its cluster. When the parameter n in Equation (3.5) is 100, the edge object set is equivalent to the cluster. The rest of this section shows the index construction method and the range search method of the MMMP-Index.

3.3.2 Index Construction

Indexing

The MMMP-Index is constructed by performing the following steps:

1. Randomly select k objects from the data.

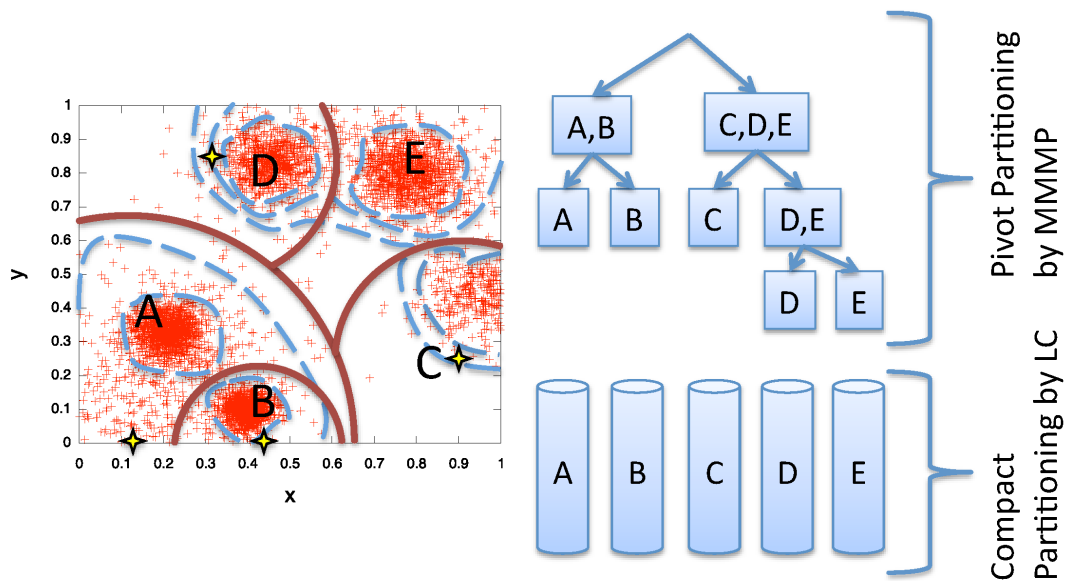


Figure 3.5: MMMP-Index

2. Discover hierarchical clusters from the objects by using OPTICS.
3. For each branch from the root of the cluster branches:
 - (a) Find the edge objects of each cluster in the branch whose reachability-distances are ranked within the top $n\%$ in the cluster.
 - (b) Select a pivot and its partitioning distance by using MMMP.
4. Create a tree structure with the pivot information.
5. For each object in the data, search for the region that the object belongs to.
6. For each region, execute compact partitioning in the manner of LC [CN05].
7. Construct two B^+ -trees to manage objects and pivots, respectively.

Algorithm 1 shows the indexing algorithm.

Object Insertion

The MMMP-Index inserts an object by performing the following steps:

1. Search for the leaf region that the object belongs to in the tree structure of the index.

2. Search for the pivot that the object belongs to in the pivot list made by LC [CN05], and add the object to the B^+ -tree.

Object Deletion

The MMMP-Index deletes an object by performing the following steps:

1. Search for an appropriate region for the object in the tree structure of the index.
2. Search for an appropriate pivot in the pivot list of the region, and delete the object [CN05].

3.3.3 Range Search

The MMMP-Index searches for objects whose distances to a query object q are within the distance r_q by performing the following steps:

1. Traverse the tree structure of the index and discard irrelevant regions by using the distance from the MMMP pivot to q and the triangle inequality (Equations (3.1) and (3.2)).
2. For each remaining region:
 - (a) Discard irrelevant pivots from the pivot list by using the distance from the pivot to q and the triangle inequality [CN05].
 - (b) For each remaining pivot:
 - i. Extract the objects which may be relevant to the range query from the B^+ -tree by using the distance from the pivot to q and the triangle inequality.
 - ii. Measure the distance between each object and q . If the distance is within r_q , add the object to the result set.

Algorithm 2 shows the range search algorithm.

3.3.4 Index Construction Cost

The MMMP-Index is constructed by

- (a) the hierarchical clustering,
- (b) selecting a pivot for each partition of a cluster in the hierarchy,
- (c) and ball partitioning of the leaf clusters.

Step (a) requires from $O(n \log n)$ to $O(n^2)$ calculations when using OPTICS [ABKS99]. Step (b) requires $O(n^2)$, and step (c) requires from $O(m \log m)$ to $O(m^2)$, where m is the size of the maximum leaf cluster [CN05]. Generally m is much less than n , so steps (a) and (b) are the dominant parts of the indexing construction. On the other hand, LC requires $O(n^2)$ [CN05]. Hence, the computational complexity for constructing the MMMP-Index is the same as the worst case of LC. In my experiments, the indexing time for MMMP-Index was almost the same as that for LC.

Algorithm 1 Indexing Algorithm**Function Indexing(Dataset D)**Select Objects S from D Cluster $C \leftarrow \text{OPTICS}(S)$ IndexTree $\leftarrow \text{MMMP}(S, C)$ **for** obj in D **do** IndexKey $id, depth \leftarrow \text{IndexTree}(obj)$ $S_{id,depth}.add(obj)$ **end for****for each** $S_{id,depth}$ **do** ListOfClustersIndex($S_{id,depth}$)**end for****return****Function MMMP(S, C)**Sub-partitioning($S, C, 0, \text{len}(S)$)**return** PivotSet**Function Sub-partitioning($S_{sub}, C, first, last$)****if** $\text{length}(C(first, last)) \geq 2$ **then** $C_1, C_2 \leftarrow C(first, last)$ pivot $p \leftarrow \text{argmax}_{p \in S} (\min_{o_f \in \text{Edge}(n, C_{Far})} d(p, o_f) - \max_{o_n \in \text{Edge}(n, C_{Near})} d(p, o_n))$ distance $r \leftarrow \text{Distance}(p)$ PivotSet.add(p, r) Sub-partitioning($C_1, C, C_1.first, C_1.last$) Sub-partitioning($C_2, C, C_2.first, C_2.last$)**end if****return****Function ListOfClustersIndex($S_{id,depth}$)****if** $\text{length}(S_{id,depth}) < \text{fixsize}$ **then** **return****end if** $p \leftarrow \text{argmax}(\text{sum}(d(obj, p_{prev}) | obj \in S_m, p_{prev} \in \text{PivotSet}_{id,depth}))$ Select r ($\text{length}(I) = \text{fixsize}$) $I \leftarrow \{d(obj, p) \leq r | obj \in S_{id,depth}\}$ $E \leftarrow S_{id,depth} - I$ PivotSet $_{id,depth}.add(p, r, \text{BTree}(I))$ ListOfClustersIndex(E)PivotListSet(PivotSet $_{id,depth}$)**return**

Algorithm 2 Range Search Algorithm

Function RangeSearch(query q , range r_q)
return Sub-search($q, r_q, 0, 0$)

Function Sub-search($q, r_q, id, depth$)
if not exist PivotSet($id, depth$) **then**
 return SearchListOfCluster($q, r_q, id, depth, 0$)
end if
 $p_{id, depth}, r_{id, depth} \leftarrow$ PivotSet($id, depth$)
distance_to_pivot \leftarrow $d(p_{id, depth}, q)$
if (distance_to_pivot $\leq r_{id, depth} + r_q$) **then**
 Sub-search($q, r_q, id.in, depth + 1$)
end if
if (distance_to_pivot $> r_{id, depth} - r_q$) **then**
 Sub-search($q, r_q, id.out, depth + 1$)
end if
return

Function SearchListOfCluster($q, r_q, id, depth, level$)
if not exist PivotSet $_{id, depth}(level)$ **then**
 return
end if
 $p, r, BTree(I) \leftarrow$ PivotSet $_{id, depth}(level)$
distance_to_pivot \leftarrow $d(p, q)$
if distance_to_pivot $\leq r + r_q$ **then**
 Search($I, distance_to_pivot, p, r, q, r_q$)
end if
if distance_to_pivot $> r - r_q$ **then**
 SearchListOfCluster($q, r_q, id, depth, level + 1$)
end if
return

Function Search(BTree(I), distance_to_pivot, p, r, q, r_q)
KeySet \leftarrow {key \in BTree(I).keys \mid $\max(distance_to_pivot - r_q, 0) \leq$
key $\leq \min(distance_to_pivot + r_q, r)$ }
for key in KeySet **do**
 $obj \leftarrow$ BTree(I).get(key)
 if $d(obj, p) \leq r_q$ **then**
 Result.add(obj)
 end if
end for
return

3.4 Performance Evaluation

3.4.1 Outline of Experiments

I conducted experiments to evaluate the MMMP-Index together with MMMP. I compared the MMMP-Index with iDistance [JOT⁺03], D-Index [DGSZ03], and LC [CN05].

As in the related work [ZADB05], the indexes' performances were evaluated in terms of the page access and computational costs. The page access cost of a region r_i $\text{Pagecost}(r_i)$ is estimated by

$$\text{Pagecost}(r_i) = \lfloor N_{r_i} / P_{ave} \rfloor, \quad (3.8)$$

where N_{r_i} is the number of accessed objects in r_i , and P_{ave} is the average number of objects on a page. The page access cost of a query q is the sum of the costs of the accessed regions. On the other hand, I measured the computational cost in terms of the total number of distance calculations during the search. Actually, the computational time depends not only on the number of distance calculations but also on the cost of each calculation. In addition, I show the search time of some data sets for reference, although the time depends on the machine resource. I conducted the experiment on a Linux PC equipped with an Intel(R) Quad Core Xeon(TM) X5492 3.40GHz CPU and 64GB memory. I implemented the MMMP-Index in C and compiled it with GCC 4.2.

The performance of a similarity search index depends on the distribution of the data. Thus, this experiment used datasets containing various objects described in Section 3.4.2. According to the previous studies [JOT⁺03], the page size needed for estimating the page access cost is 4,096 bytes. Since the computational cost of iDistance to find the cluster centers in the data uses the computationally heavy edit distance or Jaccard distance, I didn't measure the performance of iDistance. Each result was the average of over 1,000 queries of its dataset.

3.4.2 Data Set

I used three types of data to evaluate my scheme: vectors, sets, and strings. The datasets consisted of synthetic vectors generated by us, vectors of a Julia set, the real vector dataset named Corel Image Features, which were downloaded from the UCI KDD Archive [UCI],

photo tag sets which were retrieved with the queries “tokyo” from flickr [flib], and English words, which were downloaded from WordNet [Wor]. We used the Euclid distance for the vector datasets, the Jaccard distance for the set dataset, and the Levenshtein distance for the string dataset.

Synthetic vectors are generated in 2, 8, 16, and 30-dimensional feature spaces according to uniform and Gaussian mixture distributions. The cluster centers of the clustered vectors were randomly selected. The centers numbered 10, 20, and 30. The number of objects for each cluster was randomly chosen. The objects in the cluster were based on a normal distribution. The standard deviations were randomly set from 0 to 0.10 and from 0 to 0.20. Furthermore, I generated synthetic vectors with noise objects. I mixed the clustered vectors with uniform vectors as noise. By referring to the previous study [CN05], I chose the size of the data to be 100,000. Queries were randomly chosen from the same distribution as the data set. When the chosen query happened to be the same point in the data set, we discarded it.

The Julia set is a fractal figure [CG93]. Let us consider a sequence of complex numbers defined by the recurrent form $z_{n+1} = z_n^2 + c$ where c is a complex constant. Then, the quadric Julia set is the set of the initial complex numbers z_0 in a sequence that does not diverge. I set c to be $0.40 - 0.35i$. Figure 3.13(a) shows its shape.

Corel Image Features included in the Corel image collection [UCI]. They consist of 32 subspaces that divide up the HSV color space (32 colors: 8 ranges of H and 4 ranges of S). The value of each dimension indicated the density of each color in the entire image

The Photo tag set had 3,343 kinds of tags. The average number of tags per photo was 7.42. All the photos had the tag “tokyo”.

The English words are all entry words in the WordNet [Wor]. I did not adjust the lengths of the words and used the strings as they were.

Table 3.1 lists the details of the datasets.

Table 3.1: Data Sets

Data	Distance	No. of data	No. of queries	Data source	Dimension	Standard deviation range	No. of clusters
Vectors	Euclid distance	100,000	1,000	synthetic	2	(0,0,10)	20
					8	Uniform	-
					8	(0,0,10)	10
					8	(0,0,10)	20
					8	(0,0,10)	30
					8	(0,0,20)	20
					16	(0,0,10)	20
					30	(0,0,10)	20
Julia Set				synthetic	8	(0,0,10) + noise	20
	Euclid distance	100,000	1,000		2	-	-
Corel							
Image Features	Euclid distance	67,040	1,000	UCI KDD Archive [UCI]	32	-	-
Photo tag sets	Jaccard distance	50,000	1,000	Flickr (query: "tokyo") [frib]	-	-	-
English words	Edit distance	205,941	1,000	WordNet [Wor]	-	-	-

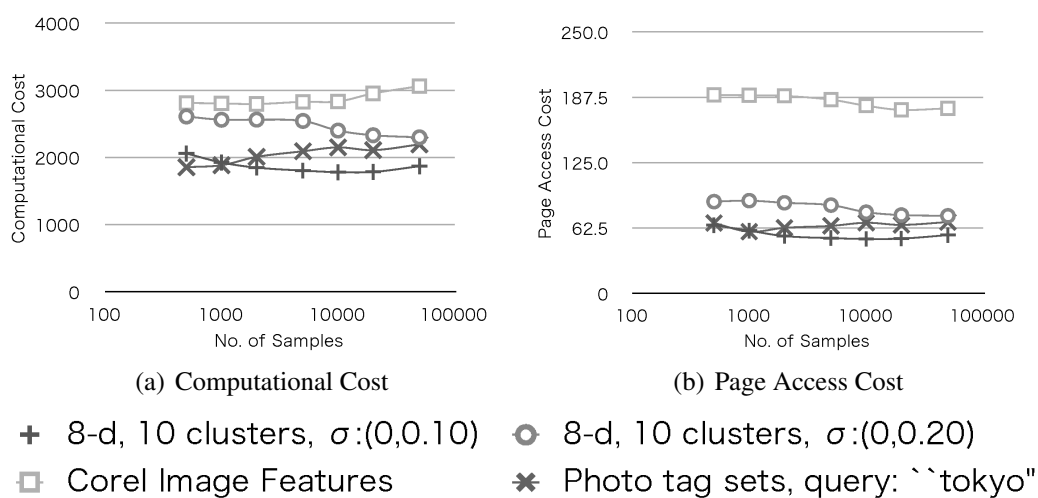


Figure 3.6: Samples for Clustering

3.4.3 Synthetic Data

Parameter Tuning

I first evaluated how the sample size of the MMMP affects the index performance using four kinds of datasets, i.e.,

- synthetic vector datasets in an 8-dimensional space with 20 clusters of vectors with standard deviations of from 0 to 0.1 and from 0 to 0.2,
- real vectors, and
- the tag set.

I constructed indexes using 500 to 50,000 samples. The radius of a query in the experiment was set to the distance to the 20-nearest neighbor objects. The edge objects of each cluster were the objects whose reachability-distances were ranked within the top 5% in the cluster. Figure 3.6 plots the computational and page access costs with respect to sample size. The result indicates that the index performances are not so much affected by the number of samples. Presumably, this is because the vector data has clear clusters. I decided that the clustering in the MMMP would be computed using 20,000 random samples from the dataset for the index performance evaluations.

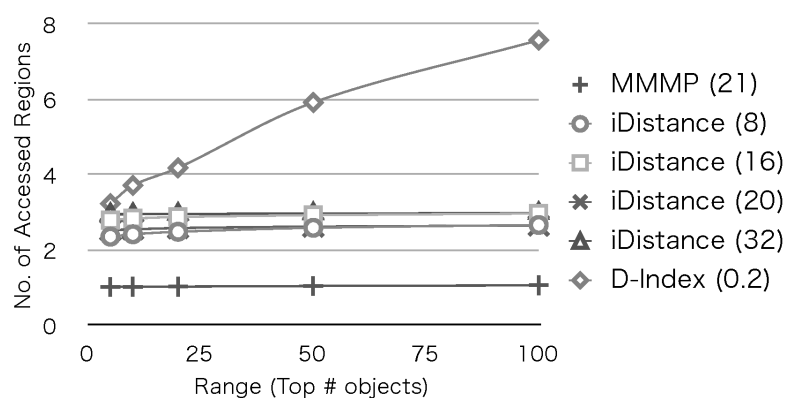
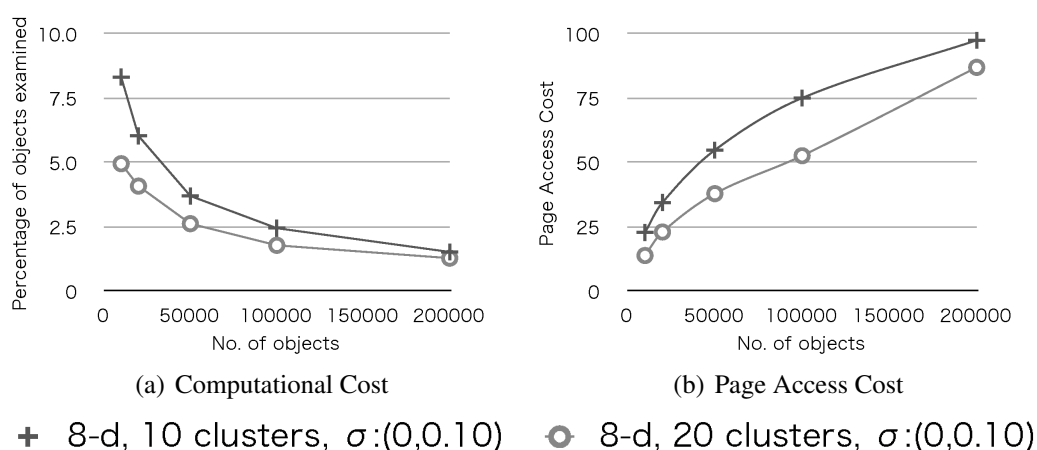
Figure 3.7: Partitioning Performance (8-d, 20 clusters, $\sigma:(0,0.10)$)

Figure 3.8: Index Performance w.r.t. No. of objects

Partitioning Performance

To evaluate the effectiveness of MMMP separately, I counted the number of accessed regions during query processing. The numbers of objects were fixed to 100,000. I compared MMMP with D-Index [DGSZ03] and iDistance [JOT⁺03]. D-Index selects a pivot based on the mean distances between the pivot and objects and recursively divides the space by using excluded middle partitioning. iDistance divides the space with a Voronoi partition based on k-means clustering and sets the cluster centers as the pivots.

Figure 3.7 shows the number of the accessed regions while searching for the 8-dimensional clustered vector. The vertical axes represent the number of accessed regions. The horizon-

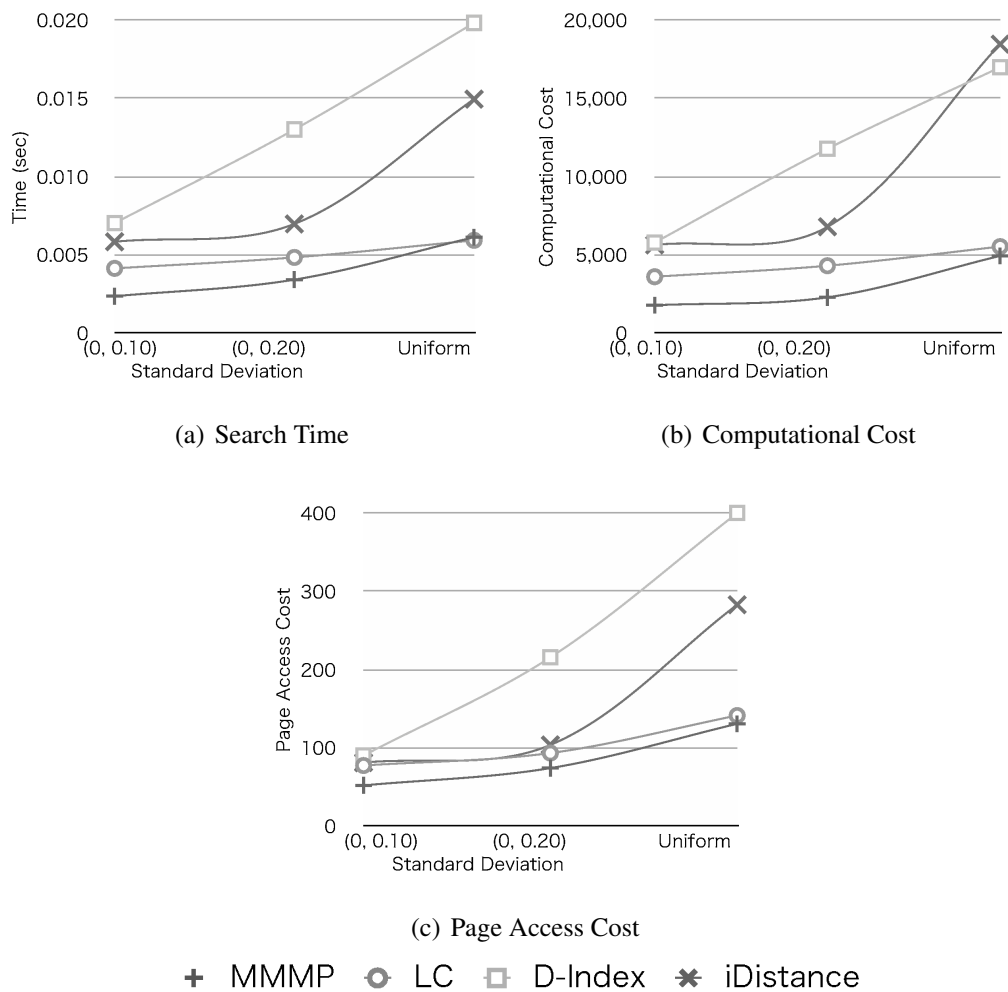


Figure 3.9: Index Performance w.r.t. Variance (8-d, 20 clusters and Uniform)

tal axis is the query range. The number for each line in the legend represents the parameter for indexing. The numbers for MMMP and iDistance are the numbers of clusters, and the numbers for D-Index are the partitioning distances of the exclusion sets. The results show that MMMP minimizes the number of accessed regions while searching. The number of accessed regions was approximately one in the MMMP results. This result indicates that the partitioning based on the cluster edges is better than partitioning based on cluster centers as in iDistance.

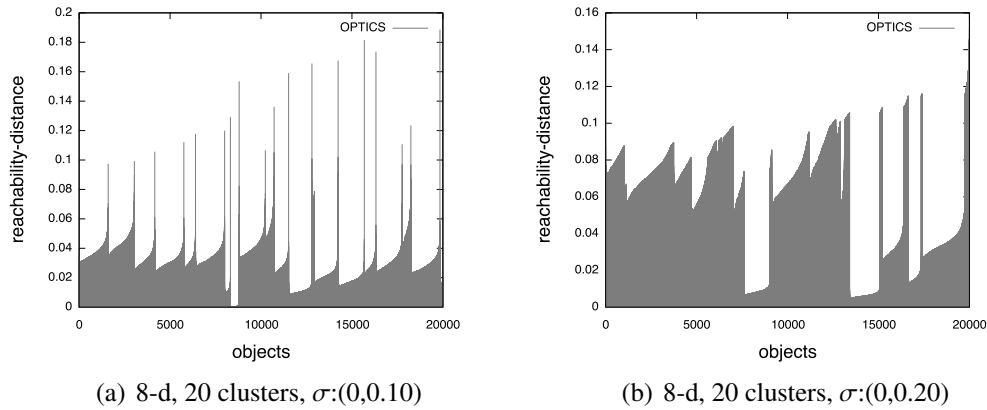


Figure 3.10: Clustering Result

Variance

I evaluated the index structure and performance with respect to the variances. I used $8d$ -clustered vectors generated by the mixture distributions with 20 Gaussian models and the uniform distribution. The radius of a query in the experiment was set to the distance to the 20-nearest neighbor objects. The numbers of objects was 100,000.

Figure 3.9 shows the index performance. It is clear that the MMMP-Index is superior to the other schemes for indexing clustered data. For example, the computational cost of the MMMP-Index is less than two thirds that of the other schemes. The page access cost of the MMMP-Index is also less than the other schemes. These results prove that the search cost of the MMMP-Index is lower for the clustered data with lower standard deviations. Figure 3.10 depicts the results of OPTICS clustering. Deeper valleys in the reachability plots mean denser clusters. Higher reachability distance points between valleys mean that clusters are far from each other. From the clustering results, we can clearly recognize that the clustered data with lower standard deviations in Figure 3.10 (a) has 18 clusters that is almost the same number of clusters as its parameter. We can also see that the clusters in the data with lower standard deviations are denser and more isolated than the other data. It is more difficult to recognize clusters when the data have higher standard deviations. Thus, I think the difference is because the data with lower standard deviations has a sparser density space and MMMP could easily create a good partitioning boundary. MMMP achieves effective partitioning by exploiting knowledge about the distribution of the clustered data.

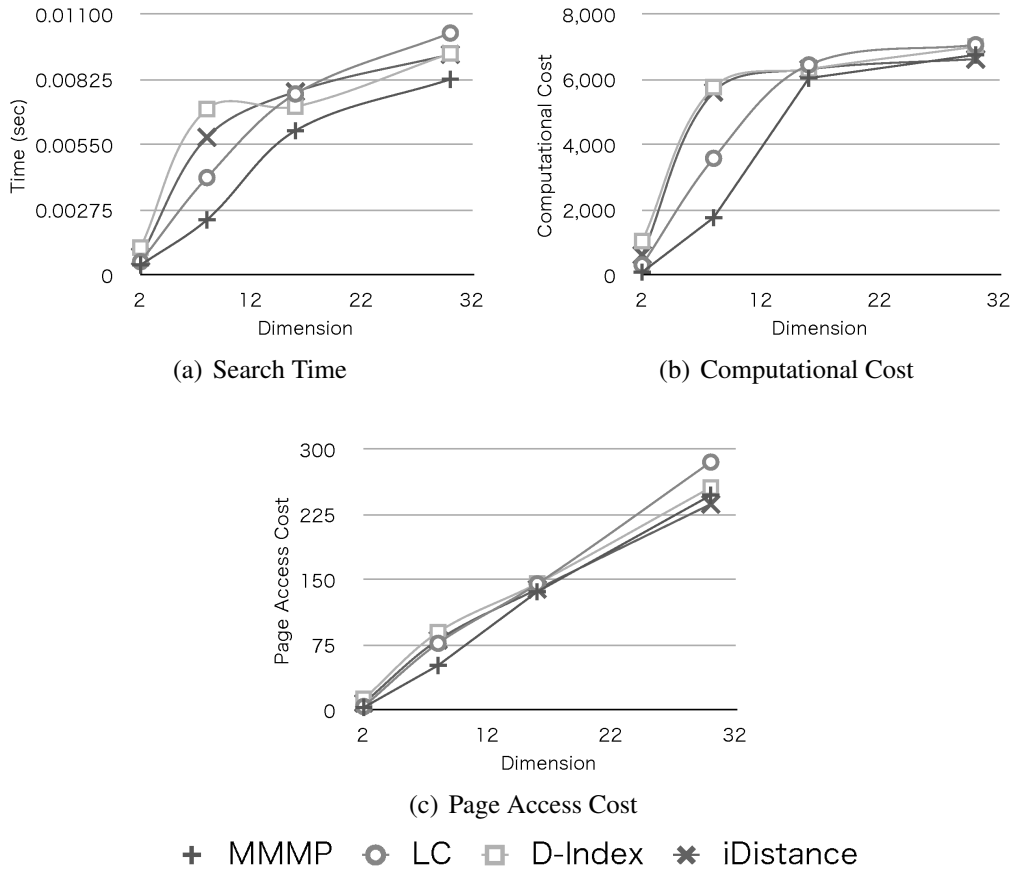


Figure 3.11: Index Performance w.r.t. Dimension (20 clusters, $\sigma:(0,0.20)$)

Dataset size

I evaluated the index structure and the performance with respect to the number of objects by using $8d$ -clustered vectors. The vectors have 20 clusters of vectors with standard deviations of from 0 to 0.1 and from 0 to 0.2, The radius of a query in the experiment was set to the distance to the 20-nearest neighbor objects. The number of objects was varied from 10,000 to 200,000.

Figure 3.8 shows the index performance. As in the related work [Nav02], the computational cost was evaluated by the percentage of objects examined. The vertical axis of (a) represents the percentage of objects examined within certain distances from the pivots. The vertical axis of (b) is the page cost. The horizontal axis is the number of objects. From these results, we can see that the search cost of the MMMP-Index is lower for datasets with more clusters, although its cost scales up almost linearly as the size of dataset increases.

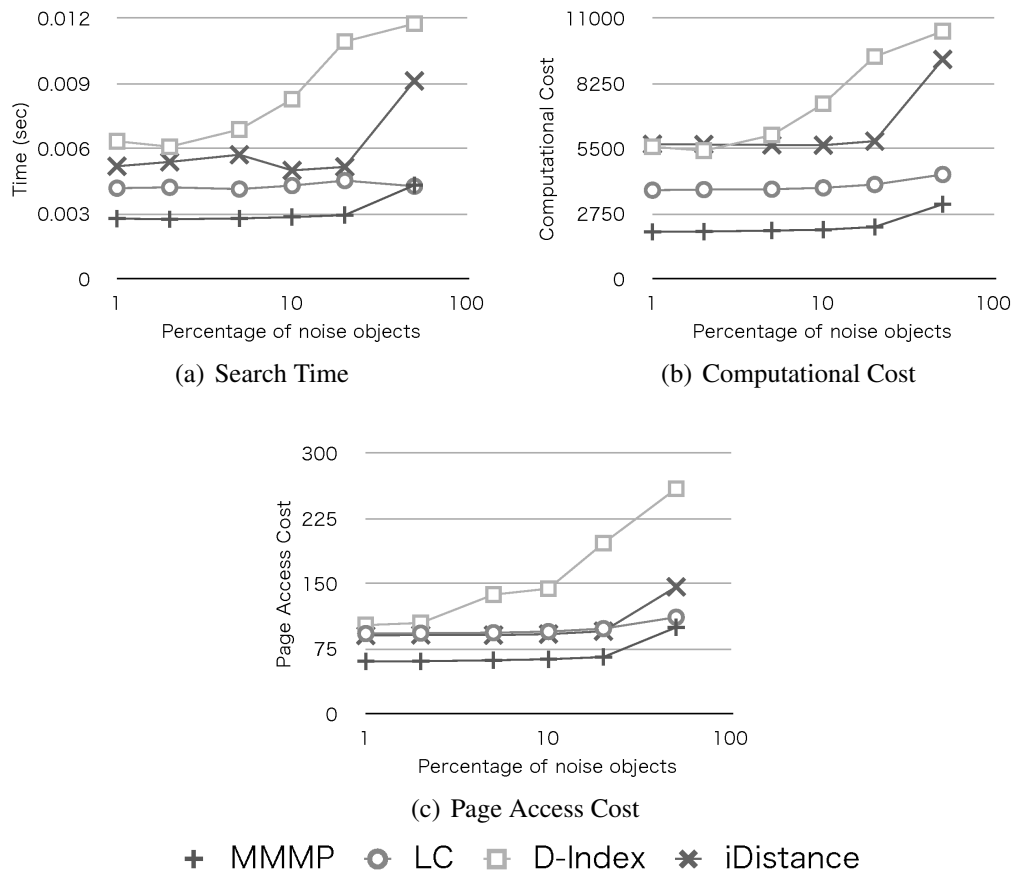


Figure 3.12: Index Performance w.r.t. Noise (8-d, 20 clusters, $\sigma:(0,0.20)$)

The results indicate that the computational cost of the MMMP-Index for the clustered data with 20 clusters is almost half that of the data with 10 clusters. I interpret this behavior as follows: the MMMP-Index correctly accesses the only region which is relevant to the query (as shown in Figures 3.10 and 3.7); however, the cost of checking the objects in the region varies almost linearly because the objects are indexed by LC. For much larger datasets, I will have to use a more efficient indexing scheme.

Dimension

I evaluated the index structure and the performance with respect to the number of dimensions. I used $2d$, $4d$, $8d$, $16d$, and $30d$ -clustered vectors with 20 clusters. The radius of a query in the experiment was set to the distance to the 20-nearest neighbor objects. The number of objects was fixed to 100,000.

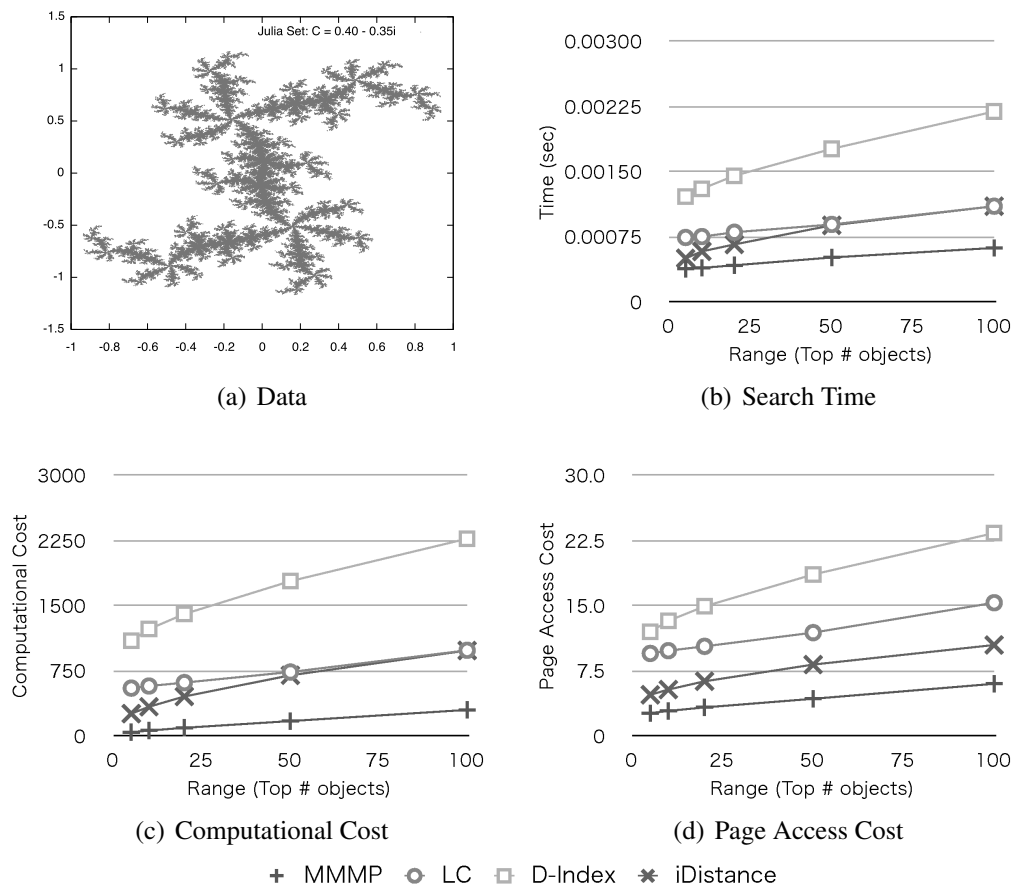


Figure 3.13: Julia set

Figure 3.11 shows the results. We can see that the MMMP is superior to the other methods on vectors of various dimensions. We can also see that MMMP performs well for lower dimensional vectors. The computational cost has more influence on the search time than the computational cost. I interpret that this is because the size of the vector is small.

Noise

I conducted an experiment for checking the robustness of the proposed method against noise. I added noise objects that were generated randomly according to a uniform distribution to 8-dimensional clustered vectors consisting of 20 clusters with standard deviations of from 0 to 0.1. The radius of a query in the experiment was set to the distance to the 20-nearest neighbor objects. The number of objects was fixed to 100,000.

I plotted the search performance with respect to the percentage of noise objects in Fig-

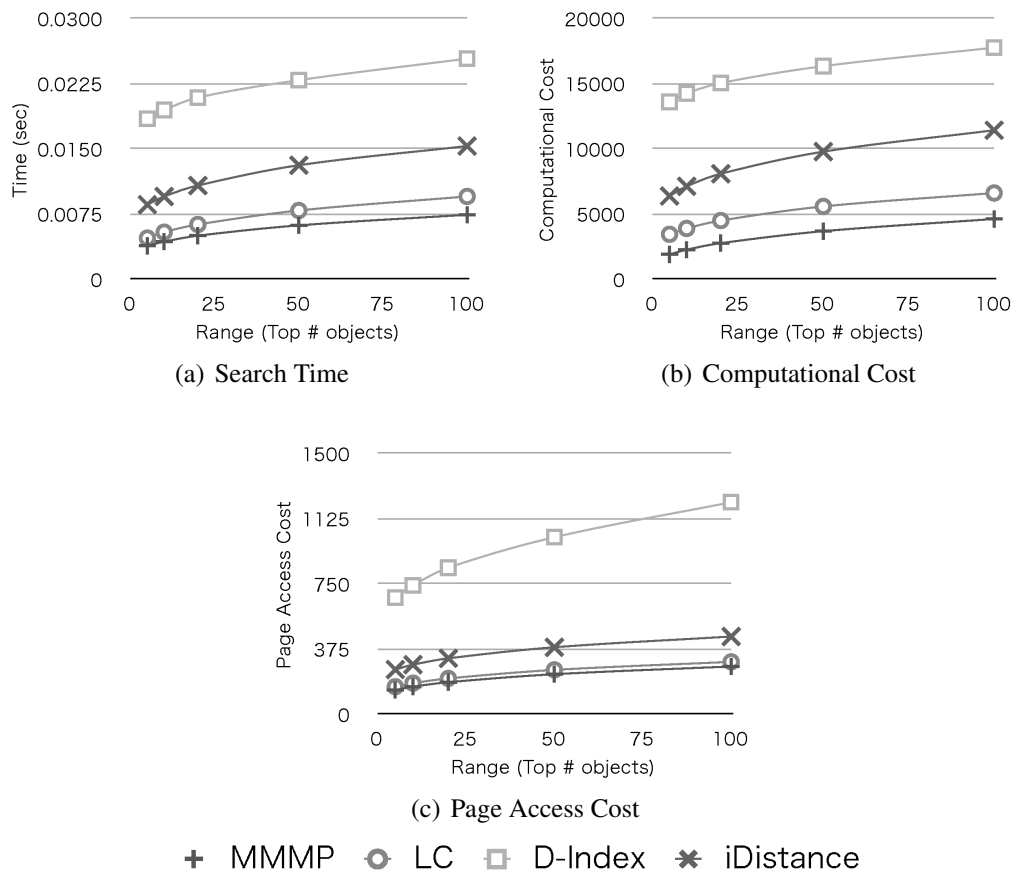


Figure 3.14: Corel Image Features

ure 3.12. It is clear that the MMMP-Index outperforms the other methods for wide range of noise ratios. The results show that my method is the most stable. D-Index is the most affected by noise. These results indicate that the clustering technique of MMMP is stable in the presence of noise.

Julia Set

The above synthetic vectors have ball-shaped clusters, because they are generated on the basis of a normal distribution. Ball-shaped clusters are easy to classify by pivot partitioning because a pivot makes a spherical boundary. Thus, to evaluate the indexing performance for clusters whose shape is not spherical, I compared the MMMP-Index with other indexes by using the Julia set. I selected it because it has hierarchical clusters of various sizes that are not ball-shaped and it can be easily generated.

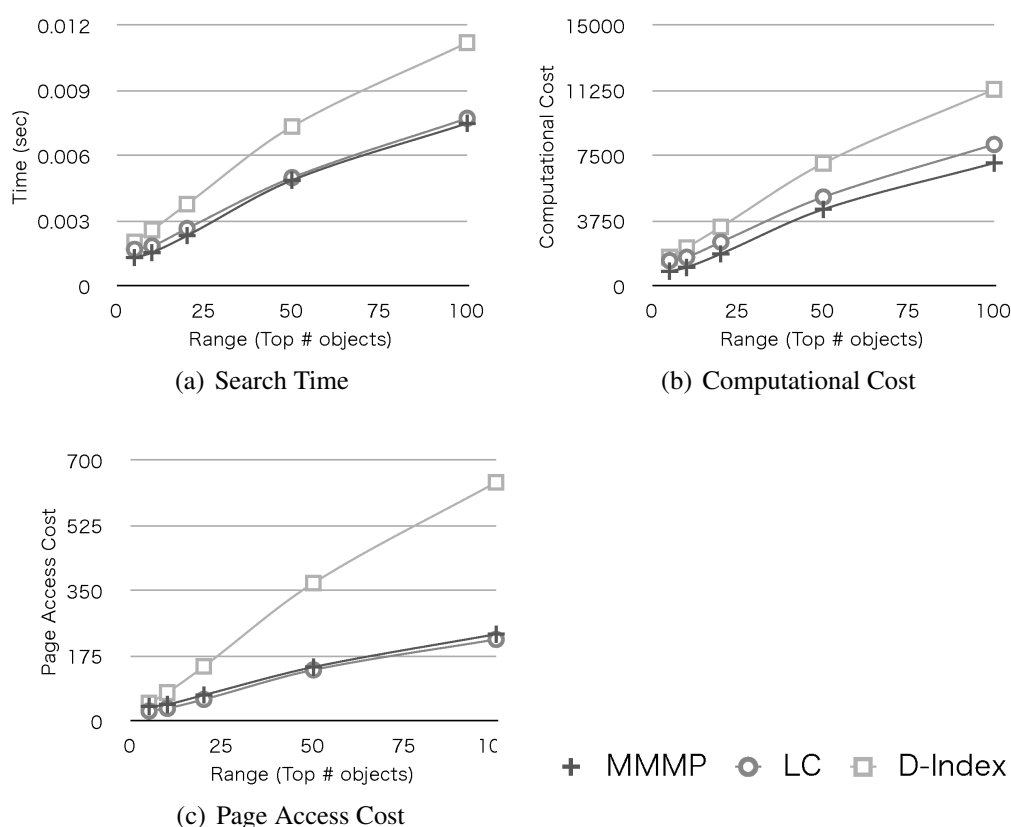


Figure 3.15: Photo tag sets, query: “tokyo”

Figure 3.13 shows the index performances of these methods on the Julia set. It is clear that the MMMP-Index is better than the other methods. Although iDistance sets the cluster centers as pivots, the MMMP can select a pivot from all the objects on the basis of its partitioning boundary and render a good partition for search pruning regardless of the clusters’ shapes.

3.4.4 Real Data

I compared the MMMP-Index with other methods on three real data sets. The datasets have different distance functions. In this experiment, I evaluated the effect of varying the query radius on the index performance. I set the radiuses to the distances to the 5-nearest through 100-nearest neighbors. Figures 3.14, 3.15, and 3.16 show the index performance for the Corel image features, the photo set, and the English words, respectively. The MMMP-Index outperforms other methods on the Corel Image Features and the photo tag sets (Figures 3.14

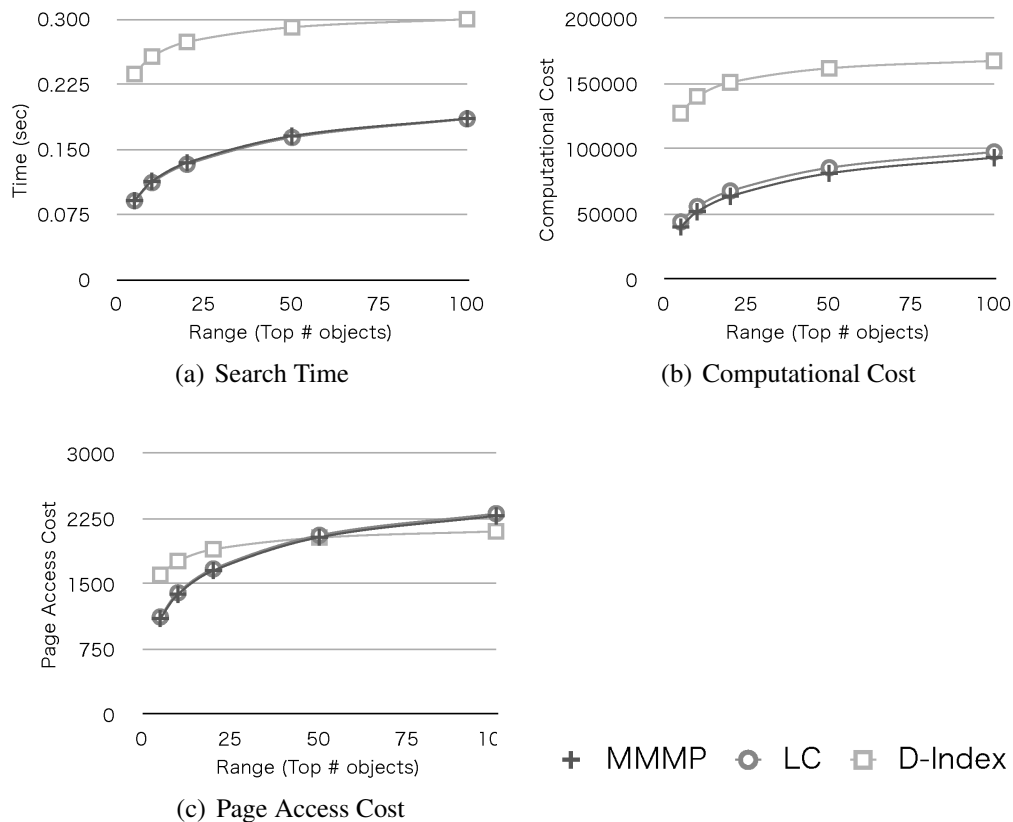


Figure 3.16: English words

and 3.15). This proves that the MMMP-Index is not only effective on synthetic vectors but also on the data whose distances obey metric space postulates.

The MMMP-Index and LC perform about the same for English words, because the datasets have few clusters. This means that MMMP does not work well on them. Since MMMP requires enough clusters for selecting pivots, it would select few pivots for pivot partitioning and the index would mainly depend on pivot filtering in those datasets.

3.5 Summary

I developed MMMP as a means of pivot partitioning for the purpose of pruning in a similarity search index. MMMP divides data on the basis of the shapes of clusters created by using OPTICS. During partitioning, MMMP searches for an object to be the best pivot whose partitioning boundary is between clusters and maximizes the distances from the cluster edges. I also created an index, named the MMMP-Index. MMMP is most effective when the indexed data is clustered.

I think that the MMMP has three problems. One problem is the pivot selection cost. The current MMMP reduces the relevance evaluation of an object p to the pivot by using edge objects of the clusters. However, the pivot candidate is not filtered from the data. I should improve the pivot selection method. Another problem is the performance on a few clusters worth of data and for high-dimensional data. The distances between objects in such data are not widely varied. MMMP cannot extract clusters or margins between clusters from them, and doesn't work well in this case. The other problem is the tree balance. The partitioning of MMMP is based on the clustering results; thus, the index tree may be imbalanced. The imbalanced tree increases the average search cost. I have to improve the indexing scheme to resolve these problems.

Chapter 4

PCTree:

**Pivot Selection Scheme for Optimizing
both Pruning and Balancing**

4.1 Overview of PCTree

Here, I present a similarity search index called the Pivot Capacity Tree (PCTree). PCTree automatically optimizes the pivot selection on the basis of the balance of the regions partitioned by a pivot as well as on the basis of the estimated effectiveness of the pruning achieved by the pivot. As a result, PCTree reduces the search cost for various data distributions.

As I mentioned in Chapter 2, the methods of selecting pivots and dividing up the space by using these pivots determine the index structure and pruning performance. The existing pivot selection studies have mainly focused on increasing the number of pruned objects at a branch in the tree [BO99]. However, most previous studies do not take into account the balance of the tree. Tree balancing techniques are used in database management systems (DBMSs), and it is known that the tree height reduction contributes to reducing the average search cost [AVL62]. Thus, even if the pivots of the methods are enough good for pruning objects at the branch, the effect of the search cost reduction may be limited by the data distribution of a database. Therefore, I focused on developing a pivot selection method for optimizing both the pruning and the balance.

The main contributions of this work are as follows.

- I devised a new information theoretic criterion called *Pivot Capacity (PC)* for pivot selection. PC takes into account both the object pruning effect and index tree balance.
- I developed a metric space index called the *Pivot Capacity Tree (PCTree)*. PCTree provides the necessary functions for constructing an index tree based on PC and for efficiently retrieving similar objects by using the index tree.
- I demonstrated the efficiency of PCTree empirically through several experiments where I compared PCTree with several metric space indexes using synthetic multi-dimensional data and five real datasets.

I experimentally compared PCTree with four indexes using synthetic data and five real datasets. The experimental results show that PCTree reduces the search cost in comparison with other indexes.

The rest of this chapter is organized as follows. Section 4.3 describes PC and PCTree. Section 4.4 discusses the experimental results. I summarize the chapter in Section 4.5.

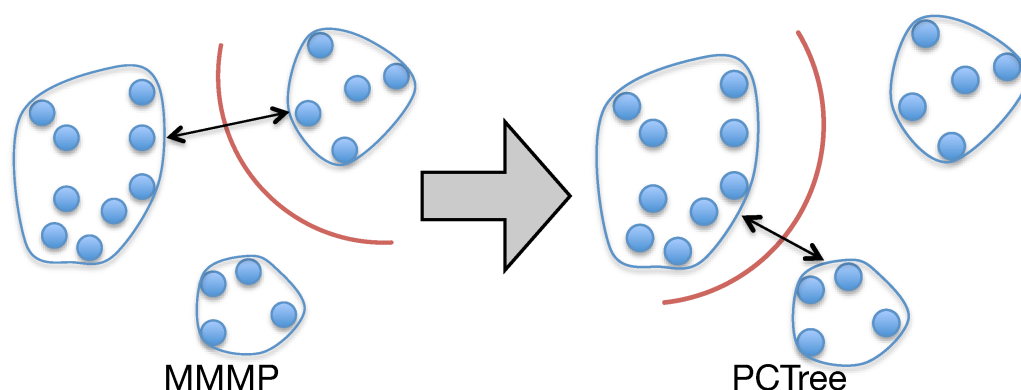


Figure 4.1: MMMP and PCTree

4.2 Improvement on the MMMP

I improved MMMP and resolves two problems affecting it. One problem relates to the tree balance. MMMP aims at selecting a pivot and its partitioning boundary based on the clusters in the dataset. Although it can select better pivots for pruning than the simple statistics-based approach [Yia93; BO99] and the clustering-based approach [JOT⁺03] can, it does not consider the tree balance. An imbalanced index tree increases the average search cost. Thus, I aimed at reducing the search cost by taking into account both object pruning and the tree balance. Figure 4.1 shows an example of the difference between MMMP and PCTree.

The other problem is the dependence on density-based clustering. MMMP does not work well when the number of clusters in the data space is very small or the size of any one cluster is very large. The number of objects managed by a pivot affects the performance of the index. Accordingly, I try to extract even more information concerning the data distribution from the dataset.

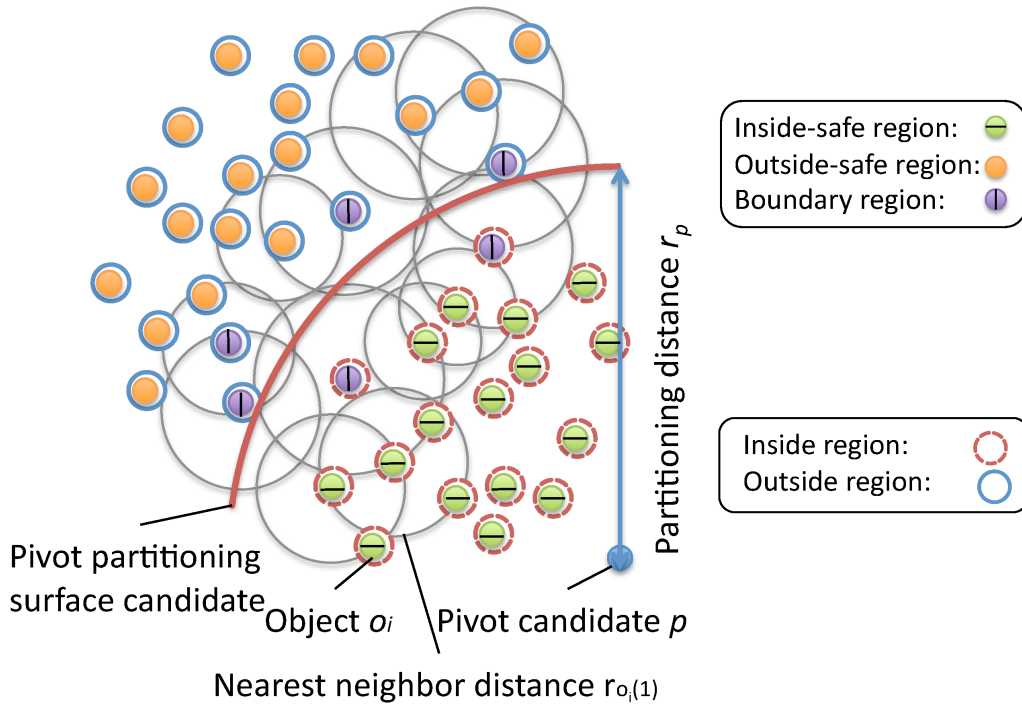


Figure 4.2: Labels for measuring the PC

4.3 PCTree

This section describes PCTree, a similarity search index for a metric space. The index is designed for nearest neighbor searches and aims at reducing the search cost for various data distributions. It selects pivots that have an effect on pruning objects as well as on balancing the index tree. It classifies the space with two criteria for measuring a pivot (Figure 4.2).

4.3.1 Pivot Capacity

The Pivot Capacity (PC) represents the expected index performance for a given query distribution. In defining this capacity, I assume that the queries and objects in the database are drawn from the same probability distribution.

For a metric space $M = (D, d)$ and a region $R \subseteq D$ in the space, suppose that a pivot p and its partitioning distance r_p divide R into two subregions:

$$\begin{aligned} R_1(p, r_p) &= \{o \in R \mid d(o, p) \leq r_p\}, \\ R_2(p, r_p) &= \{o \in R \mid d(o, p) > r_p\}. \end{aligned} \quad (4.1)$$

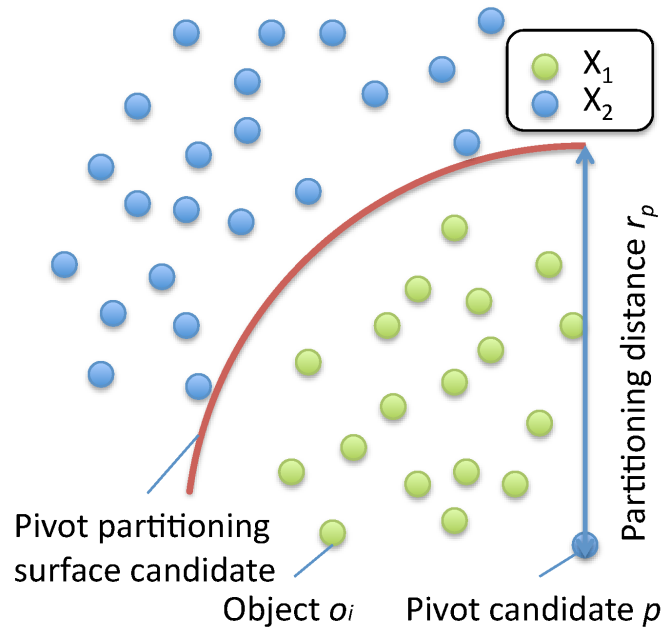


Figure 4.3: Inside Region and Outside Region

I respectively refer to $R_1(p, r_p)$ and $R_2(p, r_p)$ as the *inside region* and *outside region* with respect to pivot p and distance r_p . Figure 4.3 shows the concept of the inside region and the outside region.

For a query q , let X be the random variable that represents the region in which q is included, that is,

$$X = X_{i,p,r_p} \quad \text{if } q \in R_i(p, r_p) \quad (i = 1, 2). \quad (4.2)$$

I call $P(X)$ a *partitioning distribution*. Since I assume a query is drawn from the same distribution as the database, I can estimate the partitioning distribution as follows:

$$P(X_{i,p,r_p}) = \frac{|S_{R_i(p,r_p)}|}{|S_R|} \quad (i = 1, 2), \quad (4.3)$$

where S_R denotes the sets of database objects that are in region R .

Similarly, I define the *pruning distribution* of the query. For a set S of objects in the database and an object o in the region R , let $r_{o,k,S}$ denote the distance between o and o 's k th

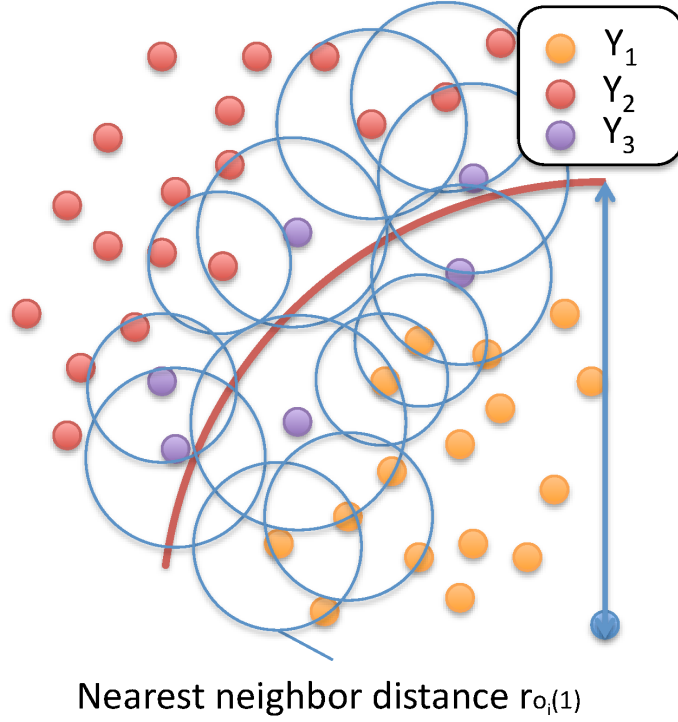


Figure 4.4: Inside-safe Region, Outside-safe Region, and Boundary Region.

nearest neighbor in S . Let us consider the following three regions:

$$R'_1(p, r_p, k) = \{o \in R \mid d(o, p) + r_{o,k,S} \leq r_p\}, \quad (4.4)$$

$$R'_2(p, r_p, k) = \{o \in R \mid d(o, p) - r_{o,k,S} > r_p\}, \quad (4.5)$$

$$R'_3(p, r_p, k) = R - R'_1(p, r_p, k) - R'_2(p, r_p, k). \quad (4.6)$$

Intuitively, $R'_1(p, r_p, k)$ is the set of objects whose k -nearest neighbor objects are within $R_1(p, r_p)$. This means that, if a query q belongs to the region $R'_1(p, r_p, k)$, we can prune $R_2(p, r_p)$ when executing the k -nearest neighbor search. Similarly, $R'_2(p, r_p, k)$ is the set of the objects whose k -nearest neighbor objects are within $R_2(p, r_p)$. I respectively refer to $R'_1(p, r_p, k)$, $R'_2(p, r_p, k)$, and $R'_3(p, r_p, k)$ as the *inside-safe region*, *outside-safe region*, and *boundary region* w.r.t the pivot p , the distance r_p , and the number k of nearest neighbors. Figure 4.4 shows the three regions. Let Y be a random variable that represents the region in which the k -nearest neighbor ranges of q is included, that is,

$$Y = Y_{i,p,r_p,k} \quad \text{if } q \in R'_i(p, r_p, k) \quad (i = 1, 2, 3). \quad (4.7)$$

I call $P(Y)$ a *pruning distribution*. I estimate the distribution as follows:

$$P(Y_{i,p,r_p,k}) = \frac{|S_{R'_i(p,r_p,k)}|}{|S_R|} \quad (i = 1, 2, 3). \quad (4.8)$$

By using the two random variables X and Y , the PC for a pivot p is defined as

$$\begin{aligned} PC_k(p) &\equiv \max_{r_p} I(X; Y) \\ &= \max_{r_p} \left(\sum_i \sum_j \left(P(X_{i,p,r_p}, Y_{j,p,r_p,k}) \cdot \log \left(\frac{P(X_{i,p,r_p}, Y_{j,p,r_p,k})}{P(X_{i,p,r_p})P(Y_{j,p,r_p,k})} \right) \right) \right), \end{aligned} \quad (4.9)$$

where $I(\cdot; \cdot)$ is the mutual information. I choose the object p (resp. distance r_p) that maximizes Equation (4.9) as a pivot (resp. partitioning distance).

The mutual information satisfies

$$I(X; Y) = H(X) - H(X | Y) \leq H(X), \quad (4.10)$$

$$I(X; Y) = H(Y) - H(Y | X) \leq H(Y), \quad (4.11)$$

where $H(\cdot)$ is the entropy. PC represents the mutual information of the random variables for the partitioning and pruning distributions. I define the PC by using Equation (4.10). $H(X)$ is the entropy of the random variable X for the partitioning distribution. It represents the balance of the partitioned regions. That is, the index tree with a larger entropy is more balanced and is closer to being a complete binary tree. A balanced tree is good for reducing the average search cost. $H(X | Y)$ represents the entropy of the probability X under the condition Y , where Y is the random variable for the pruning distribution. The pruning distribution classifies a region into three subregions. For a query in the inside-safe region (resp. the outside-safe region), the pivot can prune the outside region (resp. the inside region). For a query in the boundary region, the pivot cannot prune objects. From the definition, the inside-safe region (resp. the outside-safe region) is included in the inside region (resp. the outside region). The boundary region is part of the inside and the outside region. Thus, when the boundary region gets smaller, $H(X | Y)$ decreases. I use $H(X|Y)$ for estimating the effectiveness of the pruning. I define the effectiveness of both balancing and pruning in terms of entropy, and I do not weight these entropies. The criterion for the PC is

inspired by the formula of the channel capacity of a binary erasure channel in information and coding theory [JJ00].

Let us define the following three variables:

$$c \equiv \frac{|S_{R_1(p,r_p)}|}{|S|}, \quad (4.12)$$

$$1 - p_1 \equiv \frac{|S_{R'_1(p,r_p,k)}|}{|S_{R_1(p,r_p)}|}, \quad (4.13)$$

$$1 - p_2 \equiv \frac{|S_{R'_2(p,r_p,k)}|}{|S_{R_2(p,r_p)}|}, \quad (4.14)$$

where c means the balance of the partition and $(1 - p_1)$ and $(1 - p_2)$ are the pruning performances of regions R_1 and R_2 . The mutual information used for PC is described as follows:

$$\begin{aligned} I(X; Y) &= -c(1 - p_1) \log c - (1 - c)(1 - p_2) \log (1 - c) \\ &\quad -cp_1 \log \frac{cp_1 + (1 - c)p_2}{p_1} \\ &\quad - (1 - c)p_2 \log \frac{cp_1 + (1 - c)p_2}{p_2}. \end{aligned} \quad (4.15)$$

4.3.2 Construction

PCTree has a tree consisting of *internal nodes* and *leaves*. An internal node is associated with one pivot and its partitioning distance whereas a leaf node is associated with one pivot and the objects. This subsection outlines the index construction process. The following subsections describe some of the steps in more detail.

In the indexing phase, PCTree recursively divides a region into its inside and outside subregions by using pivots. For a metric space $M = (D, d)$ and a set S of objects, the PCTree constructs an index by

1. selecting a pivot candidate set S_p from S (Section 4.3.4 for details), and
2. dividing D recursively into inside and outside regions.

PCTree selects the most effective pivot candidates S_p because a database usually contains large amount of objects and calculating a PC for all these objects is infeasible. Note that

this pivot candidate selection is done once before constructing an index.

For a region R and set of objects S , PCTree recursively executes the following procedure $\mathcal{P}(R)$:

1. if the stopping condition described in Section 4.3.3 is satisfied,
 - (a) randomly choose a pivot p from S_R ,
 - (b) make a sorted list L of objects in S_R according to the distance from p , and
 - (c) return a leaf node associated with p and L
2. otherwise
 - (a) randomly choose a predefined number m of samples T from S_R ,
 - (b) select the pivot $p \in S_p$ and its partitioning distance r_p which maximize the PC given by Equation (4.9) for the random samples T ,
 - (c) call the region partitioning function with the inside and the outside regions w.r.t. p and r_p , and obtain the respective node $v_1 := \mathcal{P}(R_1(p, r_p))$ and $v_2 := \mathcal{P}(R_2(p, r_p))$, and
 - (d) return an internal node v associated with the pivot p , partitioning distance r_p and child nodes v_1, v_2 .

PCTree uses random samples T instead of R_S in step 2(a) to reduce the computational cost of pivot selection.

Algorithms 3 and 4 respectively show the steps of pivot selection and indexing. In these algorithms, $\text{RandomSamples}(S, m)$ is a function for sampling m objects from S . $\text{CalcKNN}(T, k, S)$ is a function for computing the k -nearest neighbor range in S for each object in T . $\text{MutualInformation}(X;Y)$ is as in Equation (4.15). $\text{MinimumPC}(T)$ is as in Equation (4.16) in Section 4.3.3. $\text{PivotCandidateSample}(S)$ is the procedure described in Section 4.3.4.

4.3.3 Condition for Stopping the Partition

If we do not use a similarity search index, the linear scan method requires $O(N)$ distance computations, where N is the number of objects in the database [Use the rewrite only if

it is correct. If it is correct, the original was redundantly written]. Therefore, we should not use PCTree if its estimated number of distance calculations is larger than those of the naive sequential access method. Thus, I define the minimum PC for filtering inappropriate pivot candidates that have less of an effect on the search than the naive method has. For measuring the PC of the pivots in the linear scan method, we can consider all the objects in the dataset to be pivots and each pivot prunes itself while searching. Therefore, I set the minimum PC as

$$\text{MinimumPC}(S) = -\frac{1}{|S|} \cdot \log \frac{1}{|S|} - \frac{|S| - 1}{|S|} \cdot \log \frac{|S| - 1}{|S|}, \quad (4.16)$$

where S is the object set. When the PCs of the pivot candidates in a region are less than the minimum PC, the partitioning stops and the region is set as a leaf node in the tree. For example, when $|S|$ is 100, $P(X_{1,p,r_p})$ and $P(X_{2,p,r_p})$ are 0.5, $P(Y_{1,p,r_p,k})$ and $P(Y_{3,p,r_p,k})$ are 0.04, and $P(Y_{3,p,r_p,k})$ is 0.92, the partitioning stops because the PC is about 0.080 and the minimum PC is about 0.081.

4.3.4 Sampling the Pivot Candidates

The indexing algorithm of PCTree is a greedy algorithm. It incurs a heavy construction cost for finding the optimal pivot and its partitioning distance. I reduce the cost by sampling the pivot candidates.

I select pivot candidates such that they are separated from each other by a certain distance d_p . I decided to sample the pivot candidates for the following reasons. For pivot candidates p_1 and p_2 , the distance from p_2 to an object o satisfies

$$|d(p_1, o) - d(p_2, o)| \leq d(p_1, p_2). \quad (4.17)$$

From the inequality, we can expect that the larger $d(p_1, p_2)$ is, the larger the difference between $d(p_1, o)$ and $d(p_2, o)$ will be. Thus, I choose objects with the following two policies:

- the pivot candidates are separate from each other in the space, and
- the pivot candidates are not selected from the partitioned region but from all the objects in the dataset.

Therefore, one should choose a pivot candidate set such that the distance between any pivot pair in the set is more than $d_{\text{ave}} \cdot s$, where d_{ave} is the approximate average distance between objects and s is a parameter. The approximate average distance is calculated by using randomly sampled objects. Pivot candidates are sampled from the data set by performing the following steps:

1. Let S_p , the pivot candidate set, be an empty set. Insert all the objects in the data set into the object set S .
2. Randomly select an object p from the object set S and add p to S_p .
3. While S is not empty:
 - Remove all objects from S whose distances to p are less than $d_{\text{ave}} \cdot s$.
 - Select the object p' that is nearest to p , add p' to S_p , and set p' to p .

The cost is at most $O(N \cdot n)$, where N (resp. n) denotes the number of objects in the dataset (resp. pivot candidates).

4.3.5 Indexing Cost

The indexing cost consists of the pivot selection cost and the PC calculation cost. The pivot selection needs to be done once for an index. On the other hand, the PC calculation needs to be done for every partition.

As shown in Section 4.3.4, the pivot candidate selection requires $O(N \cdot n)$ distance computations.

The required computations for a partition at a node are as follows. For an internal node v , let S_v denote the set of objects managed by the subtree rooted at v , and T_v denote the randomly sampled m objects from S_v . As shown in Section 4.3.1, two distributions are needed for measuring the PC. The distance from each object in T_v to its k th nearest neighbor object in S_v and the distances from each pivot candidate in S_p to the objects in T_v are needed for measuring the distributions at a node. The former distance requires at most $O(|S_v| \cdot |T_v|)$ computations. The latter requires $O(|T_v| \cdot |S_p|)$ computations. The total cost for the partitions $\text{Cost}_{\text{partition}}$ is

$$\text{Cost}_{\text{partition}} = \sum_v (|S_v| + |S_p|) \cdot |T_v|. \quad (4.18)$$

The indexing cost depends on the data distribution and the sampling parameters. The maximum number of nodes is $O(N)$. Thus, the indexing cost is at most $O(N^2)$.

4.3.6 k-Nearest Neighbor Search and Range Search

PCTree deals with range queries and k -nearest neighbor queries.

For the k -Nearest Neighbor search, PCTree receives a query q and the number k of results. The search procedure is almost the same as for VPT [Yia93] and uses the following steps:

1. create an empty set as a result set S_r and set the query range r_q to ∞ .
2. traverse from the root node in the PCTree and recursively access the nodes in a depth-first way:
 - (a) if the node is a leaf,
 - i. find the objects whose distances to q are within r_q in the node while using the object-pivot distance constraint [ZADB05], and add them to S_r .
 - ii. update r_q to be the k -nearest neighbor radius of q in S_r .
 - (b) otherwise
 - i. read the pivot p and its partitioning distance r_p associated to the node.
 - ii. if the inequality $d(p, q) \leq r_p$ is satisfied,
 - A. access the child node for the inside region and update r_q .
 - B. After this, if the inequality $d(p, q) + r_q > r_p$ is satisfied, access the other child node and update r_q .
 - iii. if the inequality $d(p, q) \geq r_p$ is satisfied,
 - A. access the child node for the outside region and update r_q .
 - B. if the inequality $d(p, q) - r_q \leq r_p$ is satisfied, access the other child node and update r_q .
3. answer the k closest objects.

When the balancing and the pruning of the index work best, the search cost is $O(\log N)$.

Algorithm 5 shows the steps in executing a k -nearest neighbor query.

A range query of a query object q and a query radius r_q is executed as follows:

1. create an empty set as a result set S_r .
2. traverse from the root node in the PCTree and recursively accesses the nodes in the depth-first way:
 - (a) if the node is a leaf,
 - i. find the objects whose distances to q are within r_q in the node while using the object-pivot distance constraint [ZADB05], and add them to S_r .
 - (b) otherwise
 - i. read the pivot p and its partitioning distance r_p associated to the node.
 - ii. if the inequality $d(p, q) \leq r_p$ is satisfied,
 - A. access the child node for the inside region and update r_q .
 - iii. if the inequality $d(p, q) \geq r_p$ is satisfied,
 - A. access the child node for the outside region and update r_q .
3. answer the result set S_r .

Algorithm 6 shows the steps in executing a range query.

Algorithm 3 PivotSelection

```

Function PivotSelection(ObjectSet  $S$ , ObjectSet  $S_p$ )
   $T \leftarrow$  RandomSamples( $S$ ,  $m$ )
  CalcKNN( $T$ ,  $k$ ,  $S$ )
   $PC \leftarrow 0$  ;  $D \leftarrow \text{nil}$  ;  $P \leftarrow \text{nil}$  ;
  for candidate  $p$  in  $S_p$  do
     $B \leftarrow$  empty array
    for object  $o_i$  in  $S$  do
      push a twin ( $\max(0, d(p, o_i) - r_{o_i(k)}, 0)$ ) into  $B$ 
      push a twin ( $d(p, o_i), 1$ ) into  $B$ 
      push a twin ( $d(p, o_i) + r_{o_i(k)}, 2$ ) into  $B$ 
    end for
    sort  $B$ 
     $\text{prev} \leftarrow -\infty$ 
     $e \leftarrow 0$  ;  $x_1 \leftarrow 0$  ;  $x_2 \leftarrow |T|$  ;  $y_1 \leftarrow 0$  ;  $y_4 \leftarrow |T|$  ;
    for twin  $(x, y)$  in  $B$  do
      if  $x_1 > 0$  and  $x_2 > 0$  and ( $y_1 > 0$  or  $y_4 > 0$ ) and  $x$  is not  $\text{prev}$  then
         $c \leftarrow \frac{x_1}{|T|}$  ;  $p_1 \leftarrow 1 - \frac{y_1}{x_1}$  ;  $p_2 \leftarrow 1 - \frac{y_4}{x_2}$  ;
         $\text{eval} \leftarrow$  MutualInformation( $X; Y$ ) ;
        if  $\text{eval} >$  MinimumPC( $T$ ) and  $\text{eval} >$   $PC$  then
           $PC \leftarrow \text{eval}$  ;  $D \leftarrow \frac{x + \text{prev}}{2}$  ;  $P \leftarrow p$  ;
        end if
      end if
      if  $y=0$  then
        Increment  $e$  ; Decrement  $y_4$  ;
      else if  $y=1$  then
        Increment  $x_1$  ; Decrement  $x_2$  ;
      else
        Decrement  $e$  ; Increment  $y_1$  ;
      end if
       $\text{prev} \leftarrow x$ 
    end for
  end for
  return ( $D, P$ )

```

Algorithm 4 BuildTree

Function BuildTreeSub(ObjectSet S , ObjectSet S_p) $(D, P) \leftarrow \text{PivotSelection}(S, S_p)$ **if** $P = \text{nil}$ **then** $b \leftarrow \text{MakeBucket}$ $p \leftarrow \text{RandomSamples}(S, 1)$ **for** p in S **do** push a twin $(d(p, o_i), o_i)$ into b sort b **end for** node.pivot $\leftarrow p$ node.bucket $\leftarrow b$ node.flag $\leftarrow 1$ **return** node**else** node.pivot $\leftarrow P$ node.dist $\leftarrow D$ node.in $\leftarrow \text{BuildTreeSub}(\{o_i \in S \mid d(o_i, P) \leq D\}, S_p)$ node.out $\leftarrow \text{BuildTreeSub}(\{o_i \in S \mid d(o_i, P) > D\}, S_p)$ **end if****return** node**Function BuildTree(ObjectSet S)** $S_p \leftarrow \text{PivotCandidateSample}(S)$ Tree $\leftarrow \text{BuildTreeSub}(S, S_p)$ **return** Tree

Algorithm 5 kNN Search

Function kNNSearchSub(Node node, Query q , Neighbor k , Result R)**if** node.flag = 1 **then** R.obj.add(SearchBucketNN(node.bucket, $d(\text{node.query}, q)$), R.dist) **if** $|R.obj| > k$ **then** SizeRefine(R.obj, k) **end if** **if** $|R.obj| = k$ **then**

R.dist ← MaxDist(R.obj)

end if**else** dist ← $d(\text{node.query}, q)$ **if** dist ≤ node.dist **then** kNNSearchSub(node.in, q, k, R) **if** Size(R.obj) < k or dist + R.dist ≥ node.dist **then** kNNSearchSub(node.out, q, k, R) **end if** **else** kNNSearchSub(node.out, q, k, R) **if** Size(R.obj) < k or dist - R.dist < node.dist **then** kNNSearchSub(node.in, q, k, R) **end if** **end if****end if****return****Function** kNNSearch(Node node, Query q , Neighbor k , Result r)

Result.obj ← empty

Result.dist ← ∞ **return** kNNSearchSub(node, q, k, Result)

Algorithm 6 Range Search

Function RangeSearchSub(Node node, Query q , Range r_q)Result \leftarrow empty**if** node.flag = 1 **then** Result.add(SearchBucket(node.bucket, $d(\text{node.pivot}, q)$))**else** dist $\leftarrow d(\text{node.query}, q)$ **if** dist $-r_q \leq \text{node.dist}$ **then** Result.add(RangeSearch(node.in, q, r_q)) **end if** **if** dist $+r_q > \text{node.dist}$ **then** Result.add(RangeSearch(node.out, q, r_q)) **end if****end if****return** Result**Function RangeSearch(Index Tree, Query q , Range r_q)****return** RangeSearchSub(Tree, q, r_q)

4.4 Performance Evaluation

4.4.1 Outline of Experiments

I implemented PCTree on the Metric Space Library [Met]. The library is written in C, and provides several indexing algorithms, metric spaces, and datasets. I compared PCTree with GHT [Uhl91], MVP [Yia93; BO99], LC [CN05], and SAT [Nav02] in the library. As in the related work [Nav02], the performance of these indexes was evaluated by the number of the distance computations divided by the total number of objects in the datasets. I conducted the experiments on a Linux PC equipped with an Intel(R) Quad Core Xeon(TM) X5492 3.40GHz CPU and that had 64GB of memory. The library and my codes were compiled with GCC 4.2. All the indexes fit in the memory.

4.4.2 Data Set

I used two types of datasets to evaluate my scheme: vectors and strings. The vector datasets consisted of synthetic vectors and four real vector datasets. The string dataset contained English words. I used the Euclid distance for the vector datasets and Levenshtein distance for the string dataset.

Synthetic vectors were generated in 2, 4, 8, 16, 32, and 64-dimensional feature spaces according to uniform and Gaussian mixture distributions. Hereafter, I refer to a dataset consisting of vectors in n -dimensional space generated according to a uniform (resp. Gaussian mixture) distribution as *nd-uniform vectors* (resp. *nd-clustered vectors*). As for the nd -clustered vectors, I used 100-component Gaussian distributions in the following experiments, if not explicitly mentioned otherwise. The mean vector of the component Gaussian distribution was randomly selected, whereas the covariance matrixes were a diagonal matrix whose diagonal components were set to 0.02. The number of objects for each cluster was randomly chosen. The objects themselves were based on a Gaussian. By referring to the previous study [CN05], I chose the size of the data to be 100,000. I randomly chose 1,000 queries from the same distribution as the dataset. When the chosen query happened to be the same point in the dataset, it was discarded. I shall use the average number of distance calculations over these 1,000 queries as the search cost in the following discussion.

NASA is a set of feature vectors made by NASA [Met]. It consists of 40,150 vectors in a 20-dimensional feature space.

Color histograms are the color histograms of 112,544 images represented by vectors in 112-dimensional space [Met].

Corel Image Features contains color histogram vectors generated from the Corel image collection [UCI]. It consists of 68,040 vectors in a 32-dimensional space.

English dictionary consists of 69,069 English words in the form of strings, and it was in the library [Met].

flickr is a set of 2,000,000 images that I downloaded from flickr. I extracted 960-dimensional features from each image by using a GIST image descriptor [OT01] provided by [gis]. This dataset is available from [flia].

For each real dataset, I randomly selected 1,000 objects for the queries and conducted the k -nearest neighbor queries on the remaining objects. All the real datasets were taken from the Web. Table 4.1 lists the details of the datasets.

4.4.3 Synthetic Data

Parameter Tuning

PCTree requires three parameters during indexing. One parameter k is for calculating the PC in Equations (4.4), (4.5) and (4.6), and the other parameters s and m are for sampling as described in Section 4.3.2 and Section 4.3.4. I evaluated how varying the sampling parameters affect the search cost by using the $8d$ -uniform, $8d$ -clustered, and $16d$ -clustered vectors. I also evaluated how the sampling method affects the search cost by using $2d$, $4d$, $8d$, $16d$, $32d$, and $64d$ -clustered vectors.

The results are shown in Figure 4.5. The vertical axes show the cost for the 1-nearest neighbor search represented by the relative distance computation. Figure 4.5 (a) includes the cost for the 5-nearest neighbors as well. The horizontal axes are the parameters. The reference search cost was that for $k = 1$, $s = 0.8$, and $m = 500$, and the plots show the ratio of each search cost to the reference search cost..

Table 4.1: Data Sets

Data	Distance	No. of data	No. of queries	Data source	Dimension	Variance	No. of Clusters
Vectors	Euclid distance	100,000	1,000	synthetic	2	0.02	100
					2	Uniform	-
					4	0.02	100
					4	Uniform	-
					8	0.02	100
					8	Uniform	-
					16	0.02	100
					16	Uniform	-
					32	0.02	from 10 to 200
					32	Uniform	-
NASA	Euclid distance	39,150	1,000	Metric Space Library [Met]	20	-	-
					112	-	-
Color histograms Corel	Euclid distance	111,544	1,000	Metric Space Library [Met]	32	Uniform	-
					64	0.02	100
Image Features flickr	Euclid distance	67,040	1,000	UCI KDD Archive [UCI]	32	-	-
					960	-	-
		2,000,000	1,000	flickr [flia]	960	-	-

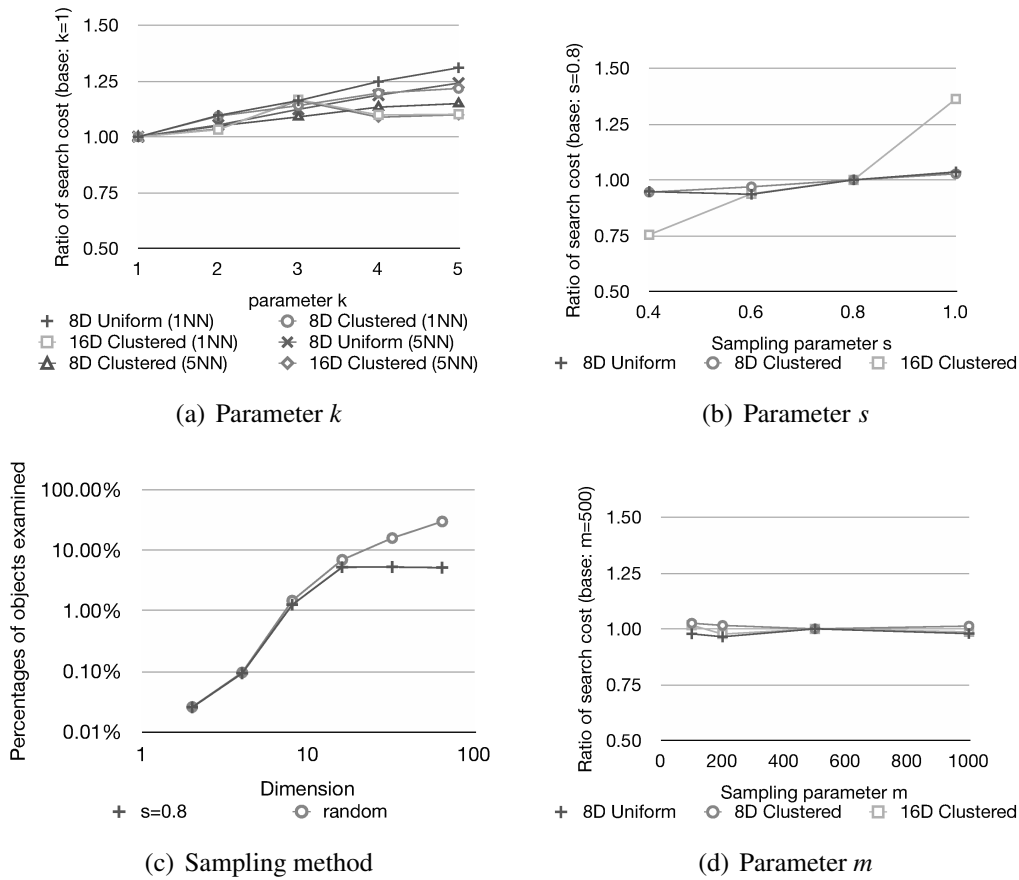


Figure 4.5: Search Cost w.r.t. Sampling Parameters and Sampling Methods

As we can see in Figure 4.5 (a), a smaller k is better for all the vectors. Figure 4.5 (b) shows the search cost w.r.t the parameter s that controls the candidate pivot size (see Section 4.3.4). Intuitively, a smaller s means more candidates are selected, and consequently, the query cost will be lower. However, a smaller s requires more computations for constructing the index. As we can see in the figure, s has the largest influence on the search cost. This indicates that more pivot candidates are better for PCTree. Figure 4.5 (c) shows the search cost of two different sampling methods. I compared my method (see Section 4.3.4) using $s = 0.8$ with random sampling. We can see that my sampling technique outperforms random sampling on higher dimensional vectors. Figure 4.5 (d) shows the search cost w.r.t the parameter m that determines the sample size when estimating the Pivot Capacity described in Section 4.3.2. As we can see in the figure, m only slightly affects the search cost and 500 samples seems to be sufficient.

From these experiments, I decided to set k , s , and m to 1, 0.8, and 500, respectively, in the index performance evaluations.

Dataset size

I evaluated the index structure and the performance with respect to the number of objects on $2d$, $4d$, $8d$, $16d$, and $32d$ -clustered vectors.

Figure 4.6 (a) shows the search cost of PCTree. Figure 4.6 (b) compares the search cost for $8d$ -clustered vectors. As in the related study [Nav02], the search cost was evaluated as the percentage of objects examined. The vertical axes represent the cost for the 1-nearest neighbor queries. The horizontal axes are the number of objects.

The results indicate that the percentage of objects examined decreases as the dataset increases. The search cost of PCTree is sublinear with respect to the number of object. Furthermore, PCTree outperforms the other methods at all object numbers. Figure 4.6 (c) shows how the dataset size affects the pivot candidate selection. The vertical axis represents the number of pivot candidates. The horizontal axis is the number of objects. We can see that the number of pivot candidates does not vary with the dataset size. The dimension has more influence on it.

Figures 4.6 (d), (e) , and (f) show the index construction cost, the number of nodes, and the index height, respectively. The construction cost was evaluated by using the number of distance computations per object during indexing. Since PCTree does not cache calculated distances, the distances between two objects may be calculated more than once. As we can see in the figure of the index construction cost, the number of required distance calculations is almost sublinear to the number of objects. From the figures of the index structure, we can see that both the number of nodes and index height are almost sublinear to the number of objects. I interpret these results to mean that the index tree is imbalanced for clustered vectors and the index does not increase linearly with respect to the number of objects. The index cost of PCTree is at most $O(N^2)$ (see Section 4.3.5), and the actual cost is substantially smaller than that. However, as I said, the coefficient is large and the indexing takes a long time. One reason is that all the PC calculations use the same pivot candidate set and the same sampling parameters in the PCTree construction. I have to improve the sampling and the indexing algorithm and reduce the indexing cost.

Dimension

I evaluated the index structure and the search cost with respect to the number of dimensions. The experiment used $2d$, $4d$, $8d$, $16d$, $32d$, and $64d$ -clustered vectors as well as uniform vectors in each dimensional space in this experiment.

Figures 4.7 (a) and (b) show the search costs of the clustered and uniform vectors, respectively. The vertical axes represent the number of distance computations for the 1-nearest neighbor queries whereas the horizontal axes are the number of dimensions. We can see that PCTree, GHT, and MVP perform well for lower dimensional vectors regardless of the data distribution. For higher dimensional clustered vectors, PCTree outperforms all other methods. However, the costs of PCTree, GHT, and MVP are large for higher dimensional uniform vectors. In contrast, while LC costs a lot for lower dimensional vectors, its cost is smaller for higher dimensional vectors. PCTree is superior to the other methods, except for the uniformly distributed high dimensional vectors whereby the performance of the PCTree is larger than LC, but is almost the same as those of the other methods.

Figure 4.7 (c) (resp. Figure 4.7 (d)) show the number of nodes and height of the index for clustered (resp. uniform) vectors. The horizontal axes are the numbers of dimensions. The difference between the maximum tree height and average tree height is smaller for lower dimensional vectors. That is, the index trees of the low-dimensional vectors are well-balanced binary trees. On the other hand, the trees of the high-dimensional vectors are imbalanced. The small number of nodes relative to the tree height also indicates the trees of high-dimensional vectors are imbalance. I interpret this result to mean that the PCTree changes the weights of the pruning and balancing according to the data distribution. PCTree did not partition the $64d$ -uniform vectors because the PC of the first partition was less than the minimum PC and hence PCTree judged there was no effective partition for the dataset. The PCTree for the low-dimensional vectors is similar to that of MVP; MVP is a completely balanced tree. The PCTree for the high-dimensional vectors is close to LC; LC is a list and is the most imbalanced index. GHT doesn't use the partitioning distance and its balance depends on only its pivots. Thus, GHT cannot adjust the index balance and is fundamentally different from PCTree. SAT is also different from the PCTree because the fan out of SAT tends to increase for the high-dimensional vectors and consequently its height decreases.

Figure 4.7 (e) shows the index construction cost for clustered vectors. The vertical axis

is the number of distance computations per object during indexing. We can see that PCTree has the largest indexing cost for all the vectors, whereas MVP has the lowest. As shown in Section 4.3.5, the PC calculation entails a lot of distance computations. The cost of the PCTree is larger for higher dimensional vectors. This is because the number of pivot candidates is larger for a higher dimensional vectors (Figure 4.6 (c)).

Number of Clusters

I evaluated the index structure and the search cost with respect to the number of clusters. I used $32d$ -clustered vectors generated by the mixture distributions with 10 to 200 Gaussian models whose variance was 0.02.

Figure 4.8 (a) shows the search cost. The vertical axis represents the number of distance computations for the 1-nearest neighbor queries. The horizontal axis is the number of clusters. PCTree achieved the optimal performance for 50 clusters. This means that the margins between clusters would be large enough until the number of clusters reached 50, and the search cost is smaller for the vectors with more clusters. PCTree could divide up the spaces of vectors with less than 50 clusters, and it could correctly access the small number of regions that were relevant to the query. However, PCTree could not find good partitions for pruning for the vectors with more than 50 clusters because many clusters overlapped each other. It seems that the distribution of the vectors with 200 clusters is similar to that of the uniform vectors. Note that PCTree outperforms the other methods for a wide range of cluster numbers, i.e., numbers less than 200. Even in the case of 200 clusters, it outperformed all the other methods except for LC. Figure 4.8 (b) shows the index structure. We can see the index tree height and the number of nodes increases as the number of clusters increases. This means that the PCTree is likely to find more partitions that satisfy the partition condition defined by the minimum PC for data consisting of a lot of clusters.

4.4.4 Real Data

I compared PCTree with other methods on five real datasets. Figure 4.10 shows the index performances for the real dataset. The vertical axis represents the number of distance computations for the k -nearest neighbor queries. The horizontal axis is k .

Table 4.2: Distance Density Properties of Real Datasets

Data	Distance	Dimension	Average	Variance	Skewness	Kurtosis
Nasa	Euclid distance	20	1.48	0.211	0.0447	2.39
Color histograms	Euclid distance	112	0.415	0.0310	0.828	3.57
Corel Image Features	Euclid distance	32	0.564	0.0332	0.444	3.08
English dictionary	Levenshtein distance	-	8.35	4.10	0.271	3.17
flickr	Euclid distance	960	2.12	0.591	0.972	5.02

Figure 4.9 shows the distance density of each dataset, and Table 4.2 lists the properties of the distances between the objects in each dataset. We note the following features from the figure and table.

- The English dictionary uses the Levenshtein distance whereas the others use the Euclid distance.
- NASA is in the lowest dimensional feature space and flickr is in the highest.
- The skewness of the Color histograms and flickr is high.
- The kurtosis of flickr is the highest.
- NASA has a wide range of distances and its skewness is the lowest.

PCTree in Figure 4.10 outperforms the other methods for the four vector datasets, and it outperforms all the others except for LC for the English dictionary dataset. Compared with the other methods, the PCTree is better than the other methods for a wide range of data distributions. For example, MVP is good for NASA and flickr, but is weak for the English dictionary. On the other hand, LC is good for the English dictionary, but is weak for NASA. GHT and SAT are not good for any of the datasets. The flickr results show that all the indexes find similar objects by examining less than 20 % of the objects in the dataset. I interpret the flickr results and Section 4.4.3 to mean that the metric space indexes can handle a high dimensional dataset even if some of its dimensions are meaningless. To make a conclusion about the relationship between the distribution and the index performance, I need to conduct more experiments and more deeply analyze their results.

I guessed that the PCTree would need more samples for calculating the PC for English dictionary. Therefore, I plotted the search costs for the PCTrees and without sampling in Figure 4.10 (d). The figure shows that the search cost without sampling is close to that of the LC. It is difficult to determine the appropriate parameters without more knowledge about the data distribution. Hence, parameter tuning of the PCTree remains a topic of study.

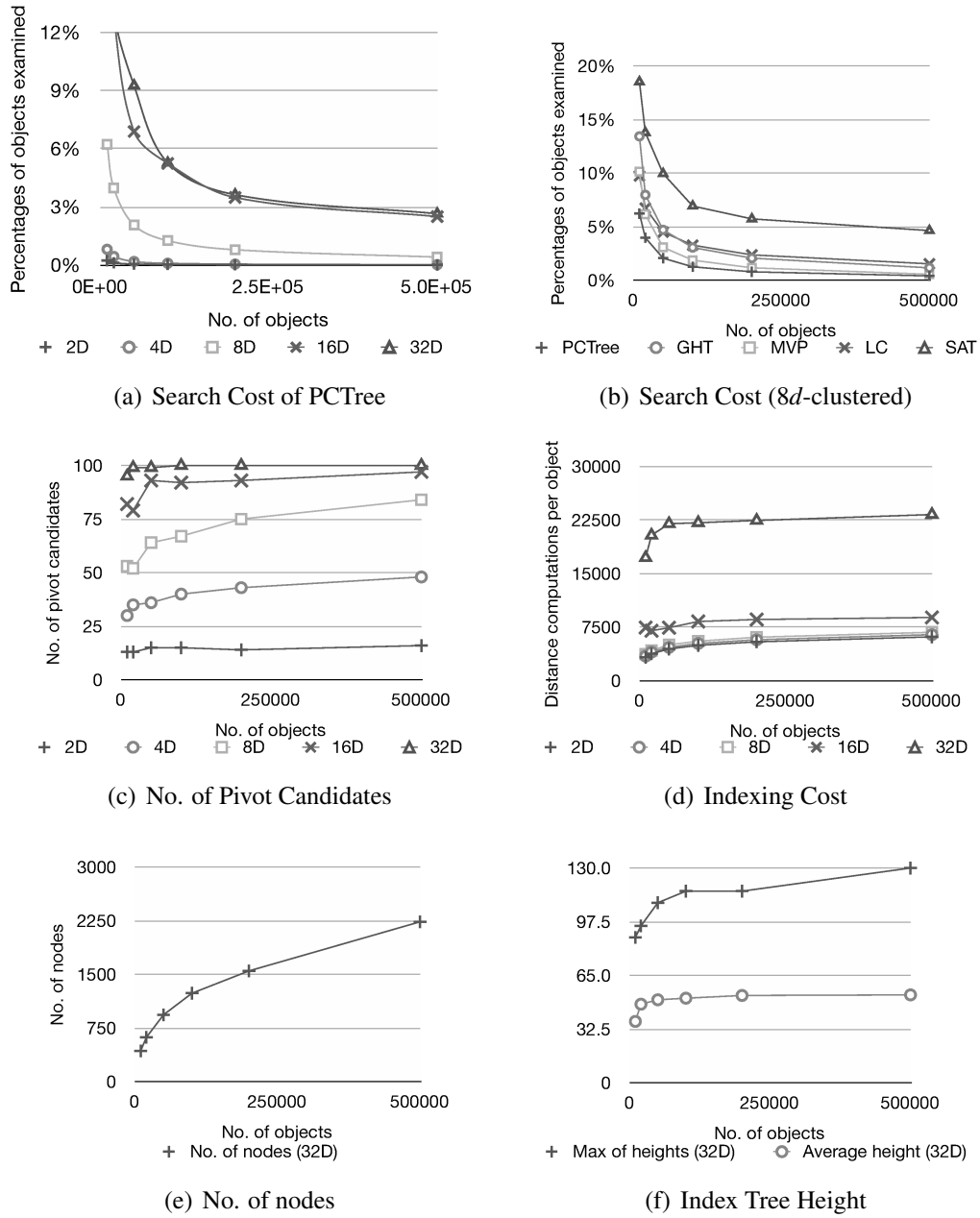


Figure 4.6: Index Performance w.r.t. Number of Objects

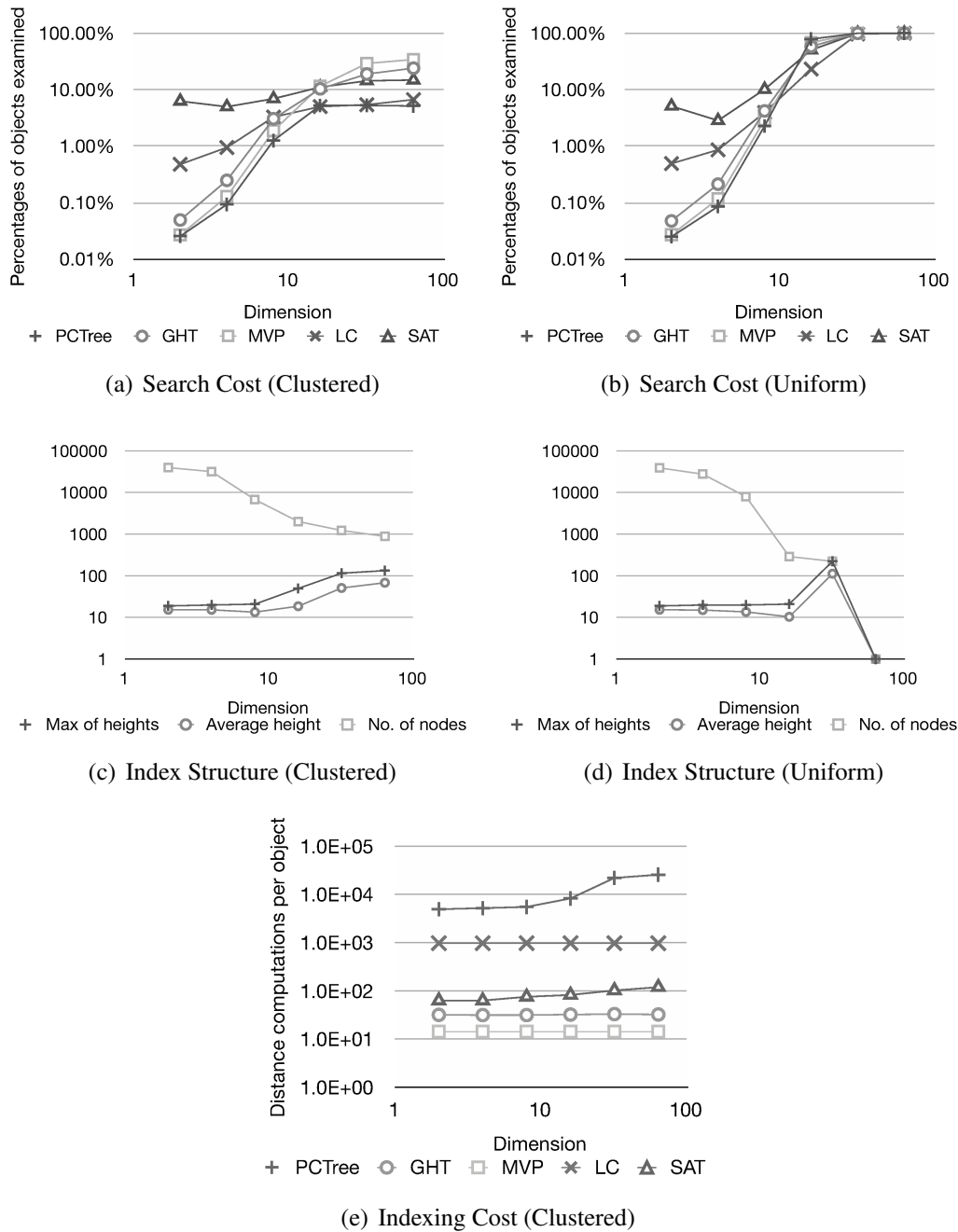


Figure 4.7: Index Performance w.r.t. Number of Dimensions

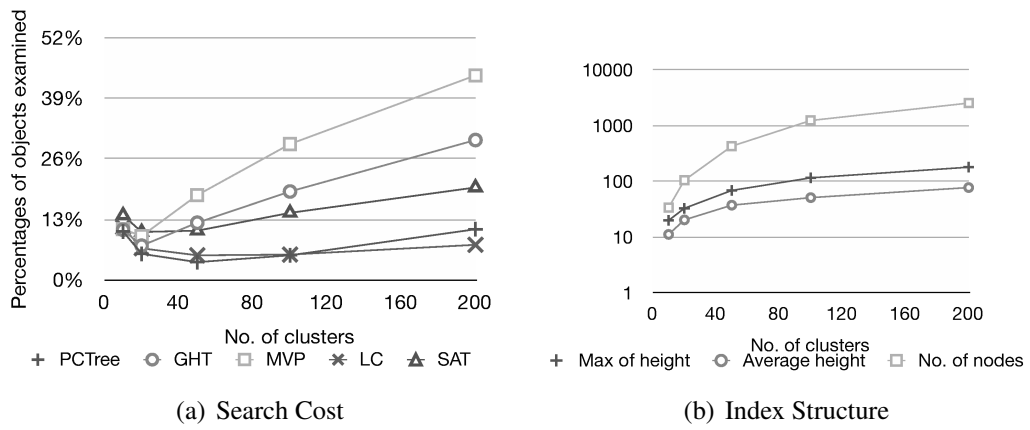


Figure 4.8: Index Performance w.r.t. the Number of Clusters

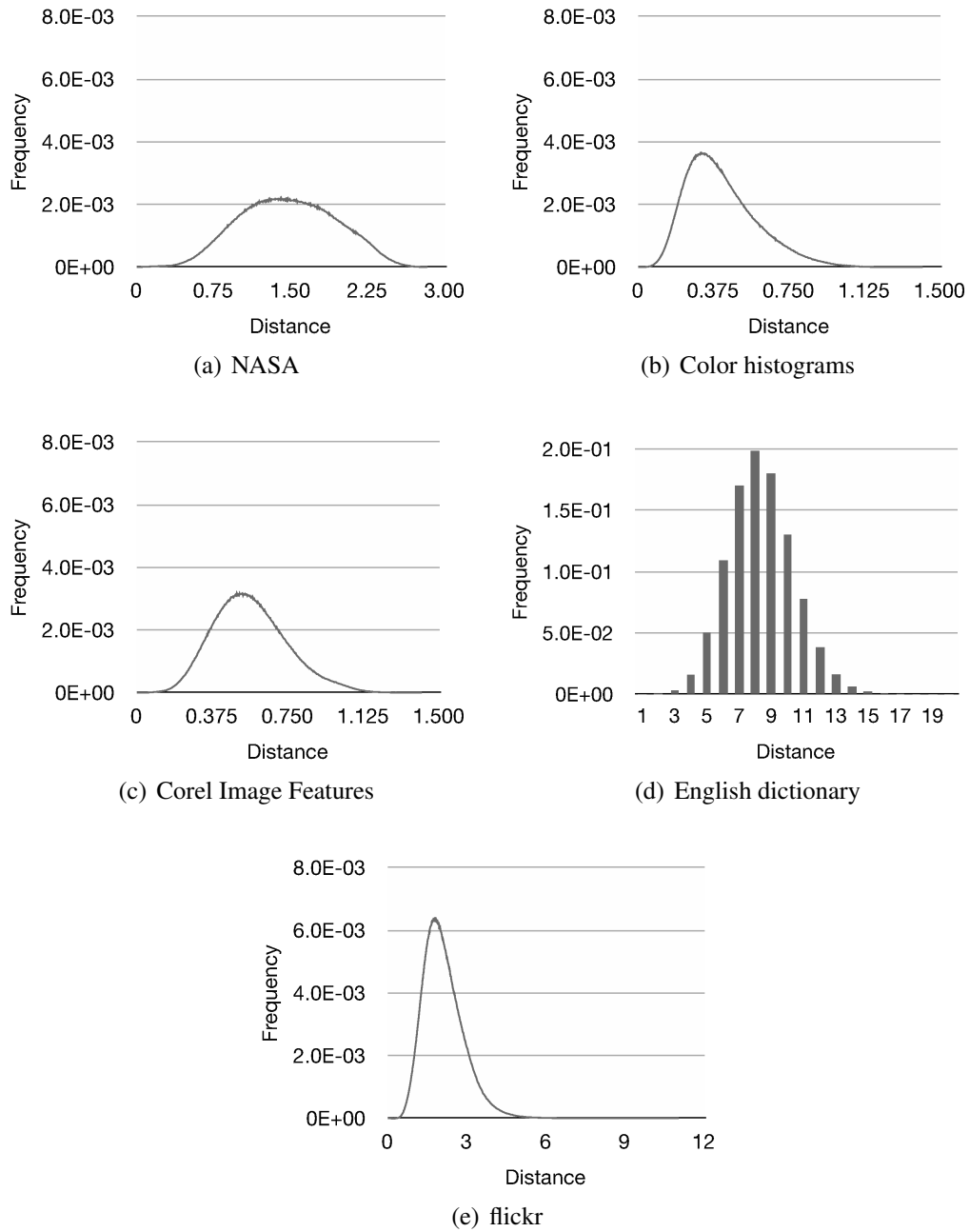


Figure 4.9: Distance densities

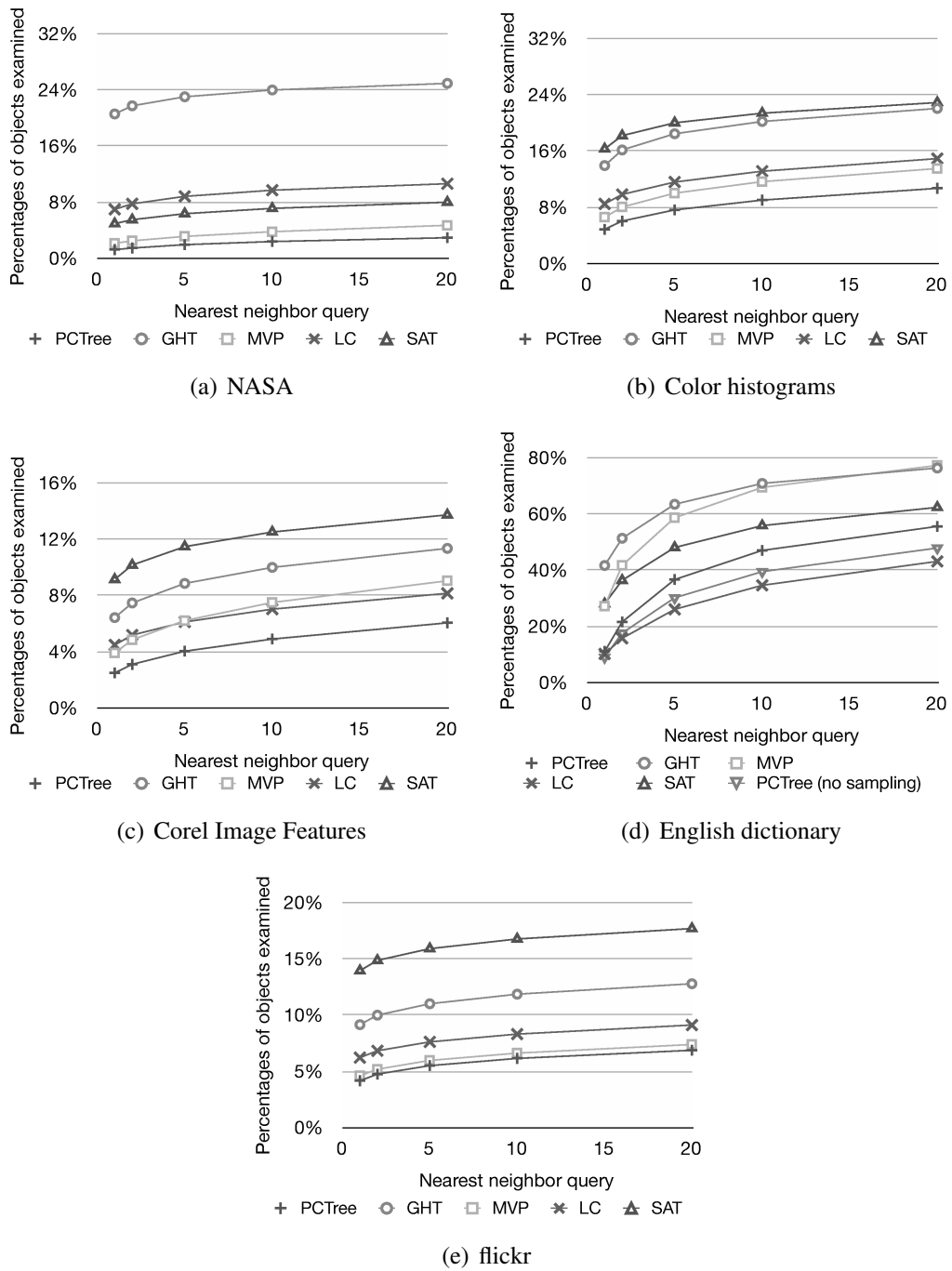


Figure 4.10: Index Performance on Real Datasets

4.5 Summary

PCTree attempts to maximize pruning and create a balanced tree. I defined the Pivot Capacity (PC) for selecting a pivot and its partition. By using the PC, PCTree automatically optimizes the index structure according to the data distribution and reduces the search cost.

I am currently attempting to improve the sampling scheme for the pivot candidates. Having more pivot candidates can help to reduce the search cost. However, as the number of candidates increases, the indexing cost also increases. I have to reduce the indexing cost of the PCTree before we can use it in practical situations.

Chapter 5

Conclusion

This thesis described the work on similarity search techniques. I presented two methods for pivot partitioning, named MMMP and PCTree, that aim at reducing the query execution cost of a similarity search in metric spaces. Pivot partitioning is a core technique for making fast similarity searches. This chapter draws conclusions from the research results obtained and looks at some future work on similarity searches.

This thesis proposed a new pivot selection approach called the “*data distribution approach*”. It considers the data distribution pattern for selecting a pivot. It aims at choosing a good pivot that prunes dissimilar objects. I developed MMMP and PCTree on the basis of the data distribution approach. MMMP is a margin-based pivot selection method for similarity search indexes. It searches for a pivot object whose partitioning boundary is between clusters and also maximizes the distances to the cluster edges. On the other hand, PCTree is a pivot selection method based on maximizing both the pruning and balance. PCTree optimizes the partitioning boundary of a pivot based on the balance of the regions partitioned by a pivot and the estimated effectiveness of the search pruning by the pivot. For PCTree, I defined Partitioning Capacity, which is an information theoretic criterion. PC represents the expected index performance for a given query distribution. I conducted several experiments with synthetic datasets and real datasets and compared my pivot selection method with existing methods. The results revealed that the data distribution approach reduces the query execution cost of a similarity search. MMMP is effective when the indexed data is clustered and has a sparser density space. It could easily create a good partitioning boundary for the data. On the other hand, PCTree is effective even if the data is not so clustered. PCTree changes the weights of the pruning and balancing according to the data distribution.

Regarding future work, there are still many things that are not well understood about the ideal pivot selection and indexing. I consider the following issues:

- The indexing cost with the data distribution approach is heavy. A large part of the cost is the data distribution extraction cost. We need to develop a low-cost distribution extraction technique for pivot selection.
- Most existing pivot selections, including my methods, assume that the distribution of queries is the same as that of the objects in the dataset. We should think about cases in which these distributions are different.

- Most pivot selections handle with a static dataset. Few studies provide the techniques for frequent insertions and deletions. A dynamic index construction method should be studied.
- We have to apply the similarity search techniques in a metric space to indexes of a specific space.

Bibliography

- [ABHY04] L. Arge, M. Berg, H. J. Haverkort, and K. Yi. The priority r-tree: A practically efficient and worst-case optimal r-tree. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data (SIGMOD 2004)*, 2004.
- [ABKS99] M. Ankerst, M. M. Breunig, H. Kriegel, and J. Sander. Optics: ordering points to identify the clustering structure. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD 1999)*, 1999.
- [ASS⁺09] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski. Building rome in a day. In *Proceedings of the 12th IEEE International Conference on Computer Vision (ICCP 2009)*, 2009.
- [AVL62] G. M. Adel'son-Vel'skii and E. M. Landis. An algorithm for the organization of information. *Soviet Mathematics Doklady*, 3:1259–1262, 1962.
- [BK73] W. A. Burkhard and R. M. Keller. Some approaches to best-match file searching. *Communications of the ACM*, 16(4):230–236, 1973.
- [BKP06] S. Brecheisen, H. Kriegel, and M. Pfeifle. Multi-step density-based clustering. *Knowledge and Information Systems*, 9(3):284–308, 2006.
- [BKSS90] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data (SIGMOD 1990)*, 1990.
- [BMS07] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *Proceedings of the 16th international conference on World Wide Web (WWW 2007)*, 2007.

- [BNC03] B. Bustos, G. Navarro, and E. Chevez. Pivot selection techniques for proximity searching in metric spaces. *Pattern Recognition Letters*, 24(14):2357–2366, 2003.
- [BNFZ06] M. Batko, D. Novak, F. Falchi, and P. Zezula. On scalability of the similarity search in the world of peers. In *Proceedings of the 1st International Conference on Scalable Information Systems (InfoScale 2006)*, 2006.
- [BO97] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. *SIGMOD Record*, 26(2):357–368, 1997.
- [BO99] T. Bozkaya and Z. M. Ozsoyoglu. Indexing large metric spaces for similarity search queries. *ACM Transactions on Database Systems (TODS)*, 24(3):361–404, 1999.
- [Bri95] S. Brin. Near neighbor search in large metric spaces. In *Proceedings of the 21th International Conference on Very Large Data Bases (VLDB 1995)*, 1995.
- [BYCMW94] R. A. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixed-queries trees. In *Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching (CPM 1994)*, 1994.
- [CG93] L. Carleson and T. W. Gamelin. *Complex Dynamics*. Springer-Verlag New York, Inc., 1993.
- [CMN01] E. Chevez, J. L. Marroguin, and G. Navarro. Fixed queries array: A fast and economical data structure for proximity searching. *Multimedia Tools Applications*, 14(2):113–135, 2001.
- [CN05] E. Chevez and G. Navarro. A compact space decomposition for effective metric indexing. *Pattern Recognition Letters*, 24(9):1363–1376, 2005.
- [CPZ97] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB 1997)*, 1997.
- [DGSZ03] V. Dohnal, C. Gennaro, P. Savino, and P. Zezula. D-index: Distance searching index for metric data sets. *Multimedia Tools and Applications*, 21(1):9–33, 2003.

- [DVKV07] C. Doulkeridis, A. Vlachou, Y. Kotidis, and M. Vazirgiannis. Peer-to-peer similarity search in metric spaces. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB2007)*, 2007.
- [flia] Flickr gist image features, <http://www.adl.nii.ac.jp/file/flickr/flickrgist2001000.ascii.gz>.
- [flib] flickr, <http://www.flickr.com/>.
- [GCM⁺08] J. F. Gantz, C. Chute, A. Manfrediz, S. Minton, D. Reinsel, W. Schlichting, and A. Toncheva. *The Diverse and Exploding Digital Universe*. IDC, 2008.
- [gis] Lear's gist implementation, <http://lear.inrialpes.fr/software>.
- [Gut84] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data (SIGMOD 1984)*, 1984.
- [HE07] J. Hays and A. A. Efros. Scene completion using millions of photographs. In *Proceedings of the 34th International Conference and Exhibition on Computer Graphics and Interactive Techniques (SIGGRAPH 2007)*, 2007.
- [HKR93] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge. Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):850–863, 1993.
- [HSW07] T. N. Herzog, F. J. Scheuren, and W. E. Winkler. *Data Quality and Record Linkage Techniques*. Springer-Verlag, 2007.
- [iPh] <http://www.apple.com/ilife/iphoto/> iPhoto.
- [JJ00] G. A. Jones and J. M. Jones. *Information and Coding Theory*. Springer-Verlag, 2000.
- [JOT⁺03] H. V. Jagadish, B. C. Ooi, K. L. Tran, C. Yu, and R. Zhang. idistance: An adaptive b+-tree based indexing method for nearest neighbor search. *ACM Transactions on Database Systems (TODS)*, 30(2):364–397, 2003.
- [KM83] I. Kalantari and G. McDonald. A data structure and an algorithm for the nearest point problem. *IEEE Transactions on Software Engineering*, 9(5):631–634, 1983.
- [KS97] N. Katayama and S. Satoh. The sr-tree: An index structure for high-dimensional nearest neighbor queries. In *Proceedings of the 1997 ACM*

- SIGMOD International Conference on Management of Data (SIGMOD 1997)*, 1997.
- [Lev65] V. I. Levenshtein. Binary codes capable of correcting spurious insertions and deletions of ones. *Problems of Information Transmissions*, 1(1):8–17, 1965.
- [Met] Metric spaces library, http://www.sisap.org/metric_space_library.html.
- [MOV94] M. L. Mico, J. Oncina, and E. Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear pre-processing time and memory requirements. *Pattern Recognition Letters*, 15(1):9–17, 1994.
- [Nav02] G. Navarro. Searching in metric spaces by spatial approximation. *The VLDB Journal*, 11(1):28–46, 2002.
- [NR08] G. Navarro and N. Reyes. Dynamic spatial approximation tree. *Journal of Experimental Algorithmics*, 12:1–68, 2008.
- [NZ06] D. Novak and P. Zezula. M-chord: A scalable distributed similarity search structure. In *Proceedings of the 1st International Conference on Scalable Information Systems (InfoScale 2006)*, 2006.
- [OT01] A. Oliva and A. Torralba. Modeling the shape of the scene: a holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3):145–175, 2001.
- [PB07] O. Pedreira and N. R. Brisaboa. Spatial selection of sparse pivots for similarity search in metric spaces. In *Proceedings of the 33rd International Conference on Conference on Current Trends in Theory and Practice of Informatics (SOFSEM)*, 2007.
- [UCI] Uci kdd archive, <http://kdd.ics.uci.edu/>.
- [Uhl91] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40(4):175–179, 1991.
- [Vid94] E. Vidal. New formulation and improvements of the nearest-neighbour approximating and eliminating search algorithm (aesa). *Pattern Recognition Letters*, 15(1):1–7, 1994.

- [VLKJ06] J. Venkateswaran, D. Lachwani, T. Kahveci, and C. Jermaine. Reference-based indexing of sequence databases. In *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB 2006)*, 2006.
- [whi10] *2010 WHITE PAPER Information and Communications in Japan*. Ministry of Internal Affairs and Communications, 2010.
- [WJ96] D. A. White and R. Jain. Similarity indexing with the ss-tree. In *Proceedings of the 12th International Conference on Data Engineering (ICDE 1996)*, 1996.
- [Wor] Wordnet, <http://wordnet.princeton.edu/>.
- [XWL08] C. Xiao, W. Wang, and X. Lin. Ed-join: an efficient algorithm for similarity joins with edit distance constraints. In *Proceedings of the 34th International Conference on Very Large Data Bases (VLDB2008)*, 2008.
- [XWLS09] C. Xiao, W. Wang, X. Lin, and H. Shang. Top-k set similarity joins. In *Proceedings of the 25th International Conference on Data Engineering (ICDE2009)*, 2009.
- [Yia93] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the 4th Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms (SODA 1993)*, 1993.
- [Yia99] P. N. Yianilos. Excluded middle vantage point forests for nearest neighbor search. In *Proceedings of the 6th DIMACS Implementation Challenge Workshop: Near Neighbor Searches*, 1999.
- [YOTJ01] C. Yu, B. C. Ooi, K. L. Tran, and H. V. Jagadish. Indexing the distance: An efficient method to knn processing. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB 2001)*, 2001.
- [ZADB05] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach (Advances in Database Systems)*. Springer-Verlag, 2005.
- [ZRL96] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: an efficient data clustering method for very large databases. *SIGMOD Record*, 25(2):103–114, 1996.

- [ZZL⁺08] Y. Zhuang, Y. Zhuang, Q. Li, L. Chen, and Y. Yu. Indexing high-dimensional data in dual distance spaces: a symmetrical encoding approach. In *Proceedings of the 11th International Conference on Extending Database Technology (EDBT 2008)*, 2008.

Publication

Journal Articles

- [1] Hisashi Kurasawa, Daiji Fukagawa, Atsuhiko Takasu, and Jun Adachi. Optimal Pivot Selection Method based on the Partition and the Pruning Effect for Metric Space Indexes. IEICE Transactions on Information and Systems, E94-D, No.3, 2011. (to be appeared)
- [2] Hisashi Kurasawa, Daiji Fukagawa, Atsuhiko Takasu, and Jun Adachi. Margin-based Pivot Selection for Similarity Search Indexes. IEICE Transactions on Information and Systems, E93-D, No.6, pp.1422-1432, 2010.
- [3] Hisashi Kurasawa, Atsuhiko Takasu, and Jun Adachi. Load Balancing Scheme on the basis of Huffman coding for P2P Information Retrieval. IEICE Transactions on Information and Systems, Vol.E92-D, No.10, pp.2064-2072, 2009.
- [4] 倉沢 央, 若木 裕美, 高須 淳宏, 安達 淳. P2P 情報検索における単語の頻度情報に基づくデータ配置手法. 情報処理学会論文誌データベース (TOD), Vol.1, No.3, pp.1-10, 2008.

International Conference Proceedings

- [5] Hisashi Kurasawa, Atsuhiko Takasu, and Jun Adachi. Finding the k-Closest Pairs in Metric Spaces. In Proceeding of 1st International Workshop on New Trends in Similarity Search (NTSS 2011), 2011. (Submitted)
- [6] Hisashi Kurasawa, Daiji Fukagawa, Atsuhiko Takasu, and Jun Adachi. Pivot Selection Method for Optimizing both Pruning and Balancing in Metric Space Indexes. In Proceeding of 21th International Conference on Database and Expert Systems Applications (DEXA 2010), pp.141-148, Short paper, 2010. (採択率 Full paper22.8%+Short paper18.3% (45+ 36/197))
- [7] Hisashi Kurasawa, Daiji Fukagawa, Atsuhiko Takasu, and Jun Adachi. Maximal Metric Margin Partitioning for Similarity Search Indexes. In Proceed-

- ings of 18th ACM Conference on Information and Knowledge Management (CIKM2009), pp.1887-1890, Short paper, 2009. (採択率 Full paper14.5%+Short paper20.2% (123+ 171/847))
- [8] Hisashi Kurasawa, Atsuhiko Takasu, and Jun Adachi. Huffman-DHT: Index Structure Refinement Scheme for P2P Information Retrieval. In Proceedings of 2008 International Symposium on Applications and the Internet (SAINT2008), pp.111-117, 2008. (採択率 27% (18/67))
- [9] Hisashi Kurasawa, Hiromi Wakaki, Atsuhiko Takasu, and Jun Adachi. Data Allocation Scheme Based on Term Weight for P2P Information Retrieval. In Proceedings of the 9th ACM International Workshop on Web Information and Data Management (ACM WIDM 2007), pp.33-40, 2007. (採択率 25% (20/80))
- [10] Yoshihiro Kawahara, Hisashi Kurasawa, and Hiroyuki Morikawa. Recognizing User Context Using Mobile Handsets with Acceleration Sensor. In Proceedings of IEEE International Conference on Portable Information Devices (IEEE Portable 2007), pp.1-5, 2007.
- [11] Hisashi Kurasawa, Yoshihiro Kawahara, Hiroyuki Morikawa, and Tomonori Aoyama. A Dynamic User Posture Inference Scheme for Mobile Devices In Proceedings of the 8th International Conference on Ubiquitous Computing (Ubi-Comp2006), Demo Paper, 2006.
- [12] Hua Si, Yoshihiro Kawahara, Hisashi Kurasawa, Hiroyuki Morikawa, and Tomonori Aoyama. A Context-aware Collaborative Filtering Algorithm for Real World Oriented Content Delivery Service. In Proceedings of the 7th International Conference on Ubiquitous Computing (UbiComp2005), Metapolis and Urban Life workshop, pp.65-69, 2005.

Research Reports

- [13] 倉沢 央, 高須 淳宏, 安達 淳. メトリック空間における最近傍ペア探索アルゴリズムの高速化. 情報処理学会全国大会論文集, 4B-1, 2011. (to be appeared)
- [14] 倉沢 央, 深川 大路, 高須 淳宏, 安達 淳. 類似検索の高速化を目的とした Pivot 選択手法の実験評価. 第9回情報科学技術フォーラム (FIT 2010) 論文集, D-008, 2010.
- [15] 倉沢 央, 深川 大路, 高須 淳宏, 安達 淳. 索引木の均衡を考慮した類似検索索引手法. 情報処理学会全国大会論文集, 2K-2, 2010.
- [16] 倉沢 央, 深川 大路, 高須 淳宏, 安達 淳. マージン最大化によるメトリック空間分割手法. WebDB forum 2009, 招待 (情報爆発)-1, 2009.

- [17] 倉沢 央, 深川 大路, 高須 淳宏, 安達 淳. マージン最大化によるメトリック空間分割手法. 電子情報通信学会研究報告, DE2009-3, 2009.
- [18] 倉沢 央, 高須 淳宏, 安達 淳. 情報爆発時代における P2P 情報検索向きデータ配置手法. 情報処理学会全国大会論文集, 6ZK-4, 2008.
- [19] 倉沢 央, 若木 裕美, 正田 備也, 高須 淳宏, 安達 淳. P2P 情報検索における単語の重みに基づいたデータ配置手法. 情報処理学会 マルチメディア, 分散, 協調とモバイル (DICOMO2007) シンポジウム, 2G-4, 2007.
- [20] 倉沢 央, 正田 備也, 高須 淳宏, 安達 淳. P2P 情報検索における索引とファイルの分散配置手法. 情報処理学会研究報告, OS-105-21, 2007.
- [21] 倉沢 央, 川原 圭博, 森川 博之, 青山 友紀. センサ装着場所を考慮した 3 軸加速度センサを用いた姿勢推定手法. 情報処理学会研究報告, UBI-11-3, Jun 2006.
- [22] 倉沢 央, 川原 圭博, 森川 博之, 青山 友紀. 装着場所を考慮した 3 軸加速度センサを用いた姿勢推定手法. 電子情報通信学会総合大会, B-15-8, 2006.
- [23] 倉沢 央, 川原 圭博, 森川 博之, 青山 友紀. 単一の無線加速度センサを用いたユーザコンテキストの推定. 電子情報通信学会ソサイエティ大会, B-19-23, 2005.

Honors

- FIT2010 ヤングリサーチャー賞, 2010
- DICOMO2007 ヤングリサーチャー賞, 2007
- 第 11 回 UBI 研究発表会優秀論文賞, 2006