CLC__________ Number__________

UDC__________ Classification Level__________

SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Undergraduate Thesis

Cover Ratio Maximization

Author :	Keming Li
Student ID :	11612126
Department :	Computer Science and Engineering
Major :	Computer Science and Technology
Supervisor :	Prof. Bo Tang
Finished Time :	May, 2020

诚信承诺书

1. 本人郑重承诺所呈交的毕业设计（论文），是在导师的指导下，独立进行研究工作所取得的成果，所有数据、图片资料均真实可靠。

2. 除文中已经注明引用的内容外，本论文不包含任何其他人或集体已经发表或撰写过的作品或成果。对本论文的研究作出重要贡献的个人和集体，均已在文中以明确的方式标明。

3. 本人承诺在毕业论文（设计）选题和研究内容过程中没有抄袭他人研究成果和伪造相关数据等行为。

4. 在毕业论文（设计）中对侵犯任何方面知识产权的行为，由本人承担相应的法律责任。

作者签名: 李可明

2020 年 5 月 14 日

Contents

ABSTRACT

Decision making problem is popular among these days. In rank aware processing, a user will only choose the option that ranks top-kF for himself. Concretely, preferences of users are usually represented as weight vectors. Each attribute of weight vector means how important is that attribute to that user. The score of an option respecting to a user is the dot product between the user's preference weight vector and the option. Only the options with top-k scores can attract the user. An option covers a user if and only if it can rank top-k for that user. Usually, a company has many products (options), each of which covers some of the users. A user covered by a company means at least one of its products covers this user. The company has to develop new product that satisfies a constraint and make its all products including this new product cover as more users as possible. In this paper, we study how to determinate which newly added option can maximize the cover ratio of the company. This problem is essential in developing new product, advertising, etc. We refer this problem as k-Cover Ratio Maximization. In this paper, we begin from top-k problem's computational geometric nature using Cell Tree to represented option spaces, and then from the relationship among constraint, options and user preference weight vectors to more efficiently solve this problem returning the exact optimal solution. We set a lot of experiments to show the efficiency of our optimizations and at the same time we found interesting relationship between the constraint and running time. Combining with the experience of decision making, we found that it is hard to tell what kinds of product could cover the most users when the constraint intersects with most the users top-k condition.

Keywords: User Cover ratio, Introduce new option, Top-k query, Weight vector

摘 要

近年来关于如何做决策的问题十分热门。在排名处理系统中，会假设一个用户只会从排入他（她）前 k 的产品中作出选择。具体地，用户对于产品各种属性的爱好会用权重向量来表示。权重向量每个维度的大小表现了该维度对于该用户的重要性。一个产品的对于一个用户的分数就是产品的各个指标得分组成的向量与该用户的权重向量的点乘。只有排进该用户前 k 的产品才有可能影响用户最后的选择。当且仅当一个用户的产品排该用户前 k 时我们称该产品覆盖这个用户。对于一个公司，它可能有很多产品，每个产品都有各自的覆盖用户群。当公司的至少一款产品能覆盖某个用户时，我们称该公司用户覆盖该用户。随着市场以及自身业务的发展，公司要研发一款新的满足一定约束条件的产品，使得公司的所有产品包括新产品总的尽可能覆盖最多的用户。在本篇论文中我们会讨论如何精确找到这个能使公司覆盖率最大的新产品。本篇论文讨论的问题在例如研发一款怎么样的新产品让公司总收益最大，如何加强广告宣传使总覆盖用户最多等非常多领域都有很大的应用价值，同时我们把这个问题命名为 k-Cover Ratio Maximization。在本篇论文中，我们从前 k 问题的计算几何性质入手用 Cell Tree 这种数据结构表达候选的产品空间，然后从约束条件、产品、用户、前 k 条件等内在联系做优化大大提高解决问题效率, 最后返回最优解的解集。我们做了许多实验证明我们的优化的有效性，同时发现了运行效率与约束条件的关系，与现实决策结合得出在付出成本与成为大多数人前 k 的条件相交时是比较难决定什么样的产品会覆盖更多的用户的。

关键词： 用户覆盖率，新产品决策，前 k 查询，权重向量

Chapter 1 INTRODUCTION

Take smartphone market as an example, there are different models of smartphones (D =$\{r_1, r_2, ..., r_m\}$). Each model ($r_i = (r_i[1], r_i[2], ..., r_i[d])$) has different prices, pixels, battery capacities, cooling capacities and etc. Now a company P owns x models (P =$\{p_1, p_2, ..., p_x\}$$\in D$) and a user data set W =$\{w_1, w_2, ..., w_n\}$. Different users prefer in different aspects, for example some of them may prefer smartphones that with large battery capacity but some prefer those with better quality of screen, so each $w_i = (w_i[1], w_i[2], ..., w_i[d])$ represents the preference weight vector corresponding to a single user. The score of a model r_i respecting to a user w_i is the dot product $r_i \cdot w_i$. Usually a user will only make a choice from his own view of top-k, so a product ranks top-k for the users is quite important. A product covers a user only when its score ranks top-k among all existing products D. With the development of company and market, company has to develop a new product to cover more users and make more profit. But with the limitation of technology, money and other factors, one can't develop a perfect product to cover all users. It can only develop a product that covers as more user as possible under a constraint. For a company, some of its products have covered some users, so what it wants to do is how to make this new product cover more users that uncovered before.

In real life, k-Cover Ratio Maximization ($kCRM$) can solve the problems that how to decide the next generation product for companies. In advertising industry, it can help merchants how to cover specific group such as students, pregnant women and children. Besides it can tell advertiser where to set up new advertising board and if do so it can cover which group of people. Data analysts can use it to discover which group of people is ignored by the market. It can tell vedio makers to make which kinds of vedios to attract users in YouTube, TikTop or other vedio platform.

Generally speaking, it is not a product covers a user but a group of products covers a user. And for different users, there are different groups of products. Among these groups of products, there are uncertain number of identity products. In our problem, we are aim to find this new product in continuous product space, which means there are infinite candidate products, so it is difficult to tell which product covers the most users.

In this paper, we will use computational geometric nature of $kCRM$ to explain how to find the exact optimal options (products) with the data structure $CellTree$ mentioned in $kSPR$[1]. Besides, from some observations of this problem, we propose advance method that ignore irrelevant users and candidate products to save time. At last we sample products that satisfy the constraint and use their maximal cover count to prune the $CellTree$.

Chapter 2 RELATED WORK

Based on the summary of $kSPR$[1], preference-based querying which based on the value of each attribute of products are mainly two kinds, skyline [2–5] and top-k query [6–10]. Skyline is also named as non-dominated set [11], which including all the data that each of them isn't dominated by any other data in the dataset. "X dominates Y" means X is better them or equivalent to Y in all dimensions and there is at least one dimension that X is better than Y. In our paper, we are more closed to top-k query. Top-k query will return k products such that their scores are ranking top-k respecting to the input user. Our problem is to find the region that where the new product lies will it rank top-k for most of the rest given users. Another related problem is reverse top-k [12–14], which is to return the users that input product can rank top-k respecting to them. Based on the output of top-k query, why-not top-k query [15] is proposed to change the user weight vector by advertisements, correcting wrong user information or other ways with the minimum penalty to make an input product ranks top-k. At the same time, based on the output of reverse top-k query, why-not reverse top-k query [16] is proposed to how to change the k in top-k, the user weight vector w or the product p's attribute values so as to make w shown in the reverse top-k result of p.

One of the related studies for our problem is $k - hit$ query[17], which attempts to find k products from given product dataset so as to rank top-1 of users as more as possible. The candidate solution of $k - hitquery$ is discrete and finite while $kCRM$ is to find a new product ranks top-k for as more users as possible. At the same time, CRM may return unknown number optimal products or even infinite products from continuous candidate space if only they are all optimal at the same time.

A recent study $TopRR$[18] is very closed to our problem, which is attempting to return the product region that each of whose products can rank top-k for all input users. The major different between $TopRR$ and $kCRM$ is that the candidate space is not complete in the domain of product for $kCRM$ since $kCRM$ can only choose the product that satisfies the constraint, which means in most cases, the optimal products can't cover all users and it is unknown that the optimal products will cover how many or which users.

In our paper, we use the CTA approach as mentioned in $kSPR$ as our baseline. $kSPR$ is to return the user region that an input product can rank top-k and it uses a data structure $CellTree$ to exactly identify in which user space the product's score is better than product dataset's another product's score, so $CellTree$ can take the regions that the input product not worse than other k products as results and return them. For $kCRM$, we transform our problem into return the product region that covers input users as more as possible. We find that to cover one or multiple users product also lies in regions and $CellTree$ can store and process them either. $CellTree$ will record each region cover how many users and return the region covers the most users as $kCRM$'s answer.

Chapter 3 PROBLEM DEFINITION

In this section, we will firstly introduce related concept and then propose our problem.

Definition 1. A product p's score respecting to a user w is the dot product $p \cdot w$.

Without loss of generality, w satisfies $w[i] \in [0, 1]$, p satisfies $p[i] \in [0, 1]$ and $\Sigma_{i=1}^{d} w[i] = 1$. We also take the values as larger as better for products and so the larger the score the better.

Definition 2. A product p covers a user w when the score respecting to w ranks top-k among D.

Problem 1. The $k - CoverRatioMaximization$ takes product dataset D, $P \subset D$, user data set W and a positive integer k as inputs. It introduces how to determine a new product p such that satisfies the constraint $C(p) \leq B$ and maximizes the cover ratio of $P \cap \{p\}$:

$$cp(p, P, k) = \frac{|\{w|\forall w \in W, \{P \cup \{p\}\} \cap TopK(w) \neq \emptyset\}|}{|W|}$$

For the sake of convenient, we define constraint $C(p) \leq B$ as $\Sigma_{i=1}^{d} p[i] \leq B$.

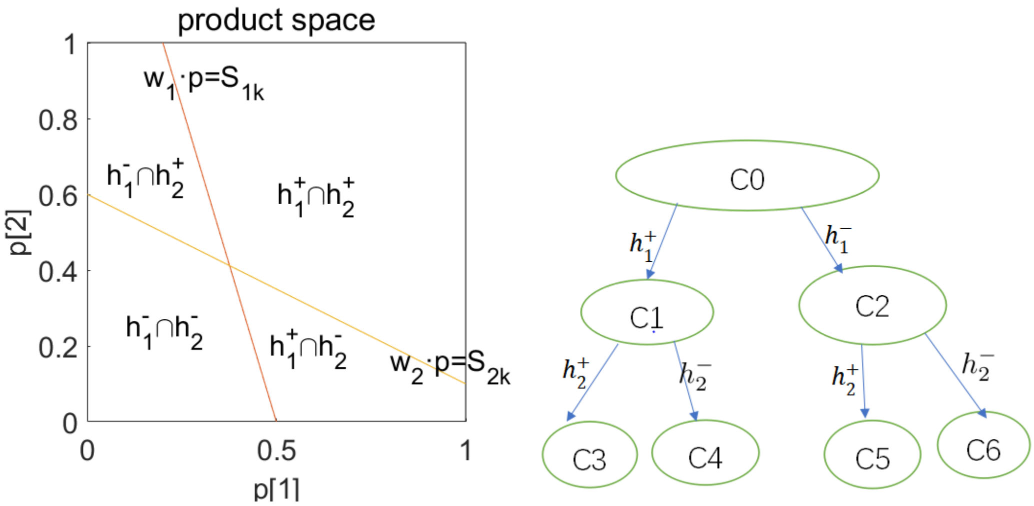
Chapter 4 GEOMETRIC COMPUTATION NATURE

4.1 To Cover Users

Definition 3. The k_{th} score S_{ik} represents the scores to ranks top-k corresponding to user w_i.

For simplicity, we mark

1. $w_i \cdot p = S_{ik}$ as h_i
2. $w_i \cdot p > S_{ik}$ as h_i^+
3. $w_i \cdot p < S_{ik}$ as h_i^-



(a) Hyper-plane insertion **(b)** $CellTree$ insertion

Figure 4.1: hyper-plane insertion and $CellTree$ insertion

As shown in Figure 4.1a, when it comes to multiple users, such as 2 users $\{w_1, w_2\}$, firstly h_1 divides product space into 2 half-space h_1^+ and h_1^-; then h_2 divides h_1^+ into $h_1^+ \cap h_2^+$ and $h_1^+ \cap h_2^-$ and divides h_1^- into $h_1^- \cap h_2^+$ and $h_1^- \cap h_2^-$. Region $h_1^+ \cap h_2^+$ covers both w_1 and w_2; region $h_1^+ \cap h_2^-$ could only cover w_1; region $h_1^- \cap h_2^+$ could only cover w_2; region $h_1^+ \cap h_2^-$ can't cover any user.

4.2 Cell Tree Representation

The tree in Figure 4.1b is called $CellTree$, which is firstly proposed by $kSPR$. We use the root node to represent the whole candidate space. After the insertion of h_1, the root node (cell) c_0 generates 2 child cells c_1 and c_2 while the space is divided into 2 parts; c_1 and c_2 represent h_1^- and h_1^- respectively. After the insertion of h_2, the cell c_1 generates 2 child cells c_3 and c_4; the cell c_2 generates 2 child cells c_5 and c_6. From root cell c_0 to cell c_3, we can clearly see that c_3 is $h_1^+ \cap h_2^+$, which means c_3 covers w_1 and w_2. Similarly, c_4 is $h_1^+ \cap h_2^-$, which means c_3 covers w_1. Among all the cells, c_3 covers the largest number of users. If we change the root cell as $C(p) \leq B$ and remove all those users that already covered by P then we can use $CellTree$ to solve $kCRM$.

4.3 Baseline Solution

In the below paragraph, we will introduce our baseline approach to get the optimal solution for $kCRM$, which follows these steps:

1. Calculate the top-k score S_{ik} for each $w_i \in W$.
2. Find all the $w_i \in W$ that P covers and mark their set as W^*.
3. Update $W = W - W^*$.
4. Using $CellTree$ to find the cell that with maximal cover count and return the optimal cells.

4.4 Time Comlexity

As proposed in $kSPR$, the $CellTree$ approach's time complexity is $O(n^d)$, which n is the product dataset cardinality and d is the dimensionalty of data. For our problem, baseline solution time complexity is $O(n^d + nm \log m)$ or $O(n^d + nmk)$. n means the cardinality of users that take part in $CellTree$ halfspace insertion. d means the dimentionality of data. m means the cardinality of product dataset. $O(nm \log m)$ is corresponding to the process of finding S_{ik} for each user, which needs calculating the dot product of users, sorting the scores and return the k_{th} score. We could also use seletion sorting instead of sorting methods with time complexity $O(m \log m)$ when product dataset is huge while k is small to make getting S_{ik} of all users with time complexity $O(nmk)$. In most cases $n^d \gg nm \log m$, so we can take baseline's time complexity as $O(n^d)$.

Chapter 5 ADVANCE SOLUTION

5.1 Lemmas That Help to Prune

Baseline solution basically is just a brute force method. Now we introduce some lemmas that prunes and accelerates the baseline solution.

Definition 4. A product p dominates another product q if and only if $\forall i \in [1, d]$, the i_{th} dimension $p[i] \geq q[i]$ and $\exists\, i \in [1, d]$, the i_{th} dimension $p[i] > q[i]$.

Lemma 1. If $\forall q$ that $C(q) < B, \exists p$ that $C(p) = B$ and p dominates q, then there must be at least one optimal solution on $C(p) = B$.

Because for most of the constraint $C(p) < B$ is bounded by $C(p) = B$ and as for our defined constraint $\Sigma_{i=1}^{d} p[i] \leq B$ satisfies condition of Lemma 1, in experiments we only need to consider region $C(p) = B$ as our candidate space which is also the root node of $CellTree$.

On the base of Lemma 1 that only consider $C(p) = B$, which also means only h_i will divide space $C(p) = B$ would affect where the optimal solutions are, we define Lemma 2.

Lemma 2. Ignore the users w such that $w \cdot p = S_k$ doesn't intersect with constraint $C(p) = B$ won't affect the $kCRM$ result.

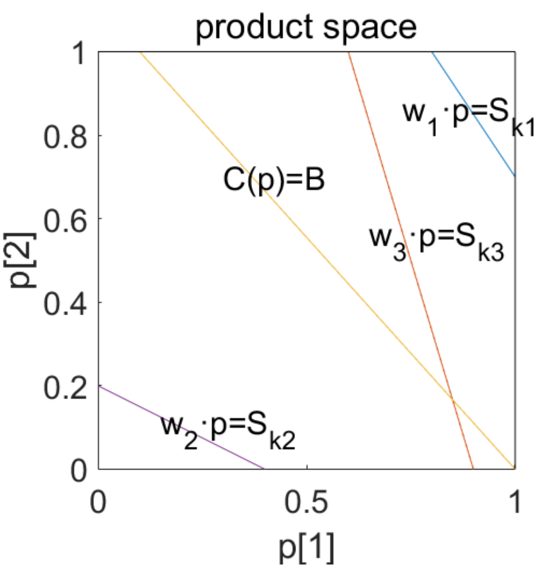


Figure 5.1: User Intersect With constraint

As shown in Figure 5.1, if we decide to choose the new product from $C(p) = B$, we can see that h_1 also divides the product space into 2 halfspaces, but all the products on $C(p) = B$ are in the "-" halfspace, which means all of them can not cover w_1. Different from w_1, the products that on $C(p) = B$ all can cover w_2. From this observation, we can move out all the users w such that $w_i \cdot p = S_{ik}$ doesn't intersect with $C(p) = B$.

Definition 5. The negative space count for a $CellTree$ node is the negative spaces from this node to root node traversing by ancestor node one by one.

Take Figure 4.1b as an example, the negative space count

1. For cell c_4 is 1 because of h_2^-,

2. For cell c_5 is 1 because of h_1^-,

3. For cell c_6 is 2 because of h_1^- and h_2^-.

Lemma 3. If the cover count of optimal solution in $kCRM$ is at least β and $card(W) = n$, then all the nodes with more than $n - \beta$ can't become the optimal solution and they can be pruned.

Lemma 3 means that if we can judge there is no solution in a space can become optimal solution because them can't cover at least β users, then we can prune this space. Take Figure 4.1b as example, if $\beta = 2$ which means the optimal solution should at least cover 2 users and $W = \{w_1, w_2, w_3\}$, then the nodes c_3, c_4, c_5 can be pruned because they can never cover at least 2 users even after the insertion of h_3. Actually, β is the lower bound of optimal solution.

Definition 6. Pruning number α is defined as $\alpha = n - \beta$ which based on Lemma 3.

Pruning number α means that if a node's negative space count exceeds α, than it is safe to prune this node.

5.2 Get Pruning Number

Based on Lemma 3, to find a proper β, we simply uniformly generate new products in candidate space and then find the maximal cover count of them. The procedure is:

1. Uniformly generate new products $P' = \{p'_1, ..., p'_y\}$ on $C(p') = B$.

2. Calculate cover count of each of P'.

3. Find the maximal cover count of P' as β

4. Pruning number $\alpha = card(W) - \beta$

5.3 Insertion Order of Users

Definition 7. Maximal likely cover count of a user means the maximal cover count of the sampled products that cover this user.

As mentioned in Lemma 3, we prune the cell nodes that with more that α negative halfspaces. To be earlier prune the nodes that itself and its sub-tree leaf nodes can't be optimal solution, we can firstly insert the halfspace that its positive halfspace not likely be the component of optimal solutions. Our method to determinate the insertion order of halfspace is by the maximal likely cover count of users (we write CoverCount in short as CC):

1. Initialize the cover count of each user as 0.
2. Uniformly generate new products $P' = \{p'_1, ..., p'_y\}$ on $C(p') = B$.
3. for p' in P'
   (a) W' is the user set covered by p
   (b) $p'.CC = card(W')$
   (c) for w' in W'
      i. $w'.CC = max(w'.CC, p.CC)$
4. update $W = AscendingSortByCC(W)$
5. return W

Lemma 4. Insert users by the order of their maximal likely cover counts in ascending.

The assumption that proposes Lemma 4 is that we don't want those users positive halfspaces h_i^i hide with the user positive halfspaces can be part of optimal because that would make us use α to prune nodes quit late for it has generate many nodes can't be part of optimal solutions.

5.4 Summary of Lemmas

1. Lemma 1 prunes the candidate space from $C(p) \leq B$ to $C(p) = B$.
2. Lemma 2 moves out the users w that $w \cdot p = S_k$ doesn't intersect with $C(p) = B$
3. Lemma 3 state that in the process of $CellTree$ we can prune the nodes whose negative space count is more than pruning number α.
4. Lemma 4 introduces a heuristic trick that forces pruning some nodes using Lemma 3.

5.5 Advance Solution

The main procedure for advance solution in short is:

1. Remove users covered by P.
2. Remove users by Lemma 2.
3. Apply Lemma 4 change the insertion order of users' halfspace.
4. Apply Lemma 1 on root node of $CellTree$.
5. Apply Lemma 3 prune nodes when doing $CellTree$ insertion.

For more details:

1. Calculate the top-k score S_{ik} for each $w_i \in W$.
2. Find all the $w_i \in W$ that P covers, mark their set as W^*.
3. Update $W = W - W^*$.
4. Find all the $w_i \in W$ that $w_i \cdot p = S_{ik}$ doesn't intersect with $C(p) = B$, mark their set as W^{**}.
5. Update $W = W - W^{**}$.
6. Generate new products on candidate space and find their maximal cover count as β
7. Let pruning number $\alpha = card(W) - \beta$
8. Update candidate space from $C(p) \leq B$ to the part of $C(p) = B$.
9. Chnage order of users W as defined in Lemma 4
10. For $w_i \in W$, try to insert h_i for existing $CellTree$ using depth first search.
    (a) If the current node is marked as pruned, return.
    (b) Else if the node is in h_i^-, increase its negative space count by 1.
        i. If the node's negative space count exceeds α mark it as pruned.
        ii. Increase its sub-tree nodes' negative space count by 1.
    (c) Else if the node is in h_i^+, increase its child nodes' and its cover count by 1.
    (d) Else if there is no child nodes of current node, generate two child nodes as h_i^- and h_i^+, return.

(e) Else if two child nodes marked as pruned, the current node is also marked as pruned

(f) Else traverse to its child nodes.

11. Return the node with maximal cover count.

5.6 Time Complexity

As memtioned in Section 4.4, the baseline's time complexity is $O(n^d)$. For advance solution, we reduce the candidate space from $C(p) \leq B$ to $C(p) = B$, which actually reduces our time complexity to $O(n^{d-1})$ because the candidate space reduces by 1 dimension and the insertion halfspace reduces by 1 dimension together. Besides, we remove the users that doesn't intersect with $C(p) = B$. Actually, some of these users may related to the optimal solution, for example their halfspace may enclose the optimal solution region but since we give up all the candidate space of $C(p) < B$ they then just look unrelated to the optimal solution. In fact, if we want an optimal solution with the lower cost we need those users that h_i^- are totally cover by $C(p) \leq B$. Back to the thesis of this section, the n in $O(n^{d-1})$ reduces users quit a lot. We use n_{new} to represented the cardinality of user data after removing the user by Lemma 3, then our time complexity is $O(n_{new}^{d-1})$.

Chapter 6 EXPERIMENT RESULTS

6.1 Experiment Setting

Table 6.1: Product data set

Dataset	d	n	Attributes	Source
HOTEL	4	186,637	hotels-base.com	No. of stars, No. of rooms, No. of facilities, Price
HOUSE	6	315,063	ipums.org	Gas, Electricity, Water, Heating, Insurance, Property tax

Table 6.2: Experiment parameters and default setting

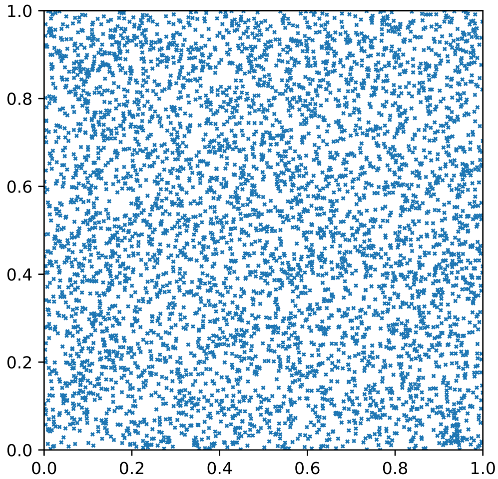
Lemma 3 product samples	10k, 100K, 1M, **10M**
$card(P)$	0, 5000, **10000**, 15000, 20000
Product dataset	**HOTEL**, HOUSE
User data size	1000, **5000**, 10000
User data distribution	**Uniform**, Correlated, Anti-correlated
k	5, **10**, 20, 30
B for HOTEL	1, **1.25**, 1.5, 2 , 2.5, 3
B for HOUSE	3, 3.5, 4 ,4.5, 5, 5.5 , 5.7 , 5.9

There are 3 kinds of datasets, the product data set D, $m = card(D)$; the product data set P, which uniformly generated by D and so $P \subset D$; the user data set W, all the attributes of data $w \in W$ satisfy $\Sigma w[i] = 1$. The dimensionality of data is marked as d. Table 6.1 shows all the product datasets. Table 6.2 shows the parameters' setting and our default setting of them. Figure 6.1 shows how different distribution data looks like. For example, if we want to sample products according to correlated distribution with 2 attributes, then the final generated data will looks like Figure 6.1b.

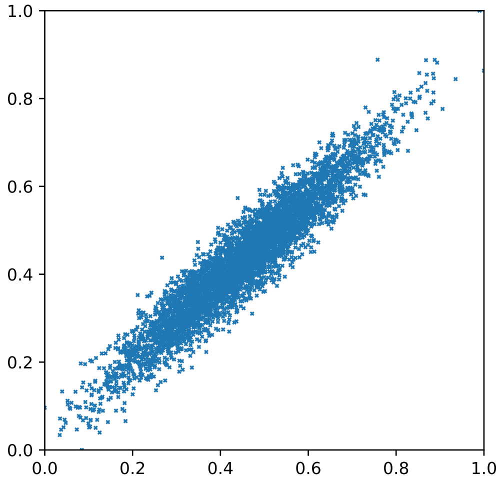
All codes are implemented by C++; the procedure of insertion of h_i needs LP solver and we use $lp\_solve(http://lpsolve.sourceforge.net/5.5/)$ to do it and the effciency of lp_solve has been proved by $kSPR$. The running machine is with Intel Xeon Gold 5122 - 3.60 GHz CPU, 128GB DDR4 RAM.

6.2 Effectiveness of Lemmas and Tricks

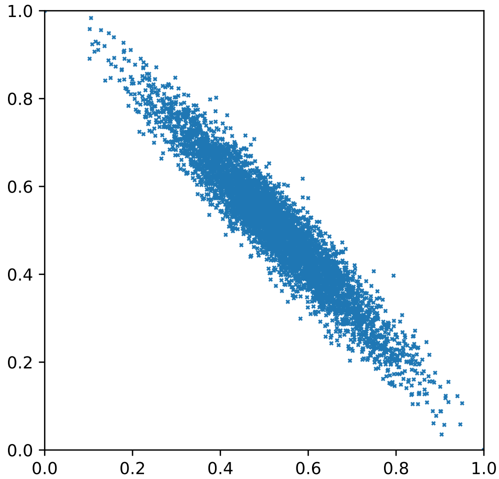
In this section, we mainly discuss that how much does each lemma effetively help us solve $kCRM$. And because there are some parameters within our advance algorithm, we



(a) Uniform Distribution Data



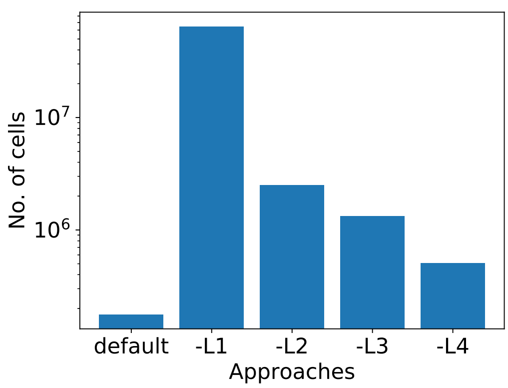
(b) Correlated Distribution Data



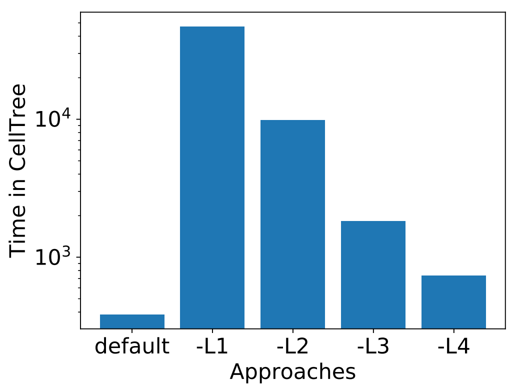
(c) Anti-correlated Distribution Data

Figure 6.1: Different Distribution Data in 2d

would also explore how and why they could influence the efficiency of our algorithm the way shown in our experiments.



(a) No. of cells



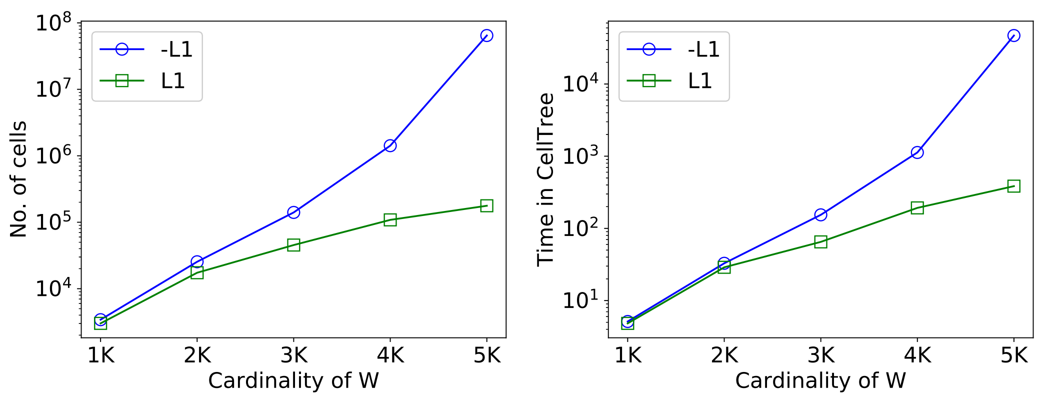
(b) Time spent in $CellTree$

Figure 6.2: Effectiveness of Lemmas

Firstly, we would show a global view of each lemmas as in Figure 6.2. -L1 means for advance solution, we apply $C(p) \leq B$ instead of $C(p) = B$ and the other optimization will be remained. For -L2, it will take account with those users w_i such that $w_i \cdot p = S_{ik}$ doesn't intersect with $C(p) = B$. For -L3, it won't generate any lower bound of optimal solution or anything related to pruning number α and it will just do insertion in $CellTree$ no matter how many negative spaces the nodes in. For -L4, it will randomly insert users into $CellTree$. The default running means, we will

- only consider $C(p) = B$
- remove unrelated users respecting to $C(p) = B$
- sample new product on $C(p) = B$, get lower bound of optimal solution and then use α to prune nodes in $CellTree$.
- based on the less likely to be one of the covered users of optimal solution to insert users so as to earlier prune them.

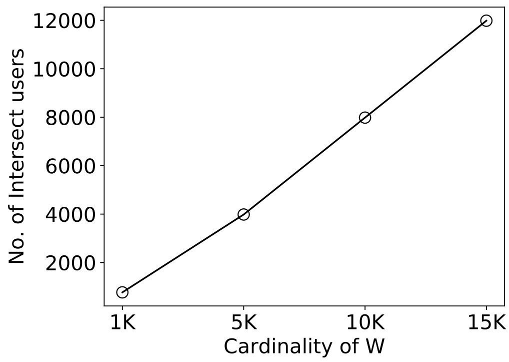
As shown in Figure 6.2a, because Lemma 1 tells that we only take account $C(p) = B$, which means we reduce most of the candidate space and reduce problem from d dimension to $d-1$ dimension, it will improve our approach by hundreds of times of cells and running time. For the other lemmas, it also improves our algorithm by dozens of times.



(a) No. of cells **(b)** Time spent in $CellTree$

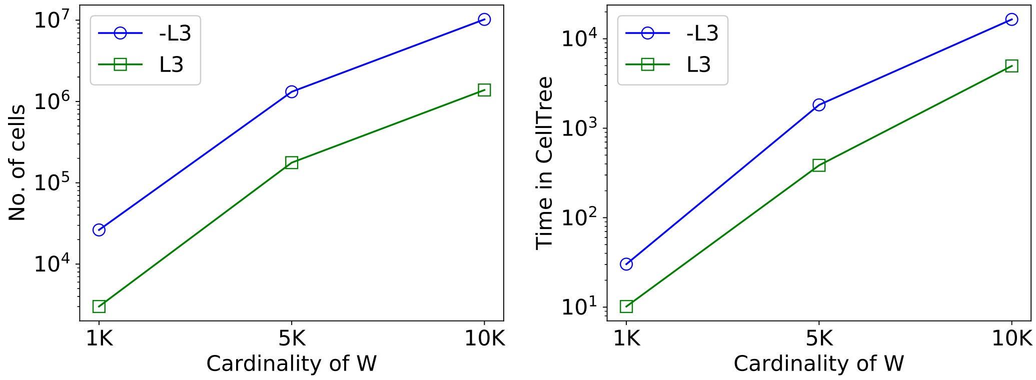
Figure 6.3: Effectiveness of Lemma 1

For each of lemmas, we make some experiments for details of how efficient they are. In Figure 6.3, $-L1$ means when the situation without applying Lemma 1 but keeping the other lemmas work. With the change of cardinality of user dataset W, $CellTree$'s cell number and running time grow fast for $-L1$ while grow more slower for $L1$. In Figure 6.4, we skip the previous step that remove the users that covered by P and clearly see that the number users, whose $w_i \cdot p = B$ would intersect constraint $C(p) = B$, grows linearly



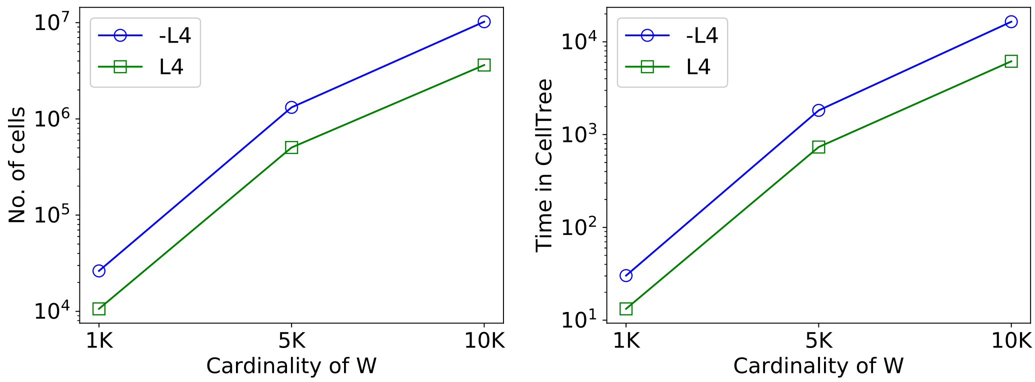
(a) No. of cells

Figure 6.4: Effectiveness of Lemma 2



(a) No. of cells **(b)** Time spent in $CellTree$

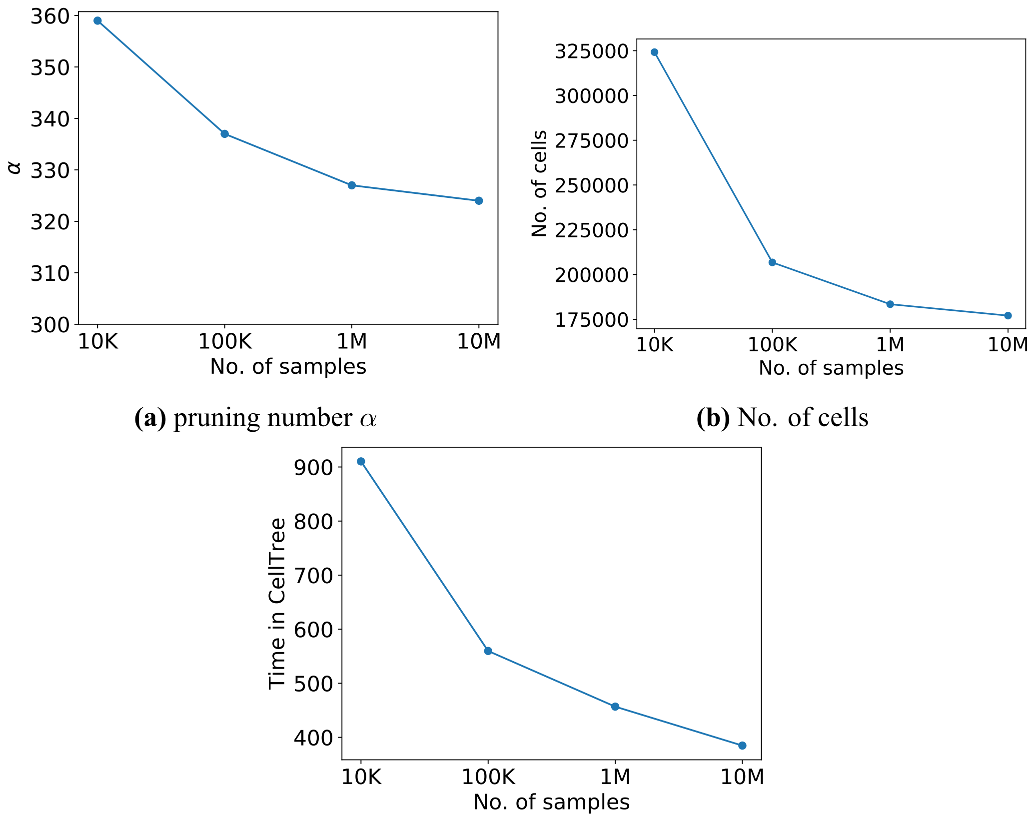
Figure 6.5: Effectiveness of Lemma 3



(a) No. of cells **(b)** Time spent in $CellTree$

Figure 6.6: Effectiveness of Lemma 4

with cardinality of W. In Figure 6.5, we applying other lemmas but without Lemma 3, which using lower bound of optimal solution to prune tree cells of $CellTree$, comparing applying Lemma 3 shows that Lemma 3 help us save space and time efficiently. Figure 6.6 also shows the effectiveness of Lemma 4 which changes the insertion order of users to $CellTree$ in order to earlier prune the nodes that unlikely to be the ancestor cell of optimal



(a) pruning number α **(b)** No. of cells

(c) Time spent in $CellTree$

Figure 6.7: Effect of No. of newly sampled products

solution nodes.

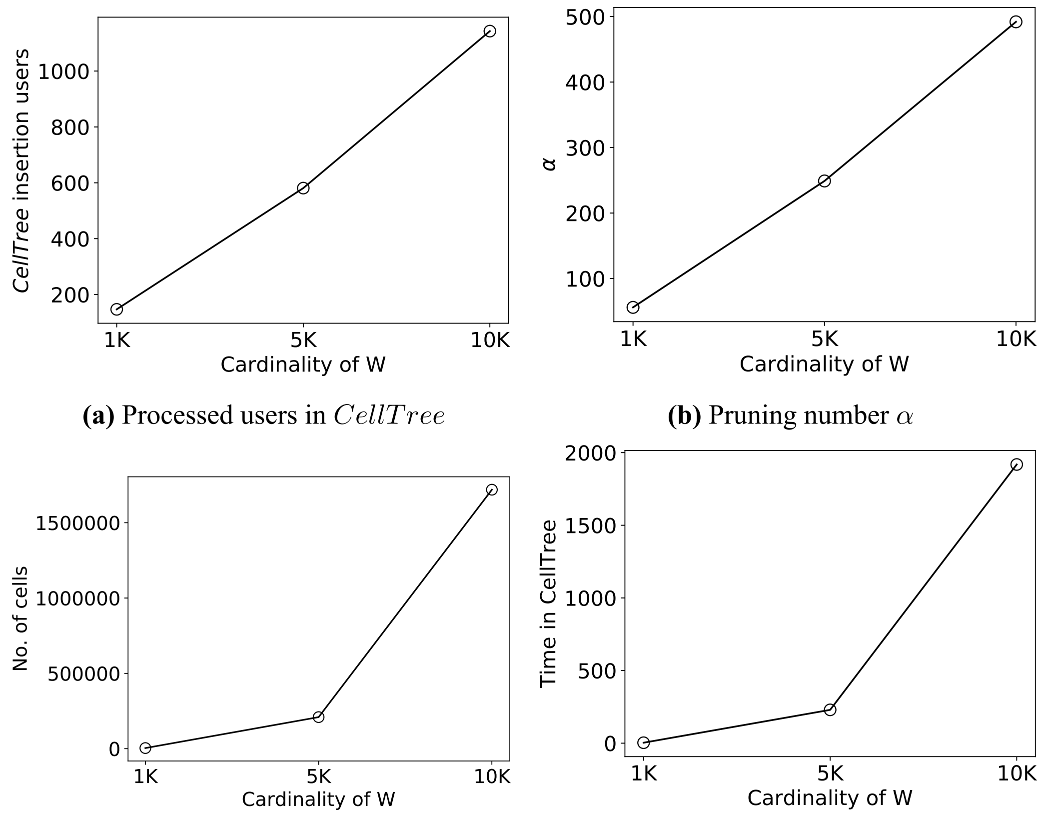
In Lemma 3 we say that we would sample new products on $C(p) = B$ to get pruning number and next we will show the cardinality of newly generated new products influences our algorithm. The sampling time is negligible for time in $CellTree$ insertion and we won't discuss the time spent by sampling in this section. As shown in Figure 6.7a, α slowly decreases with the increment of samples. We can also see from Figure 6.7b and Figure 6.7c though the change of α is small, but it makes a huge impact on resulting cells number in $CellTree$. This can explained by our algorithm's time complexity, $O(n^d)$, which means a litter change of users improves response time huge.

6.3 Influence of Inputs

In this section, we would show the compact of input parameters to our algorithm.

Since the time complexity is $O(n^d)$, we can see from Figure 6.8 that the users remain to be inserted in $CellTree$ grow linearly while the resulting cells and response time growing exponentially.

Product dataset P will influence the efficiency of solving problems because in some extreme condition, or unluckily P only covers a few users; but sometime, P will covers



(a) Processed users in $CellTree$ **(b)** Pruning number α

(c) No. of cells **(d)** Time spent in $CellTree$

Figure 6.8: Effect of cardinality of user dataset W

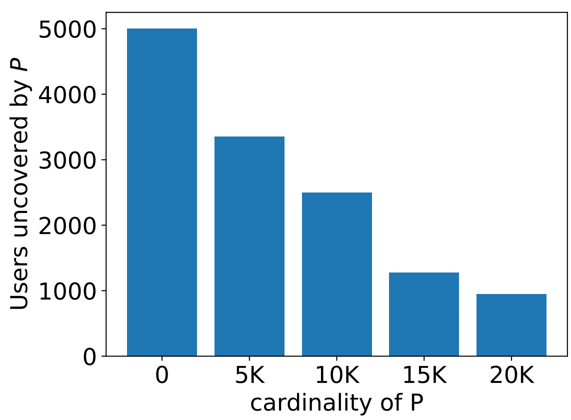
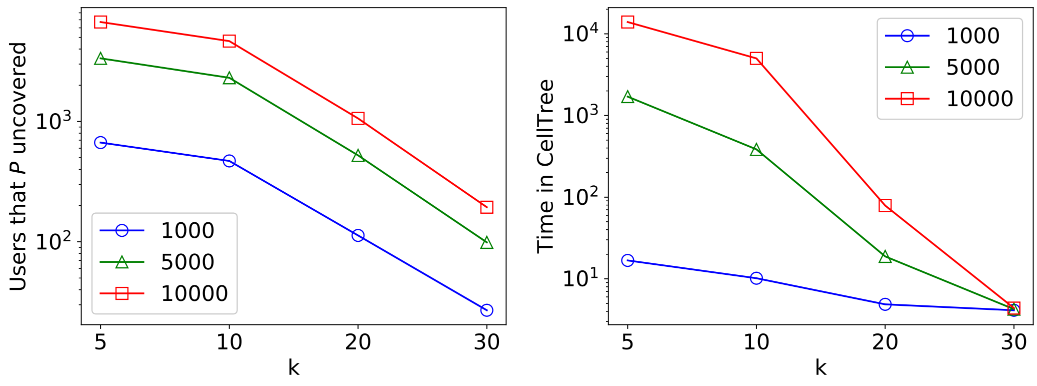


Figure 6.9: Effect of $card(P)$ on user uncovered by P

most of users. To complete the experiment of explore the global view impact of cardinality of P, we run 20 times sampling different P for each attribute as shown in Figure 6.9. The original user data sizes are all 5000 and are the same. The y axis means how many users is left that uncovered by P. We can see from Figure 6.9 such that with the increment of cardinality of P, uncovered users number decreases slower and slower.

Now we explore the impact of k. Our problem is to find a new product that rank as more as possible users' top k as possible, and we have a prerequisite such that product



(a) User uncovered by P **(b)** Time spent in $CellTree$

Figure 6.10: Effect of k (HOTEL)

dataset P already covers a part of the users. With the increasing k, unchanged product dataset and user dataset, the products in dataset P are to cover more users easily because the the threshold to be a user's top-k is lowering down. With P covering more users, no matter which of the lemmas we propose or the baseline $CellTree$ approach will all greatly benefit from the reducing users. We show the experiment in Figure 6.10. The y axis of Figure 6.10a means the users number that we need to handle after removing the users that covered by P. We can see that user dataset with size 1000, 5000 or 10000, their uncovered users decrease exponentially with the change of k. And because of the decreasing uncovered users, our response time also decreases exponentially.

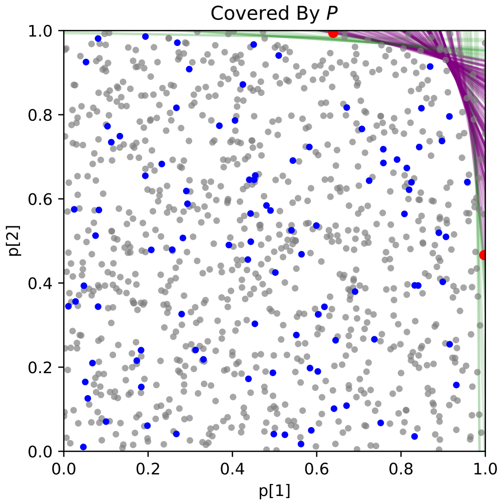
6.4 Effect of Product Dataset Distribution and B

To straight forward presents our algorithm and shows the interesting finding in experiments, we use a 2d experiment and visualize it as in Figure 6.11, 6.12 and 6.13.

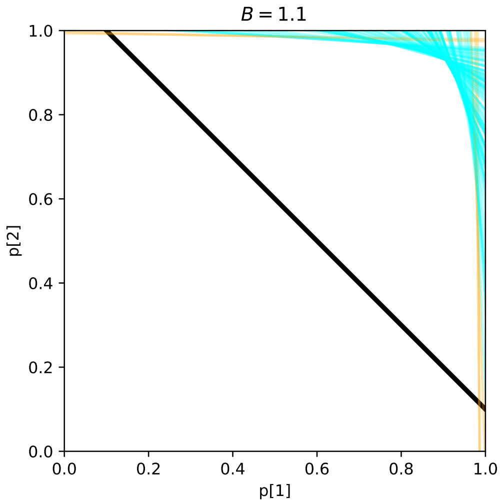
First of all, the users are all generated uniformly from $w[1] + w[2] = 1$.

To explain Figure 6.11, 6.12 and 6.13, we take 6.11a as an example, this figure is drawn in product space. Product dataset D consists with the grey, blue and red points ; product dataset P consists with the blue and red points; the red points are the points that each of them at least covers one user. Specially, we have to expand the size of red points to clearly show them. We can see the red points of Figure 6.11a at about $(0.9, 0.9)$ and $(0.95, 0.8)$. The lines are all $w_i \dot{p} = S_{ik}$. The blue lines mean the corresponding $w_i \dot{p} = S_{ik}$ of users that covered by P and the rest users' halfspaces are orange lines. From Figure 6.11b to Figure 6.11f, the bold black line means the constraint $\Sigma_{i=1}^{d} p[i] \leq B$ when B equals 1.1, 1.3, 1.5, 1.7, 1.9 respectively. The orange lines mean the $w_i \dot{p} = S_{ik}$ that intersects with constraint and the blues aren't intersect with constraint.

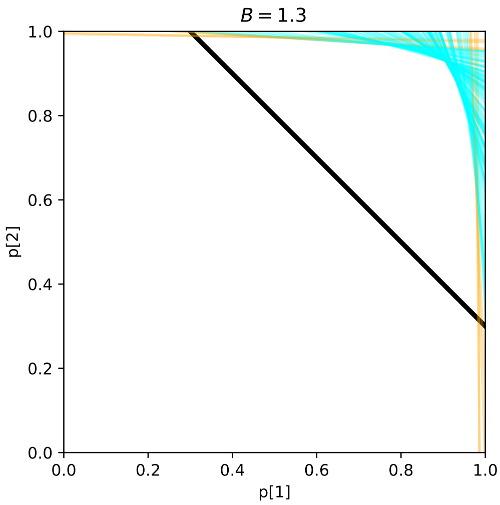
In Figure 6.11a, because the distribution of products is uniform so for each user weight vector w_i, there is a proper product p_{wi} nearby the extend of w_i ranks top-k for it. And



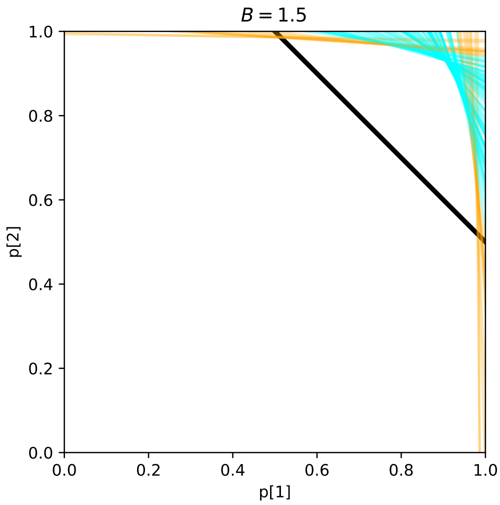
(a) Products and users covered by P



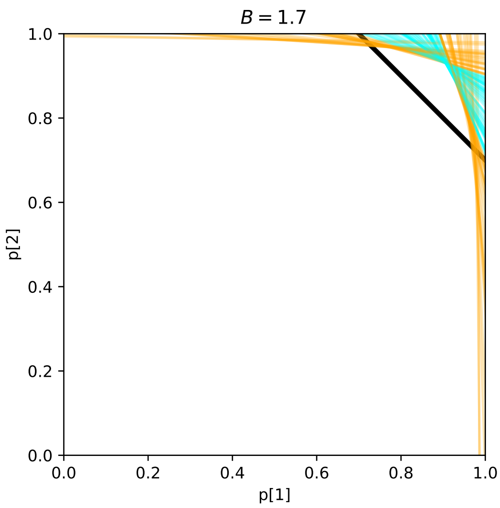
(b) Uniform, B=1.1



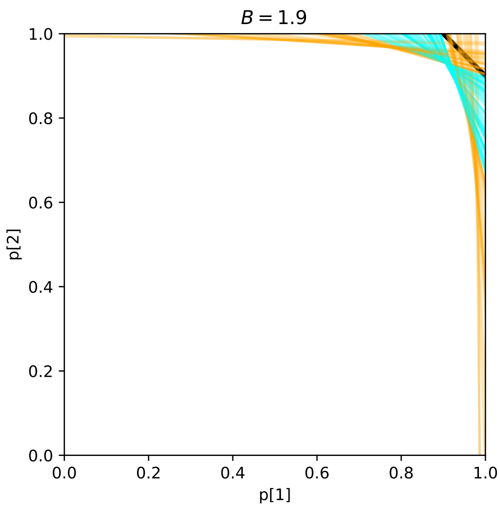
(c) Uniform, B=1.3



(d) Uniform, B=1.5

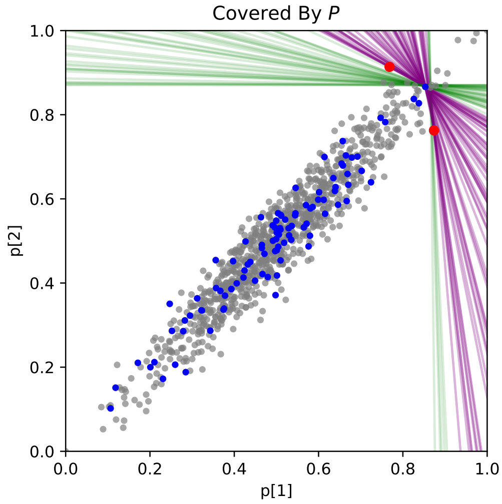


(e) Uniform, B=1.7

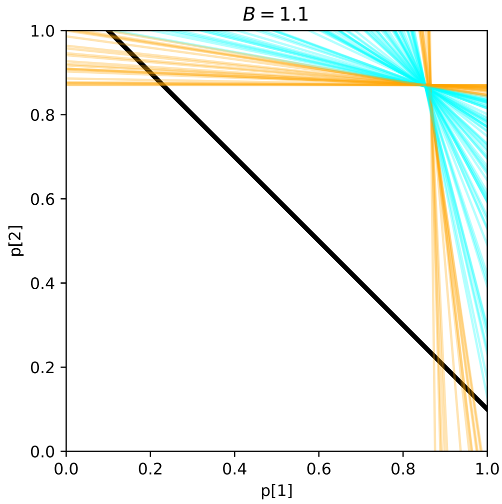


(f) Uniform, B=1.9

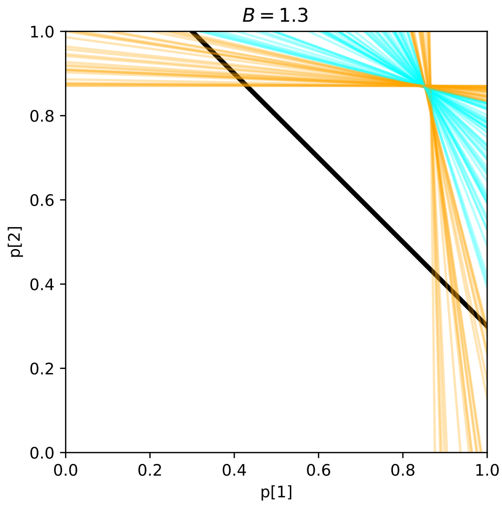
Figure 6.11: Effect of B on uniform products



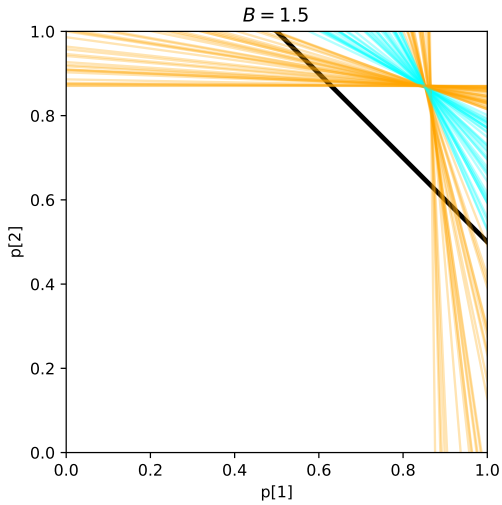
(a) Products and users covered by P



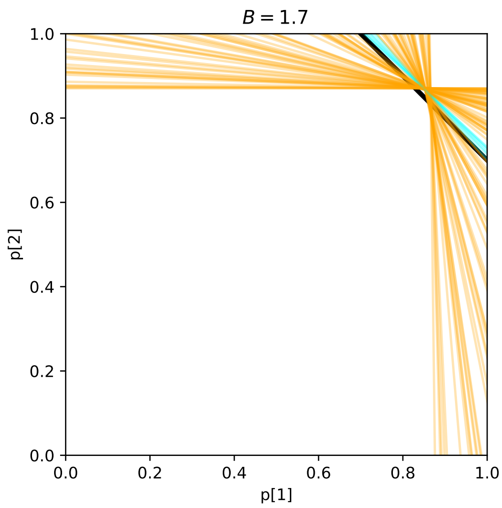
(b) Corr, B=1.1



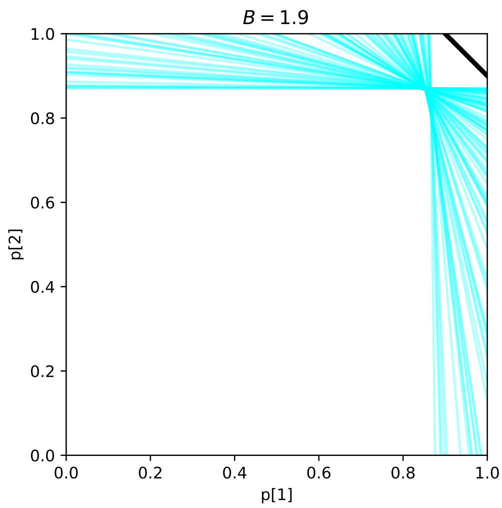
(c) Corr, B=1.3



(d) Corr, B=1.5

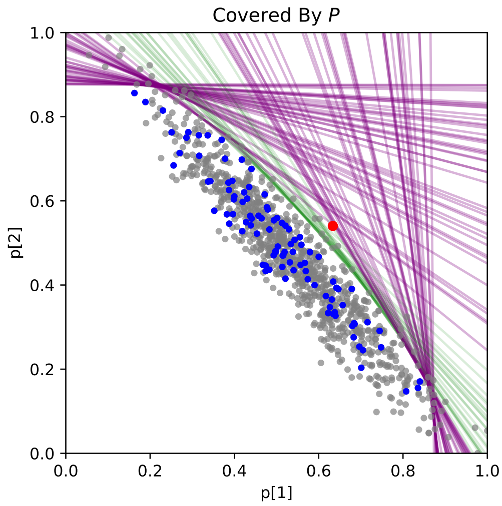


(e) Corr, B=1.7

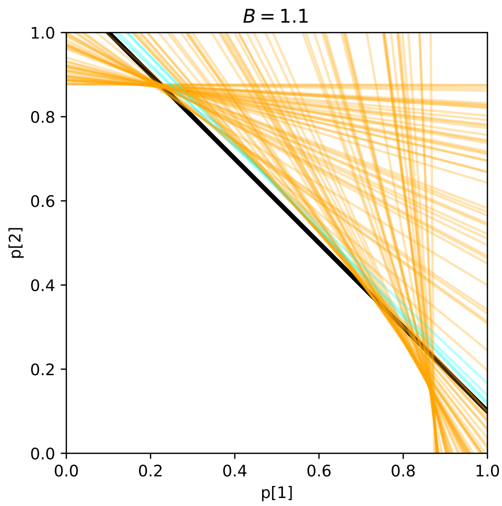


(f) Corr, B=1.9

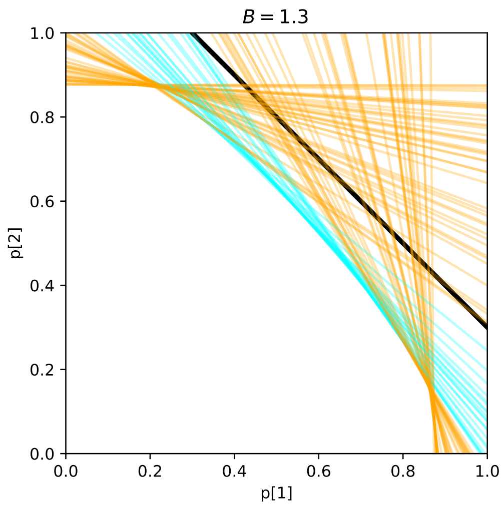
Figure 6.12: Effect of B on corr products



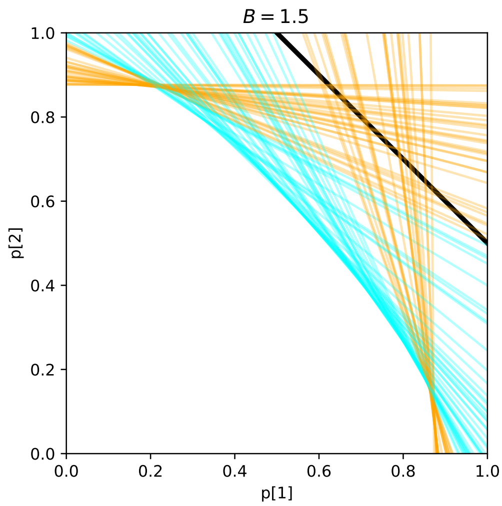
(a) Products and users covered by P



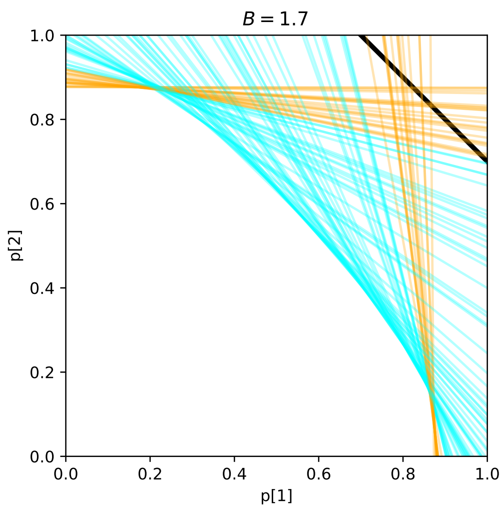
(b) Anti, B=1.1



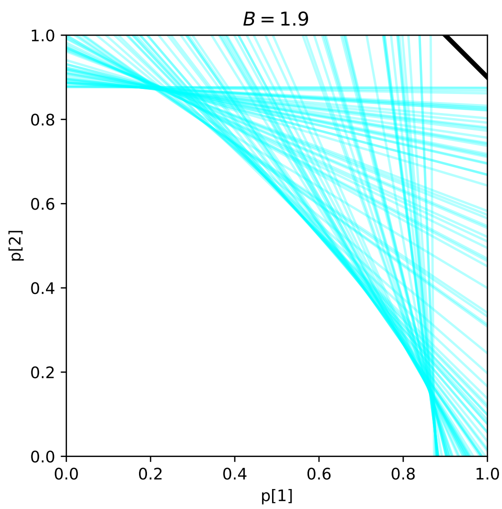
(c) Anti, B=1.3



(d) Anti, B=1.5



(e) Anti, B=1.7



(f) Anti, B=1.9

Figure 6.13: Effect of B on anti products

because the halfspace $w_i \cdot p = S_{ik}$ getting through the point p_{wi}, so the lower bounds of $w_i\dot{p} = S_{ik}$ makes a smooth curve along $p[2] = 1$ and $p[1] = 1$. From Figure 6.11b to Figure 6.11f, the number of the halfspaces that intersect with constraint increases with the increasing B because more and more products become some of the users' top-k and so as will more halfspaces be there.

In Figure 6.12a, the red point is around $(0.9, 0.9)$. In correlated distribution products as shown in 6.12a, almost all the products that rank k for users are at the location that is closed to $(1, 1)$. Because the halfspaces will get through the points that rank exactly k and products ranking k are almost the same or very closed to each other, it seems all halfspaces intersect at one point. Be careful that they aren't intersect at exact one point and it looks so just because of the size of figure we can show. The halfspaces that intersect with constraint will gradually increase and at a special B decrease suddenly as we can see from Figure 6.12b to Figure 6.12f.

In Figure 6.13a, the red points are about at $(0.1, 0.95)$, $(0.3, 0.8)$, $(0.8, 0.3)$. The products that exactly rank k for users concentrate on $(0.2, 0.9)$ and $(0.9, 0.2)$. This could be explained by the Lemma 4 of $k - hitquery$, which says that for any weight vector w if only a point p_ix inside the convex hull made by $P_i = \{p_i1, p_i2, ..., p_{i(x-1)}\}$, then there must be a point in P_i such that its dot product with w is higher than p_ix's. For our case of anti-correlated distribution products, most of products are between the points nearby $(0, 1)$ and $(1, 0)$ which means for any weight vector, its top-k is nearby $(0, 1)$ or $(1, 0)$ when k is small and $card(D)$ is also large. Therefore, in Figure 6.13 we can see most of halfspaces getting through $(0.2, 0.9)$ and $(0.9, 0.2)$ which is closed to $(0, 1)$ or $(1, 0)$. With the increasing of B, the number of halfspaces that intersect with constraint suddenly increases and then gradually decreases.

In Figure 6.11, 6.12 and 6.13, we show the process of our algorithm to solve $kCRM$. Firstly, remove the users that covered by P. Remove the halfspaces $w_i \cdot p = S_{ik}$ that doesn't intersect with constraint. Change the insertion order of halfspaces by some heuristic. Use $CellTree$ proposed in $kSPR$ to find the region that cover the most of the rest of users. During the process in $CellTree$, prune the tree nodes with lower bound of optimal solution.

Figure 6.14 shows the exact changes of halfspaces.

For uniform generated products, intersect halfspaces increases gradually. If the new proposed product wants to cover as more user as possible, its attributes should balance and all with high values. And if the new product is already a top-k option for some users, it is hard for it to cover more because each aspect of it is already high and the cost of develop such product is too high to afford or too hard to realize. But still, we could introduce the product that only satisfies some kind of users. For example, to cover users that care more about $p[1]$, we can introduce the new product as $p = (1, 0.5)$ under the constraint

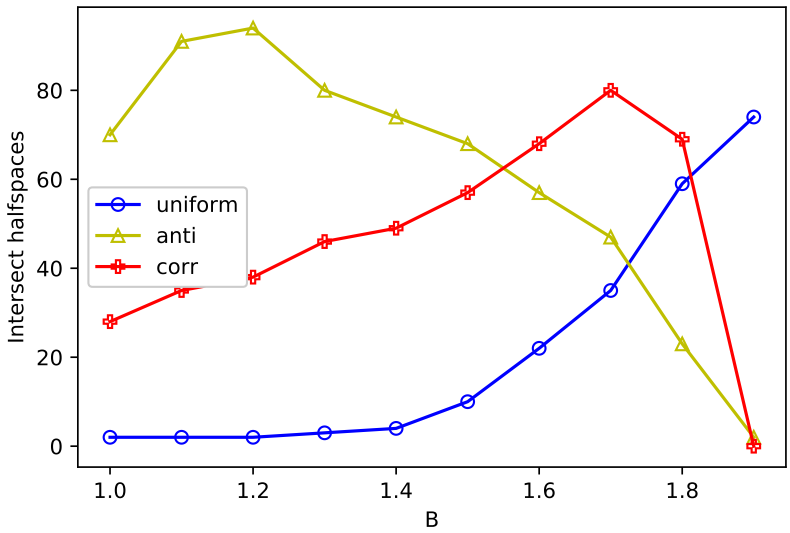


Figure 6.14: No. of halfspaces intersect with constraint

$p[1] + p[2] \leq 1.5$.

For correlated generated products, intersect halfspace increases gradually and then suddenly decrease. For this kind of product distribution, it is also recommended to introduce new products that with high value in some attribute. It is easy for correlated products to introduce a new product that cover all of the users since there are already existing several products cover all of users but for uniformly products it is not likely to find product that cover all users.

For anti-correlated generated products, intersect halfspaces firstly increases for a short time and then gradually decreases. In real world, most of the product datasets are based on this distribution, such as $HOTEL$ and $HOUSE$ data proposed in this paper. For each attribute, there is a certain value that if the product's corresponding attribute exceeds it then this product will cover a kind of users. To cover different kinds of users, the new product has to balance each attribute. Because in real world the users that favor in each attribute are unbalance. For example, there are more users prefer computers with powerful computation ability than with large memory. Consider the case $P = \emptyset$, to cover more users the new product just needs to with high value in the attribute that considered more important by most users. When $P \neq \emptyset$, as we can see from Figure 6.13a, to covered all of the rest user(the orange lines), we need to introduce products such as $\{(0.1, 0.95), (0.95, 0.1)\}$. To covered them by a single product, we need a product for example no worse than $(0.9, 0.7)$. But in realistic, the option decision maker isn't likely to introduce such single product under the limitation of technology, money and other factors. The best strategy is to firstly introduce product $(0.95, 0.1)$ and then is $(0.1, 0.95)$. It is hard to cover all the users and company should take good evaluation of market so as to step by step make more profits.

CONCLUSION

Motivated by the need of introducing new product, we proposal our problem kCRM, which is aim to find the exact optimal solution that covers the most rest users which product dataset P can't cover. We use data structure $CellTree$ introduced in $kSPR$ as our baseline. Then we base on the relationship between constraint and user insertion halfspace to prune unrelated users and get lower bound of optimal solution to prune the nodes in $CellTree$. Besides, we change the insertion order of user halfspaces to more efficiently process insertion reducing the possible useless nodes. In our paper, we use experiments to show our optimization for the baseline did improve it a great deal and we find some inner connection between efficiency of our approach and constraint. For the future work, we can perform further study the effect of insertion order of halfspaces to more efficiently solve $kCRM$ and so that we can deal with larger user datasets.

References

[1] TANG B, MOURATIDIS K, YIU M L. Determining the Impact Regions of Competing Options in Preference Space[J/OL]. Proceedings of the 2017 ACM International Conference on Management of Data - SIGMOD 17, 2017.
http://dx.doi.org/10.1145/3035918.3064044.

[2] BORZSONY S, KOSSMANN D, STOCKER K. The Skyline operator[J/OL]. Proceedings 17th International Conference on Data Engineering, .
http://dx.doi.org/10.1109/icde.2001.914855.

[3] PAPADIAS D, TAO Y, FU G, et al. Progressive skyline computation in database systems[J/OL]. ACM Transactions on Database Systems, 2005, 30(1) : 41-82.
http://dx.doi.org/10.1145/1061318.1061320.

[4] TAN K-L, ENG P-K, OOI B. Efficient Progressive Skyline Computation.[J], 2001 : 301 – 310.

[5] KOSSMANN D, RAMSAK F, ROST S. Shooting Stars in the Sky[J/OL]. VLDB 02: Proceedings of the 28th International Conference on Very Large Databases, 2002 : 275-286.
http://dx.doi.org/10.1016/b978-155860869-6/50032-9.

[6] CHANG Y-C, BERGMAN L, CASTELLI V, et al. The onion technique[J/OL]. ACM SIGMOD Record, 2000, 29(2) : 391-402.
http://dx.doi.org/10.1145/335191.335433.

[7] HRISTIDIS V, PAPAKONSTANTINOU Y. Algorithms and applications for answering ranked queries using ranked views[J/OL]. The VLDB Journal The International Journal on Very Large Data Bases, 2004, 13(1) : 49-70.
http://dx.doi.org/10.1007/s00778-003-0099-8.

[8] HRISTIDIS V, KOUDAS N, PAPAKONSTANTINOU Y. PREFER: A System for the Efficient Execution of Multi-parametric Ranked Queries.[J/OL]. SIGMOD Record (ACM Special Interest Group on Management of Data), 2001, 30 : 259 – 270.
http://dx.doi.org/10.1145/375663.375690.

[9] XIN D, CHEN F, HAN J. Towards Robust Indexing for Ranked Queries.[J]. VLDB 2006 - Proceedings of the 32nd International Conference on Very Large Data Bases, 2006 : 235 – 246.

[10] ZOU L, CHEN L. Dominant Graph: An Efficient Indexing Structure to Answer Top-K Queries[J/OL]. 2008 IEEE 24th International Conference on Data Engineering, 2008.
http://dx.doi.org/10.1109/icde.2008.4497462.

[11] Deb K, Pratap A, Agarwal S, et al. A fast and elitist multiobjective genetic algorithm: NSGA-II[J]. IEEE Transactions on Evolutionary Computation, 2002, 6(2) : 182 – 197.

[12] VLACHOU A, DOULKERIDIS C, NøRVåG K, et al. Branch-and-bound algorithm for reverse top-k queries[J/OL]. Proceedings of the 2013 international conference on Management of data - SIGMOD 13, 2013.
http://dx.doi.org/10.1145/2463676.2465278.

[13] VLACHOU A, DOULKERIDIS C, KOTIDIS Y, et al. Reverse top-k queries[J/OL]. 2010 IEEE 26th International Conference on Data Engineering (ICDE 2010), 2010.
http://dx.doi.org/10.1109/icde.2010.5447890.

[14] VLACHOU A, DOULKERIDIS C, KOTIDIS Y, et al. Monochromatic and Bichromatic Reverse Top-k Queries[J/OL]. IEEE Transactions on Knowledge and Data Engineering, 2011, 23(8) : 1215-1229.
http://dx.doi.org/10.1109/tkde.2011.50.

[15] He Z, Lo E. Answering Why-Not Questions on Top-K Queries[J]. IEEE Transactions on Knowledge and Data Engineering, 2014, 26(6) : 1300 – 1315.

[16] GAO Y, LIU Q, CHEN G, et al. Answering why-not questions on reverse top- k queries[J/OL]. Proceedings of the VLDB Endowment, 2015, 8(7) : 738-749.
http://dx.doi.org/10.14778/2752939.2752943.

[17] PENG P, WONG R C-W. k-Hit Query[J/OL]. Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data - SIGMOD 15, 2015.
http://dx.doi.org/10.1145/2723372.2723735.

[18] TANG B, MOURATIDIS K, YIU M L, et al. Creating top ranking options in the continuous option and preference space[J/OL]. Proceedings of the VLDB Endowment, 2019, 12(10).
http://dx.doi.org/10.14778/3339490.3339500.

[19] ILYAS I F, BESKALES G, SOLIMAN M A. A survey of top- k query processing techniques in relational database systems[J/OL]. ACM Computing Surveys, 2008, 40(4): 1-58. http://dx.doi.org/10.1145/1391729.1391730.
[20] FAGIN R, LOTEM A, NAOR M. Optimal aggregation algorithms for middleware[J/OL]. Journal of Computer and System Sciences, 2003, 66(4): 614-656. http://dx.doi.org/10.1016/s0022-0000(03)00026-6.
[21] MOURATIDIS K, ZHANG J, PANG H. Maximum rank query[J/OL]. Proceedings of the VLDB Endowment, 2015, 8(12): 1554-1565. http://dx.doi.org/10.14778/2824032.2824053.
[22] DAS G, GUNOPULOS D, KOUDAS N, et al. Answering Top-k Queries Using Views.[J]. VLDB 2006 - Proceedings of the 32nd International Conference on Very Large Data Bases, 2006: 451–462.

Acknowledgements

四年的光阴不长不短，在春节辅导员顺路接送回家，跟校学生会文娱部的大家组织过很多活动有过许多快乐的时光，在风景美丽的校园上体育课，生物课老师对我差劲英语的包容，物理课老师对我不耐烦的教导，细心认真的 Stéphane Faroult 带领我走入计算机专业，时常上课给同学人生建议的王琦老师，书院导师杨柳青给与我们生活上的指导，书院班主任史玉回老师给我们分享的自己的人生经历，Hisao 教授带领我进入科研，唐博教授教懂我如何做项目，邵宣杰同学四年中对我数学上的帮助，经常一起谈笑风生的黄旭以及张思宇同学，华为实习期间一直帮助我的工作导师苏林达，组长林吉生以及同事王玮，有幸遇到各位优秀 DBGroup@SUSTech 的成员, 感谢学校给与的丰厚资助，强大的师资与硬件，非常感谢让我一路上经历的这一切美好与困难。最感谢的是我的父母，在我大学期间我体会到学校无微不至的关怀，真诚的关心，认真的付出，这些我都将铭记在心。

李可明

2020 年 5 月 20 日