國立政治大學資訊科學系
Department of Computer Science
National Chengchi University

碩士論文

Master's Thesis

針對複合式競賽挑選最佳球員組合的方法

Selecting the Best Group of Players for a Composite Competition

研 究 生：鄧雅文

指導教授：陳良弼

中華民國九十九年七月

July 2010

針對複合式競賽挑選最佳球員組合的方法

Selecting the Best Group of Players for a Composite Competition

研 究 生：鄧雅文　　Student：Ya-Wen Teng

指導教授：陳良弼　　Advisor：Arbee L. P. Chen

國立政治大學

資訊科學系

碩士論文

A Thesis

submitted to Department of Computer Science

National Chengchi University

in partial fulfillment of the Requirements

for the degree of

Master

in

Computer Science

中華民國九十九年七月

July 2010

國立政治大學

博碩士論文全文上網授權書

National ChengChi University

Letter of Authorization for Theses and Dissertations Full Text Upload

(提供授權人裝訂於紙本論文書名頁之次頁用)

(Bind with paper copy thesis/dissertation following the title page)

本授權書所授權之論文為授權人在國立政治大學資訊科學學系系所 _____組98學年度第二學期取得 碩士學位之論文。
This form attests that the _____ Division of the Department of Graduate Institute of Computer Science at National ChengChi University has received a Master degree thesis/dissertation by the undersigned in the _____ semester of 98 academic year.

論文題目（Title）：針對複合式競賽挑選最佳球員組合的方法 (Selecting the Best Group of Players for a Composite Competition)

指導教授（Supervisor）：陳良弼

立書人同意非專屬、無償授權國立政治大學，將上列論文全文資料以數位化等各種方式重製後收錄於資料庫，透過單機、網際網路、無線網路或其他公開傳輸方式提供用戶進行線上檢索、瀏覽、下載、傳輸及列印。國立政治大學並得以再授權第三人進行上述之行為。
The undersigned grants non-exclusive and gratis authorization to National ChengChi University, to re-produce the above thesis/dissertation full text material via digitalization or any other way, and to store it in the database for users to access online search, browse, download, transmit and print via single-machine, the Internet, wireless Internet or other public methods. National ChengChi University is entitled to reauthorize a third party to perform the above actions.

論文全文上載網路公開之時間（Time of Thesis/Dissertation Full Text Uploading for Internet Access）：
網際網路（The Internet） ■ 中華民國 100 年 8 月 5 日公開

● 立書人擔保本著作為立書人所創作之著作，有權依本授權書內容進行各項授權，且未侵害任何第三人之智慧財產權。
The undersigned guarantees that this work is the original work of the undersigned, and is therefore eligible to grant various authorizations according to this letter of authorization, and does not infringe any intellectual property right of any third party.

● 依據96年9月22日96學年度第1學期第1次教務會議決議，畢業論文既經考試委員評定完成，並已繳交至圖書館，應視為本校之檔案，不得再行抽換。關於授權事項亦採一經授權不得變更之原則辦理。
According to the resolution of the first Academic Affairs Meeting of the first semester on September 22nd, 2007,Once the thesis/dissertation is passed after the officiating examiner's evaluation and sent to the library, it will be considered as the library's record, thereby changing and replacing of the record is disallowed. For the matter of authorization, once the authorization is granted to the library, any further alteration is disallowed，

立 書 人：鄧雅文
簽　　名(Signature)：

中華民國　　年　　月　　日
Date of signature：_____/_____/_____ (dd/mm/yyyy)

針對複合式競賽挑選最佳球員組合的方法

摘要

在資料庫的處理中，*top-k* 查詢幫助使用者從龐大的資料中萃取出具有價值的物件，它將資料庫中的物件依照給分公式給分後，選擇出分數最高的前 *k* 個回傳給使用者。然而在多數的情況下，一個物件也許不只有一個分數，要如何在多個分數中仍然選擇出整體最高分的前 *k* 個物件，便成為一個新的問題。在本研究中，我們將這樣的物件用不確定資料來表示，而每個物件的不確定性則是其帶有機率的分數以表示此分數出現的可能性，並提出一個新的問題：*Best-kGROUP* 查詢。在此我們將情況模擬為一個複合式競賽，其中有多個子項目，每個項目的參賽人數各異，且最多需要 *k* 個人參賽；我們希望能針對此複合式競賽挑選出最佳的 *k* 個球員組合。當我們定義一個較佳的組合為其在較多項目居首位的機率比另一組合高，而最佳的組合則是沒有比它更佳的組合。為了加快挑選的速度，我們利用動態規劃的方式與篩選的演算法，將不可能的組合先剔除；所剩的組合則是具有*天際線*特質的組合，在這些天際線組合中，我們可以輕易的找出最佳的組合。此外，在實驗中，對於在所有球員中挑選最佳的組合，Best-*k*GROUP 查詢也有非常優異的表現。

Selecting the Best Group of Players for a Composite Competition

Abstract

In a large database, *top-k* query is an important mechanism to retrieve the most valuable information for the users. It ranks data objects with a ranking function and reports the *k* objects with the highest scores. However, when an object has multiple scores, how to rank objects without information loss becomes challenging. In this paper, we model the object with multiple scores as an uncertain data object and the uncertainty of the object as a distribution of the scores, and consider a novel problem named *Best-kGROUP query*. Imagine the following scenario. Assume there is a composite competition consisting of several games each of which requires a distinct number of players. Suppose the largest number is *k*, and we want to select the *best* group of *k* players from all the players for the competition. A group x is considered *better* than another group y if x has higher aggregated probability to be the top ones in more games than y. In order to speed up the selection process, the groups worse than another group definitely should first be discarded. We identify these groups using a dynamic programming based approach and a filtering algorithm. The remaining groups with the property that none of them have higher aggregated probability to be the top ones for all games against the other groups are called *skyline groups*. From these skyline groups, we can easily compare them to select the best group for the composite competition. The experiments show that our approach outperforms the other approaches in selecting the best group to defeat the other groups in the composite competitions.

誌謝

首先，我要先歸榮耀給神，在我的求學過程中，一路都有祂最美好的安排；特別在碩二的緊張和壓力中，祂不斷地用愛激勵我、帶領我，雖然遇到困難和挫折，但是感謝神，祂依然將我安置在高處。

這篇論文的完成最感謝的人是我的指導教授陳良弼老師，從碩一就開始教導我做研究的嚴謹態度，一直到碩二開始討論問題，老師特地每週一次的討論時間對我來說非常珍貴也非常感謝；還有每次的論文修改討論，都帶給我很大的幫助。除此之外，在生活上不管是受傷或生病都有老師的關心，很謝謝老師在研究所這兩年所有的付出。

也很感謝實驗室的學長姊、學弟妹，每次的報告都給我很好的建議，不僅如此，也陪伴我渡過兩年快樂的研究所生活，和你們相處的時間雖然不長，但是不管在課業或玩樂上都給我留下十分美好的回憶。

謝謝一直陪伴著我的家人：爸爸、媽媽、大姨和姊姊，你們的關心、代禱、鼓勵是我的支持和力量，每一通電話、每一封簡訊、每一次一起禱告，都給我極大的安慰；還有我的男朋友家祺，雖然我們常常一起打電動以致於必須熬夜趕報告、一起吃吃喝喝直到錢包空空如也，但是很高興、也很感謝你的陪伴。教會的牧者、小組長、同組的姊妹，還有許許多多的朋友，謝謝你們在我研究不順利的時候為我加油打氣，也包容我的低潮，也願我的喜悅與你們一同分享。

最後，我要再一次地謝謝所有師長、家人、朋友，過去的日子有你們的陪伴是神的恩典，願上帝也大大祝福你們！

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

There are many approaches to help the users to retrieve important data objects. Ranking objects and reporting the ones with the highest scores is called top-*k* query. Traditional top-*k* queries identify top-*k* objects by a scoring function which gives each object a unique score. For example, there are three players and their pin-fall records of a bowling game in Table 1-1. The bowling is scored for each pin knocked over. The players will get bonuses in two cases. One is that he/she knocked down all 10 pins with the first ball, called a *strike* and recorded as an X, while the other is that no pins are left after the second ball, called a *spare* and recorded as a slash. The bonus is that points scored for the next two and next ball after a strike and spare are doubled, respectively. By the rules above, we obtain the scores for each frame, shown in Table 1-2, and the scoring function to rank these three players in a bowling game is the aggregated scores. In this example, we can simply claim the player A gets the first place.

Table 1-1: Pin-fall records of A, B, and C.

Player	**1**		**2**		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**		**10+1**		**10+2**	
A	X		X		X		9	/	X		X		X		X		X		X		X		X	
B	X		X		5	/	X		X		X		X		X		X		X		X		X	
C	X		X		X		X		X		8	/	7	/	X		X		X		X		X	

Table 1-2: Bowling scores of A, B, and C.

Player	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Score**
A	30	29	20	20	30	30	30	30	30	30	279
B	25	20	20	30	30	30	30	30	30	30	275
C	30	30	30	28	20	17	20	30	30	30	265

In the traditional top-*k* query, the answer to a top-*k* query is the extension of that to a top-(*k*-1) query. In the example of the data in Table 1-2, the top-1 player is A and the top-2 players are A and B. When we define a *composite competition* consisting of various games each of which requires distinct number of players and suppose the largest number of players in the games is *k*, we can simply submit a top-*k* query to the database of players and take the answer as the best group of *k* players for the competition since these *k* players must contains the answers of top-1, top-2, …, and top-(*k*-1) queries.

However, when a player has more competing experiences, he/she will have not only one score, shown in Table 1-3. To rank such data objects with multiple scores, we can simply use another scoring function to obtain the average scores or the expected scores, in other words. When we replace the multiple scores with a single score, some information is omitted. For example, a player has the following scores: 100, 1000 and 1000, while another player has 50, 50 and 2000. The expected score is the same, 700, but the range and the distribution of the scores are not the same for these two players. Therefore, this way to model the multi-score data is not appropriate. In this paper, we use an *uncertainty model* to model the data objects with multiple scores. The uncertainty of each object is the probabilistic scores which keep the original score distribution, as shown in Table 1-4

Table 1-3: Records of player A, B, and C.

Player	**Past records**				
A	279	252	252	252	266
B	275	300	275	200	200
C	265	265	265	265	265

Table 1-4: Modeled as uncertain data.

Player	**Probabilistic score**	
	Score	**Probability**
A	279	0.2
	266	0.2
	252	0.6
B	300	0.2
	275	0.4
	200	0.4
C	265	1.0

After we model the objects as uncertain data, the top-*k* objects are defined as the k objects with the highest probability to be the top-*k* ones. For example, when a top-1 query is submitted to the dataset in Table 1-4, the probability to be the top-1 object for each object

needs to be obtained first and then the object with the highest probability can be decided. We compute the probability of A to be the top-1 player in three cases. One is when A gets 279 (with probability 0.2) and B and C get scores not greater than A (with probability 0.4+0.4=0.8 of B and probability 1.0 of C). A has probability 0.2*0.8*1.0=0.16 to be the top-1 player. Another is 0.2*0.4*1.0=0.08 and the other is 0.6*0.4*0=0 since C cannot be less than A when A gets 252. Therefore, the total probability of A to be the top-1 player is 0.16+0.08+0=0.24. We can obtain the probability for each player in the same way to have the top-1 player. When a top-*k* query is submitted, we view *k* objects as a group and compute the aggregated probability of these objects to be the top-*k* objects and select the group with the highest aggregated probability as the answer. Here we also use the data in Table 1-4 as an example to explain more clearly. When a top-2 query is submitted to the dataset in Table 1-4, we group these three players into 3 groups as following: {A, B}, {A, C}, and {B, C}. The aggregated probability of each group should be computed as the way in the above example. The aggregated probability of A and B to be the top-2 players is computed in nine cases. When A gets 279 (with probability 0.2), B gets 300 (with probability 0.2) and C gets the score not greater than A and B (with probability 1.0), the probability of A and B as the top-2 players is 0.2*0.2*1.0=0.04. We can compute other eight cases as the same way. Note that, when A gets 252 or B gets 200, C has no chance to get a score smaller than A and B. That is, A and B are not the top-2 players in the cases so that the probability is 0 and it does not contribute to the aggregated probability. Hence, the aggregated probability of A and B to be the top-2 players is 0.24. Again, we compute the aggregated probability for other groups to obtain the two players with the highest aggregated probability to be the top-2 ones.

However, we have an observation of the top-k queries on uncertain data. The answer of top-*k* query does not always include the answers of top-*i* queries, where *i* is less than *k*. We

take the data in Table 1-4 as an example. The answer of top-1 query is B but that of top-2 query is B and C, shown in Table 1-5 and Table 1-6. That is, when we have a composite competition consisting of two games which include one game with 1 player while the other with 2 players, the traditional method is not appropriate since the top-2 query reports B and C, but none of them is the answer of top-1 query. In other words, the top-*k* query on uncertain data cannot satisfy the need for a composite competition.

Table 1-5: The aggregated probability to be the top-1 player.

Player	**Aggregated probability**
{B}	0.52
{C}	0.24
{A}	0.24

Table 1-6: The aggregated probability to be the top-2 players.

Players	**Aggregated probability**
{A, C}	0.40
{B, C}	0.36
{A, B}	0.24

Therefore, we propose a novel problem named *Best-kGROUP query* to retrieve the best groups for the composite competition. Suppose the largest number of players in the games of the composite competition is *k*. In a game with *i* players, if a group x of *i* players has a

higher aggregated probability to be the top-*i* players than another group y, we say there is a *preference* of x over y. A group P is said better than another group Q in a composite competition if there are more preferences of the sub-groups in P than in Q. The best groups are those that are not worse than any other groups.

2 RELATED WORK

In the following, we introduce the background of uncertain data, the top-*k* processing and nearest-neighbor queries. On the other hand, we also mention the work of skyline and the reason why we choose skyline to solve our problem.

2.1 Top-*k* Queries

Top-*k* queries are useful when users are interested in the most important objects, especially in large databases. In [1], I. F. Ilyas et al. classify the top-*k* processing techniques. There are five categories as following: query model, data and query certainty, data access, implementation level, and ranking function.

There are three types of query models. One is Top-*k* Selection Query which is to report the *k* tuples with the highest score according to some scoring function. Another is Top-*k* Join Query which is the variance of Top-*k* Selection Query. Its scoring functions are attached to join results. The other is Top-*k* Aggregate Query which focuses on groups of tuples rather than single tuples.

In the data and query certainty issue, the authors furthermore classify the techniques into three types. The first two types are related to certain data and queries. The difference between them is that the first type reports the exact answers of queries while the second types reports the approximate answers instead. The last type is related to uncertain data which will be more discussed in the following sections.

In data access field, sorted access and random access are discussed. Difference data access assumptions affect the methods to retrieve the underlying data sources.

The way implementing top-*k* queries can be classified into two types. One is on application level and the other is on query engine level. The difference between these two types is the modification of the core of database engines.

The ranking functions in most techniques are assumed to be monotone while a few ones are generic form. However, in recent researches, some are without scoring function. It is called a skyline query and we would discuss it in Section 0.

2.2 Uncertain Data

Recently, the research on uncertain data has attracted a lot of interest. It is because the problems related to uncertainty cannot be addressed as traditional approaches. In [2], C. C. Aggarwal et al. survey the sources of uncertainty. It is from errors, incompleteness, and multiple records so that the data objects have probabilistic attributes. The main research area of uncertain data includes three types. One is data modeling, another is data management, and the other is data mining. Here we focus on the data management, and we use the discrete probability distribution to model our uncertain data.

2.2.1 Uncertain Top-*k* Queries

The traditional top-*k* processing is to retrieve the *k* tuples with the highest scores. In uncertain top-*k* definition, we need to consider the tradeoff between scores and probabilities. Here we will introduce two definitions proposed in [3]. Before we explain the definitions, we need the some preliminaries. Due to the uncertainty, an object may have different behaviors, and each behavior has its existent probability. Thus, when every object acts as its own behavior, we multiply the probability as *p* and claim this case is one of the possible worlds and the existent probability of it is *p*. For instance, Table 2-1 shows the example

uncertain database. We can obtain all possible worlds from it, shown in Table 2-2.

Table 2-1: The example database D.

Player	**Probabilistic score**		
	Tuple ID	**Score**	**Probability**
A	t_1	279	0.2
	t_2	266	0.2
	t_3	252	0.6
B	t_4	300	0.2
	t_5	275	0.4
	t_6	200	0.4
C	t_7	265	1.0

Table 2-2: All possible worlds of D.

Possible World	Members	Probability
PW_1	$B(t_4)$, $A(t_1)$, $C(t_7)$	0.2*0.2*1.0 = 0.04
PW_2	$A(t_1)$, $B(t_5)$, $C(t_7)$	0.2*0.4*1.0 = 0.08
PW_3	$A(t_1)$, $C(t_7)$, $B(t_6)$	0.2*0.4*1.0 = 0.08
PW_4	$B(t_4)$, $A(t_2)$, $C(t_7)$	0.2*0.2*1.0 = 0.04
PW_5	$B(t_5)$, $A(t_2)$, $C(t_7)$	0.2*0.4*1.0 = 0.08
PW_6	$A(t_2)$, $C(t_7)$, $B(t_6)$	0.2*0.4*1.0 = 0.08
PW_7	$B(t_4)$, $C(t_7)$, $A(t_3)$	0.6*0.2*1.0 = 0.12
PW_8	$B(t_5)$, $C(t_7)$, $A(t_3)$	0.6*0.4*1.0 = 0.24
PW_9	$C(t_7)$, $A(t_3)$, $B(t_6)$	0.6*0.4*1.0 = 0.24

In U-Top*k*, we sum up all probabilities over all possible worlds for a top-*k* tuples. Note that, the U-Top*k* only considers the concept of tuples. From Table 2-2, we can furthermore obtain all possible U-top2 answer in Table 2-3. Therefore, $<t_5, t_7>$ and $<t_7, t_3>$ with the highest probability 0.24 to be top-2 over all possible worlds would be the answer of U-Top2. M. Soliman et al. transform the U-Top*k* query into a state search problem. They start from the tuple with the highest score. Each time they scan a new tuple, they extend the current state with the highest probability until the length of the retrieved current state is *k*. The tuples kept in the state is the answer of U-Top*k*.

Table 2-3: All possible U-top2 of D.

Top-2 vector	**Probability**
$<t_5, t_7>$	0.24
$<t_7, t_3>$	0.24
$<t_4, t_7>$	0.12
$<t_1, t_5>$	0.08
$<t_1, t_7>$	0.08
$<t_5, t_2>$	0.08
$<t_2, t_7>$	0.08
$<t_4, t_1>$	0.04
$<t_4, t_2>$	0.04

M. Soliman et al. also define U-*k*Ranks which is totally different from U-Top*k*. U-*k*Ranks retrieves the top-*k* tuples separately. That is, we scan the every ranking position for the tuple with highest existent probability. In this example, although the U-2Ranks answer, $<t_5, t_7>$, is the same as the U-Top2 one, the semantic meaning is very different. Since the tuple of each ranking is picked separately, we might choose the same tuple in different ranking or tuples from the same object. That is, the U-*k*Ranks definition does not care of the relation between each tuple in the answer and the answer might not be valid in any possible world.

Table 2-4: All possible rank 1 and rank 2 answers.

Rank 1	**Probability**	**Rank 2**	**Probability**
t_5	0.32	t_7	0.52
t_7	0.24	t_3	0.24
t_4	0.20	t_2	0.12
t_1	0.16	t_5	0.08
t_2	0.08	t_1	0.04

In addition, M. Hui et al. propose another definition about uncertain top-*k* queries in [4], and it is called PT-*k*. For each tuple, they first define a top-*k* probability which is the possibility of the tuple as a member of the top-*k* ones over all possible worlds. When we compute the top-*k* probabilities, the straightforward approach suffers from the exclusive rules. M. Hui et al. use the compressed dominant set of each tuple to simplify the steps. In a compressed dominant set of a tuple t, all tuples are independent of t. As we know, all the probabilities can be simply multiplied when all tuples are independent. Hence, the computing process can be efficiently simplified. The answer of PT-k is all the tuples with top-*k* probability higher than an input threshold *p*. In the example of Table 2-1, when *p*=0.5, the answer of PT-2 is only one tuple $\{t_7\}$. From the definition, the answer of PT-*k* is simply a set of tuples with top-*k* probability higher than *p*. The relation of each tuple is not considered when PT-*k* reports the answer.

Table 2-5: The top-*k* probability of all tuples.

Tuple ID	**Top-*k* probability**
t_1	0.20
t_2	0.20
t_3	0.24
t_4	0.20
t_5	0.40
t_6	0.00
t_7	0.76

The uncertain top-*k* queries above focus on the most probable tuples to be the top-*k*. However, the answers might not have the highest score. In [5], T. Ge et al. observed the answers of uncertain top-*k* queries might be atypical. It means the answer might have relative low score. For example, the average score of the answer tuples might not be higher than the expected score. Or combinations with aggregated scores of tuples higher than the answer are many and their total probability is much higher than the answer is. In these situations, the uncertain top-*k* answers are perhaps not appropriate in score concept. Therefore, T. Ge et al. propose *c*-Typical-Top*k* to retrieve *c* combinations to represent the whole score distribution the dataset. It first uses a dynamic programming based approach to generate the whole distribution. Then, it maps the process of retrieving *c* combinations from all combinations of score distribution to the *p*-median problem and resolves by two recursion

functions.

2.2.2 Uncertain Top-*k* Queries on Data Streams

Data streaming has been focused for a long time because it models the way data collected from the real-world applications. In a streaming problem, the challenge is the rapid growth of the amount of data. Both the execution time and storing space makes the approaches designed for the static databases useless. We can classify the streaming models into two types. One is unbounded streaming and the other is bounded streaming. The former collects data from the beginning while the latter would remove the expired data. This is more challenging when we need to not only append new coming data but also remove old ones. We call this a sliding-window model since the valid data only appear in the window and the window would slides on the stream.

In [6], C. Jin et al. propose a framework to process difference types of top-*k* queries on uncertain data streams. They claim the previous definitions, such as U-Top*k*, U-*k*Ranks, PT-*k*, and so on, can be plugged into this framework. In their framework, they first define a compact set which keeps the minimal tuples to retrieve the uncertain top-*k* answers even if new tuples are appended. However, the problem is defined on sliding-window model, so we use multiple compact sets at different time-stamps to implement the removing operation. The authors furthermore propose other compression approaches to reduce the space the large amount of compact sets would need.

2.2.3 Uncertain Nearest Neighbor Queries

In addition to the uncertain top-*k* queries, there are uncertain NN queries to retrieve the most important objects. The NN (nearest neighbor) queries report the closest object to the

query object. However, in the uncertain database, every object has its probability to be the NN and we define this probability as PNN. G. Beskales et al. propose Top*k*-PNN to report the *k* objects with the highest probability to be the NN in [7]. They consider both I/O and computing time to design an efficient algorithm for retrieving the *k* objects. Instead of computing the exact PNN of all objects, they use a lazy bound to simplified the computation and reduce the cost. Here we can transform the NN queries into top-*k* queries. We let the scoring function be the distance of the object and the query, and the score is the lower the better. Now we can find out the PNN of an object is the same as the probability of a player to be the top-1 team in 1-game. We will take Top*k*-PNN for comparison in our experiments.

2.3 Skyline

The skyline query is also a useful approach to retrieve important data. Before we introduce the definition of skyline, we have to explain what the term “dominate” is. Suppose the value is the smaller the better. A point A is said to dominate another point B if and only if for every dimension, the value of A is less than or equal to that of B, and at least in one dimension the value of A is less than that of B. For example, A is <5, 6, 7, 8> and B is <8, 6, 7, 8>. Then, we can easily tell that A dominates B. Moreover, if there is a point cannot be dominated by any other points in the database, we call the point as a skyline point.

However, as the number of dimensions increases, a point becomes more difficult to be dominated. That is, the number of skyline points becomes huge. Therefore, M. L. Yiu et al. combine the advantages of skyline and top-*k* queries to define a new query named top-*k* dominating query in [8]. It computes the number of points dominated by each point as the ranking score, and then returns the points with the highest scores. Users do not need to worry about how to define a scoring function and can simply give a value *k* as parameter to

restrict the size of the answers.

Another work on high dimension skyline operation is proposed by C.-Y. Chan et al. in [9]. They define *k*-dominate to describe a point A has the dominance property of another point B in a subspace of original space. According to the *k*-dominance, they also define *k*-dominant skyline consisting of points which cannot be *k*-dominated. Note that, the problem does not only reduce the dimension into some subspace since the dominance relationship between any two points is not specified in a fixed subspace. We have the property that if a point A (*k*+1)-dominates another point B, A must *k*-dominates B. It implies that every *k*-dominant skyline point must be one of the members of (*k*+1)-dominant skyline. Hence, we know the size of the answers of *k*-dominant skyline must be less than that of original skyline.

3 METHODOLOGY

In this section, we will first define our problem and then show the basic algorithms to address it. In the basic algorithms, the aggregated probability for all groups should be computed so that we use a dynamic programming based approach to achieve the goal. However, in consideration of the time and space complexity, we turn to heuristics to reduce the cost and compare the performance in the Experiment Section. Besides, we also exploit the property of skyline to prune those with no chance to be the answer. The number of the remaining ones is much smaller after the pruning process so that we can compare them directly to retrieve the best as the answer.

3.1 Problem Definition

Before giving the definition of our problem, we first define the terms used in the following.

Definition 3-1 ***i*-game.** A game with *i* players in one team.

Definition 3-2 ***i*-group.** A set of *i* players.

Definition 3-3 **aggregated probability of an *i*-group.** The probability is the sum of the probabilities of the players in this *i*-group to be the top-*i* players. In other words, we sum up the U-Top*i* probabilities of the tuples related to this *i*-group.

Definition 3-4 ***k*GROUP**. A *k*GROUP is a set of *k* groups, which includes one each for all sizes of groups. For example, a 3GROUP should include three groups which are a 1-group, a 2-group, and a 3-group. Besides, for an *i*-group in the *k*GROUP, it should be the sub-group of the *k*-group in the same *k*GROUP, and its aggregated probability is the highest among all sub-groups with size *i* of the *k*-group mentioned above.

Definition 3-5 **HiRank vector of a *k*GROUP.** It is a vector with *k* dimensions. Assume that we sort all *i*-groups by aggregated probability and define that the one whose rank is higher has higher aggregated probability. The HiRank vector keeps the rank in dimension *i* for each *i*-group in this *k*GROUP. For example, in Table 3-3, the 2GROUP, {{B}, {B, C}}, has <1, 2> as the HiRank vector since {B} has the highest aggregated probability so that {B} is ranked first in 1-game and {B, C} has the second highest aggregated probability so that it is ranked second in 2-game.

Definition 3-6 **better *k*GROUP**. For an *i*-game, if an *i*-group x has a higher aggregated probability than another *i*-group y, we define there is a preference of x over y. Hence, a *k*GROUP P is a better *k*GROUP than another *k*GROUP Q if there are more preferences of the *i*-groups in P than in Q.

Definition 3-7 **best *k*GROUP**. A *k*GROUP is the best *k*GROUP if no other *k*GROUP is better than it.

Next, we introduce the concept of skyline to help retrieve all *k*GROUPs with chance to be the best one.

Definition 3-8 s**kyline *k*GROUP.** For a *k*GROUP, if there are no other *k*GROUPs dominating this *k*GROUP with respect to the HiRank vector, we call it as a skyline *k*GROUP. A *k*GROUP P is said to *dominate* another *k*GROUP Q if and only if in the HiRank vector V_P and V_Q, $V_P.i$ is less than or equal to $V_Q.i$ for every dimension *i*, and $V_P.j$ is less than $V_Q.j$ in

some dimension *j*. We also use the term non-skyline *k*GROUP in opposition.

Now, we can give the definition of our problem based on the following uncertainty model. For a database D, there are *n* uncertain players each of which has probabilistic scores, shown in Table 2-1. We can also represent D as the concept of tuples as shown in Table 3-1.

Table 3-1: The tuple concept of D.

Tuple ID	**Related player**	**Score**	**Probability**
t_1	A	279	0.2
t_2	A	266	0.2
t_3	A	252	0.6
t_4	B	300	0.2
t_5	B	275	0.4
t_6	B	200	0.4
t_7	C	265	1.0

Definition 3-9 **Best-*kGROUP* query.** The Best-*k*GROUP query returns the best *k*GROUPs.

Here is an example to illustrate the Best-*k*GROUP query. According to the database D in Table 2-1, we can obtain a table keeping all groups sorted by their aggregated probabilities, shown in Table 3-2. Suppose we submit a Best-2GROUP query to D. Then, we need to find the HiRank vectors of all 2GROUP. The information of finding results of the

Best-2GROUP query is shown in Table 3-3. Thus, {{A}, {A, C}} and {{B}, {B, C}} are both the answers of the Best-2GROUP query.

Table 3-2: The table with all groups and their aggregated probabilities.

1-game		**2-game**		**3-game**	
{B}	0.52	{A, C}	0.40	{A, B, C}	1.000
{A}	0.24	{B, C}	0.36		
{C}	0.24	{A, B}	0.24		

Table 3-3: Information of selecting the best 2GROUPs of the dataset in D.

2GROUP	**HiRank vector**	
{{A}, {A, C}}	<2, 1>	
{{B}, {B, C}}	<1, 2>	
{{A}, {A, B}}	<1, 3>	{{B}, {B, C}} is better than it

3.2 The Basic Algorithms

In comparison with our basic algorithms, we first introduce a naïve approach to discover the results of the Best-*k*GROUP query.

In the naïve approach, we generate all the *k*-groups, and then, for each *k*-group, create a *k*GROUP having this *k*-group and it sub-groups according to Definition 3-4. After we obtain all HiRank vectors of all *k*GROUPs, we can compare which is better with each other. Then, after those *k*GROUPs worse than the others are removed, the remaining ones are the best.

However, this approach suffers from the high cost in computation. Therefore, we propose the basic algorithms to improve it.

First, we generate all groups with their aggregated probability based on dynamic programming. Here is our algorithm *GroupGen* in Figure 3-1. The input of *GroupGen* is a database in tuple concept ordered by score. In a dynamic programming based approach, tuples should be independent; otherwise, we cannot reuse the previous computation to reduce the computing time. However, tuples in D are often related to other tuples, i.e. they belong to the same player. That is, we cannot compute the correct probabilities from only one scan. Instead, we perform an incremental method. Each time we retrieve a new tuple in D, we perform a new scan reversely starting from the new tuple (line 5-8) and stopping at the first tuple in D. However, in each scan, tuples related to this new tuple should be removed and others can be unified into ones with respect to its player. We use MEHash to keep the information of all unified probabilities (line 4). Hence, only unified probabilities except the one related to the new tuple needs computing (line 9-12). Furthermore, in one scan, *GroupGen* would generate only part of aggregated probability for those groups related to the tuples scanned so far, so we keep these probabilities in cHash temporarily (line 1 and 13). After we retrieve all tuples in D, the aggregated probabilities of the groups in cHash become complete. Then, all groups are distinguished from their size and sorted by aggregated probability as the result to return (line 15-16).

```
     Input: D: a database in tuple concept ordered by score
     Output: all groups distinguished from size and sorted by aggregated probability

 1   cHash: temporarily keeps all groups and their current probability
 2   MEHash: keeps all players and their current probability
 3   for each t in D do
 4       Update t to MEHash
 5       DPTable[|MEHash|][k]: the dynamic programming table
 6       index = |MEHash|-1
 7       DPTable[index][1] = new Group(t)
 8       index = index-1
 9       for each player in MEHash but not owning t do
10           UpdateDPTable(DPTable, Pr(player), index)
11           index = index-1
12       end for
13       put all groups in DPT[0] into cHash
14   end for
15   classify all groups by size and sort each class by probability
16   return all groups and their probability

17   Function UpdateDPTable(DPTable, P, index)
18   for i = 1 to k do
19       multiply all groups in DPT[index+1][i-1] by P, and add to DPT[index][i]
20       multiply all groups in DPT[index+1][i] by 1-P, and add to DPT[index][i]
21   end for
```

Figure 3-1: Algorithm *GroupGen*.

In Figure 3-2, we illustrate the process of updating the dynamic programming table. In each scan of a new tuple, we would put the object related the new tuple onto the last row and other objects appeared so far onto the higher rows in the dynamic programming table. Note that, the order of other objects does not matter since they are all independent. We start from

the bottom left cell and append a new group {O'} to the cell. Since any 2-group cannot be generated by only one object, we stop the update of the last row and scan upwards. For a cell X, the groups inside are generated by two ways. One is from the lower left cell A. In this case, all groups in A should be extended with O_j and appended to X, and the probability of each group should be multiplied by the existence probability of O_j. The other is from the lower cell B. This time we simply append all groups in B to X with their probability multiplied by the non-existence probability of O_j $(1 - \Pr(O_j))$. When we finish updating all cells in the table, the groups in the first row (as the gray region) are the ones with correct probability in this scan.

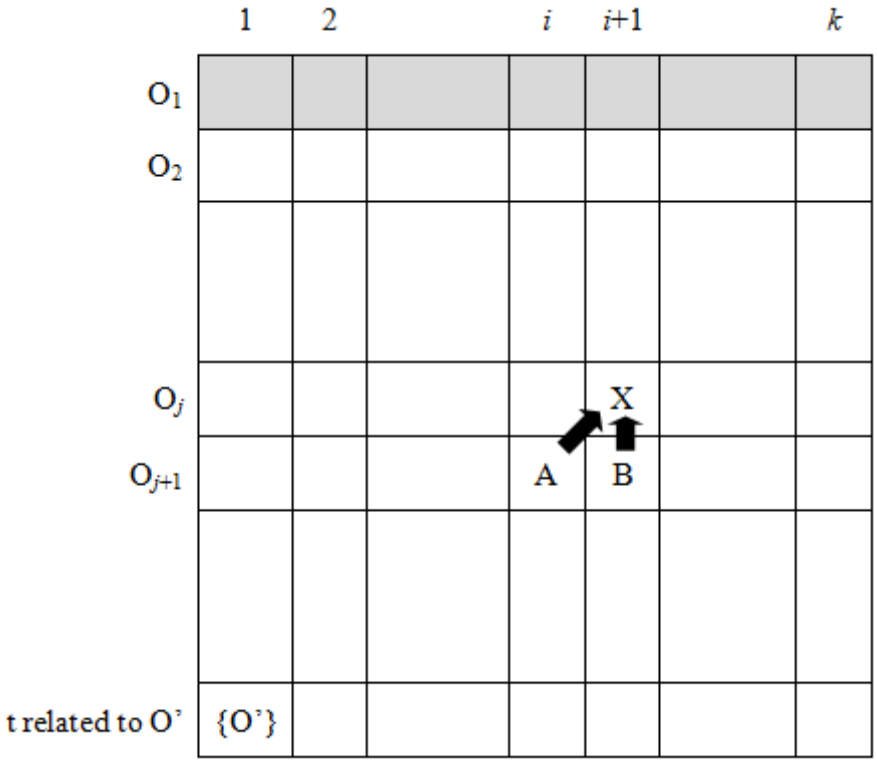


Figure 3-2: Illustration of updating the dynamic programming table.

Since the non-skyline *k*GROUPs must be worse than a specific *k*GROUP by Definition 3-6 and Definition 3-8, we can ignore them is this step. We will introduce *SubsetFilter* to

retrieve the skyline *k*GROUPs without computing all HiRank vectors. However, before we discuss the algorithm *SubsetFilter*, we need to show some properties of the *k*GROUPs.

Property 1. For a *k*GROUP G and its HiRank vector V_G, any other *k*GROUP having an *i*-group whose rank is higher than $V_G.i$ (the value in dimension *i* of V_G) would never be dominated by G.

Proof: Assume that a *k*GROUP A is dominated by G and having some *i*-group whose rank is higher than the *i*-group in G. Here we denote the HiRank vector of A as V_A. We can obtain two formulas from this assumption. One is $(\forall j, V_A.j \geq V_G.j) \wedge (\exists j, V_A.j > V_G.j)$, and the other is $V_A.i < V_G.i$. However, it is contradiction between these two formulas. Hence, the assumption is false and we know a *k*GROUP having some *i*-group whose rank is higher than $V_G.i$ must not be dominated by G.

It is easy to verify the correctness of this property. Recall the definition of dominating. We can formulate the equivalence proposition of it. A vector A is said not to dominate another vector B if and only if the value of A is great than that of B in any dimension. Hence, the *k*GROUPs with the *k*-group extended from any *i*-group whose rank is higher than $V_G.i$ must have an *i*-group whose rank is higher than G in dimension *i*. This is why we claim they must not be dominated by G.

Property 2. If a *k*GROUP G has a *k*-group with the highest aggregated probability and there is no other *k*-group with the same probability as it, this *k*GROUP G must be a skyline *k*GROUP.

Proof: We prove this by the equivalence proposition. If G is a non-skyline *k*GROUP, there must be a *k*GROUP A dominating G. That is, $(\forall i, V_A.i \leq V_G.i) \wedge (\exists i, V_A.i < V_G.i)$. In dimension *k*, there are two situations. One is when $V_A.k < V_G.k$. It turns out that G does not have a *k*-group with the highest aggregated probability. The other is when $V_A.k = V_G.k$.

It makes that there are at least two *k*GROUPs having the *k*-group with the same aggregated probability. Since the equivalence proposition is true, Property 2 is proven.

To use this property, we assume that each time distinguishing a skyline *k*GROUP, we would prune all *k*GROUPs dominated by it. That is, the *k*GROUPs not being removed so far must not be dominated by any reported skyline *k*GROUP. Moreover, the *k*GROUP having the *k*-group with the highest aggregated probability must not be dominated by any other *k*GROUP having the *k*-group whose rank is lower than it. Since this *k*GROUP cannot be dominated, it must be a skyline *k*GROUP. Note that, if there are more than one *k*GROUPs having the *k*-group with the highest aggregated probability, at least one of them must be a skyline *k*GROUP.

In algorithm *SubsetFilter*, shown in Figure 3-3, we take all groups as input and it outputs all skyline *k*GROUPs. At first, S is used to keep all the possible answers (line 1). We start from the *k*GROUP G having the *k*-group with the highest aggregated probability according to Property 2 (line 3-8). Then, we use Property 1 to filter out those *k*-groups extends from *i*-groups whose ranks are higher than *v.i* for some *i*-game and remove other *k*-groups since the *k*GROUPs created from them must be non-skyline ones (line 15-19). After adding G into S (line 20), we repeat the whole process until the input set K is empty. Finally, we can return all skyline *k*GROUPs.

```
Input: K: all k-groups; O: other groups
Output: all skyline kGROUPs

1   S = φ : keeps all skyline kGROUPs
2   while K is not empty do
3       if more than one k-groups with the highest aggregated probability then
4           create kGROUPs for all these k-groups according to Definition 3-4
5           G = any skyline kGROUP of them
6       else
7           G = a kGROUP having the k-group and its sub-groups according to Definition 3-4
8       end if
9       remove the k-group chosen by G
10      v = the HiRank vector of G
11      Subset = φ : a set to filter out possible skyline kGROUPs
12      for each dimension i do
13          add all i-groups whose rank is higher than or equal to v.i to Subset
14      end for
15      for each g whose rank is lower than or equal to v.k in K do
16          if g is not a superset of any of Subset then
17              remove g from K
18          end if
19      end for
20      add G into S
21  end while
22  return all kGROUP in S
```

Figure 3-3: Algorithm *SubsetFilter*.

After we retrieve all skyline *k*GROUPs, we can directly compare each other to obtain the best one since the size of skyline *k*GROUPs is much smaller than the size of original *k*GROUPs. The algorithm to retrieve the best *k*GROUPs is shown in Figure 3-4.

```
    Input: S: keeps all skyline kGROUPs
    Output: the best kGROUPs

1   Answer: keeps all possible answers
2   for each g in S do
3       for each a in Answer do
4           if g is better than a then
5               a is removed from Answer and g is appended to Answer
6           else if a is better than g then
7               break
8           else
9               g is appended to Answer
10          end if
11      end for
12  end for
13  return all kGROUPs in Answer
```

Figure 3-4: Algorithm *BestGROUP*.

Note that, the best *k*GROUPs from all players are supposed to have the largest chance to be better than other ones. That is, when we randomly select a *k*GROUP x from any other ones, the probability of the best *k*GROUPs being better than x should be high. In another consideration, the number of *k*GROUPs dominated by a skyline *k*GROUP can be viewed as the lower bound of its number of *k*GROUPs being worse than it. A skyline *k*GROUP x being better than another skyline one y must have larger number than y since when x is better than y, x is also better than any dominated by y. In this paper, the best *k*GROUPs are defined as the ones not being worse than any other ones. That is, no other *k*GROUPs have larger number than the best ones. And it leads to the largest chance of the answer to be the best *k*GROUPs.

3.3 The Heuristic Approaches

We analyze the time complexity of *GroupGen* and obtain the time complexity $O(nkN)$. However, in each iteration, the cell of the dynamic programming table contains $\binom{n}{i}$ groups, where *i* is the size of groups. That is, when we consider the computation in each cell, the total complexity becomes $O(n^k \cdot nkN)$. It is why we turn to heuristics to reduce the cost.

An observation is that the probabilities of many groups are low or even equal to zero. We have an attempt to ignore the groups with the probability 0 and it is called *IgnoreGroupGen*. *IgnoreGroupGen* makes no difference in the membership of skyline *k*GROUPs. However, the cost of this attempt might still be high. Another attempt, *LimitGroupGen*, is to restrict the size of each cell. *LimitGroupGen* would lead to an amount of inaccuracy but benefit the cost greatly. It only keeps *m(α)* groups with the highest probability in each cell where the equation *m(α)* is defined as following.

$$\mathrm{m}(\alpha) = \begin{cases} \min\left(\binom{n}{k}, \alpha \cdot \log\binom{n}{k}\right), & k < \frac{n}{2} \\ \min\left(\binom{n}{\lceil \frac{n}{2} \rceil}, \alpha \cdot \log\binom{n}{\lceil \frac{n}{2} \rceil}\right), & k \geq \frac{n}{2} \end{cases}$$

Equation 1

In Equation 1, we define the equation *m(α)* with respect to the log of the maximum value of $\binom{n}{i}$, where *α* is a scale parameter and *i* is from 1 to *k*. If *k* is less than $\frac{n}{2}$, the maximum value will be $\binom{n}{k}$, otherwise $\binom{n}{\lceil \frac{n}{2} \rceil}$. Note that, when *k* or *n* is too small, the log of $\binom{n}{k}$ (or $\binom{n}{\lceil \frac{n}{2} \rceil}$) might be less than $\binom{n}{k}$ (or $\binom{n}{\lceil \frac{n}{2} \rceil}$), and we should choose $\binom{n}{k}$ (or $\binom{n}{\lceil \frac{n}{2} \rceil}$) in these cases.

More analyses of the heuristic approaches will be shown in Experiment Section.

4 EXPERIMENTS

In this section, we set up experiments to compare our Best-*k*GROUP query and other uncertain top-*k* queries proposed in previous work. Here we consider two algorithms as following: U-Top*k* and Top*k*-PNN. Besides, we use the expected value approach as the baseline of our experiments since it is the most straightforward one.

In our experiments, we generate 20 players and their scores. Each player has at least one score and at most ten scores in the distribution. Note that, there are no two or more players with the same scores.

Here we compare the probability of defeating other groups in a composite competition and execution time. In addition, we also examine the accuracy of *LimitGroupGen*.

4.1 Experiment Setup

Before we start the experiments, we should first define the inputs. The input is the *k*SET which is a set of *k* groups and includes one each for all sizes of groups. For an *i*-group in this *k*SET, it must be the sub-group of the *k*-group in the same *k*SET. From the definition above, we know the *k*SET is the superset of the *k*GROUP and we can take the answer of the Best-*k*GROUP query as the *k*SET. For each algorithm except our Best-*k*GROUP query, the *k*SET consists of the *i*-group which is the answer of the query on parameter *i* for each *i*. However, when the answer is not the sub-group of the *k*-group in the *k*SET, we randomly choose one of the sub-groups of the *k*-group since the algorithm does not provide other answers.

In the competition experiments, we make 20 composite competitions in iteration and in

each iteration, we randomly generate a *k*SET to compete with the one of the algorithm to be compared. These 20 composite competitions consist of different number of games. A competition with *k* players is composed of *k* distinct games, from 1-game to *k*-game. In a composite competition with *k* players, we have to input two *k*SETs and then obtain the better of this competition. Note that, the better *k*SET here is defined the same as the better *k*GROUP, so is the best *k*SET. We sum up the counts of preference of these two *k*SETs. Here the *i*-group of each *k*SET is for the *i*-game. The one with more preferences defeats another in this competition. We use the probability of defeating other *k*SETs over 100 iterations to compare all algorithms.

Here we introduce the way to perform the experiment in each *i*-game. Since each player has many scores as a discrete distribution, we need to model the distribution rather than use the expected value to simplify it. We use a straightforward approach to model all the distribution of all players. We randomly pick a score from his/her distribution as the performance of this player. Assuming that the randomly-picking approach would fit the probability model in large quantities, we perform this step MAX_BOUND times (line 21). Then, in each iteration, we sort all players with their randomly-picked scores. Any *i*-group has exactly the top-*i* players would obtain one count. The *i*-group with more counts after MAX_BOUND iterations is the one gaining a preference. The detail process of the experiment is shown in Figure 4-1.

```
    Input: two kSETs, A and B
    Output: the better of A and B

 1  BetterCountOfA = 0
 2  BetterCountOfB = 0
 3  for each i-game do
 4      preferred = MorePreferences(i-group of A, i-group of B, i)
 5      if A is the prefered then
 6          BetterCountOfA ++
 7      else
 8          BetterCountOfB ++
 9      end if
10  end for
11  if BetterCountOfA > BetterCountOfB then
12      return A is the better
13  else if BetterCountOfA < BetterCountOfB then
14      return B is the better
15  else
16      return cannot be distinguished
17  end if

18  Function MorePreferences(A, B, i)
19  PreferenceCountOfA = 0
20  PreferenceCountOfB = 0
21  while count < MAX_BOUND do
22      for each player in database do
23          randomly pick a score from the distribution of player
24      end for
25      sort all players with score
26      if members in A is exactly the top-i of all players then
27          PreferenceCountOfA ++
28      else if members in B is exactly the top-i of all players then
29          PreferenceCountOfB ++
30      end if
31  end while
32  if PreferenceCountOfA > PreferenceCountOfB then
33      return A
34  else if PreferenceCountOfA < PreferenceCountOfB then
35      return B
36  else
37      MorePreferences(A, B, i)      // perform again to decide which to return
38  end if
```

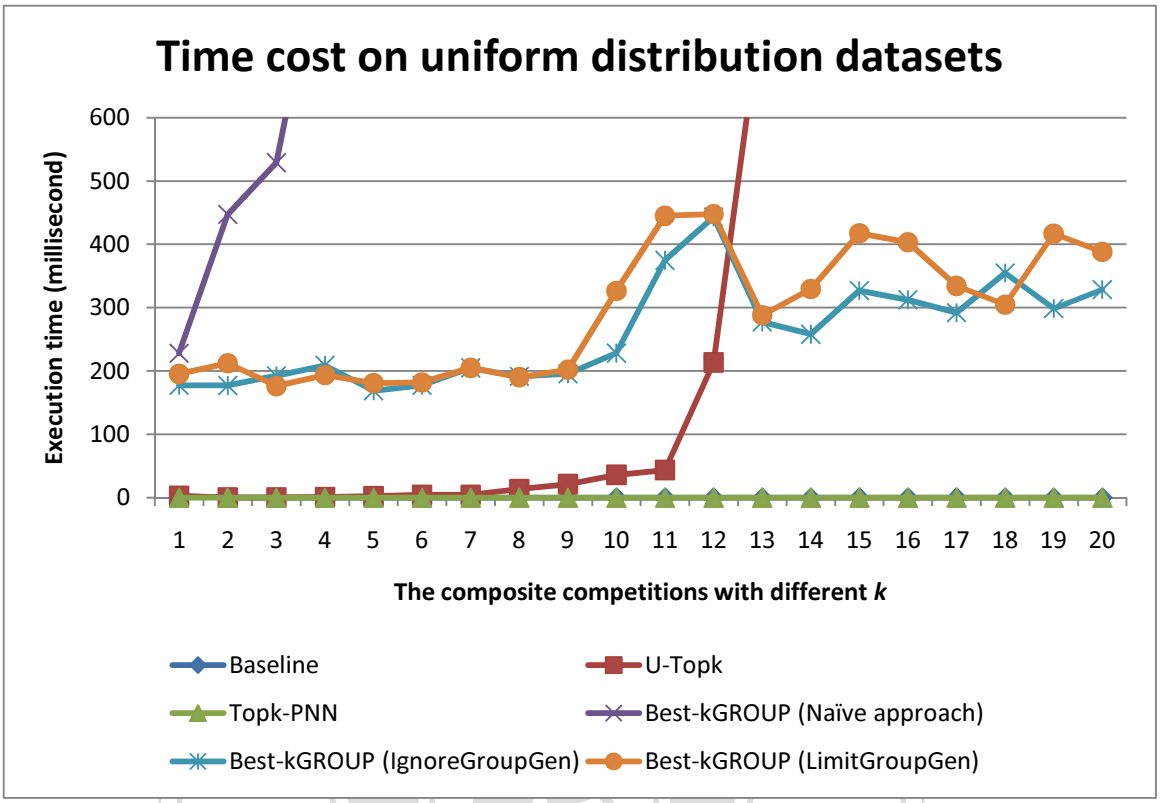
Figure 4-1: Experiment process.

4.2 On Execution Time

In Figure 4-2, we first compare the execution time among these algorithms in two types of datasets. One is uniform distribution and the other is normal distribution. Here we do not perform the algorithm *GroupGen* because it costs too much space. But we can know exactly that it must take longer time than *IgnoreGroupGen* does.

The average execution time of normal distribution dataset is larger since we have to model the normal distribution by almost 10 scores to keep the characteristic of the normal distribution while the datasets of uniform distribution is unrestricted.

The baseline and Top*k*-PNN cost a constant time for every *k* value on datasets of both distributions. However, in the dataset of uniform distribution, the time U-Top*k* costs raises up severely when *k* increases because U-Top*k* needs to scan the whole database to find the answers. Also, the time *IgnoreGroupGen* and *LimitGroupGen* cost is almost the same because the score range of each player overlaps a lot and the probability might be little but hardly be 0. In this case, the drawback of *LimitGroupGen* sorting all groups in each cell becomes more obvious. On the other hand, in the dataset of normal distribution, the case of overlapping is reduced so that U-Top*k* runs with a constant time, too. Since the probability of a group becomes 0 when the scores of some player not in this group have all appeared, less overlapping leads to more groups with probability 0. Here, *IgnoreGroupGen* takes almost twice the time of *LimitGroupGen* does. In other words, the number of groups ignored of *IgnoreGroupGen* might be also about twice of that of *LimitGroupGen*.



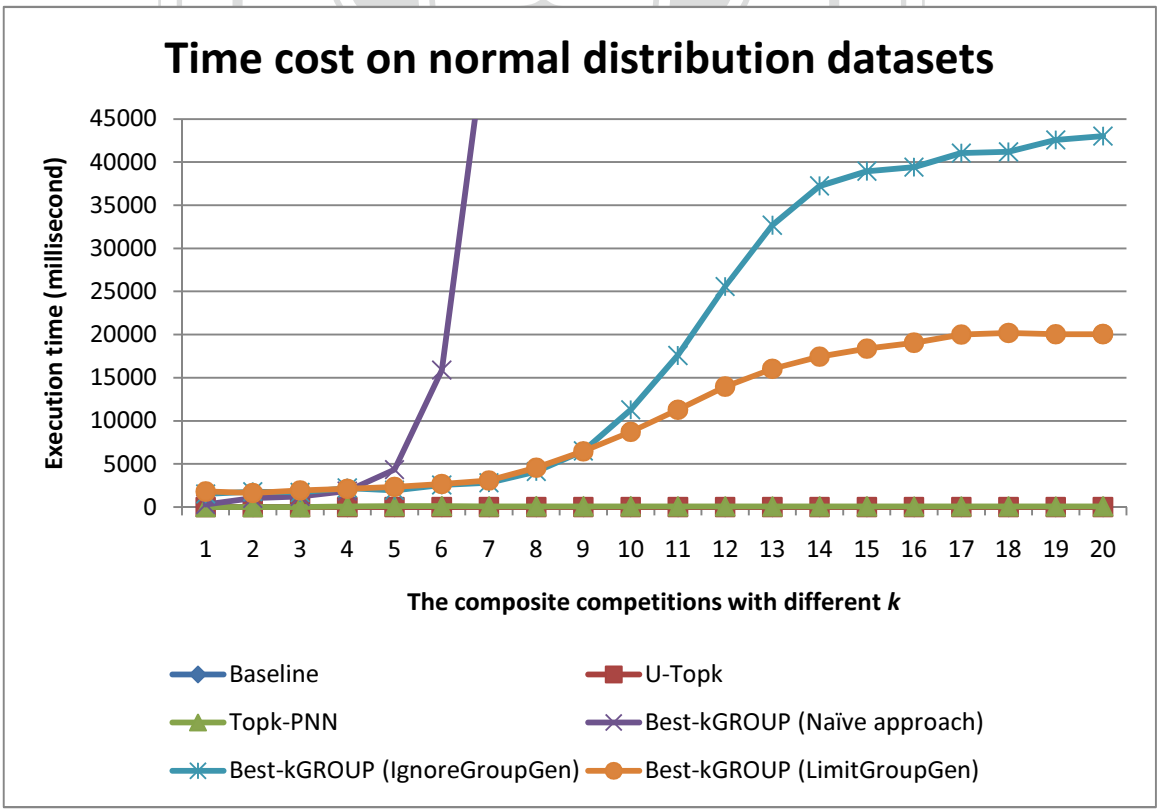


Figure 4-2: Execution time among different algorithms.

4.3 On Accuracy

Here we compare the accuracy between *GroupGen* and *LimitGroupGen*.

We can see the accuracy from the Figure 4-3 below. The F1-measure is computed by the value of precision and recall. We find all best *k*GROUPs from *GroupGen* and *LimitGroupGen*, separately. The precision is the percentage of the *k*GROUPs matched in two approaches over all generated by *LimitGroupGen* while the recall is over all generated by *GroupGen*.

Here the value of α can be 5, 100, 500, and 1000. The larger the value of α is, the higher the accuracy is. However, the difference is not obvious in Figure 4-3. It might be because of the small size of our datasets. The values of F1-measure are almost 1 but when *k* is 8, the value is relatively low. It is because the groups ignored in the cells of a dynamic programming table affect the rankings of the related groups so that some *k*GROUPs have wrong HiRank vectors and the algorithm reposts the wrong answer. Hence, the value of F1-measure is only related to the reason and we cannot estimate the trend in F1-measure.

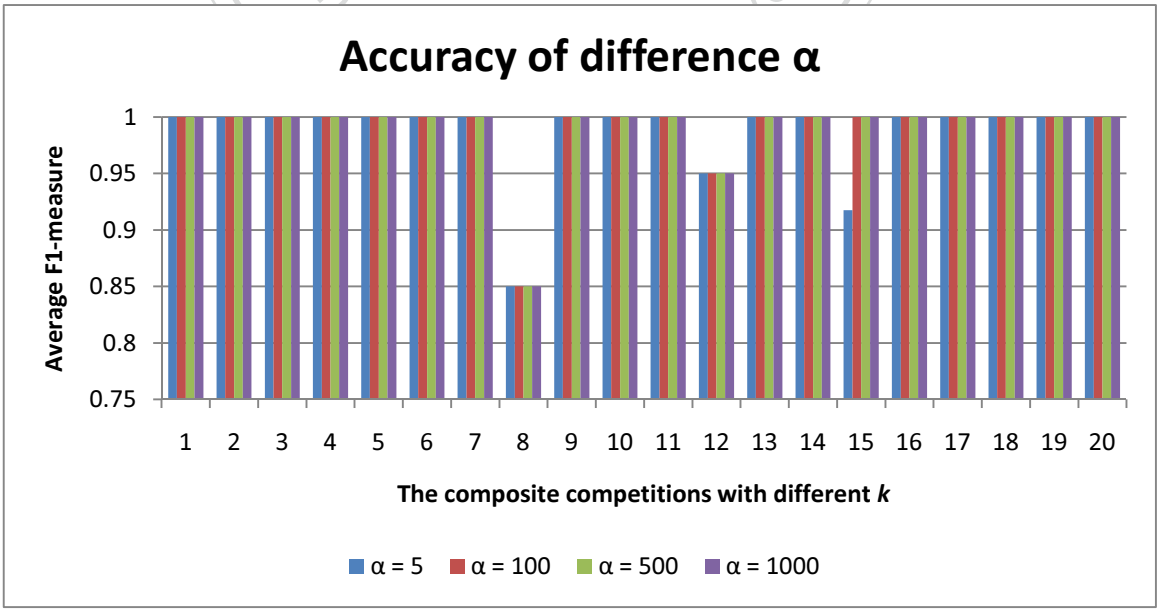


Figure 4-3: Accuracy of *LimitGroupGen*.

4.4 On Performance of the Composite Competition

In this section, we would compare the performance of the expected value approach (baseline), U-Top*k*, Top*k*-PNN, and *k*-group. We use the experiment process mentioned in Section 4.1.

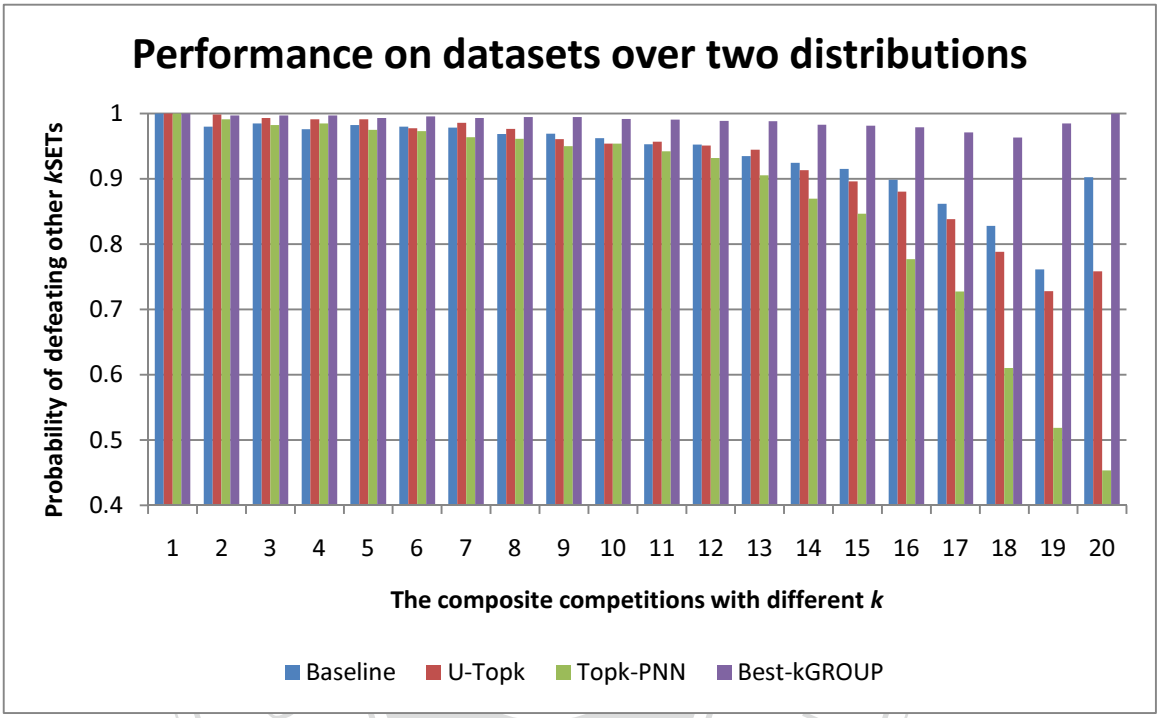


Figure 4-4: Probability of defeating other *k*SETs among different algorithms.

From Figure 4-4, we can see the probability of defeating other *k*SETs is lower down when the value of *k* increases in baseline, U-Top*k*, and Top*k*-PNN. The baseline considers the multiple scores as an expected value and although the answer in *i*-game is the *i* players with the highest expected value, the real probability to be the top-*i* ones is not considered. In Top*k*-PNN, although the real probability is computed, it does not consider to put *k* players in one group. Instead, it puts each player in a single group and independently computes the probability of each group. However, U-Top*k* considers the real probability and put *k* players into one group, but its algorithm is *tuple-wise* and incoherent. The term “tuple-wise” means that each *i*-group represents a specific performance of *i* players and the consideration is not

comprehensive. Moreover, when a group is reported as the answer of U-Top*k*, the players of any of its sub-groups may not have high probability to be the top-*i* players. When *k* increases and the competition becomes complicated, these three algorithms cannot give a good answer for the competition.

Therefore, the Best-*k*GROUP query outperforms other algorithms in selecting the best *k*GROUP for a composite competition. Also, when the competition becomes more and more complicated, the Best-*k*GROUP query shows the distinction.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we propose a novel problem to select the best *k*GROUP in a composite competition. The concept of the composite competition is not considered in previous work so that their answers are not suitable for addressing this problem.

In addition, we use heuristics to reduce the cost of generating all groups with aggregated probability. By a skyline filter, we can remove *k*GROUPs with no chance to be the answer. The *SubsetFilter* algorithm furthermore exploits the relationship among dimensions to retrieve the skyline without redundant steps. Also, the experiments show the value of our heuristic approaches and the outperformance of our Best-*kGROUP* query.

Our future work will be solving the problem in a more realistic environment. The databases should be updated when players play new games and have new records. Reusing the computing results in the past would reduce the cost but is challenging. Besides, current uncertain model uses a discrete distribution. If we discuss objects with continuous distribution such as moving objects, what the best uncertain model is should be first considered. Moreover, the algorithm to generate all groups with probability is costly, in this paper, we use heuristics to reduce the cost and we are looking for other algorithms to improve the complexity. Overall, this paper only describes the first and simple step of our work and we believe further research would also be interesting.

6 REFERENCES

[1] I. F. Ilyas, G. Beskales, and M. A. Soliman. 2008. A Survey of Top-k Query Processing Techniques in Relational Database System. *ACM Computing Surveys*, Vol. 40, No. 4, pp. 11:1-11:58.

[2] C. C. Aggarwal and P. S. Yu. 2009. A Survey of Uncertain Data Algorithms and Applications. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 21, No. 5, pp. 609-623.

[3] M. Soliman, I. Ilyas, and K. Chang. 2007. Top-k Query Processing in Uncertain Databases. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE)*, pp. 896-905.

[4] M. Hua, J. Pei, W. Zhang, and X. Lin. 2008. Efficiently Answering Probabilistic Threshold Top-k Queries on Uncertain Data. In *Proceedings of the 24th International Conference on Data Engineering (ICDE)*, pp. 1403-1405.

[5] T. Ge, S. Zdonik, and S. Madden. 2009. Top-k Queries on Uncertain Data: On Score Distribution and Typical Answers. In *Proceedings of the 35th International Conference on Management of Data (SIGMOD)*, pp. 375-388.

[6] C. Jin, K. Yi, L. Chen, J. X. Yu, and X. Lin. 2008. Sliding-Window Top-k Queries on Uncertain Streams. In *Proceedings of the VLDB Endowment*, Vol. 1, No. 1, pp. 301-312.

[7] G. Beskales, M. A. Soliman, and I. F. Ilyas. 2008. Efficient Search for the Top-k Probable Nearest Neighbors in Uncertain Databases. In *Proceedings of the VLDB Endowment*, Vol. 1, No. 1, pp. 326-339.

[8] M. L. Yiu and N. Mamoulis. 2007. Efficient Processing of Top-k Dominating Queries on

Multi-Dimensional Data. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, pp. 483-494.

[9] C.-Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. 2006. Finding k-Dominant Skylines in High Dimensional Space. In *Proceedings of the 32nd International Conference on Management of Data (SIGMOD)*, pp. 503-514.