# “Anything You Can Do, I Can Do Better.” – Team Recommendation for Fantasy Games


Chengkai Li, UT Arlington
Nan Zhang, George Wash. U
Naeemul Hassan, UT Arlington
S. Rajasekaran, George Wash. U
Gautam Das, UT Arlington



## ABSTRACT

Motivated by the booming multi-billion dollar industry of fantasy sports, we propose and investigate the novel problem of team recommendation for fantasy sports games. The problem is modeled as finding the *skyline $k$-tuple groups* from an $n$-tuple dataset - i.e., groups of $k$ tuples which are not dominated by any other group of equal size, based on certain notion of group dominance relationship. We consider *aggregate-based* group dominance relationships of interest in practice. In finding skyline groups according to aggregate-based relationships, the major technical challenge is to identify effective anti-monotonic properties for pruning the search space of skyline groups. To this end, we first show that the anti-monotonic property used in the well-known *Apriori* algorithm does not hold for skyline group pruning. Then, we identify two anti-monotonic properties with varying degrees of applicability: *order-specific property* which applies to SUM, MIN, and MAX as well as *weak candidate-generation property* which applies to MIN and MAX only. Experimental results on both real and synthetic datasets verify that the proposed algorithms achieve orders of magnitude performance gain over the baseline method.


## 1. INTRODUCTION

**Motivation:** In online fantasy sports games, gamers manage teams that consist of real-world professional or collegiate athletes. Teams are of equal size and are compared based on the performance of their real-world athletes in real games. Such comparison is often made according to multiple criteria (e.g., points, rebounds, and assists made by teams of NBA players). During the course of real-world sports events (e.g., NBA regular season games), a game server may host hundreds of thousands of leagues, each of which containing multiple teams. Gamers compete by entering into leagues and selecting and managing their teams to achieve better performance than teams of other gamers in the same league.

Fantasy sports has become a booming multi-billion dollar industry. Game servers are hosted by a variety of providers, ranging from major media corporations such as ESPN, Yahoo!, and CBS to small startup companies. The covered sports include all major team sports as well as golf, racing, swimming, wrestling, surfing, bowling, poker, etc. It is estimated that one in five males and 32 million people aged 12 and above in the U.S. and Canada played fantasy sports in 2010 [1]. With the advent of social networks, fantasy sports will see explosive growth, due to its social nature. Its growth has also triggered the invention of non-sports games in recent years, including fantasy congress, fantasy celebrity gossip, fantasy stock, fantasy movie games, and so on.

Gamers spend significant time and money on fantasy sports - on average \$500 per year and 3 hours per week according to a 2006 survey [21]. They pay for entries into games, real-time statistics, news, and other tools. For better outcome in competition, gamers study the previous performance of athletes, investigate news about their current status (such as injuries and performance dip), and predict their performance in future games. The key to success in fantasy games is team formation based on player performance, which takes place both before and during a game season. Typically a gamer enters into a fantasy league by creating a roster of athletes. The fantasy game often runs for a whole season of the real-life sport, during which gamers can rearrange their team rosters.

Having an automatic tool that recommends teams will be desirable to both gamers and game providers. Such a tool will not only save time for gamers but also enhance their competitive edges. Hence gamers may be willing to pay for it. The convenience brought by it will also attract more people to play, which further boosts the profit of game providers. To the best of our knowledge, existing assistance tools provided by real-world fantasy game servers are limited to only individual player ranking.

**Problem Definition:** In this paper we propose and investigate the novel problem of *team recommendation* for fantasy games. The problem is modeled as computing the **skyline groups** of a dataset, in analogy to the traditional skyline tuples problem [6, 7, 9, 11, 12, 18, 23]. Consider a database table $D$ of $n$ tuples $\{t_1, \ldots, t_n\}$ and $m$ numeric attributes $A_1, \ldots, A_m$. For example, in an NBA basketball fantasy game, the table consists of the pool of available NBA players. Each player is represented as a tuple consisting of several statistical categories: points per game, rebounds per game, assists per game, steals per game, blocks per game, etc. The domain of each attribute has an application-specific preference order, with “better” values being preferred over “worse” values. For instance, the natural order of real numbers may be used for attribute points per game, while the exact opposite order may be used for attribute turnovers per game. We refer to any subset of $k$ tuples in the table, i.e., $G : \{t_{i1}, \ldots, t_{ik}\} \subseteq D$, as a $k$*-tuple group*. Our objective is to find, for a given $k$, all $k$-tuple *skyline groups*, i.e., $k$-tuple groups that are not *dominated* by any other $k$-tuple groups.

The notion of dominance between groups is analogous to the dominance relationship between tuples in skyline analysis. A tuple $t_1$ *dominates* $t_2$ if and only if every attribute value of $t_1$ is either better than or equal to the corresponding value of $t_2$ according to the preference order and $t_1$ has better value on at least one attribute. The set of skyline tuples are those tuples that are not dominated by any other tuples in the dataset. Analogously the dominance relationship between two groups of $k$ tuples each is defined by comparing their aggregates. To be more specific, we calculate for each group a single aggregate tuple, whose attribute values are aggregated over the corresponding attribute values of the tuples in the group. The groups are then compared by their aggregate tuples using traditional tuple dominance. While many aggregate functions can be considered, in this paper we focus on three commonly used functions - SUM (i.e, AVG, since groups are of equal size), MIN and MAX. Intuitively, SUM captures the collective strength of a group, while MIN/MAX compares groups by their weakest/strongest member on each attribute.

**Our Goal:** The goal of this paper is not to develop a competitive automated fantasy gamer, which requires solving many challenging problems. For instance, to accurately predict the future performance of an athlete, the automated gamer would have to collect and analyze subtle sports news (e.g., an athlete is not getting along with the team coach and hence may not get much play time). Apparently these issues are well beyond the scope of this paper.

Our goal is not to directly choose a team for a gamer either, since the real-world scenario can be much more complex than any chosen model. Different sports, different game providers, and different leagues may use vastly different scoring systems. Many scoring systems fall into the following three broad categories - (Rotisserie): The score for a gamer's team on an attribute is its ranking position in the league on that attribute. The overall score of the team is the summation of its ranking positions on all relevant attributes; (Fantasy points): The score of a team is calculated by a scoring function, e.g., weighted sum over all attribute values, where the weights can be decided by the league owner; (Head-to-head): A team will face a different opponent team in each week, and the team's score for that week is the number of its winning categories (i.e., attributes) in the head-to-head comparison.

Instead of aiming at any particular sport, game server, or league, our goal is to provide the general approach of finding skyline groups out of the prohibitively large number of possible groups. The (significantly smaller) returned skyline groups will be the input to further (manual or automated) process that will ultimately recommend a team to a gamer. The further process will be tailored for the scoring systems of particular sports, servers, and leagues. Examples of such postprocessing include eyeballing the skyline groups and narrowing down to the gamer's specific requirements, and returning the top-$k$ groups according to specific ranking functions.

In fact, under the aforementioned three categories of scoring systems, the set of skyline groups will always contain at least a team that attains the best result. Suppose a group $G$ is not in the skyline groups, i.e., it is dominated by another group $G'$. In a fantasy points scoring system, as long as the scoring function is *monotonic* on the attributes (which is almost always true in practice), $G$ will have a worse score than $G'$. Hence only skyline groups can possibly attain the highest score. Similarly, it is easy to see that, under Rotisserie and head-to-head systems, $G$ cannot get a better score than $G'$, i.e., $G'$ will be at least equally good. Hence the skyline groups contain at least one best team, although not necessarily all.

The solution developed in this paper may also find applications in other places where the need for team recommendation arises, such as expert finding and crowdsourcing. Consider the task of finding a panel of experts to evaluate a research paper or a proposal. An expert can be modeled as a tuple in the multi-dimensional space defined by the paper's topics, to reflect the expert's strength on these topics. The collective expertise of a panel of experts can thus be modeled as a set of tuples. The goal is to form teams corresponding to strong tuples. Similarly the problem of forming collaborative teams for software development projects can be viewed as finding groups of programmers whose corresponding tuples are strong in the multi-dimensional space of desired skills for the project. This can be extended to the more general context of crowdsourcing tasks to users in a community.

**Technical Challenges:** To find $k$-tuple skyline groups in a table of $n$ tuples, there can be $\binom{n}{k}$ different candidate groups. *How do we compute the skyline groups of $k$ tuples each from all possible groups?* Interestingly, in this paper we demonstrate that our skyline group problem is significantly different from the more traditional skyline tuples problem, to the extent that algorithms for the later (or any simple extensions) are quite inapplicable in solving the former.

A simple solution to the problem is to first list every possible combination of $\binom{n}{k}$ tuples, compute the aggregate tuple for each combination, and then call any traditional skyline tuple algorithm to identify the skyline groups. The main problem with such an approach is the significant computational and storage overhead of having to create this huge intermediate input for the traditional skyline tuple algorithm (i.e., $O(\binom{n}{k})$ for an $n$-tuple input dataset). The skyline groups problem also has another idiosyncrasy that is not shared by the traditional skyline tuples problem. For certain aggregate functions, specifically MAX and MIN, even the output size– i.e., the number of skyline groups produced– while significantly smaller that $\binom{n}{k}$, may be nevertheless too large to explicitly compute and store. To address these two problems, we develop novel techniques, namely *output compression*, *input pruning*, and *search space pruning*. We discuss these briefly in next subsection.

Techniques developed in several prior works are not applicable or fully effective. For example, [13] considers the problem of forming expert teams to solve tasks, but they focus on measuring if the members in a team can effectively collaborate with each other. Zhang et al. [26] studied set preferences where the preference relationships between $k$-subsets of tuples are based on features of $k$-subsets. Their general framework can model many different queries, including our skyline group problem. The optimization techniques for that framework, when customized and applied for our specific problem, becomes equivalent to the input pruning in our solution as well as merging identical tuples. However, the important search space pruning properties and output compression proposed in this paper are specific to our problem, and were not studied before. These ideas bring substantial performance improvement for our specific problem, as the comparison with [26] in Section 4 shall demonstrate. We leave more detailed discussion of related work to Section 5.

**Outline of Technical Results:** For MAX and MIN aggregates, we observe that numerous groups may share the same aggregate tuple. Our approach to compressing the output is to list the distinct aggregate tuples, each representing possibly many skyline groups, but also providing enough additional information so that the actual skyline groups can be reconstructed if required. Interestingly, there is a difference between MIN and MAX in this regard: while the compression for MIN is relatively efficient, the compression for MAX requires solving the NP-Hard *Set Cover Problem* (which fortunately is not an issue in practice, as we shall show in the paper).

Our approach to input pruning is to filter the input tuples and significantly reduce the input size to the search of skyline groups. Our main observation is that if a tuple $t$ is dominated by $k$ or more tuples in the original table, then we can safely exclude $t$ from the input without influencing the distinct aggregate tuples found at the end. We also find that for MAX, we can safely exclude any non-skyline-tuple from the input without influencing the results.

Our final ideas (perhaps, technically the most sophisticated of the paper) are on search space pruning– i.e., instead of enumerating each and every $k$-tuple combination, we quickly exclude from consideration a large number of combinations which are not on the skyline. To enable such candidate pruning, we identify and leverage a number of anti-monotonic properties similar to the one used in the well-known *Apriori* algorithm for frequent itemset mining [2]. However, it is important to emphasize here that the anti-monotonic property leveraged by the Apriori algorithm– i.e., every subset of a group "of interest" (e.g., a group of frequent items or a skyline group) must also be "of interest" itself– *does not hold* for skyline groups defined by SUM, MIN or MAX. Thus, a significant part of our technical contribution is the identification of alternate anti-monotonic properties which serve our algorithms. In particular, we identify two different anti-monotonic properties with varying degrees of applicability: (a) *Order-Specific Anti-Monotonic Property*, a generic anti-monotonic property that applies to SUM, MIN and

MAX, and (b) *Weak Candidate-Generation Property* which applies to MIN and MAX but not SUM. Based on the two properties, we develop algorithms to compute skyline groups. These algorithms iteratively generate larger candidate groups from smaller ones and prune candidate groups by the anti-monotonic properties. For each individual property, a different candidate generation and pruning algorithm is devised. In particular, we develop a dynamic programming based algorithm that leverages the order-specific property, and an iterative algorithm that leverages the weak candidate-generation property.

The contributions of this paper can be summarized as follows.

- We introduce and motivate the novel problem of team recommendation for fantasy games.
- We modeled the problem as computing skyline groups, and discuss the inapplicability of traditional skyline tuple algorithms in solving this problem.
- We develop novel algorithmic techniques for output compression, input pruning, and search space pruning. In particular, search space pruning requires the identification and leveraging of interesting anti-monotonic properties to filter out candidate groups from consideration.
- We run comprehensive experiments on real NBA dataset and synthetic datasets to evaluate the proposed algorithms.

## 2. SKYLINE GROUPS PROBLEM

Table 1 depicts a 5-tuple, 2-attribute database table which we shall use as a running example throughout this section. Figure 1 depicts the five tuples on a 2-dimensional plane defined by the two attributes. The symbols corresponding to MIN and MAX skyline aggregate vectors shall be explained in the later part of the paper.

Consider a database table $D$ of $n$ tuples $\{t_1, \ldots, t_n\}$ and $m$ attributes $A_1, \ldots, A_m$. We refer to any subset of $k$ tuples in the table, i.e., $G : \{t_{i1}, \ldots, t_{ik}\} \subseteq D$, as a *$k$-tuple group*. Our objective is to find, for a given $k$, all $k$-tuple groups "of interest" according to certain application-specific preferences. In particular, we capture such preferences as a combination of total orders for all attributes, where each total order is defined over (all possible values of) an attribute, with "larger" values always preferred over "smaller" values. In our running example, we consider the natural order of real numbers as the preference order for all attributes. According to such total orders of attribute values, our goal is to find a *skyline* of $k$-tuple groups, defined in analogy to the traditional definition of skyline tuples [6].

| | $A_1$ | $A_2$ |
|---|---|---|
| $t_1$ | 3 | 0 |
| $t_2$ | 0 | 3 |
| $t_3$ | 2 | 1 |
| $t_4$ | 2 | 2 |
| $t_5$ | 0 | 2 |

Table 1: Running Example

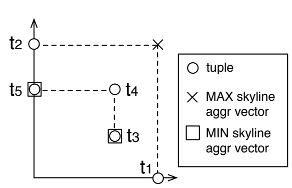



Figure 1: Running Example

In particular, similar to the case of skyline tuples, whether a $k$-tuple group belongs to the skyline or not is determined by the comparison, i.e., the "dominance relationship", between this group and other $k$-tuple groups in the database table. Such a *group comparison function*, when taking two groups $G_1$ and $G_2$ as input, produces (one of) three possible outputs: $G_1$ dominates $G_2$, $G_2$ dominates $G_1$, or neither dominates the other. According to the output of a group comparison function, a $k$-tuple group is a *skyline $k$-tuple group*, or *skyline group* in short (without causing ambiguity), if and only if it is *not dominated* by any other $k$-tuple group in $D$.

One can easily observe from the definition the analogy to skyline tuples: in particular, all existing work on skyline tuples used a uniform definition of "tuple comparison function" - i.e., tuple $t_1$ dominates $t_2$ if and only if every attribute value of $t_1$ is either larger than or equal to the corresponding value of $t_2$ according to the preference order (e.g., in the running example, $t_2$ dominates $t_5$ while neither $t_2$ nor $t_3$ dominates each other).

**Aggregate-based group comparing function:** To define the dominance relationship between two groups of $k$ tuples each, we compare groups by their aggregates. To be more specific:

1. For each input group, compute an *aggregate vector*, i.e., an $m$-dimensional vector with the $i$-th element being an aggregate value of $A_i$ over all $k$ tuples in the group, as the summary of the group.
2. Compare the aggregate vectors for the two groups according to the traditional definition of tuple dominance relationship, and output the result.

One can compute the aggregate vectors for input groups using many aggregate based functions. In this paper we focus on three commonly used aggregate functions: SUM (i.e, AVG, since groups are of equal size), MIN, and MAX. Table 2 shows a sample case for each aggregate function according to the running example.

| | Tuples | SUM | MAX | MIN |
|---|---|---|---|---|
| $G$ | $t_2\langle 0,3\rangle$ $t_3\langle 2,1\rangle$ $t_4\langle 2,2\rangle$ | $\langle 4,6\rangle$ | $\langle 2,3\rangle$ | $\langle 0,1\rangle$ |
| $G'$ | $t_3\langle 2,1\rangle$ $t_4\langle 2,2\rangle$ $t_5\langle 0,2\rangle$ | $\langle 4,5\rangle$ | $\langle 2,2\rangle$ | $\langle 0,1\rangle$ |
| Dominance Relationship | | $G \succ G'$ | $G \succ G'$ | $G = G'$ |

Table 2: Examples of aggregate-based comparison

Our methods allow a mixture of different aggregate functions applied on different attributes. For example, if we use SUM on the first attribute and MAX on the second attribute, then for the two groups in Table 2, the aggregated vectors for $G$ and $G'$ are $\langle 4,3\rangle$ and $\langle 4,2\rangle$, respectively. Hence $G \succ G'$. Our order-specific property (OSM) (Section 3.4.1) can handle arbitrary mixture of all three functions considered in this paper, i.e., SUM, MIN, and MAX, while the weak candidate-generation property (WCM) (Section 3.4.2) can handle any mixture of MIN and MAX. Due to space limitations, we will not further discuss this issue. Instead, we will present experimental results on such mixed functions in Section 4.

**Finding All Skyline Groups:** It is non-trivial to find all skyline groups. In particular, we have the following two observations:

1. A group solely consisting of skyline tuples may *not* be a skyline group. Consider group $G = \{t_1, t_2\}$ in the running example. Note that both $t_1$ and $t_2$ are skyline tuples. Nonetheless, with SUM-based group comparison function, $G$ is dominated by $G' = \{t_3, t_4\}$, as SUM$(G) = \langle 3,3\rangle$ while SUM$(G') = \langle 4,3\rangle$. As such, $G$ is not on the skyline.
2. A group containing non-skyline tuples could be a skyline group, even if there are skyline tuples which are not included in the group. Again consider the running example, this time with $G = \{t_4, t_5\}$ and MIN-based group comparison function. Note that $t_5$ is not on the skyline as it is dominated by $t_2$ and $t_4$. Nonetheless, $G$ (with MIN$(G) = \langle 0,2\rangle$) is actually on the skyline, because the only other groups which can reach $A_2 \geq 2$ in the aggregate vector are $\{t_2, t_4\}$ and $\{t_2, t_5\}$, both of which yield an aggregate vector of $\langle 0,2\rangle$, the same as MIN$(G)$. Thus, $G$ is on the skyline despite containing a non-skyline tuple.

## 3. FINDING SKYLINE GROUPS

In this section, we develop our main ideas for finding skyline $k$-tuple groups (hereafter simply referred to as skyline groups).

## 3.1 Challenges

We start by considering a brute-force approach which first enumerates each possible combination of $k$ tuples in the input table, computes the aggregate vector for each combination, and then invokes a traditional skyline-tuple-search algorithm to find all skyline groups. This approach has two main problems. One is its significant computational overhead, as the input size to the final step - i.e., skyline tuple search - is $\binom{n}{k}$, which can be extremely large for real-world databases.

The other problem is actually on the seemingly natural strategy of listing all skyline groups as the output. The problem here is that, for certain aggregate functions (e.g., MAX and MIN), even the output size - i.e., the number of skyline groups produced - may be too large for practical use. Consider MAX as an example. If a tuple $t$ dominates all other tuples in the input table, then every $k$-tuple combination which contains $t$ is a MAX skyline group - leading to a total of $O(n^k)$ skyline groups. One can see that such a large output size makes the direct-skyline-usage case discussed in Section 1 hardly meaningful, because a human user cannot reasonably go through all skyline groups. Even for indirect-usage applications (e.g., pre-processing for top-$k$ queries), the large number of skyline groups may lead to significant storage overhead.

We address both challenges in this section. We start with developing an output-compression technique that significantly reduces the output size when the number of skyline groups is large, thereby enabling both direct and indirect skyline-usage applications. Then, we consider how to efficiently find skyline groups. In particular, we shall describe two main ideas. One is input pruning - i.e., filtering the input tuples to significantly reduce the input size to the search of skyline groups. The other is search space pruning - i.e., instead of enumerating each and every $k$-tuple combination, we develop techniques to quickly exclude a large number of $k$-tuple combinations (which are not on the skyline) from consideration. Note that the two types of pruning techniques are transparent to each other and therefore can be readily integrated.

## 3.2 Output Compression for MIN, MAX

**Main Idea:** Out of the three aggregate functions we consider in the paper, i.e., SUM, MIN and MAX, the SUM function rarely, if ever, requires output compression. The intuitive reason is that, for any attribute, the SUM aggregate of a skyline group is sensitive to all tuples in the group, while MIN (resp. MAX) aggregate is in general only sensitive to tuples with minimum (resp. maximum) values on certain attributes, making it much more likely for two groups to share the same MIN (resp. MAX) vector. Thus, we focus on MIN and MAX output compression.

A key observation driving our design of output compression is that while the number of MIN and MAX skyline groups may be extremely large, many of these skyline groups share the same *aggregate vector*. Thus, our main idea for compressing skyline groups is to store not all skyline groups, but only the (much fewer) distinct aggregate vectors as well as one skyline group for each distinct vector. One can see that the "sample" skyline groups stored here are useful for team recommendation, as the application only requires one skyline group to be found among many equivalent groups.

**Reconstructing all Skyline Groups for an Aggregated Vector:** While the distinct aggregate vectors and their accompanied (sample) skyline groups may suffice in many cases, there are scenarios where it is necessary to enumerate all skyline groups with the same aggregate vector. For example, a gamer may be willing to spend time investigate all groups equivalent to a particular aggregate vector, and to choose a group after factoring in his knowledge about the involved players or simply his preference over these players. Thus, we now discuss how one can reconstruct all skyline groups from a given aggregate vector, if required.

Consider MIN first. For a given MIN vector $v$, the search process is as simple as finding $\Omega(v)$, the set of all input tuples which dominate or are equal to $v$, because while every $k$-tuple subset of $\Omega(v)$ has an aggregate vector which either dominates or is equal to $v$ (and therefore is a skyline group), any group which contains a tuple outside $\Omega(v)$ must have an aggregate vector dominated by $v$, and therefore cannot be on the skyline. One can see that the time complexity for finding $\Omega(v)$ is $O(n)$. Given $\Omega(v)$, the only additional step needed is to output all $k$-tuple subsets of $\Omega(v)$.

For MAX, the problem is, interestingly, much harder. To understand why, consider each tuple as a set consisting of all attributes for which the tuple reaches the same value as the MAX aggregate vector. One can see that the problem is now transformed to finding all combination of $k$ tuples such that the union of their corresponding sets is the universal set of all attributes - i.e., finding all set covers of size $k$. The NP-hardness of this problem directly follows from the NP-completeness of SET-COVER, seemingly indicating that MAX skyline groups should not be compressed.

Fortunately, despite of the theoretical intractability, finding all skyline groups matching a given MAX aggregate vector $v$ is usually efficient in practice, mainly because the number of tuples which "hit" the MAX attribute values in $v$ - i.e., the input size - is in general extremely small for practical databases. As such, even a brute-force enumeration can be efficiently done, as demonstrated by the experimental results in Section 4.

**Summary:** In the rest part of the paper, we shall focus on the problem of finding all skyline $k$-tuple groups for SUM, and finding all distinct aggregate vectors and their accompanying (sample) skyline groups for MIN and MAX. Then, for MIN and MAX, we shall show through experimental results in Section 4 that the generation of all skyline groups from distinct aggregate vectors is an efficient process for practical databases. Before that, we use the term "skyline search" to refer to the process of finding all distinct aggregate vectors of $k$-tuple groups on the skyline.

Before starting the algorithmic discussions, we would like to make an important observation for the case of MAX when $k \geq m$, where $k$ is the size of a skyline group, and $m$ is the number of attributes. Since it takes at most $m$ tuples to cover the MAX values of all attributes, there is only one distinct (skyline) aggregate vector in this case - which is the vector that takes the MAX value on every attribute. Thus, we only focus on the case of $k < m$ for MAX in the rest of the paper.

## 3.3 Input Pruning

We now consider the pruning of input to skyline group searches - which is originally the set of all $n$ tuples.

An important observation here is that if a tuple $t$ is dominated by $k$ or more tuples in the original table, then we can safely exclude $t$ from the input without influencing the distinct aggregate vectors found at the end. To understand why, suppose that a skyline group $G$ contains a tuple $t$ which is dominated by $h$ ($h \geq k$) tuples. One can see that there is always an input tuple $t'$ which dominates $t$ and is not in $G$. Since $t'$ dominates $t$, the number of tuples which dominate $t'$ must be smaller than $h$. Note that if $t'$ is still dominated by $k$ or more tuples, we can always repeat this process until finding $t' \notin G$ which is dominated by less than $k$ tuples. Now consider the construction of another group $G'$ by replacing $t$ in $G$ with $t'$. For SUM, one can see that $G'$ always dominates $G$, contradicting our assumption that $G$ is a skyline group. Thus, no skyline group under SUM can contain any tuple dominated by $k$ or more tuples.

For MIN and MAX, it is also possible that the aggregate vector of $G'$ is exactly the same as $G$. Even in this case, we can still safely exclude $t$ from the input without influencing the distinct aggregate vectors found at the end. If there are other tuples in $G$ which are dominated by $k$ or more tuples, we can use the same process to remove them all and finally reach a group that (1) features the same aggregate vector as $G$, and (2) has no tuple dominated by $k$ or more other tuples. Thus, we can safely remove from the input all tuples with at least $k$ dominators for all aggregate functions - i.e., SUM, MIN and MAX.

Another observation for input pruning is that for MAX only, we can safely exclude any non-skyline-tuple $t$ from the input without influencing the distinct aggregate vectors. The reason can be explained as follows. Suppose that a skyline group $G$ contains a non-skyline-tuple $t$ which is dominated by another skyline tuple $t'$. If $t' \notin G$, then we can replace $t$ in $G$ with $t'$ to achieve the same (skyline) aggregate vector (because $G$ is a skyline group) - i.e., $t$ can be safely excluded from the input.

If $t' \in G$, then we consider the following two cases: (1) When $G$ does not reach MAX on at least one attribute, say $A_i$. Let $t_j$ be a tuple which reaches MAX on $A_i$. In this case, we can construct $G' = \{G \backslash t\} \cup \{t_j\}$ which is a $k$-tuple group and dominates $G$, contradicting our assumption of $G$ being on the skyline. (2) When $G$ reaches MAX on all attributes. One can see that there must be a group of size at most $k$ which is formed solely by skyline tuples and reaches the same aggregate vector, because any tuple which reaches MAX on at least one attribute must be a skyline tuple[1]. Thus, we can safely exclude $t$ from the input.

While the input size reduction is often significant (as we shall show in the experimental results), we cannot stop here because enumerating all $k$-tuple combinations of even the reduced input may still incur a prohibitive overhead. Thus, we consider search space pruning for further improving the efficiency of skyline search in the next subsection.

## 3.4 Search Space Pruning: Anti-Monotonicity

Our principal idea for search space pruning is to find and leverage a number of *anti-monotonic properties* for skyline search, somewhat in analogy to the Apriori algorithm for frequent itemset mining [2]. Nonetheless, it is important to note that the original anti-monotonic property used by the Apriori algorithm - i.e., every subset of a group "of interest" (e.g., a group of frequent items or a skyline group) must also be "of interest" itself - does not hold for skyline search over SUM, MIN or MAX. In fact, two examples in Section 2 can serve as proof by contradiction, to demonstrate the inapplicability for SUM and MIN, respectively. Specifically, for SUM, skyline 2-tuple group $\{t_3, t_4\}$ contains a non-skyline tuple $t_3$, i.e., a non-skyline 1-tuple group. For MIN, skyline group $\{t_4, t_5\}$ contains a non-skyline tuple $t_5$. For MAX, the inapplicability can be easily observed from the fact that the set of all tuples is always a skyline $n$-tuple group, while many subsets of it are not on their corresponding skylines of equal group size.

Thus, the key challenge here, and our focus in this subsection, is to find those anti-monotonic properties which hold for skyline search. Before presenting the detailed properties, we would like to stress that the main contribution here is not about *proving* these properties, which is often straightforward, but rather about *finding* the right ones which can effectively prune the search space. For this very reason, our following discussions mainly focus on describing the anti-monotonic properties we found and discussing their effectiveness on improving the efficiency of skyline search.

### 3.4.1 *Order-Specific Anti-Monotonic Property*

**Main Idea:** Our first idea for finding an anti-monotonic property is to make a small revision to the classic property used by the apriori algorithm - specifically, by factoring in an order of all tuples in the database. To understand how, consider a skyline $k$-tuple group $G_k$ which violates the apriori property - i.e., a $(k-1)$-tuple subset of it, $G_{k-1} \subseteq G_k$, is not a skyline $(k-1)$-tuple group by itself. We note for this case that all $(k-1)$-tuple groups which dominate $G_{k-1}$ must contain tuple $t_k = G_k \backslash G_{k-1}$. To understand why, suppose that there exists a $(k-1)$-tuple group $G'$ which dominates $G_{k-1}$ but does not contain $t_k$. Then, $G' \cup \{t_k\}$ would always dominate $G_k = G_{k-1} \cup \{t_k\}$, contradicting the skyline assumption for $G_k$. One can see from this example that while a subset of a skyline group may not be on the skyline for the entire input table, it is always a skyline group over a subset of the input table - in particular, $D \backslash \{t_k\}$ in the above example. This observation leads to the following anti-monotonic property:

**Definition 1: Order-Specific Property** An aggregate function $\mathcal{F}$ satisfies the *order-specific anti-monotonic property* if and only if $\forall k$, if a $k$-tuple group $G_k$ with aggregate vector $v$ (i.e., $v = \mathcal{F}(G_k)$) is a skyline group, then for each tuple $t$ in $G_k$, there must exist a set of $k-1$ tuples $G_{k-1} \subseteq D$ with $t \notin G_{k-1}$, such that (1) $G_{k-1}$ is a skyline $(k-1)$-tuple group over an input table $D \backslash t$, and (2) $G_{k-1} \cup \{t\}$ is a skyline $k$-tuple group over the original input table $D$ which satisfies $\mathcal{F}(G_{k-1} \cup \{t\}) = v$. ■

It may be puzzling from the definition where the "order" comes from - we note that it actually lies on the way search-space pruning can be done according to this anti-monotonic property: Consider an arbitrary order of all tuples in the input table, say $\langle t_1, \ldots, t_n \rangle$. For any $r < n$, if we know that an $h$-tuple group $G_h$ ($h \leq r$) is *not* a skyline group over $\{t_1, \ldots, t_r\}$, then we can safely prune from the search space all $k$-tuple groups whose intersection with $\{t_1, \ldots, t_r\}$ is $G_h$ - a reduction of the search space size by $O((n-r)^{k-h})$ - as Definition 1 clearly precludes such groups from being skyline $k$-tuple groups over the original input table. One can see that such a pruning technique considers all tuples in a specific order - hence the name of "order-specific" anti-monotonic property.

**Applicability of Order-Specific Property:** The following theorem shows that the order-based property holds for all three aggregate functions we consider, i.e., SUM, MIN and MAX.

**Theorem 1: (Positive Results)** SUM, MIN and MAX satisfy the order-specific anti-monotonic property. ■

We do not include the proof as it follows directly from Definition 1. We do, however, want to note a limitation of the property. One can see from the above discussion that, to prune based on this order-specific property, one has to compute for every $h \in [k, n-k]$ the aggregate vectors of all skyline 1, 2, ..., $\min(k, h)$-tuple groups over the first $h$ tuples (according to the order), because any of these groups may grow into a skyline $k$-tuple group for the database when latter tuples (again, according to the order) are brought into consideration. Given a large $n$ (i.e., a long order), the order-specific pruning process may incur a significant overhead, as we shall show in Section 4. To address this problem, we consider order-free anti-monotonic properties as follows.

### 3.4.2 *Weak Candidate-Generation Property(MIN, MAX)*

We now describe an "order-free" anti-monotonic property which "loosens" the classic apriori property to one which holds for skyline search. The main idea is that, instead of requiring *every* $(k-1)$-tuple subset of a skyline $k$-tuple group to be a skyline $(k-1)$-tuple group (as in the Apriori property), we consider the following property which only requires *at least one* subset to be on the skyline.

**Definition 2: (Weak Candidate-Generation Property)** An aggregate function $\mathcal{F}$ satisfies the *weak candidate-generation property* if and only if, $\forall k$ and for any aggregate vector $v_k$ of a skyline $k$-tuple group, there must exist an aggregate vector $v_{k-1}$ for a skyline

[1]Note that if there are fewer than $k$ skyline tuples in the input, then we can immediately conclude that any skyline $k$-tuple group must reach MAX on all attributes.

$(k-1)$-tuple group, such that for any $(k-1)$ tuple group $G_{k-1}$ which reaches $v_{k-1}$ (i.e., $\mathcal{F}(G_{k-1}) = v_{k-1}$), there must exist an input tuple $t \notin G_{k-1}$ which makes $G_{k-1} \cup \{t\}$ a skyline $k$-tuple group that reaches $v$ (i.e., $\mathcal{F}(G_{k-1} \cup \{t\}) = v$). ∎

An intuitive way to understand the definition is to consider the case where every skyline group has a distinct aggregate vector. In this case, the weak anti-monotonic property holds when every skyline $k$-tuple group has at least one $(k-1)$-tuple subset being a skyline $(k-1)$-tuple group - a property that is clearly "weaker" than the classic apriori property when being used for pruning, in the sense that it allows many more candidate sets to be generated than directly (and mistakenly) applying the classic property.

In general, this property avoids the pitfall of order-specific property by removing the requirement of enumerating all tuples in order and generating skyline groups for each subset of tuples along the way. Unfortunately, this weak candidate-generation property only holds for MIN and MAX, but not for SUM.

**Theorem 2: (Positive Results: MIN and MAX)** MIN and MAX satisfy the weak anti-monotonic property. ∎

Please refer to the extended version [14] for the proof of this theorem.

**Theorem 3: (Negative Result: SUM)** SUM does not satisfy the weak candidate-generation property. ∎

We would like to note that while the only proof needed here is one counter-example, our studies showed that finding such a counter-example is non-trivial. In particular, the weak candidate-generation property indeed holds when $k \leq 3$, but fails when $k \geq 4$. For $k = 4$, we constructed through MATLAB an 8-tuple, 69-attribute table as a counter-example. We do not include the example (which constitutes a proof) here due to space limitations.

## 3.5 Algorithms OSM and WCM

Based on the order-specific (OSM) and weak candidate generation (WCM) properties derived above, one can easily devise the corresponding algorithms by integrating the anti-monotonic properties with the output compression and input pruning techniques discussed in Sections 3.3 and 3.2, respectively. Due to space limitations, we refer readers to the extended version [14] for the pseudocode and detailed engineering designs of these algorithms. Note that while OSM works for all three aggregate functions (SUM, MIN, MAX) as well as any combination of them, WCM only applies to MIN and MAX.

# 4. EXPERIMENTS

## 4.1 Experimental Setup

All the algorithms were implemented in C++. We executed all our experiments on a Dell PowerEdge 2900 III server running Linux, with dual quad-core Xeon 2.0GHz processors, 2x6MB cache, 8GB RAM, and three 250GB SATA hard drivers in RAID5.

**Datasets**: We collected 512 tuples of NBA players who played in the 2009 regular season. [2] Each tuple has 5 statistics (i.e., 5 attributes) that measure the player's performance. The statistics are points per game (PPG), rebounds per game (RPG), assists per game (APG), steals per game (SPG), and blocks per game (BPG).

To study the scalability of our methods, we also tested synthetic datasets produced by the data generator in [6] with size from 1 to 10 million tuples and 5 attributes. We used the data generator to produce datasets where the attributes are independent (uniform distribution), correlated (multivariate normal distribution), and anti-correlated, respectively.

[2] from `http://www.databasebasketball.com/`.

**Aggregate Functions and Methods Compared**: We investigated the performance of the our algorithms based on the order-specific (OSM) and weak candidate-generation (WCM) properties, respectively. We also compared them with the baseline method (BASELINE), which is a direct adaptation of the general framework in [26] for our specific skyline group problem. (The detailed discussion of [26] is in Section 5.) We tested all aggregate functions discussed in previous sections - SUM, MIN, and MAX.

**Values Measured**: For each applicable combination of aggregate function, method, and parameter values, we measured the execution time needed to find all distinct aggregate vectors and their representative groups, as well as the time to find all skyline groups. Besides execution time, we also measured the total number of candidate groups generated and number of pairwise group (aggregated vector) comparisons in the process. Due to the iterative nature of OSM and WCM, they call the basic skyline function multiple times. Hence, the total number of generated candidate groups is the cumulative sizes of inputs to all skyline function invocations. Furthermore, OSM produces candidate groups by merging two disjoint sets of smaller groups. Here input size was calculated as the summation of the sizes of disjoint sets.

| $n$ | | $k$ = 2 | | | $k$ = 4 | | | $k$ = 6 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | G | S | V | G | S | V | G | S | V |
| 1 M | SUM | | 247 | 247 | | 1654 | 1654 | | 6146 | 6146 |
| | MIN | 3240 | 187 | 141 | 7.6 M | 1914 | 436 | 22.8 B | 12816 | 870 |
| | MAX | | 1982 | 220 | | 529 | 73 | | 0.9 M | 1 |
| 4 M | SUM | | 219 | 219 | | 1610 | 1610 | | 7482 | 7482 |
| | MIN | 4005 | 179 | 131 | 32.7 M | 2182 | 461 | 197 B | 17784 | 1148 |
| | MAX | | 29256 | 274 | | 1888 | 78 | | 3.9 M | 1 |
| 7 M | SUM | | 221 | 221 | | 1374 | 1374 | | 5825 | 5825 |
| | MIN | 3916 | 188 | 134 | 38.6 M | 2193 | 455 | 225 B | 16347 | 1002 |
| | MAX | | 7991 | 323 | | 4851 | 90 | | 13 M | 1 |
| 10 M | SUM | | 210 | 210 | | 1300 | 1300 | | 4487 | 4487 |
| | MIN | 4278 | 183 | 133 | 29.1 M | 2130 | 450 | 231 B | 15442 | 913 |
| | MAX | | 6137 | 224 | | 719 | 63 | | 39 M | 1 |

Table 3: Number of all groups (G), skyline groups (S), and distinct vectors for skyline groups (V), under different $n$, $k$, and functions. Correlated synthetic dataset. M: million, B: billion.

## 4.2 Study of Different Aggregate Functions

**Size of Output under Different Functions:** Table 3 shows, for different database size $n$, group size $k$, and aggregate functions, the number of all possible groups (G), the number of all skyline groups (S), and the number of distinct aggregated vectors (V) for the skyline groups. The table is for correlated synthetic datasets. The observations made on the NBA dataset were similar. We can see that G quickly becomes very large, which indicates that any exhaustive method will suffer due to large space of possible answers.

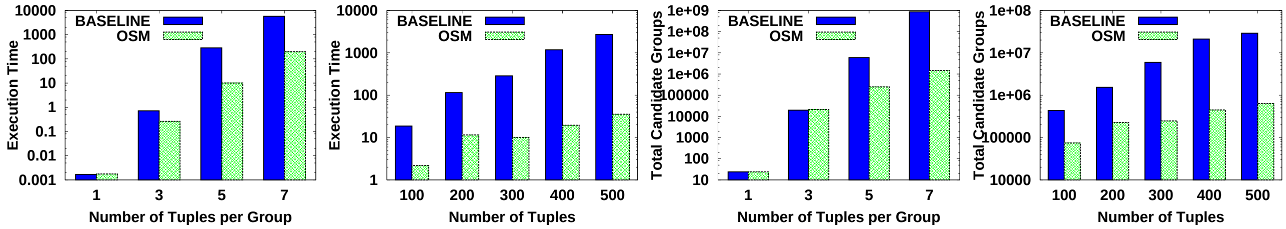
Among the 3 functions, in general SUM has the largest number of distinct vectors and MAX results in the smallest output size. This is due to the intrinsic characteristics of these functions. In computing the aggregated vector for a group, SUM reflects the strength of all group members on each dimension. Hence it is more difficult for a group to dominate or equal to another group on every dimension. On the contrary, MIN (MAX) chooses the lowest (highest) value among group members on each dimension. Hence skyline groups are formed by relatively small number of phenomenal players.

On the other hand, if we compare the numbers of all skyline groups including the equivalent ones, it is rare under SUM to have multiple skyline groups sharing the same aggregated vector. MAX results in much more equivalent groups. Moreover, under MAX, when group size $k$ is larger than or equal to the number of attributes (5 for the datasets), all skyline groups have the same aggregated vector that attains the highest value on every attribute.

**Dealing with a Mixture of Aggregate Functions:** Our methods allow a mixture of different aggregate functions applied on differ-

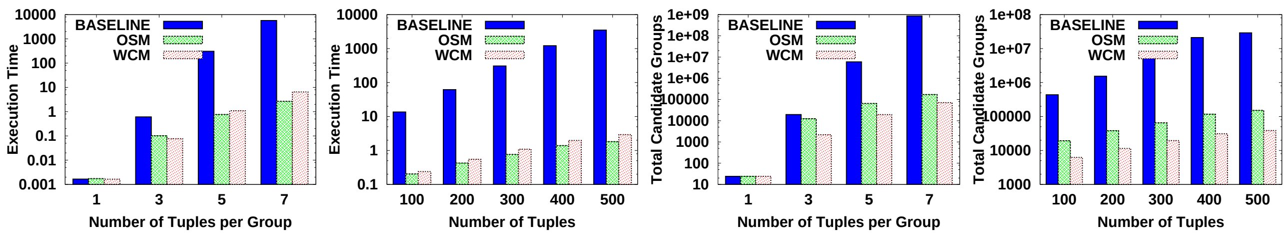
| | | | | | | PPG | RBG | APG | SPG | BPG |
|---|---|---|---|---|---|---|---|---|---|---|
| G1 | Carmelo Anthony | Kobe Bryant | Kevin Durant | LeBron James | Dwyane Wade | 283.2 | 63.4 | 52.2 | 15.2 | 7.6 |
| G2 | Andrew Bogut | Marcus Camby | Monta Ellis | Dwight Howard | Josh Smith | 166.2 | 96.4 | 32.2 | 13.4 | 19.4 |
| G3 | Trevor Ariza | Monta Ellis | Dwyane Wade | Dwight Howard | Josh Smith | 202 | 72.6 | 43.2 | 16.6 | 14 |
| G4 | Carlos Boozer | Baron Davis | LeBron James | Rajon Rondo | Chris Paul | 193.8 | 61.2 | 80.6 | 17.6 | 4.8 |
| G5 | Andrew Bogut | LeBron James | Chris Paul | Dwight Howard | Jason Kidd | 185.8 | 81 | 64 | 14 | 13.8 |

Table 4: Sample skyline groups from 512 players, 5 players per group



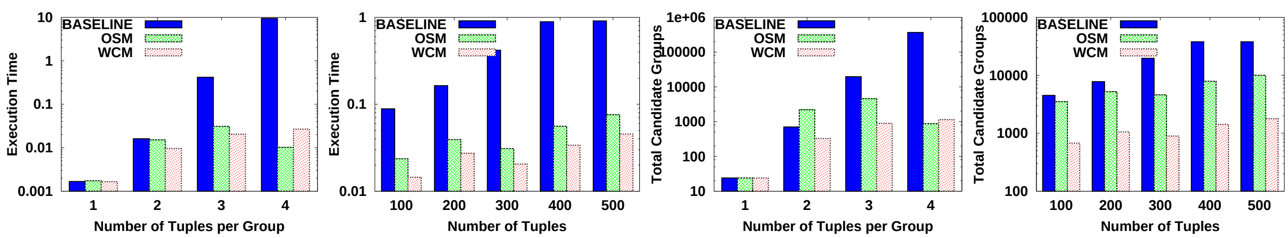

(a) $n = 300$ (b) $k = 5$ (c) $n = 300$ (d) $k = 5$

Figure 2: (a)-(b): Execution time (in seconds, logarithmic scale) and (c)-(d): number of candidate groups (logarithmic scale), SUM



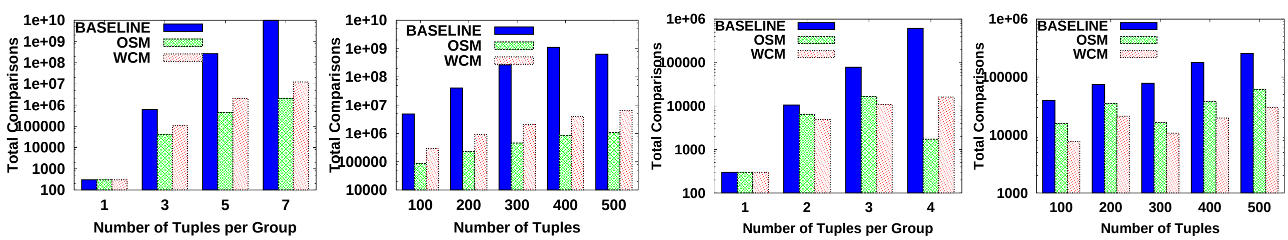

(a) $n = 300$ (b) $k = 5$ (c) $n = 300$ (d) $k = 5$

Figure 3: (a)-(b): Execution time (in seconds, logarithmic scale) and (c)-(d): number of candidate groups (logarithmic scale), MIN




(a) $n = 300$ (b) $k = 3$ (c) $n = 300$ (d) $k = 3$

Figure 4: (a)-(b): Execution time (in seconds, logarithmic scale) and (c)-(d): number of candidate groups (logarithmic scale), MAX



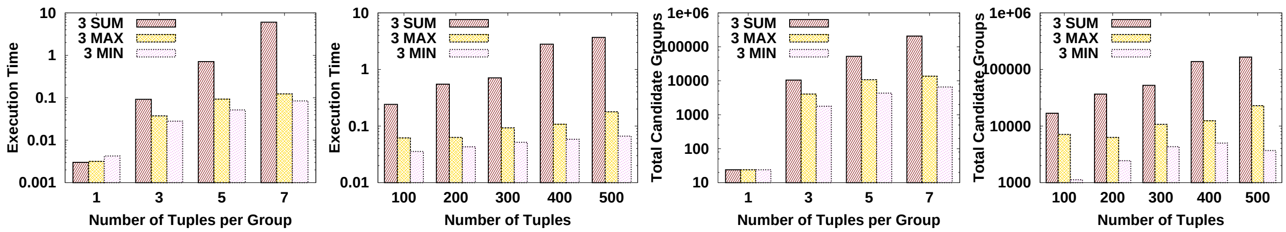

(a) $n = 300$ (b) $k = 5$ (c) $n = 300$ (d) $k = 3$

Figure 5: Number of pairwise group comparisons by different methods for MIN (a)-(b) and MAX (c)-(d)

ent attributes. OSM can handle arbitrary mixture of all three functions considered in this paper, i.e., SUM, MIN, and MAX, while WCM can handle any mixture of MIN and MAX. Figure 6 shows the execution time of OSM over the 5-attribute NBA dataset, for 3 different mixtures of functions. For example, 3SUM means SUM function on the first 3 of the 5 attributes, and MIN and MAX on the remaining 2 attributes. From Figure 6 we can see that SUM function is typically more expensive. This is because output compression has less effect on SUM, under which it is more difficult for a group to dominate other groups, as also explained above.

## 4.3 Experiments on NBA Dataset

**Sample Resultant Skyline Groups:** Table 4 shows several sample skyline 5-tuple groups under aggregate function SUM, from the 512-player NBA dataset. We see from the sample groups that they are formed by elite players and have different strengths. For




(a) $n$ = 300 (b) $k$ = 3 (c) $n$ = 300 (d) $k$ = 3

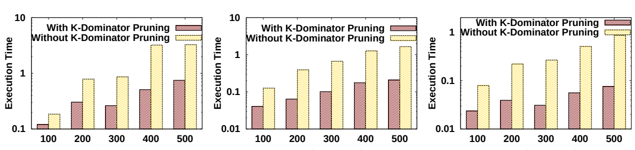
Figure 6: (a)-(b): Execution time (in seconds, log scale) and (c)-(d): number of candidate groups (log scale), mixture of SUM/MAX/MIN

instance, G1 is excellent in scoring (PPG), G2 excels in defense (RBG and BPG), and G3 is a very balanced group that is strong on many aspects although not the best on any dimension.

**Comparison of Various Methods:** Figure 2-4 show the execution time and number of generated candidate groups, by BASELINE/OSM/WCM under all applicable functions (SUM, MIN, MAX), over the NBA dataset. Figure 5 further shows the number of pairwise group (aggregated vector) comparisons performed by these algorithms under MIN and MAX, to offer a closer look at their performance. In sub-figure (a) and (c) of these figures, we fix the size of dataset ($n$) to 300 tuples and vary group size ($k$). In sub-figure (b) and (d) of these figures, we fix the group size ($k$=5 for SUM/MIN and $k$=3 for MAX) and vary dataset size. We observed that OSM/WCM performed substantially (often orders of magnitude in execution time) better than BASELINE. Without the order-specific and weak candidate-generate pruning properties, BASELINE produced much more candidate groups than OSM/WCM did and thus incurred much more pairwise group (aggregated vector) comparisons inside skyline function invocations.

| $n$ | $k$ = 1 | $k$ = 3 | $k$ = 5 | $k$ = 7 |
|---|---|---|---|---|
| 100 | 19 | 31 | 37 | 44 |
| 200 | 22 | 37 | 47 | 57 |
| 300 | 24 | 50 | 61 | 67 |
| 400 | 29 | 62 | 78 | 86 |
| 500 | 30 | 62 | 83 | 94 |

Table 5: Number of tuples that are dominated by less than $k$ tuples




(a) SUM (b) MIN (c) MAX

Figure 7: Effect of input pruning on OSM, $k$ = 3

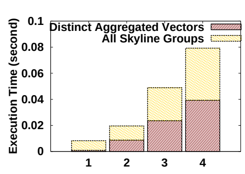
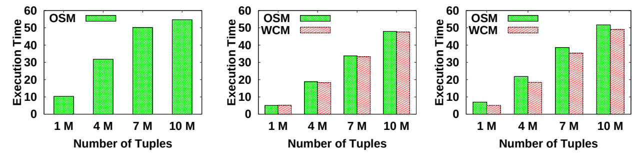


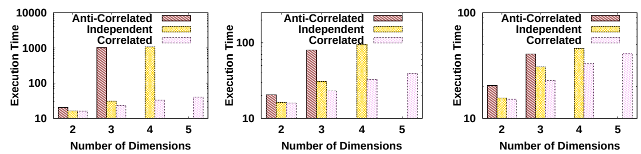

Figure 8: Finding all skyline groups from distinct aggregated vectors for MAX, $n$= 100, OSM

**Effect of Input Pruning:** Input pruning was applied in all the experiments for Figure 2-4. It had a good impact on the performance of all algorithms, since it significantly reduced the size of input. Table 5 shows that, in all considered cases on NBA dataset, less than 100 tuples remained after $k$-dominator tuple pruning was applied. Figure 7 shows the substantial saving on execution time.




(a) SUM (b) MIN (c) MAX

Figure 9: Execution time (in seconds) of OSM/WCM on correlated synthetic dataset with 5 attributes, $k$ = 4




(a) SUM (b) MIN (c) MAX

Figure 10: Execution time (in seconds, logarithmic scale) of OSM on different synthetic datasets, $k$ = 3, $n$ = 10 million

**Search Space Pruning Power of OSM and WCM:** Figure 3, 4 and 5 compare OSM and WCM, in terms of execution time, number of candidate groups produced, and number of pairwise group (aggregated vector) comparisons incurred. We observed that WCM performed better than OSM under MAX but OSM won for MIN on the NBA dataset. With regard to MAX, WCM demonstrated better pruning power in most cases because it resulted in both less candidate groups (Figure 4(c) and 4(d)) and less pairwise group comparisons (Figure 5(c) and 5(d)). With regard to MIN, even though WCM produced less candidate groups (Figure 3(c) and 3(d)), it required more group comparisons (Figure 5(a) and5(b)). Hence it lost in comparison with OSM under MIN for NBA dataset.

**Effect of Output Compression:** Figure 8 shows the cost (in execution time) of postprocessing for obtaining all skyline groups from distinct aggregate vectors, on the NBA dataset, for $n$ = 100, MAX function, and OSM algorithm. We can see that in this configuration finding all skyline groups only doubled the execution time. This verifies that, even though the problem of finding all skyline groups from distinct aggregate vectors is an NP-hard problem, in practice it is usually efficient due to the small number of tuples that can "hit" MAX attribute values, as explained in Section 3.2. As $n$ increases, naturally the cost of postprocessing will also increase. However, in reality we often only produce the equivalent groups for an aggregated vector chosen by the user, instead of for all vectors.

## 4.4 Experiments on Synthetic Datasets

Figure 9 demonstrates the scalability of our algorithm to datasets with 1 to 10 million tuples - OSM/WCM finish within a minute on these large datasets, for $k$=4 and all 3 functions.

The same methods will not be as efficient on independent or even anti-correlated data. One can see from Figure 10 that the execution time on anti-correlated and independent data increases quickly and soon the algorithm cannot finish within reasonable amount of time. (Thus the corresponding bars are not plotted.) This is not surprising - e.g., in anti-correlated dataset, values of a tuple on different attributes are negatively correlated. Hence it is more difficult to find a tuple dominating other tuples. This means input pruning cannot reduce the input size effectively, and OSM/WCM cannot prune many candidates either.

In real fantasy games the performance of athletes on multiple measures may neither be fully correlated nor fully anti-correlated. The attributes often form groups, such as rebounds and blocks, assists and steals in basketball games. The attributes within the same group are correlated, while the ones across different groups tend to be independent or anti-correlated. One direction for our future study is to investigate the performance of our methods on synthetic data following such more realistic correlation patterns.

## 5. RELATED WORK

Since Börzsönyi et al.'s pioneer work [6] on skyline query processing in database systems, skyline query has been intensively studied over the last decade - e.g., on improved algorithms [9, 11], progressive skyline computation [12,18,23], query optimization [7], to name just a few. Other variants of skyline queries have also been studied, including skyline cube [20,25], skyline computation in data streams [16], uncertain databases [19], and low-cardinality domains [17], reverse skyline queries [10,15], spatial skyline queries [22], privacy skyline [8], parallel and distributed computation [5,24], etc.

We apply skyline algorithms over candidate groups to find skyline groups but our methods are orthogonal to specific choices of skyline algorithms. The one we currently implement is a simple nested-loop algorithm, similar to the one in [6].

With regard to skyline groups, the most related work include [4] and [26]. In [4] groups defined by GROUP BY in SQL are compared with each other and groups not dominated by any other are called skyline groups. In our paper, skyline groups can be formed by an arbitrary set of $k$ tuples. Zhang et al. [26] studied a more generic problem which supports comparing two groups with any partial-ordered feature (not just the numeric aggregate functions we focused on in this paper). Nonetheless, the generality also leads to significant overhead on finding skyline groups, as we showed in Section 4, mainly because the important pruning properties (OSM and WCM) and output compression proposed in this paper are specific to our problem, and were not applicable to the settings in [26].

With regard to team formation, the most related work are [13] and [3], both studying the formation of expert teams to solve tasks. [3] differs from ours on two important aspects: (1) [3] assumes a pre-determined scoring function which determines the closeness between a team and a task. We do not have a fixed scoring function, but instead compare groups according to the skyline-based dominance relationship. (2) The goodness of teams is measured by how well they match the given tasks. Our paper chooses groups based on how good they are in an absolute sense - i.e., task-independent. In [13], the skill space is Boolean, i.e., an individual skill is either present (required) in the profile of an expert (task) or not. Instead of deciding on how well teams match tasks, this work focuses on measuring if the members in a team can effectively collaborate with each other, based on information from social networks.

## 6. CONCLUSION

In this paper, we defined a novel problem of team recommendation for fantasy games. The problem is modeled as finding skyline groups in a table. We developed novel algorithmic techniques on output compression, input pruning, and search space pruning to address the problem. In particular, for search space pruning, we identified a number of anti-monotonic properties to efficiently remove non-skyline groups from consideration. Based on the properties, we developed dynamic programming and iterative algorithms for skyline group search. Experimental results on real and synthetic datasets verify that the proposed algorithms achieve orders of magnitude performance gain over the baseline method.

## 7. REFERENCES


[1] Fantasy Sports Participation Sets All-Time Record, Grows Past 32 Million Players. Fantasy Sports Trade Association. www.fsta.org/blog/fsta-press-release/.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, 1994.

[3] A. Anagnostopoulos, L. Becchetti, C. Castillo, A. Gionis, and S. Leonardi. Power in unity: forming teams in large-scale community systems. In *CIKM*, 2010.

[4] S. Antony, P. Wu, D. Agrawal, and A. El Abbadi. Moolap: Towards multi-objective olap. In *ICDE*, 2008.

[5] W.-T. Balke, U. Guntzer, and J. Zheng. Efficient distributed skylining for web information systems. In *EDBT*. 2004.

[6] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, 2001.

[7] S. Chaudhuri, N. Dalvi, and R. Kaushik. Robust cardinality and cost estimation for skyline operator. In *ICDE*, 2006.

[8] B.-C. Chen, K. LeFevre, and R. Ramakrishnan. Privacy skyline: privacy with multidimensional adversarial knowledge. In *VLDB*, 2007.

[9] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE*, 2003.

[10] E. Dellis and B. Seeger. Efficient computation of reverse skyline queries. In *VLDB*, 2007.

[11] P. Godfrey, R. Shipley, and J. Gryz. Maximal vector computation in large data sets. In *VLDB*, 2005.

[12] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: an online algorithm for skyline queries. In *VLDB*, 2002.

[13] T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *KDD*, 2009.

[14] C. Li, N. Zhang, N. Hassan, S. Rajasekaran, and G. Das. "Anything You Can Do, I Can Do Better." – Team Recommendation for Fantasy Games (extended version). http://ranger.uta.edu/~cli/kdd12tr.pdf.

[15] X. Lian and L. Chen. Monochromatic and bichromatic reverse skyline search over uncertain databases. In *SIGMOD*, 2008.

[16] X. Lin, Y. Yuan, W. Wang, and H. Lu. Stabbing the sky: efficient skyline computation over sliding windows. In *ICDE*, 2005.

[17] M. Morse, J. M. Patel, and H. V. Jagadish. Efficient skyline computation over low-cardinality domains. In *VLDB*, 2007.

[18] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *TODS*, 30(1), 2005.

[19] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *VLDB*, 2007.

[20] J. Pei, Y. Yuan, X. Lin, W. Jin, M. Ester, Q. Liu, W. Wang, Y. Tao, J. X. Yu, and Q. Zhang. Towards multidimensional subspace skyline analysis. *TODS*, 31(4), 2006.

[21] L. Prescott. Fantasy Sports Online. http://www.imediaconnection.com/content/9599.asp/.

[22] M. Sharifzadeh and C. Shahabi. The spatial skyline queries. In *VLDB*, 2006.

[23] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *VLDB*, 2001.

[24] P. Wu, C. Zhang, Y. Feng, B. Zhao, D. Agrawal, and A. El Abbadi. Parallelizing skyline queries for scalable distribution. In *EDBT*. 2006.

[25] T. Xia and D. Zhang. Refreshing the sky: the compressed skycube with efficient support for frequent updates. In *SIGMOD*, 2006.

[26] X. Zhang and J. Chomicki. Preference queries over sets. In *ICDE*, 2011.