

國立臺灣大學電機資訊學院資訊工程研究所

碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

在分散式環境下以正交轉換技術節省頻寬

之最近鄰查詢演算法

Exact kNN-Search with Orthogonal Transformation in the
Distributed Environment

王瑞斌

Jui-Pin Wang

指導教授：林守德博士，葉彌妍博士

Advisor: Shou-De Lin, Ph.D., Mi-Yen Yeh, Ph.D.

中華民國 103 年 7 月

July, 2014

Abstract

With the popularity of mobile devices and Internet of Things, similarity search such as image querying or query by humming among data distributed in different machines has become an increasingly important problem. Moreover, in a distributed environment, the transmission cost is usually much more crucial than the computation cost as generally it consumes more resource to in transmission. Thus, this thesis proposes a framework to conduct a bandwidth-efficient, exact k-nearest neighbor search amount a large number of distributed machines through exploiting the techniques of the orthogonal transformation. On the basis of our previous work, new bounds on the Euclidean distance have been derived to prune impossible instances in the early stages of searching using partial information of the query. Moreover, three additional enhancements are devised to further save the communication cost. The experiment results show that our method significantly outperforms other competitions in bandwidth consumption in large-scale experiments of millions of instances.

Key words: Exact Similarity Search, Orthogonal Transformation, Optimization.

Contents

Abstract	i
Contents	ii
List of Figures	v
List of Tables	vi
1 Introduction	1
2 Related Works	4
2.1 Concurrent Processing	4
2.2 Probabilistic Processing	4
2.3 LeeWave	5
2.4 MsWave	5
2.5 Others	6
3 Methodology	7
3.1 Problem Setup	7
3.2 Overview of Our Framework	7
3.3 First Phase	9
3.3.1 Definition of Orthogonal Transformation	9
3.3.2 Property of Orthogonal Transformation	9
3.4 Enhance the Bounds by the Orthogonal Transformation	10
3.4.1 The Goal of the First Phase	10

3.4.2	Definition of the Bounds	11
3.4.3	Relation Between the Norms and the Bounds	11
3.4.4	Equivalent Bounds After Transformation	12
3.4.5	Reduction of the Norm with Orthogonal Transformation	13
3.4.6	Optimization over Orthogonal Constraints	14
3.4.7	Reduce the Cost of Sending Matrices	15
3.5	Second Phase	17
3.5.1	Pruning the Candidates with the Bounds	17
3.5.2	Derivation of our Bounds	18
3.5.3	Calculation of our Bounds	19
3.5.4	Finding the Threshold in Distributed Machines	20
3.6	Decision of the Pivots	21
3.6.1	Estimation of the Number of Residual Machines	21
3.6.2	Estimation of the Transmission Cost	22
3.6.3	Coordinate Descent to Decide the Pivots	23
3.7	Overall Framework	23
3.7.1	Importance-Selecting Function	23
3.7.2	Overall Framework	24
4	Experiment	26
4.1	Experiment Setup	26
4.2	Data Description	26
4.2.1	Time Series Data	26
4.2.2	Image Data	27
4.2.3	Audio Data	27
4.3	Comparison Among all Frameworks	28
4.3.1	Frameworks for Comparison	28
4.3.2	Results of Different Frameworks	28
4.4	Comparison Among our Framework with Different Configurations	32
4.4.1	Configurations for Comparison	32

4.4.2	Results of Different Configurations	32
4.5	Number of Queries for Amortizing the Cost of Matrices	35
4.6	Power of the Pruning Procedure	37
4.6.1	Methods with Different Bounds	37
4.6.2	Results of Pruning	37
5	Conclusions	42
	Bibliography	44

List of Figures

3.1	The overall flow of our framework.	8
3.2	The goal of the first phase.	10
3.3	The procedure of pruning impossible candidates.	17
3.4	The example of linear interpolation.	22
4.1	Different Frameworks on Time Series	29
4.2	Different Frameworks on ANN	29
4.3	Different Frameworks on Flickr: CSD	29
4.4	Different Frameworks on Flickr: SCD	30
4.5	Different Frameworks on Million Song: MVD	30
4.6	Different Frameworks on Million Song: TRH	30
4.7	Different Configurations on Time Series	33
4.8	Different Configurations on ANN	33
4.9	Different Configurations on Flickr: CSD	33
4.10	Different Configurations on Flickr: SCD	34
4.11	Different Configurations on Million Song: MVD	34
4.12	Different Configurations on Million Song: TRH	34
4.13	Pruning Results on Time Series	37
4.14	Pruning Results on ANN	40
4.15	Pruning Results on Flickr: CSD	40
4.16	Pruning Results on Flickr: SCD	40
4.17	Pruning Results on Million Song: MVD	41
4.18	Pruning Results on Million Song: TRH	41

List of Tables

4.1	Summary for each dataset	27
4.2	Number of queries to amortize the cost of matrices	36
4.3	Time Series, Number of residual machines after sending some dimensions for 5000 machines	38
4.4	Flickr:CSD, Number of residual machines after sending some dimensions for 1000 machines	38
4.5	Million Songs: MVD, Number of residual machines after sending some dimensions for 800 machines	38

Chapter 1

Introduction

With the arrival of big-data era, similarity search for various types of data among distributed machines such as mobile devices has become a common and important task. In a distributed environment where a large amount of local machines are involved in computation and storage, our goal is to minimize the amount of transmission cost while identifying the nearest neighbors of a query. This paper aims to generalize the current state-of-the-art on distributed pattern matching from only suitable for “time series” datasets to other types of data such as image, audio, etc.

Consider a scenario in which, we are building a search engine on a large amount of distributed machines such as smart phones or tablets. On that engine, users could search for the most similar instances from other devices as long as they also share their data. These types of these instances could be image, audio, which are very common in these devices. As a search engine, it would obtain a large number of users whose number of queries could be also very large. So it is not practical to directly send every query to the all devices. Also, since there could be a very large amount of instances on each machine, it is also inefficient to send them back to the search engine due to its huge bandwidth cost.

The proposed task is challenging in several aspects. First, the goal is to identify the *exact* k nearest neighbors with guarantee, instead of approximated k nearest neighbors. Second, we are targeting at handling endless number of queries (similar to a search engine) and assume the data are stored distributed in many (thousands or even millions) local devices. Finally, different from most of the previous works to focus only on time-series

data, we are not constraining ourselves to a specific type of data, rather emphasizing on a general-purpose algorithm.

One of the state-of-the-art method, LeeWave, [1] has been proposed to handle some of the above challenges. Unfortunately, it is designed specific for time-series and works poorly for other types of data. The main reason is that the method LeeWave used to prune the candidates assumes the smoothness of data, which is not necessary true in general case. Another problem is that the cost of the pruning procedure in LeeWave grows linearly with the number of total instances in these distributed devices. When the number of instances becomes large, so is the communication cost.

We propose a two-phase framework to solve these challenges. On the basis of LeeWave, our proposed model also derives theoretical bounds for pruning candidates in the early stages. To further improve the performance of pruning, we add a preprocessing step which utilizes the idea of orthogonal transformation to directly optimizes these bounds given data. We also devise a method which reduces the cost during the pruning procedure from linear to the number of total instances to a constant value, which could significantly reduce the communication cost given large-scale data.

We summarize our main contributions as follows:

1. We propose a general communication-efficient framework to identify exact k NN instances given a query. The model can be applied to various types of datasets which are distributed in a large amount of devices.
2. In the part of methodology, based on the ideas of upper bounds and lower bounds of similarity in [1], we derive new bounds with the help of the orthogonal transformation which enhance the power of pruning to save communication cost. Besides, we propose a method which reduces the cost for the pruning procedure from linear to the number of total instances to a constant value. Moreover, we give a method to estimate the communication cost of a query before sending it, which allows us to dynamically adjust how much information we need to send for it based on the history of past queries.

3. We conduct extensive experiments for various types of datasets to demonstrate the effectiveness of our model.

This thesis is organized as follows: In the chapter 2, we discuss the related works and their limitations. In the chapter 3, we describe the details of our framework. In the chapter 4, we provide the results of the experiments using images, audio, and time series datasets, and conclude our works in chapter 5.

Chapter 2

Related Works

In this chapter, we talk about the papers which are related to these scenarios.

2.1 Concurrent Processing

Concurrent Processing (CP) [2] is the baseline of this problem. First, the server would send the whole query to every local machine. Each machine calculates the distance between this query and every instance on it. Then, every machine return the top k instances with the lowest distance back to the server. By these $m \times k$ instances, the server could know which k instances are the k nearest neighbors of the query. We could easily notice that much communication cost is waste in this framework.

2.2 Probabilistic Processing

There is another method named Probabilistic Processing (PRP) in [2] that we could take it as an improvement of CP. The most important characteristic of PRP is that it could find the answers in two rounds with pruning some machines. In the first round, the server also sends the whole query to every local machines. But, instead of return the top k instances like CP does, each machine only return the top $\lfloor \frac{k}{m} \rfloor + 1$ instances back to the server where m is the number of total local machines. With these $m \times (\lfloor \frac{k}{m} \rfloor + 1)$ instances, the

server could prune some machines which are impossible to obtain the final k answers and then ask the other machines for the final answers in the second rounds. Although PRP might able to prune some machines in the first round, it still spends too much cost in the second round.

2.3 LeeWave

Now let's talk about the state-of-the-art method, LeeWave [1], which is the starting point of our framework. The spirit of LeeWave is to iteratively pruning impossible candidates until only k instances left by transforming a raw feature vector into an error tree with the help of the Haar wavelet transformation. Although the total number of coefficients in an error tree would be equal to length of the raw feature vector, the coefficients at the upper levels would be more important than those in the lower levels. The importance defined here is the chance to contribute more to the final Euclidean distance, And it could also be observed from the way to calculate the Euclidean distance from the error trees, the higher level the coefficient is, the heavier weight it has to multiply.

Once we have the importance of the coefficients, LeeWave sends coefficients according to their levels in the error tree transformed from the query q , from upside to down. In each round, LeeWave would send those coefficients in one level of the tree to each candidate machines. Then, these machines would return some information that allows the server to compute the bounds between q and the instances in these machines. With the help of these bounds, the server could prune some instances that they are impossible to be the final answers. If there are exactly k instances left after pruning, then we just achieve our goal to find the k NN/ k FN. Otherwise, the server would send the next level and repeat the pruning process until finding the answers or sending every level of this tree.

2.4 MsWave

MsWave [3] is our previous work which is also extended from LeeWave but toward a different direction. While the main contribution of this paper is to generalize the ideas

of LeeWave to various types of datasets, the contribution of MsWave focus on how to modify the bounds in LeeWave for multiple queries and finding k farthest neighbors. We could say that LeeWave is a special case of MsWave for a single query only with some slightly difference. Since the foundation of MsWave and this paper is the same, we could easily apply our all improvements in this paper to the scenarios in MsWave. That is, we could also solve the cases of multiple queries and find k farthest neighbors with our new bounds. Because there are detailed discussions about the situation of multiple queries and differences between finding k NN and k FN, we only conduct the experiments for the case of a single query and finding k NN.

2.5 Others

Due to the population of the P2P paradigm [4, 5], there are some methods which use the distributed computing to do similarity search over a set of machines. For example, [6–8] are P2P approaches proposed for similarity search, but they are designed for one dimensional data, not high dimensional data. SWAM [9] is a family of *Small World Access Methods*, whose goal is to build a network topology which could collect peers with similar content. However, in this framework, each peer could only obtain a single data, which is not suitable with our problem for a large amount of data. VBI-tree and SkipIndex both rely on tree-based approaches that could not scale when the dimensions of data are high. [10] leverage on LSH-based approaches for similarity search over structured P2P network for high dimensional data. Nevertheless, it only provides the approximate results, not exactly solution. But the biggest difference between these papers and our framework is the setting of the network. While their framework apply on a more general P2P system, our setting is that every local machine is only able to communicate with a single server.

Chapter 3

Methodology

In this chapter, we give the details about our framework.

3.1 Problem Setup

There are a query set $Q = \{q_1, q_2, \dots, q_T\} \subset \mathbb{R}^D$ at the server P and a dataset $X_i \subset \mathbb{R}^D$ on each local machine M_i , $\forall i = 1, 2, \dots, m$ where the total number of instances in these machines is N . For each coming query q_t , we want to find its k_{th} nearest neighborhood among these distributed datasets while reducing the transmission cost between P and each M_i .

3.2 Overview of Our Framework

In this section, we describe an overview of our framework. Then, we will give the details about the framework in the following sections.

Figure 3.1 is the overall flow on our framework. There are two main phases in our framework. For each X_i , the first phase (the red part) only needs to be done for once. On the other hand, we need to run the second phase (the cycle) for each new query q_t .

The first phase is an preprocessing procedure for the second phase. Its goal is to improve the performance of pruning in the second phase. We will prove in the section 3.5.2 that this pruning power is highly correlated to the distribution of the norm of the feature

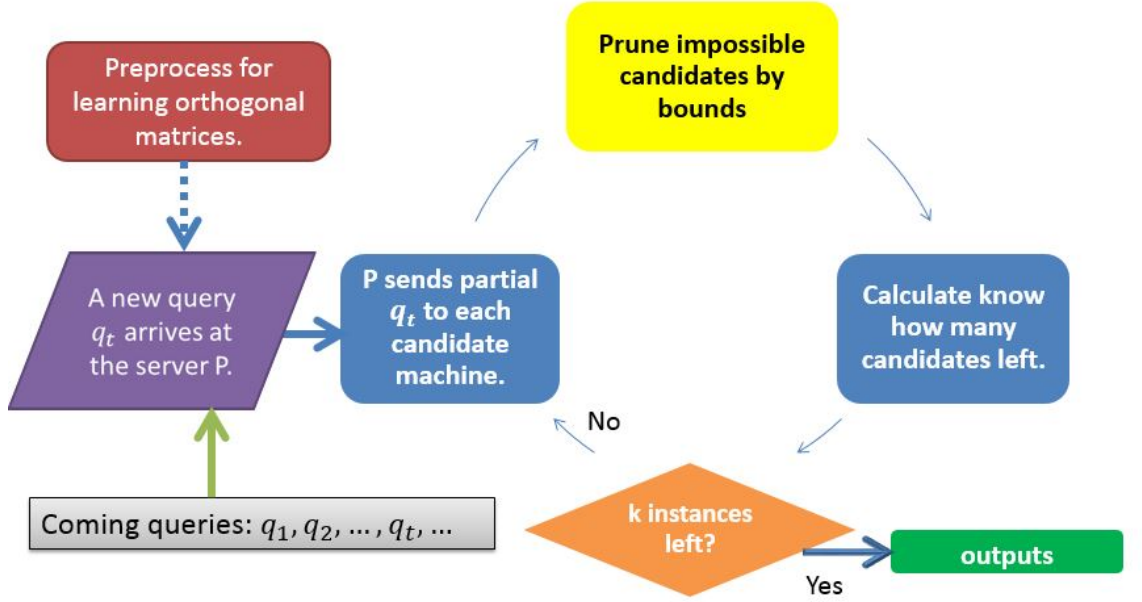


Figure 3.1: The overall flow of our framework.

vectors. As a result, each M_i would learn an ethodogonal matrix W_i for its X_i to fit our desired distribution and then send each W_i back to P . We can notice that this phase is only dependent on X_i and independent of q_t . Therefore, we only need to do the first phase for once. we give the details about how to learn W_i , how to send it back to P in the section 3.4.6.

The second phase is the main procedure of our framework. Note that P have already got W_i for each X_i in the beginning of the second phase. For each coming query q_t , we iteratively prune some candidates which are impossible to be the k NN of q_t to reduce the search space until there are only k candidates left.

To prune candidates iteratively, we divide the second phase into several rounds. For each round j , we use a function $S_i(q_t, j, \theta_t; W_i)$ to generate the values for trasmitting from P to M_i , where S_i is the importance-selecting function of M_i and θ_t is its parameters for deciding how many elements we need to send. (We put the details of S_i at the section 3.7.1.) By these values, each M_i could calculate the bounds between each candidate x_l and q_t . With these bounds, P would be able to determine which candidates are definitely not our answers and then disregards them in the following rounds. By these pruning, we could achieve the goal of saving transmission cost from avoiding to consider the unnecessary candidates.

Note that we could use the square of the Euclidean distance instead of the origin Euclidean distance to find k NN as it is non-negative. So we will use the former one in our

framework.

3.3 First Phase

In this section, we talk about the first phase of our framework.

3.3.1 Definition of Orthogonal Transformation

The most important part of the first phase is the orthogonal transformation. So we give its definition as below.

Definition 1. A matrix $W \in \mathbb{R}^{D \times D}$ is orthogonal if whose columns and rows are orthogonal vectors, i.e.

$$W^T W = W W^T = I$$

where I is the identity matrix.

3.3.2 Property of Orthogonal Transformation

There are some useful properties in orthogonal transformation. Here we introduce the one that we will use in the derivation of our bounds.

Property 1. Let $x, y \in \mathbb{R}^{D \times 1}$, and $W \in \mathbb{R}^{D \times D}$ be an orthogonal matrix. Then,

$$Dist(x, y) = \sum_{d=1}^D (x[d] - y[d])^2 = \sum_{d=1}^D (W[d, :]x - W[d, :]y)^2 = Dist(Wx, Wy)$$

where $W[d, :]$ is the d_{th} row of W .

We will use this important property in the section 3.4.4.

3.4 Enhance the Bounds by the Orthogonal Transformation

In the section 3.2, we mentioned that the first phase is an auxiliary step for the second phase. After the introduction of the orthogonal transformation, we introduce this powerful tool into the first phase in our framework.

3.4.1 The Goal of the First Phase

Our goal in the first phase is to reduce the ranges of the bounds used in the second phase. Since we will use a threshold to prune the impossible candidates according to their bounds in the pruning procedure, the ranges of the bounds would be one of the most influential factor of the pruning power. It is easy to understand that when the ranges of their bounds is short, more candidates would be pruned than those with long ranges of the bounds. In other words, the shorter the range of the bound, the higher chance this candidate would be pruned if it is not our final answer of k NN.

Moreover, we would prove in the section 3.4.5 that to reduce the range of the bounds, the final goal of the first phase would become to learn an orthogonal transformation W which could make each raw feature vector $x \in \mathbb{R}^D$ (the orange vector in the figure 3.2) transformed to be Wx (the red vector in the figure 3.2) which would make the absolute values of elements in the forward part of the vector much larger and those in the later part much smaller.

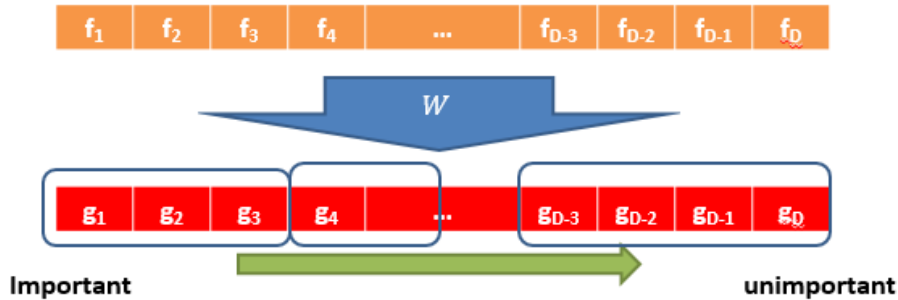


Figure 3.2: The goal of the first phase.

3.4.2 Definition of the Bounds

Before we explain why we need the orthogonal transformation to enhance our bounds, we need to define our bounds for the pruning. Recall that given a query q_t , our goal is to find its k NN in these distributed datasets X_i . Intuitively, we need to calculate the square of the Euclidean distance $Dist(q_t, x), \forall x \in \cup_i X_i$. However, to calculate $Dist(q_t, x)$, we need to send the whole q_t to the local machines or send the whole x to P , which causes a huge transmission cost. Therefore, instead of the exact value of Euclidean distances, our proposed framework uses bounds to confine this distance for finding the k NN.

Definition 1. $\forall x, y \in \mathbb{R}^D$, a lower bound $LB(x, y)$ and an upper bound $UB(x, y)$ must satisfy the following inequation:

$$LB(x, y) \leq Dist(x, y) = \sum_{d=1}^D (x[d] - y[d])^2 \leq UB(x, y)$$

3.4.3 Relation Between the Norms and the Bounds

To achieve the goal of reducing the length of the ranges of the bounds, we could look into the derivation of the bounds. Suppose there are two vectors $x, y \in \mathbb{R}^D$, but we could only observe the first s dimensions of x and $\sum_{d=s+1}^D x[d]^2$, which is the square of two norm of the unobserved part $x[s+1 : D]$. In the section 3.5.2, we derive the bounds as

$$\begin{aligned} LB(x, y) &= \sum_{d=1}^s (x[d] - y[d])^2. \\ UB(x, y) &= \sum_{d=1}^s (x[d] - y[d])^2 \\ &\quad + \sum_{d=s+1}^D x[d]^2 + \sum_{d=s+1}^D y[d]^2 \\ &\quad + 2 \times \sqrt{\sum_{d=s+1}^D x[d]^2 \times \sum_{d=s+1}^D y[d]^2}. \end{aligned}$$

Therefore, we could get the length of the range by the subtraction.

$$\begin{aligned} Len &= UB(x, y) - LB(x, y) \\ &= \sum_{d=s+1}^D x[d]^2 + \sum_{d=s+1}^D y[d]^2 \\ &\quad + 2 \times \sqrt{\sum_{d=s+1}^D x[d]^2 \times \sum_{d=s+1}^D y[d]^2}. \end{aligned}$$

Since the term $\sum_{d=s+1}^D x[d]^2$ is given, all we could do to reduce the length is to make use of the $\sum_{d=s+1}^D y[d]^2$ term. If we could reduce $\sum_{d=s+1}^D y[d]^2$, the term $\sum_{d=s+1}^D x[d]^2 \times \sum_{d=s+1}^D y[d]^2$ would also decrease and then make Len smaller. Therefore, our goal now becomes to make the term $\sum_{d=s+1}^D y[d]^2$ as small as possible, which is the square norm of the vector $y[s+1 : D]$.

Note that the lower (upper) bounds is non-decreasing (non-increasing) as s becomes larger and $LB(x, y) = UB(x, y)$ when $s = D$. This means that LB and UB would be exactly equal to $\sum_{d=1}^D (x[d] - y[d])^2$ eventually.

To reduce the length of the ranges Len for each s , we hope to make $\sum_{d=s+1}^D y[d]^2$ as small as possible. However, since the feature vector y is given from datasets, the value of $\sum_{d=s+1}^D y[d]^2$ is already determined when s is given. As a result, we introduce the orthogonal transformation to achieve this goal.

3.4.4 Equivalent Bounds After Transformation

From the section 3.3.2, we know the distance of two vectors won't be changed after an orthogonal transformation. Now we use this property to achieve our goal to reduce the length of the ranges Len given s .

Given an orthogonal transformation $W \in \mathbb{R}^{D \times D}$, we have $Dist(x, y) = Dist(Wx, Wy)$. This means that the bounds we derivated before could also be the bounds for $Dist(Wx, Wy)$. That is,

$$LB(x, y) \leq Dist(x, y) = Dist(\hat{x}, \hat{y}) \leq UB(x, y)$$

where $\hat{x} = Wx$.

This also means that we could use the same way to derivate the lower bounds and

upper bounds for $Dist(Wx, Wy)$ and these bounds are also the bounds for $Dist(x, y)$. That is,

$$LB(\hat{x}, \hat{y}) \leq Dist(x, y) = Dist(\hat{x}, \hat{y}) \leq UB(\hat{x}, \hat{y})$$

So, we could use the bounds $LB(\hat{x}, \hat{y}), UB(\hat{x}, \hat{y})$ to confine $Dist(x, y)$ and the length of range we want to reduce becomes

$$\begin{aligned} \hat{Len}(W) &= UB(\hat{x}, \hat{y}) - LB(\hat{x}, \hat{y}) \\ &= \sum_{d=s+1}^D \hat{x}[d]^2 + \sum_{d=s+1}^D \hat{y}[d]^2 \\ &\quad + 2 \times \sqrt{\sum_{d=s+1}^D \hat{x}[d]^2 \times \sum_{d=s+1}^D \hat{y}[d]^2}. \end{aligned}$$

which becomes a function of W .

As a result, instead of trying to reduce $\sum_{d=s+1}^D y[d]^2$ which is impossible as we mentioned in the section 3.4.3, our goal becomes to reduce $\sum_{d=s+1}^D \hat{y}[d]^2$ with the help of W .

3.4.5 Reduction of the Norm with Orthogonal Transformation

For $y \in \mathbb{R}^{D \times 1}$ and $W \in \mathbb{R}^{D \times D}$, given s , we want to reduce $\sum_{d=s+1}^D \hat{y}[d]^2$ as much as possible. However, since s is unknown while deciding W in the first phase of our framework, we have to handle all possible values which s could be. Moreover, since $\sum_{d=1}^D \hat{y}[d]^2$ is equal to $\sum_{d=1}^D y[d]^2$, which is independent with W , if the $\sum_{d=s+1}^D \hat{y}[d]^2$ decreases with some W , the term $\sum_{d=1}^s \hat{y}[d]^2$ must increase. Here, we use a more general strategy to deal with these problems.

We could look the term $\sum_{d=s+1}^D \hat{y}[d]^2$ from a different angle. Actually, this term is the square norm of the latter part of the vector \hat{y} . Therefore, although $\sum_{d=1}^D \hat{y}[d]^2$ is a constant for W , we could reduce the square norm of the latter part by increasing the forward part of it. In other words, we move the norm of the latter part of y to its forward part. To accomplish it, we design an objective function and then optimize this function to find our

ideal W .

$$f(W; y) = \sum_{d=1}^D w_d \times \hat{y}[d]^2 = \sum_{d=1}^D w_d \times (W[d, :]y[d])^2 \quad (3.1)$$

where $w_d = d, \forall d = 1, 2, \dots, D$.

Because w_d would give the larger penalty as d increases, the elements in the latter part of \hat{y} would be forced to become small while minimizing this objective function. This is exactly our goal to reduce $\sum_{d=s+1}^D \hat{y}[d]^2$. Therefore, our question becomes how to optimize this objective function with the constraints that W must be an orthogonal matrix.

3.4.6 Optimization over Orthogonal Constraints

Finally, we could introduce this concept of reducing the norms into our framework. In the first phase, we solve the following optimization problem to learn an orthogonal matrix W_i for each machine M_i .

$$\begin{aligned} & \underset{W}{\text{minimize}} && F_i(W) \\ & \text{subject to} && W^T W = W W^T = I \end{aligned} \quad (3.2)$$

where

$$F_i(W) = \sum_{x \in X_i} f(W; x) = \sum_{x \in X_i} \sum_{d=1}^D d \times (W[d, :]x[d])^2 \quad (3.3)$$

This is an optimization problem with constraints that its solution must be an orthogonal matrix. We could solve it efficiently with the help of the package from [11] as long as we have its gradient.

After we get the optimal W_i^* for each M_i , we send these matrices back to the server P . Since the learning of W_i^* is independent with the queries in the future, we only have to go through the procedure of learning W_i^* for once if X_i doesn't change too much. Then our first phase is done.

3.4.7 Reduce the Cost of Sending Matrices

However, the cost to sending an orthogonal matrix $W \in \mathbb{R}^{D \times D}$ is $D \times D$, which is too expensive. Therefore, in this section, we prove that we could reduce this cost to $\frac{D \times (D-1)}{2}$.

First, we introduce a lemma which would be used in our proof.

Lemma. For a vector $\mathbf{x} = (x_1, x_2, \dots, x_D) \in \mathbb{R}^D$, we could use a spherical coordinate system with a radial coordinate r and $D - 1$ angular coordinates $\phi_1, \phi_2, \dots, \phi_{D-1}$ to represent it as

- $x_1 = r \cos(\phi_1)$
- $x_2 = r \sin(\phi_1) \cos(\phi_2)$
- $x_3 = r \sin(\phi_1) \sin(\phi_2) \cos(\phi_3)$
- \vdots
- $x_{D-1} = r \sin(\phi_1) \dots \sin(\phi_{D-2}) \cos(\phi_{D-1})$
- $x_D = r \sin(\phi_1) \dots \sin(\phi_{D-2}) \sin(\phi_{D-1})$

where $\phi_i \in [0, \pi]$, $\forall i = 1, 2, \dots, D - 2$ and $\phi_{D-1} \in [0, 2\pi)$. ■

Since these row vectors $w_1, w_2, w_3, \dots, w_D$ in W are orthonormal vectors, their r in this lemma is 1, which allows us to represent an orthonormal vector in \mathbb{R}^D with $D - 1$ parameters ($\phi_1, \phi_2, \dots, \phi_{D-1}$ in the lemma). Moreover, since they are orthogonal to each other ($w_i w_j^T = 0$, $\forall i \neq j$), our goal becomes how to represent D vectors in \mathbb{R}^D which are orthogonal to each other.

Now we are ready to give the proof by mathematical induction.

Theorem. Given D orthonormal vectors $w_1, w_2, w_3, \dots, w_D \in \mathbb{R}^D$, we could use $\frac{D \times (D-1)}{2}$ parameters to represent them.

Proof. *Base case:*

If $d = 2$, we could use a single parameter θ for w_1, w_2 as

$$\begin{aligned} w_1 &= (\cos \theta, \sin \theta) \\ w_2 &= (\cos(\theta + \frac{\pi}{2}), \sin(\theta + \frac{\pi}{2})) \end{aligned} \tag{3.4}$$

Therefore, the total number of parameters here is 1 and equal to $\frac{2 \times (2-1)}{2}$.

To be clear, we give one more base case here.

Base case-2:

When $d = 3$, we need to represent three vectors $w_1, w_2, w_3 \in \mathbb{R}^3$.

In the beginning, by the lemma above, we could use $3 - 1 = 2$ parameters to represent w_3 .

Then, since w_1 and w_2 are orthogonal to w_3 , they must lie in the plane whose normal vector is w_3 . Since we already have w_3 (with two parameters), this plane is fixed. All we need to do is to decide w_1 and w_2 on this \mathbb{R}^2 plane. By projecting the x axis in the \mathbb{R}^3 to this plane, we could build a Cartesian coordinate system in two dimensions on this plane. And we know that we could use one parameter to decide w_1 and w_2 in an \mathbb{R}^2 plane from the base case above.

So, the total number of parameters for \mathbb{R}^3 case is $2 + 1 = 3$ and equal to $\frac{3 \times (3-1)}{2}$

Inductive hypothesis:

Suppose the theorem holds for all values d up to some $k, k > 3$

Inductive step:

Let $d = k + 1$, we need to represent $k + 1$ vectors $w_1, w_2, w_3, \dots, w_{k+1} \in \mathbb{R}^{k+1}$. We could use similar way like $d = 3$ to do it.

First, by the lemma above, we could use $(k + 1) - 1 = k$ parameters to represent w_{k+1} .

Then, our problem becomes to decide w_1, w_2, \dots, w_k on this hyperplane whose normal

vector is w_{k+1} . From the inductive hypothesis, we know it needs $\frac{k \times (k-1)}{2}$ parameters.

So, the total number of parameters for \mathbb{R}^{k+1} case is

$$k + \frac{k \times (k-1)}{2} = \frac{k \times (k+1)}{2} \dots \blacksquare$$

3.5 Second Phase

Now we start to discuss the second phase of our framework. In this section, we talk about how to prune the candidates if we already have bounds. Note that this mechanism is the most crucial part to achieve our goal to save the transmission cost.

3.5.1 Pruning the Candidates with the Bounds

For the query q_t , if we already know $LB(q_t, x)$ and $UB(q_t, x) \forall x \in \cup_i X_i$, we could use the k_{th} smallest upper bounds and directly prune those instances whose lower bounds are higher than this value thr . I.e., we want to prune

$$\{x \mid LB(q_t, x) > thr, \forall x \in \cup_i X_i\}$$

where thr is the k_{th} largest $UB(q_t, x) \forall x \in \cup_i X_i$. The following figure 3.3 is an example of how we prune impossible instances.

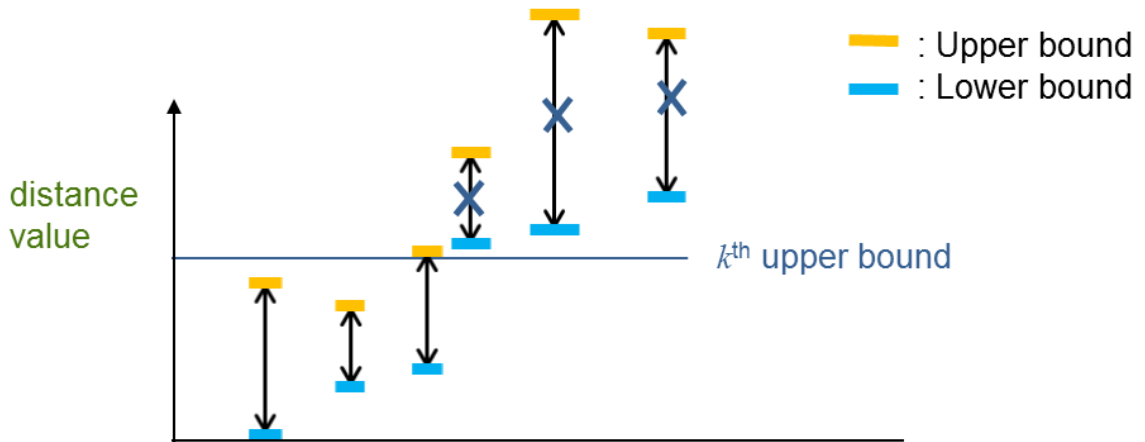


Figure 3.3: The procedure of pruning impossible candidates.

We will talk about the details of calculation these bounds and finding the threshold thr

in the section 3.5.4.

3.5.2 Derivation of our Bounds

Suppose there are two vectors $x, y \in \mathbb{R}^D$, we know the square of their Euclidean distance is

$$Dist(x, y) = \sum_{d=1}^D (x[d] - y[d])^2 \quad (3.5)$$

However, if we could only observe the first s dimensions of x , we could decompose their distance as

$$\begin{aligned} Dist(x, y) &= \sum_{d=1}^D (x[d] - y[d])^2 \\ &= \sum_{d=1}^s (x[d] - y[d])^2 + \sum_{d=s+1}^D (x[d] - y[d])^2 \end{aligned} \quad (3.6)$$

Since the first component of (3.6) is already known, all we need to do is to deal with the second term. Therefore, we further expand the second term as below:

$$\sum_{d=s+1}^D (x[d] - y[d])^2 = \sum_{d=s+1}^D x[d]^2 + \sum_{d=s+1}^D y[d]^2 - \sum_{d=s+1}^D 2 \times x[d] \times y[d].$$

By this analysis, we find the final term $\sum_{d=s+1}^D x[d] \times y[d]$ is the inner product between two partial vector $x[s+1 : D]$ and $y[s+1 : D]$, which could be approximated by Cauchy–Schwarz inequality

$$\sum_{d=s+1}^D x[d] \times y[d] \leq \sqrt{\sum_{d=s+1}^D x[d]^2 \times \sum_{d=s+1}^D y[d]^2}. \quad (3.7)$$

After combining with (3.6) and (3.7), we derive the bounds as

$$LB(x, y) = \sum_{d=1}^s (x[d] - y[d])^2. \quad (3.8)$$

$$\begin{aligned} UB(x, y) &= \sum_{d=1}^s (x[d] - y[d])^2 \\ &+ \sum_{d=s+1}^D x[d]^2 + \sum_{d=s+1}^D y[d]^2 \\ &+ 2 \times \sqrt{\sum_{d=s+1}^D x[d]^2 \times \sum_{d=s+1}^D y[d]^2}. \end{aligned} \quad (3.9)$$

We could notice that the calculation of the bounds only needs the first s dimensions of x and $\sum_{d=s+1}^D x[d]^2$. Therefore, we only need one more number $\sum_{d=s+1}^D x[d]^2$ to get the bounds for the unobserved part $x[s+1 : D]$.

3.5.3 Calculation of our Bounds

After the derivation of the bounds, we describe the procedure of calculating them in our framework.

For the query q_t , at the first round (i.e. $j = 1$), P sends the first s_1 dimensions of q_t and $\sum_{d=s_1+1}^D q_t[d]^2$ to each M_i . With these values, each M_i would be able to calculate the lower bounds $LB_1(q_t, x)$ and upper bounds $UB_1(q_t, x)$ for each $x \in X_i$. Then, after P getting the k_{th} smallest upper bounds as *thr*, we could run the pruning procedure.

In each following round (i.e. $j > 1$), P sends the next s_j dimensions of q_t to each M_i whose instances were not pruned completely. These M_i will update their bounds as follows:

$$\begin{aligned} LB_j(q_t, x) &= LB_{j-1}(q_t, x) + \sum_{d=p_{j-1}+1}^{p_j} (q_t[d] - x[d])^2. \\ UB_j(q_t, x) &= LB_j(q_t, x) \\ &+ \sum_{d=p_j+1}^D q_t[d]^2 + \sum_{d=p_j+1}^D x[d]^2 \\ &+ 2 \times \sqrt{\sum_{d=p_j+1}^D q_t[d]^2 \times \sum_{d=p_j+1}^D x[d]^2}. \end{aligned}$$

where $p_j = \sum_{i=1}^j s_i$, LB_j and UB_j indicate the lower bounds and upper bounds at the round j respectively.

We call those p_j as pivots, which mean that each machine would observe the first p_i elements of q_t at the round j .

3.5.4 Finding the Threshold in Distributed Machines

The question now is to find the threshold thr for pruning. We know that thr is the k_{th} upper bounds in all bounds of these round. However, these bounds are calculated by each local machine and lied there. That means we have to find it from a distributed data. In [3], we directly send these bounds computed in each M_i back to the server P . However, it would make the transmission cost grow linearly with the number of total instances in these distributed machines and lead to expensive cost when our dataset is extremely huge. As a result, we propose a method that could make the growth of the cost independent with the number of total instances.

To be simplified, we could think this problem as follows: given many distributed numbers N_i , we want to find the k_{th} largest number among these N_i . The N_i here actually means the upper bounds at the machine M_i in our framework. Once we model this problem as this, we could solve it through modifying the work of [2].

In [2], there are also many phases to find the kNN . In the first phase, the server P would send the whole query to every machine. Then, in the following phases, it just focuses on finding the instances with the k_{th} largest distance with the query. To make it fit our problem, we can only use the phases of PRP except the first phase to find the k_{th} largest upper bound as our threshold thr . Its cost is linear to

$$m \times \left(\left\lceil \frac{k}{m} \right\rceil + 1 \right),$$

which is much lower than [3] when the number of total instances is very large.

3.6 Decision of the Pivots

The remaining problem is to decide how many dimensions (i.e. s_j) of \hat{q}_t we have to send from the server P in each round j . If we send too few dimensions, the bounds would be too loose to prune any candidates and we will spend much unnecessary cost in finding the thresholds. On the other hand, if we send too many dimensions, although it could allow us to prune many candidates at once, it would send too many dimensions to some candidates which could be pruned by much fewer dimensions. That would also lead to the waste the transmission cost. Therefore, in this section, we propose a simple but effective method to decide how many dimensions of \hat{q}_t we should send from P in each round.

3.6.1 Estimation of the Number of Residual Machines

From the discussion above, we could notice that the decision of pivots is highly dependent on the transmission cost. Therefore, if we could estimate the cost as a cost function of the pivots p_j , we could decide these pivots by optimizing this function.


However, to estimate the cost, we need to know the number of residual candidates machines before sending those dimensions of \hat{q}_t in each round. But it is almost impossible to know how many machines would be left after we send part of \hat{q}_t before we actually send the part of \hat{q}_t . Therefore, we estimate a vector called $EstResMach \in \mathbb{R}^D$ where $EstResMach[j]$ indicates our estimate of the number of residual machines *after* we send $\hat{q}_t[1 : j]$ to each local machine.

To allow P to estimate this vector $EstResMach \in \mathbb{R}^D$ without causing any more transmission cost, we use the history information from $\hat{q}_1, \dots, \hat{q}_t$. Suppose that we just finish finding the answer for \hat{q}_t , during the procedure, we would collect some such pairs of information $(index_j, ResMach_j)$ for some j where $index_j$ means each candidate machine would observe the first $index_j$ dimensions of \hat{q}_t at the round j and $ResMach_j$ indicates the number of residual machines after pruning by $\hat{q}_t[1 : index_j]$. For those dimensions which are not in these pairs, we use linear interpolation to estimate their $ResMach$.

Figure 3.4 is an example for the above procedure. The blue vector means the pairs of information we collected for \hat{q}_t as $(2, 400), (4, 390), \dots, (D-3, 15), (D-1, 13), (D, 1)$.

And the red vector is the results after linear interpolation.

1	2	3	4	...	D-3	D-2	D-1	D
x	400	x	390	...	15	x	13	1



1	2	3	4	...	D-3	D-2	D-1	D
400	400	395	390	...	15	14	13	1

Figure 3.4: The example of linear interpolation.

By average this vector (the red one) collected from every query as $EstResMach$, P could estimate the number of residual machines *before* sending a query.

3.6.2 Estimation of the Transmission Cost

Once we have the vector $EstResMach$, we are ready to estimate the transmission cost *before* sending the query. For a query, we could estimate its transmission cost in the first round

$$Cost_1 = m \times s_1 + Cost_{PRP}(m)$$

And for those round $j > 1$ as follows,

$$Cost_j = EstResMach[p_{j-1}] \times s_j + Cost_{PRP}(EstResMach[p_{j-1}])$$

where $p_j = \sum_{i=1}^j s_i$.

The term $m \times s_1$ means the cost to send part of \hat{q}_t in the first round which is the length of this part times the total number of machines. Similarly, the term $EstResMach[p_{j-1}] \times s_j$ is the cost to send the next part of \hat{q}_t in the round j to the residual machines. On the other hand, the term $Cost_{PRP}(i)$ is the total cost to find the threshold thr and then send it to all i residual machines. Therefore, we could minimize the total cost to get the optimal pivots.

However, since there are D dimensions could be picked .there are too many variables to decide in this problem. According to our experiments, the most crucial variable is the

number of dimensions which will be sent in the first round, which is s_1 in this optimization problem. The other s_j don't have such huge influence like s_1 . Therefore, we make all s_j be equal and then simplify this optimization problem as follows,

$$\begin{aligned} & \underset{StartD, EachLenD}{\text{minimize}} \quad \sum_j Cost_j(StartD, EachLenD) \\ & \text{subject to } StartD, EachLenD \in \mathbb{N} \end{aligned} \quad (3.10)$$

where

$$Cost_1 = m \times StartD + Cost_{PRP}(m)$$

$$Cost_j = EstResMach[p_{j-1}] \times EachLenD + Cost_{PRP}(EstResMach[p_{j-1}]), \forall j > 1$$

$$p_j = StartD + (j - 1) \times EachLenD.$$

3.6.3 Coordinate Descent to Decide the Pivots

Now we have reduced the number of variables to only two variables: $StartD$ and $EachLenD$. Since we need to solve this optimization problem for every new query, we apply the Coordinate Descent method to solve this for efficiency. After solving the optimal $StartD$ and $EachLenD$ for this query, we are able to decide its pivots as below:

$$p_j = StartD + (j - 1) \times EachLenD \quad (3.11)$$

3.7 Overall Framework

3.7.1 Importance-Selecting Function

Before giving the final version of our framework, we define a selection function S_i for convenience.

At each round j for q_t , we would decide what values to send from P to each M_i and then calculate the bounds at M_i . Actually, we could use a function to indicate these values. That is, our values sent from P to M_i at the round j for q_t are the return values of

the importance-selecting function $S_i(q_t, k, \theta_t; W_i)$, which could be formulated as below.

$$S_i(q_t, j, \theta_t; W_i) = \hat{q}_t[p_{j-1}^t + 1 : p_j^t] \cup Meta_j$$

where

$$\begin{aligned} \hat{q}_t &= W_i q_t, \\ \theta_t &= (StartD_t, LenD_t) \\ p_j^t &= \min\{D, StartD_t + LenD_t \times (j - 1)\}, \forall j \geq 1, \forall t \\ p_0^t &= 0, \forall t \\ Meta_1 &= \sum_{d=StartD_t+1}^D \hat{q}[d]^2, Meta_j = \emptyset, \forall j > 1 \end{aligned} \tag{3.12}$$

Although we sent the dimensions of transformed query \hat{q}_t instead of the original query q_t to each local machine, the cost of sending the values of the importance-selecting function S_t is the length of the original query q_t plus one which is the norm of the latter part of \hat{q}_t in the first round. That is

$$\sum_j S_i(q_t, j, \theta_t; W_i) = D + 1$$

Note that this is the worst case, in our experiments, we could prune most candidates and thus don't need to send it until the last rounds.

3.7.2 Overall Framework

Finally, we give our algorithm as the following two tables.

Input:	$X_1, X_2, X_3, \dots, X_m$
Output:	$W_1, W_2, W_3, \dots, W_m$
1 for	$i = 1; i \leq m; i = i + 1$ do
2	M_i : Compute W_i with X_i by solving the optimization problem (3.2);
3	M_i : Send W_i back to P ;
4 end	

Algorithm 1: First Phase

```

Input:  $q_t, k$ 
Output:  $k$ NN of  $q_t$ 
1  $P.CandidateMach(m) = True;$ 
2  $P.Counter = N;$ 
3  $P$ : Solve  $\theta_t$  by the section 3.6.3;
4 for  $j = 1; P.Counter > k; j = j + 1$  do
5   for  $i = 1; i \leq m; i = i + 1$  do
6     if ( $CandidateMach(i) == True$ ) then
7        $P$ : Send the values of  $S_i(q_t, j, \theta_t; W_i)$  to  $M_i$ .;
8        $M_i$ :  $\forall x \in X_i$ , compute bounds by the section 3.5.3;
9     end
10  end
11   $P$ : Use PRP to find the threshold  $thr$  and send it to every candidate machine.;
12   $P.Counter = 0$ .;
13  for  $i = 1; i \leq m; i = i + 1$  do
14    if ( $CandidateMach(i) == True$ ) then
15       $M_i$ : Prune instances by  $thr$ , return number of residual instances  $n_i$  back
        to  $P$ .;
16      if ( $n_i == 0$ ) then
17         $P.CandidateMach(i) = False$ ;
18      end
19       $P.Counter+ = n_i$ ;
20    end
21  end
22 end

```

Algorithm 2: Second Phase

Chapter 4

Experiment

4.1 Experiment Setup

In this section, we discuss the results of our experiments. There are four parts in our experiments. First, we compare our framework with other frameworks in the communication cost. Second, since there are several stages of improvement in our framework, we discuss each of their influence to our final model. Third, we consider the amortization of transmitting the orthogonal matrices. Finally, we compare the power of pruning among different bounds. For every experiment, we collect the results of 100 experiments by randomly picking our 100 instances as the queries.

4.2 Data Description

The table 4.1 is the description of those datasets we used in our experiments. Note that the $n \times m$ in the final column means that there are n instances placed in each machine and m machines used in this experiments. For instance, for the image dataset ANN with SIFT feature, there are totally 5000 machines and each has 200 instances in our experiments.

4.2.1 Time Series Data

The time series datasets we used is a synthetic dataset. We use the random walk data model in [12]. Each time series is generated by a random walk whose every step size is

Table 4.1: Summary for each dataset

Type	Dataset	Feature	Num of Dimensions	Num of Instances
Time Series	Random Walk	$N(0, 1)$	128	200×5000
Image	ANN	SIFT	128	200×5000
	Flickr	CSD	256	500×2000
		SCD	256	500×2000
Audio	Million Songs	MVD	480	500×1900
		TRH	480	500×1900

a normal distributed random number with mean 0 and standard deviation 1. We also use this model to generate the synthetic dataset in the experiments of MsWave [3].

4.2.2 Image Data

We use two datasets in our experiments for images. First is the data provided in [13], which is a widely used dataset for evaluate the performance of approximate nearest neighbors search algorithms. The another one is the Flickr datasets with two kind of features used in [14]. The dataset is also a widely used dataset in the task of image retrieval. The CSD indicates *Color Structure Descriptor* while the SCD means *Scalable Color Descriptor*.

4.2.3 Audio Data

Here we use the audio data named Million Song Dataset from [15] which is a free-available collection of audio features for a million contemporary popular music tracks. For the features, MVD means *Modulation Frequency Variance Descriptor* and TRH is *Temporal Rhythm Histograms*. Please refer to [16–18] to see the details about how these features were extracted.

4.3 Comparison Among all Frameworks

4.3.1 Frameworks for Comparison

We compare our framework with those methods mentioned in the related work chapter. From [2], we use CP and PRP but with slightly modifications. In the origin CP, every machine would return the top k instances once receiving the query. But there is a trivial improvement that every machine only return the *distances* of these top k instances. Then, the server could know the distances of the k NN of this query and then ask those machines with answers to return those instances. Although it is a slight modification, it could reduce the cost of CP a lot when the number of machines is large. Also, we run LeeWave [1] in these experiments for comparison. But some of its experiments are not finished due to its too long computation time. We call our final framework as *Main* in the following figures.

Note that in the following experiments, the cost of our framework here does not include the cost of sending the orthogonal matrices. We will prove in the section 4.5 that the total cost including the matrices could be amortized by enough queries and thus to achieve the cost here. That is, we could see the cost of our framework here as the cost after amortized by enough queries. Due to the time limitations, we didn't conduct enough number of queries to achieve the amortized results for each dataset.

4.3.2 Results of Different Frameworks

The following figures are the results of our experiments. The x axis indicates the number of local machines while the y axis is the total transmission cost of the 100 queries. Since the differences among these are too large, we transform the y axis to logarithmic scale.

From these figures, we could see that our framework used the least transmission cost among all these frameworks for every dataset. And these differences between our framework and each other framework increased as the number of local machines increased. The reason is that when the number of local machines increases, there would be a higher chance to prune more local machines in the early round since the ratio of pruned machines

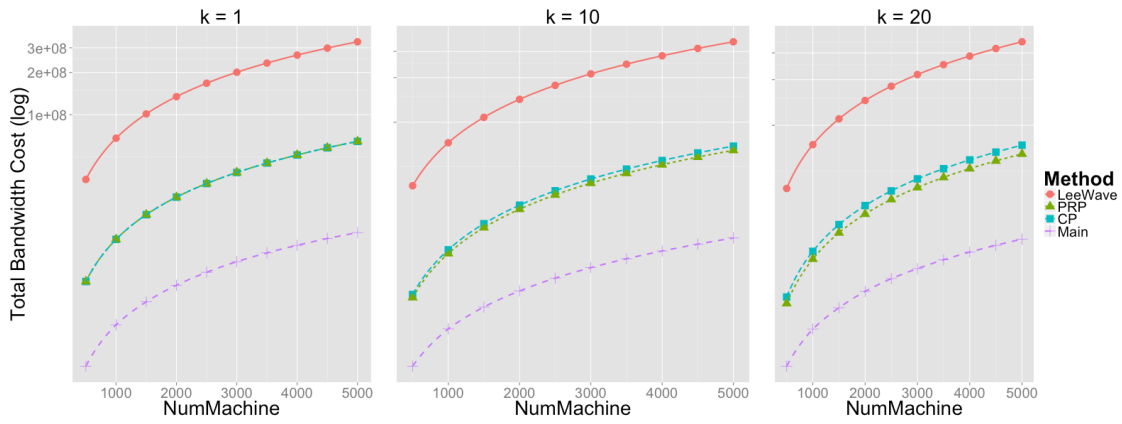


Figure 4.1: Different Frameworks on Time Series

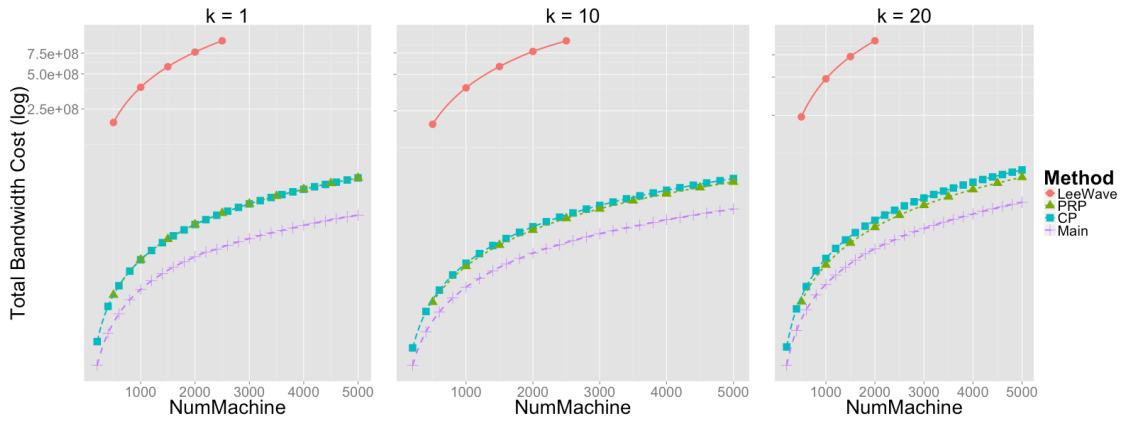


Figure 4.2: Different Frameworks on ANN

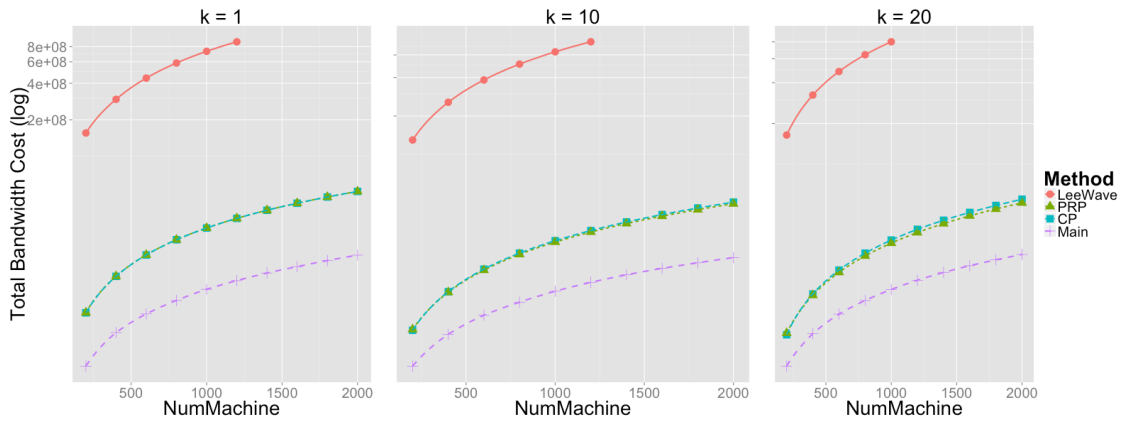


Figure 4.3: Different Frameworks on Flickr: CSD

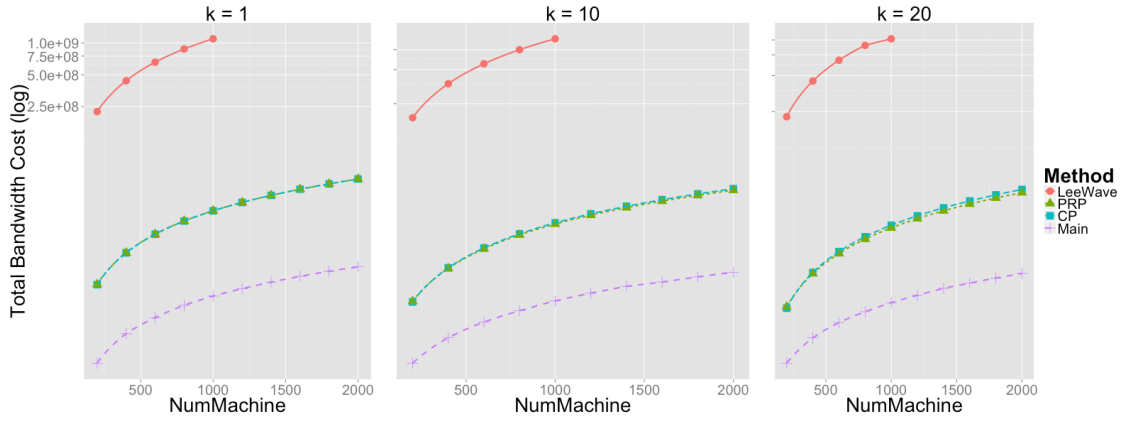


Figure 4.4: Different Frameworks on Flickr: SCD

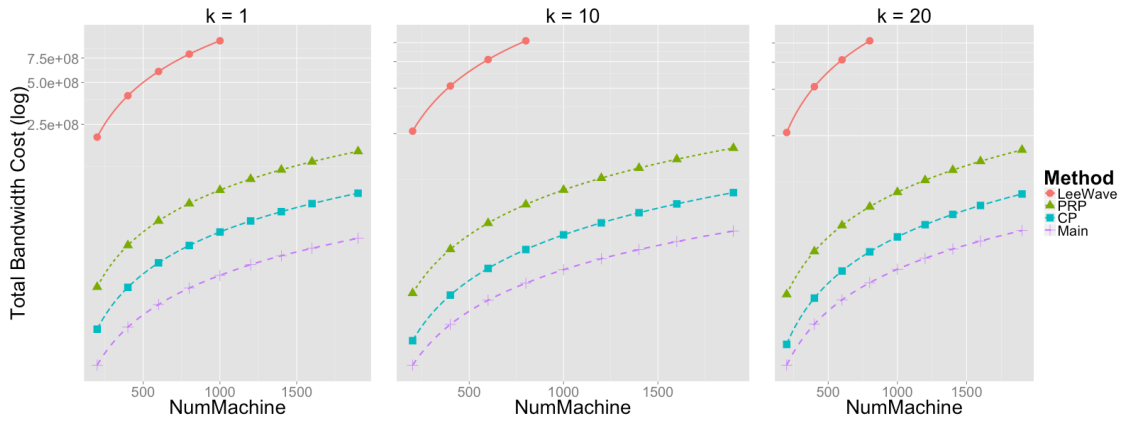


Figure 4.5: Different Frameworks on Million Song: MVD

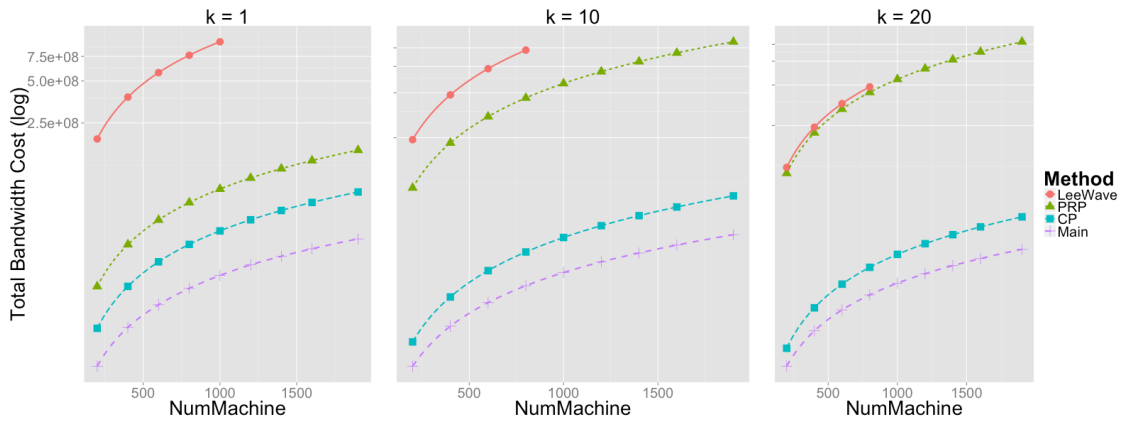


Figure 4.6: Different Frameworks on Million Song: TRH

doesn't change too much. But the other frameworks would be more sensitive to the number of local machines.

The performance of CP and PRP are similar to each other in most dataset. It is because when k is much smaller than the number of total local machines, their procedure would be almost the same except the final stage. For those cases where PRP is worse than CP, we could find that PRP would return too many instances in the final stage while CP would only return exact k instances back to the server. But these results would highly depend on the distribution of the instances among these local machines and could not be controlled by the algorithms themselves.

We could also notice that LeeWave needs the largest transmission cost to finding the k NN for the 100 queries for all datasets. When the type of the dataset is time series (figure 4.1), the difference between LeeWave and other frameworks are smaller than other datasets since its pruning power is still effective for this type of dataset. For other types of datasets like images or audio, LeeWave almost could not prune any candidate machines until the last round which would send the whole query to each machine and thus used a large amount of transmission cost.

Even though the LeeWave could prune some candidate machines when the type of the dataset is time series, there is a big gap between it and CP, PRP. The reason of the large transmission cost in LeeWave here is not the pruning power but its way to calculate the bounds. In each round, after sending the coefficients in this level of the error tree, LeeWave requires every instance to return some metadata back for calculating the bounds at the server. This causes one term in the transmission cost of LeeWave would grow linearly with the total number of instances in all local machines. Since we conducted our experiments on the datasets with about one million instances, this term would be large enough to cover the saving from the pruning. On the other hand, the transmission cost of CP and PRP is independent of the total number of instance but only dependent on the number of local machines and k . Therefore, when the total instances is large, LeeWave could use more transmission cost than CP and PRP even when its pruning power still exists.

4.4 Comparison Among our Framework with Different Configurations

4.4.1 Configurations for Comparison

There are many stages of algorithms which build the final version of our framework. Therefore, in this section, we would like to discuss the performance of our framework with or without each of these algorithms. We conducted the experiments for the following different configurations of our framework.

Main: It is the final version of our framework which obtains all algorithms mentioned in the chapter of methodology.

NoW: To test the influence of the orthogonal transformation, we consider the configuration which has all the algorithms except the orthogonal transmission. We could take it as the special case of *Main* where $W_i = I, \forall i = 1, 2, \dots, m$.

NoPRP: In the section 3.5.4, we use PRP to find the threshold for pruning in each round. Therefore, we want to compare this modification with the original version which directly returns all bounds back to the server.

NoCD: In the section 3.6.3, the number of dimensions we sent for every query are decided dynamically by solving an optimization problem with Coordinate Descent. Here we just send 10% of the transformed query in each round to the server for comparing the improvement from the optimization.

4.4.2 Results of Different Configurations

We provide the results of the experiments to compare our framework with different configurations in the following figures. The meanings of their axes are same with those in the section 4.3.2.

In these figures, we could see that although every algorithm could make our framework better, their influences are very different to each other. The algorithm which makes the largest difference with our final framework is PRP. It is very obvious since we use

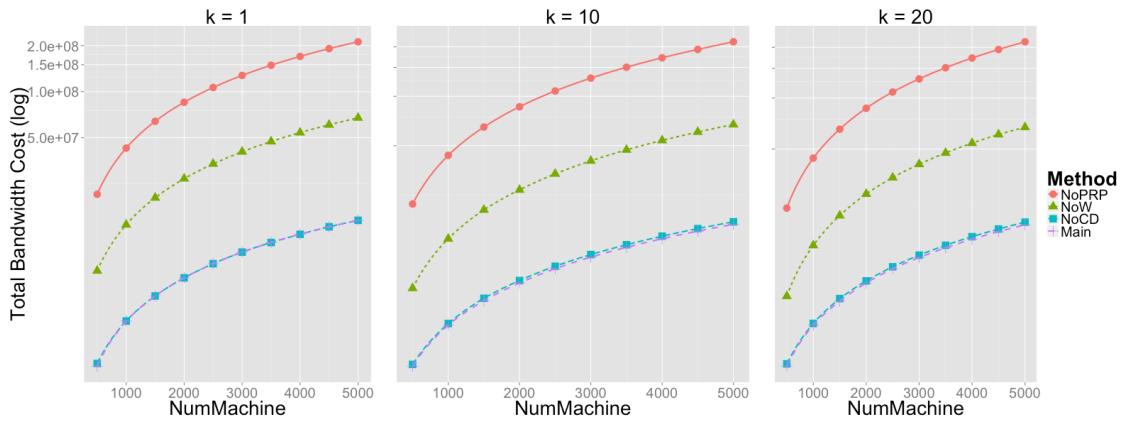


Figure 4.7: Different Configurations on Time Series

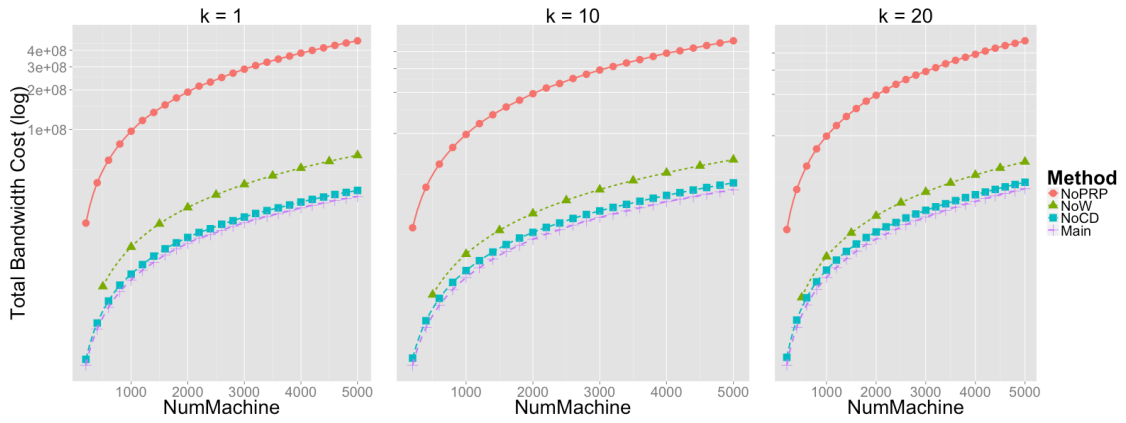


Figure 4.8: Different Configurations on ANN

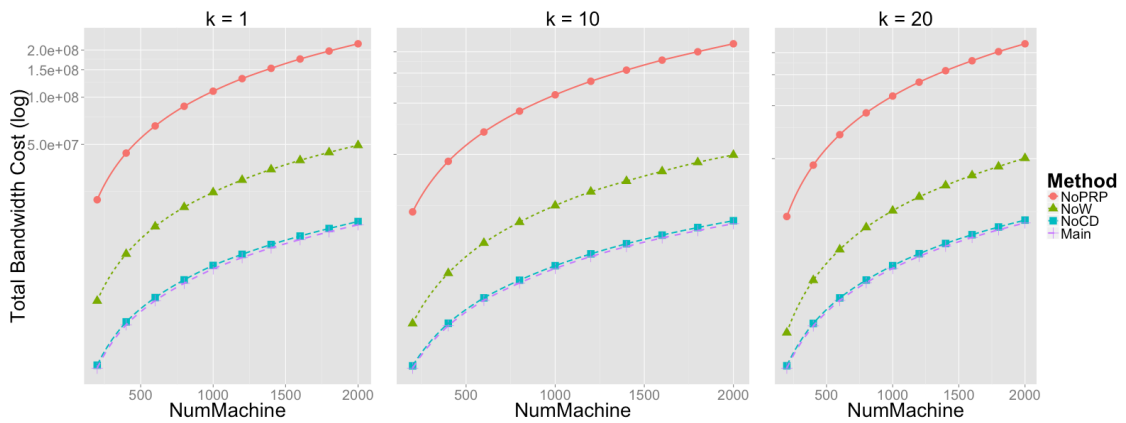


Figure 4.9: Different Configurations on Flickr: CSD

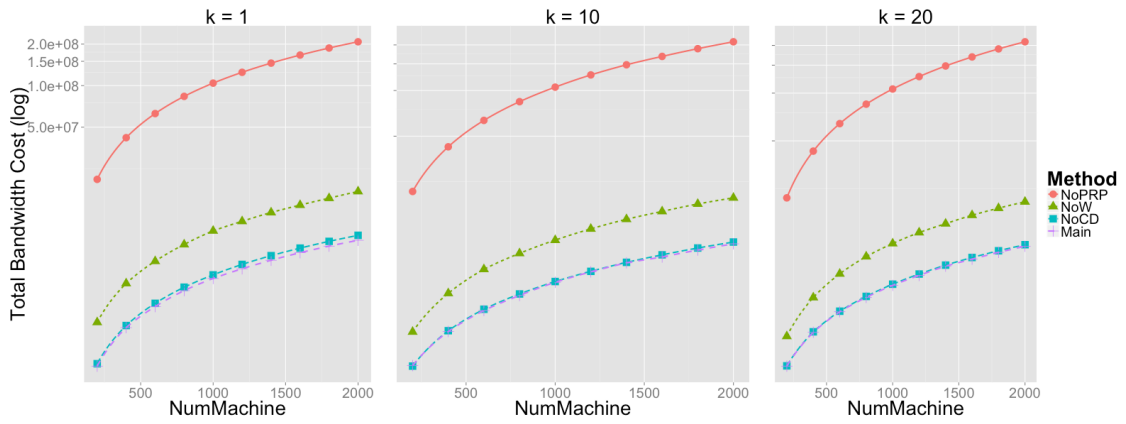


Figure 4.10: Different Configurations on Flickr: SCD

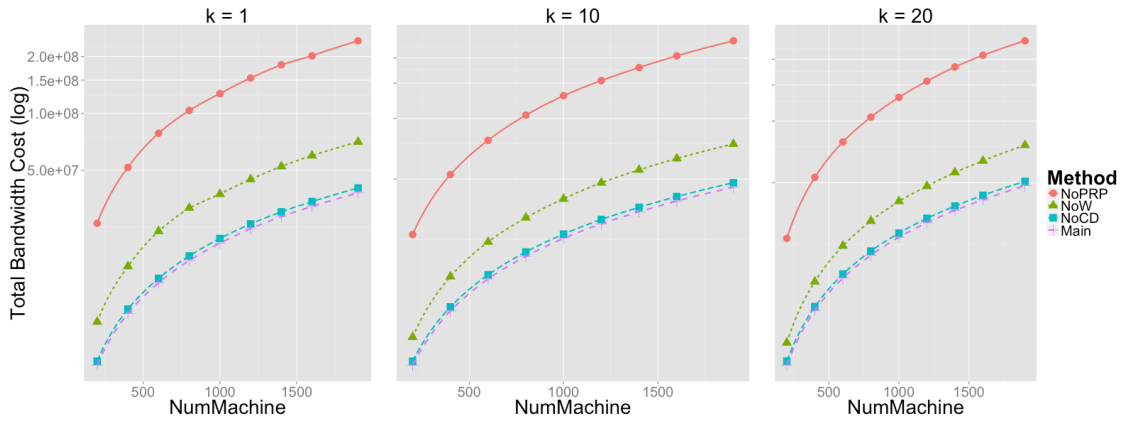


Figure 4.11: Different Configurations on Million Song: MVD

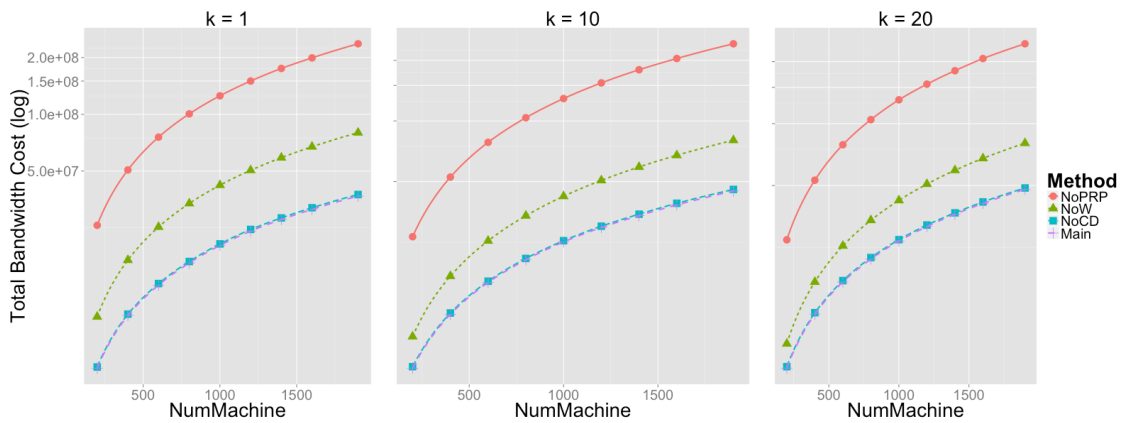


Figure 4.12: Different Configurations on Million Song: TRH

PRP to find the threshold could make the transmission cost independent with the total number of instances. This makes a big difference when the datasets here contain about one million instances. The algorithm with the second big improvement is the orthogonal transformation. This confirms our assumption that we could save transmission cost by improving the power of pruning with the help of the orthogonal transformation. Finally, the algorithm of dynamically deciding the pivots with Coordinate Descent seems to have the least influence to our final framework in these figures. However, it does contribute much to the saving of transmission even though not as significant as the others. With the help of it, we don't have to worry about deciding the pivots for every dataset.

4.5 Number of Queries for Amortizing the Cost of Matrices

In the beginning of this chapter, we mentioned that the cost of our final framework showed in the previous experiments didn't count the cost of sending those orthogonal matrices. The matrices cost from the first phase of our framework is as below.

$$\begin{aligned} Cost_{Matrix} &= m \times SingleMatrixCost \\ &= m \times \frac{D \times (D - 1)}{2} \end{aligned} \tag{4.1}$$

where we have given the proof that we could use $\frac{D \times (D - 1)}{2}$ parameters to represent an orthogonal matrix in the section 3.4.7.

Although it would cause a large cost, it only needs to be done for one time. Therefore, we could amortize it to the cost of every query in the second phase. If the number of queries is large enough, we could amortize the matrices cost to a very small ratio of the total transmission cost. But due to the time limitation, we didn't conduct the experiments for a large number of queries, therefore, we estimate the number of queries we need as below.

Given a small ratio (here we use $r_{ideal} = 5\%$), we could estimate how many queries we

need for amortizing the matrices cost to this ratio of the total transmission cost as below.

$$\begin{aligned}
Cost_{Total} &= \sum_{t=1}^{\|Q\|} Cost_{our}(q_t) + Cost_{Matrix} \\
Cost(i) &= \frac{\sum_{t=1}^{\|Q\|} Cost_{our}(q_t)}{\|Q\|} \times i + Cost_{Matrix}
\end{aligned} \tag{4.2}$$

In the equation above, we estimate the total cost of i queries as $Cost(i)$ by summing up the matrices cost and the average cost of the second phase in our past experiments multiplied by i . In other words, we estimate the cost of the second phase for a single query as $\frac{\sum_{t=1}^{\|Q\|} Cost_{our}(q_t)}{\|Q\|}$.

Therefore, our goal becomes how many queries we need to make the ratio of $Cost_{Matrix}$ lower than r_{ideal} . We could write it as the following problem:

$$\begin{aligned}
&\underset{i}{\text{minimize}} \quad \frac{Cost_{Matrix}}{Cost(i)} < r_{ideal} \\
&\text{where } i \in \mathbb{N}
\end{aligned} \tag{4.3}$$

It could be solved easily.

Table 4.2: Number of queries to amortize the cost of matrices

Type	Dataset	Feature	Total Num of Instances	Num of Queries
Time Series	Random Walk	$N(0, 1)$	1000000	4655
Image	Flickr	ANN	1000000	1997
		SIFT	1000000	6313
		CSD	1000000	12724
Audio	Million Songs	SCD	1000000	6979
		MVD	950000	7076
		TRH	950000	

We solve (4.3) for each dataset when the number of local machines reaches the maximal and $k = 10$. The results are put in the table 4.2. From this table, we could find that although the matrices cost is large, we could amortize it to less than 5% with a few thousands of queries. For a dataset with one million of instances, the number of queries could be achieved easily.

From the figure 4.13, we could notice that $\frac{Cost_{Matrix}}{Cost(i)} < r_{ideal}$ decreases fast as the number of queries increases. As our estimation, this ratio achieve 5% when the number

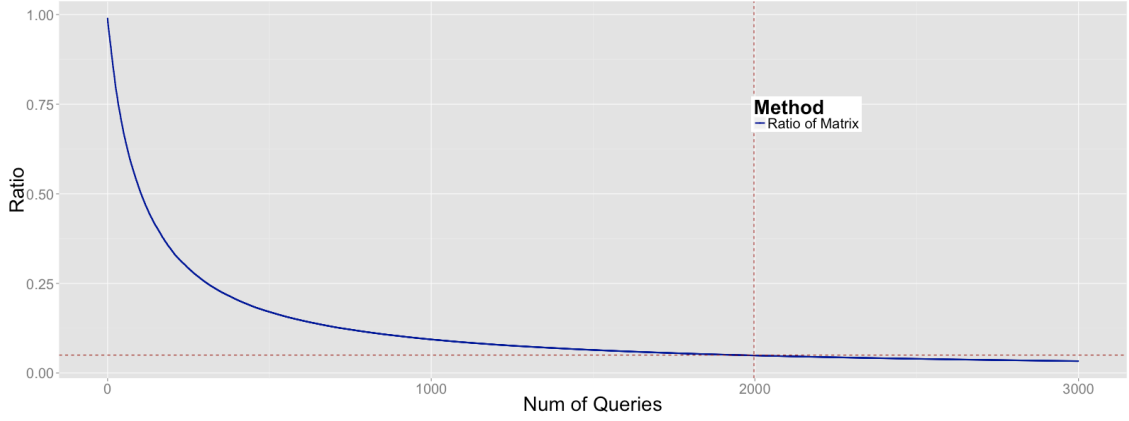


Figure 4.13: Pruning Results on Time Series

of queries reach about 2000.

4.6 Power of the Pruning Procedure

4.6.1 Methods with Different Bounds

There are three different bounds which we could compare. The first one is the bounds of LeeWave, which comes from the Haar wavelet transformation. The second one is the bounds we derived in the section 3.5.2. That is the method *NoW* we mentioned above. The final bounds is those improved by the orthogonal transformation.

4.6.2 Results of Pruning

First, we compare the bounds we derived and the bounds of LeeWave. Since the number of dimensions LeeWave sent in every round is restricted by the shape of the error tree, we could only compare the number of residual machines under these dimensions between LeeWave and our method. For LeeWave, we just average the results of each query. As for our method, we use our estimation for each dimension used in the section 3.6.1 and pick those dimesions used in LeeWave.

From the table 4.3 for time series dataset, we could see that although LeeWave could prune some local machines in the last few round, our method outperforms it by pruning machines earlier and more. This improvement mainly comes from our tighter bounds that

provide a more powerful pruning mechanism.

Table 4.3: Time Series, Number of residual machines after sending some dimensions for 5000 machines

Method \ Dim	2	4	8	16	32	64	128
Ours	4823.24	4469.71	3762.66	2344.46	540.14	69.11	9.99
LeeWave	4999.84	4984.90	4895.18	3722.49	938.80	111.00	9.99

However, for the other types of datasets, from these tables we could notice that LeeWave almost could not prune any local machine until the last round. The reason is that since the bounds of LeeWave come from the Haar wavelet transformation, which is suitable for the time series feature, but not workable for the images and audio datasets from these experiments. Although the power of pruning of our method would be different for different types of data, we could prune about half of total local machines after sending half of the query feature vector. This allows us to avoid sending redundant information to unnecessary machines.

Table 4.4: Flickr:CSD, Number of residual machines after sending some dimensions for 1000 machines

Method \ Dim	2	4	8	16	32	64	128	256
Ours	995.37	986.10	967.56	930.49	856.34	573.79	410.3	9.89
LeeWave	1000	1000	1000	999.90	995.16	967.26	808.16	9.89

Table 4.5: Million Songs: MVD, Number of residual machines after sending some dimensions for 800 machines

Method \ Dim	2	6	12	26	52	104	210	420
Ours	799.41	797.05	793.5	785.23	769.87	739.14	467.74	12.77
LeeWave	800	800	800	800	799.07	794.93	772.91	12.58

Finally, let's discuss the pruning power with or without the help of the orthogonal transformation, which is our final method and *NoW*. We plot our estimation in the section 3.6.1 and compare their estimation. However, we found that for some datasets, these could not reflect the status of pruning for *NoW* since the decision of the pivots are concentrated in

the latter part of the vector. As a result, we add another method named *NoCDW* which is same with *NoW* but instead of dynamically deciding the pivots, it sends 10% of the vector for each round. In the following figures, the red circles are our final method, the black ones are *NoW*, and the green ones are *NoCDW*.

For all these figures except the figure 4.16, we could see that our method could prune most of the residual machines with about 50% of the total dimensions. From *NoCDW*, we could find we have to send many dimensions to prune some machines without the orthogonal transformation. This difference comes from the tightness of the bounds. After optimizing with the orthogonal transformation, our method could use much tighter bounds to prune machines than the original bounds. With the enhanced pruning power, our method could save much transmission cost.

On the other hand, in the figure 4.14 and figure 4.18, the curve of *NoW* is very different with *NoCDW*. The reason is that for these datasets, the pivots learnt from the estimation cost problem concentrate at the latter part of the vector, which leads to very poor estimation for residual machines for the earlier part of the vector. So, its curve could not reflect the pruning power of the bounds without transformation. This also implies the policy of sending 10% total dimensions would not be the best policy for these datasets since the curve of *NoW* is much different with the one of *NoCDW*. For the other figures, we could see the curve of *NoW* is similar with *NoCDW*. This implies the 10% policy would be fine for these datasets. But since we could not know whether this policy would work for a dataset in advance, the best policy is to decide the pivots dynamically as we mentioned before.

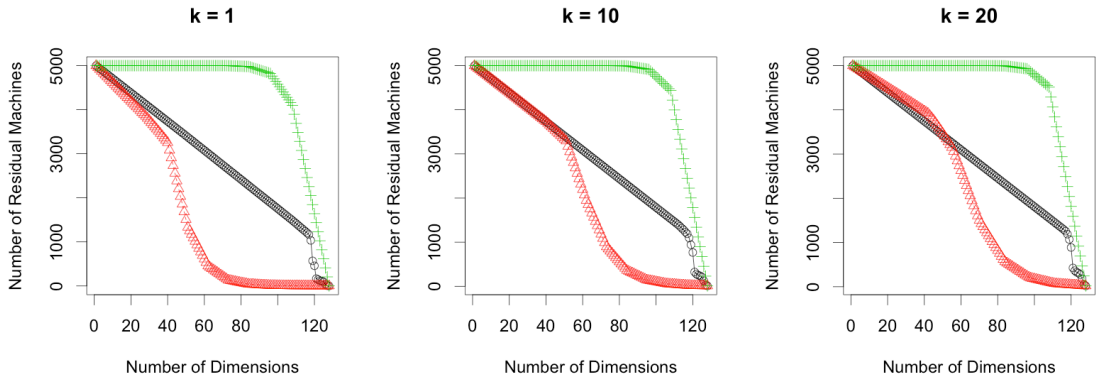


Figure 4.14: Pruning Results on ANN

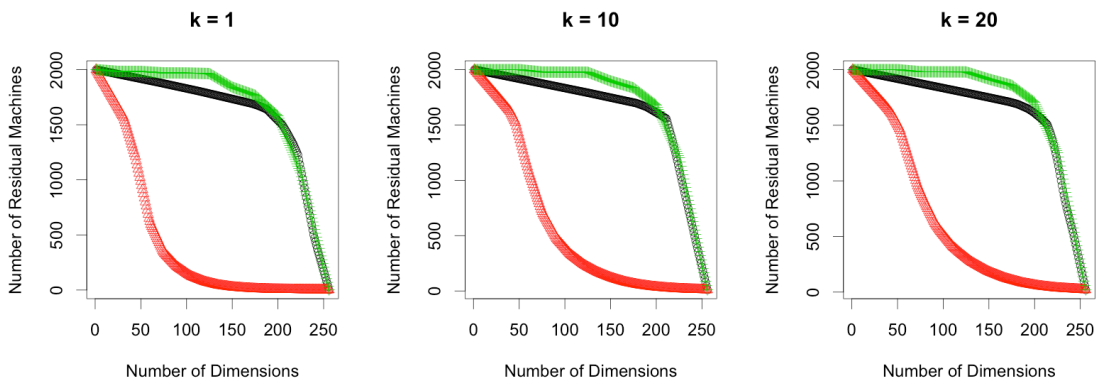


Figure 4.15: Pruning Results on Flickr: CSD

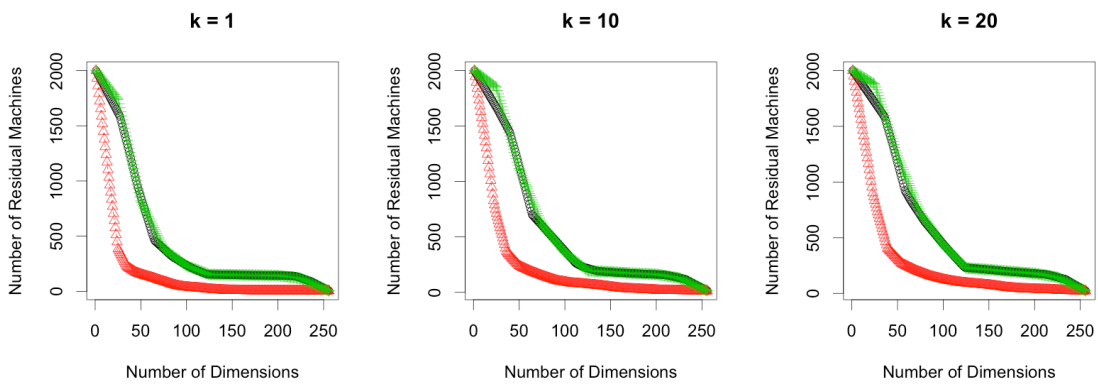


Figure 4.16: Pruning Results on Flickr: SCD

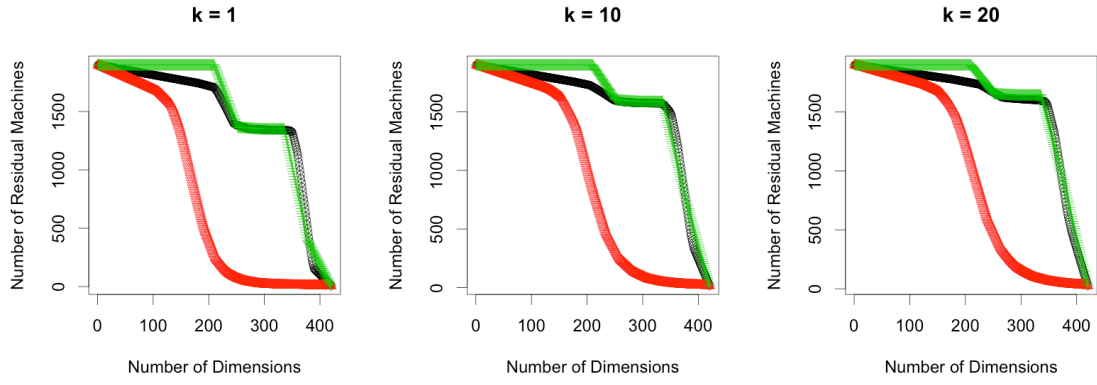


Figure 4.17: Pruning Results on Million Song: MVD

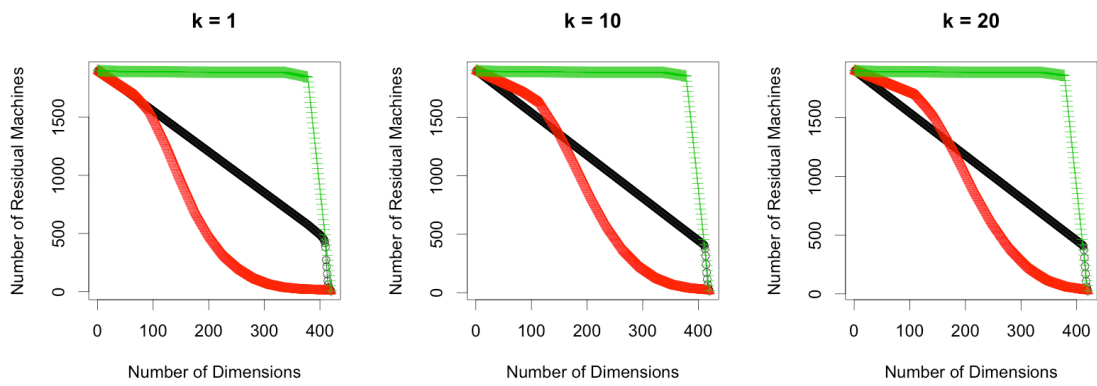


Figure 4.18: Pruning Results on Million Song: TRH

Chapter 5

Conclusions

Distributed similarity search becomes increasingly crucial as there are various types of datasets on the mobile devices today. However, we need to deal with the huge transmission cost while searching the answers among these distributed machines. In this paper, we propose a two-phase framework which is able to use much less transmission cost to find our desired answers with the help of the orthogonal transformation. We conducted experiments on various types of datasets to confirm the generalization of our framework. From the results of these experiments, we could see that our framework could maintain its performance while facing very different kinds of data.

From our framework, we could find that there are many useful properties of orthogonal transformation, especially the property to preserve the distance relationship. As long as we could design a reasonable objective function of the transformation and optimize it well, it is possible to make those values which are unchangeable in the original space become functions of the transformation in the transformed space while preserving many properties in the original space.

Although our framework outperforms other frameworks in the experiments, there are some directions which could further improve our framework.

First, the transmission cost of the first phase is still too much. If we could reduce the matrices cost more, we could use less queries to amortize this cost. We could compress these matrices and thus send them with less transmission cost. After the server receives these matrices, they could be reconstructed back to orthogonal matrices by SVD. However,

the compression might be lossy. And it would become a tradeoff between the performance of pruning and the transmission cost.

The another direction would be the estimation of the transmission cost before sending a query. Before estimating this cost, we have to estimate the number of residual machines like we mentioned in the section 3.6.1. In this paper, we use a simple linear interpolation to estimate those dimensions with empty value. If we could estimate them more precisely, we could get a better pivots to send the query.

Since our framework are comprised of several individual subproblems, its overall performance could be improved as long as any of the subproblem is solved by a better alorithm.

Bibliography

- [1] Mi-Yen Yeh, Kun-Lung Wu, Philip S. Yu, and Ming-Syan Chen. LeeWave: level-wise distribution of wavelet coefficients for processing knn queries over distributed streams. *Proc. VLDB Endow.*, 2008.
- [2] Apostolos N. Papadopoulos and Yannis Manolopoulos. Distributed processing of similarity queries. *Distributed and Parallel Databases*, 2001.
- [3] Jui-Pin Wang, Yu-Chen Lu, Mi-Yen Yeh, Shou-De Lin, and Phillip B. Gibbons. Communication-efficient distributed multiple reference pattern matching for m2m systems. *Proc. of IEEE ICDM*, 2013.
- [4] Sylvia Ratnasamy, Paul Francis, Mark Handley, and Richard Karp Scott Shenker. A scalable content-addressable network. *SIGCOMM*, 2001.
- [5] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM*, 2001.
- [6] Adina Crainiceanu, Prakash Linga, Ashwin Machanavajjhala, Johannes Gehrke, and Jayavel Shanmugasundaram. P-ring: an efficient and robust p2p range index structure. *SIGMOD*, 2007.
- [7] Ashwin R. Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury: Supporting scalable multi-attribute range queries. *SIGCOMM*, 2004.

- [8] Erik Buchmann and Klemens Bohm. Efficient evaluation of nearest-neighbor queries in content-addressable networks. *From Integrated Publication and Information Systems to Virtual Information and Knowledge Environments*, 2005.
- [9] Farnoush Banaei-Kashani and Cyrus Shahabi. Swam: A family of access methods for similarity-search in peer-to-peer data networks. *CIKM*, 2004.
- [10] Parisa Haghani, Sebastian Michel, Philippe Cudre-Mauroux, and Karl Aberer. Lsh at large – distributed knn search in high dimensions. *International Workshop on Web and Databases*, 2008.
- [11] Zaiwen Wen and Wotao Yin. Optimization with orthogonality constraints. <http://optman.blogs.rice.edu/>, Aril 2012.
- [12] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2012.
- [13] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2011.
- [14] Nitish Srivastava and Ruslan R. Salakhutdinov. Multimodal learning with deep boltzmann machines. In *Neural Information Processing Systems*, 2012.
- [15] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.
- [16] Thomas Lidy and Andreas Rauber. Evaluation of feature extractors and psycho-acoustic transformations for music genre classification. In *Proceedings of the Sixth International Conference on Music Information Retrieval (ISMIR 2005)*, pages 34–41, London, UK, September 11-15 2005.

- [17] A. Rauber, E. Pampalk, and D. Merkl. The SOM-enhanced JukeBox: Organization and visualization of music collections based on perceptual models. *Journal of New Music Research*, 2003.
- [18] A. Rauber and M. Frühwirth. Automatically analyzing and organizing music archives. In *Proceedings of the 5. European Conference on Research and Advanced Technology for Digital Libraries (ECDL 2001)*, Springer Lecture Notes in Computer Science, Darmstadt, Germany, Sept. 4-8 2001. Springer. <http://www.ifs.tuwien.ac.at/ifs/research/publications.html>.