On Skyline Groups

Chengkai Li
University of Texas at Arlington

Nan Zhang
George Washington University

Naeemul Hassan
University Texas at Arlington

Sundaresan Rajasekaran
George Washington University

Gautam Das
University of Texas at Arlington
Qatar Computing Research Institute

ABSTRACT

We formulate and investigate the novel problem of 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 aggregate-based group dominance relationship. 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 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

In this paper we formulate and investigate the novel problem of computing the *skyline groups* of a dataset. While the traditional skyline tuple problem has been extensively investigated in recent years [5, 6, 8, 10, 12, 17, 21], the skyline group problem surprisingly has not been studied in prior work.

Consider a database table of n tuples and m numeric attributes. The domain of each attribute has an application-specific preference order, with "better" values being preferred over "worse" values. We refer to any subset of k tuples in the table 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 calculating aggregate tuples, in this paper we focus on three distinct functions that are commonly used in database applications – 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.

Many real-world applications require to choose groups of objects. In the booming multi-billion dollar industry of online fantasy sports, gamers compete by forming and managing team rosters of real-world athletes, aiming at outperforming other gamers' teams. They select teams based on prediction of player performance. The teams are compared by aggregated performance of the athletes in real games. For example, consider a table of the pool of available NBA players in a basketball fantasy game. Each player is represented as a tuple consisting of several statistical categories: points per game, rebounds per game, assists per game, etc. The strength of a team is thus captured by the corresponding aggregates of these statistics. Other motivating examples include the applications where the need for choosing groups arises, such as expert finding and crowdsourcing. Consider the task of choosing a panel of experts to evaluate a research paper or a grant 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 is modeled as the aggregate of the corresponding tuples. The goal is to select panels attaining strong aggregates. 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.

The capability of recommending groups is valuable in the abovementioned applications. An attractive property of skyline groups is that a skyline group cannot be dominated by any other group. In contrast, given a non-skyline group, there always exists a better group in the skyline. Hence the skyline groups present those groups that are worth recommending. They become the input to further (manual or automated) process that ultimately recommends one group. Examples of such post-processing include eyeballing the skyline groups, more systematic browsing and visualization of the skyline groups, and filtering and ranking the groups according to user preference. For instance, if groups are ranked by a *monotonic* scoring function on attributes $A_1, \ldots, A_m$, regardless of the specific scoring function, the skyline always contains a group attaining the best score.

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, the skyline group problem is significantly different from the traditional skyline tuple problem, to the extent that algorithms for the later are quite inapplicable in solving the former.

A simple solution to the problem is to first list all $\binom{n}{k}$ groups, compute the aggregate tuple for each group, and then use 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 group problem also has

another idiosyncrasy that is not shared by the skyline tuple 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*.

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 the solution to the NP-Hard Set Cover Problem (which fortunately is not a real 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. Instead of enumerating each and every k-tuple combination, we exclude from consideration a large number of combinations. To enable such candidate pruning, we identify a number of properties inspired by the anti-monotonic property in the well-known *Apriori* algorithm for frequent itemset mining [1]. However, it is important to emphasize here that the anti-monotonic property in Apriori *does not hold* for skyline groups defined by SUM, MIN or MAX. More specifically, a subset of a skyline group may not necessarily be a skyline group itself. 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 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 these properties. For each individual property, a different candidate generation and pruning algorithm is devised. In particular, we develop a dynamic programming algorithm that leverages the order-specific property and an iterative algorithm that leverages the weak candidate-generation property.

We briefly summarize our contributions as follows.

- We motivate and formulate the novel problem of 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, for search space pruning, we identify interesting anti-monotonic properties to filter out candidate groups from consideration.
- We run comprehensive experiments on real and synthetic datasets to evaluate the proposed algorithms.

2. RELATED WORK

Skyline query has been intensively studied over the last decade. Kung et al. [11] first proposed in-memory algorithms to tackle the skyline problem. Börzsönyi et al. [5] was the original work that studied how to process skyline queries in database systems. Since then, this line of research includes proposals of improved algorithms [8, 10], progressive skyline computation [12, 17, 21], query optimization [6], and the investigation of many variants of skyline queries [4, 7, 9, 14–16, 18–20, 22, 23].

With regard to the concept of skyline groups, the most related previous works are [3] and [24]. In [3] the groups are defined by GROUP BY in SQL, while the groups in our work are formed by combinations of k tuples in a tuple set. Zhang et al. [24] studied set preferences where the preference relationships between k-subsets of tuples are based on features of k-subsets. The features are much more general than numeric aggregate functions considered in our work. The preferences given on each individual feature form a partial order over the k-subsets instead of a total order by numeric values. Their general framework can model many different queries, including our skyline group problem. The optimization techniques for that framework, namely the *superpreference* and *M-relation* ideas, when instantiated for our specific problem, are essentially equivalent to input pruning in our solution as well as merging identical tuples. Hence such an instantiation is a baseline solution to our problem. However, the important search space pruning properties (OSM and WCM) and output compression in Section 4 are specific to our problem and were not studied before. These ideas bring substantial performance improvement, as the comparison with the baseline in Section 6 shall demonstrate.

With regard to the problem of forming expert teams to solve tasks, the most related prior works are [13] and [2]. In [2] teams are ranked by a scoring function, while in our case groups are compared by skyline-based dominance relationship. Hence the techniques proposed in [2] are not applicable to our setting. In [13], instead of measuring how well teams match tasks, the focus was on measuring if the members in a team can effectively collaborate with each other, based on information from social networks.

3. SKYLINE GROUP PROBLEM

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 the skyline of k-tuple groups. In particular, 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. The dominance test, 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. 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.

More specifically, groups are compared by their aggregates. Each group is associated with 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. The aggregate vectors can be computed by different aggregate 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. The aggregate vectors for two groups are compared according to the traditional tuple dominance relationship used in all existing work on skyline tuples. Such traditional tuple dominance relationship is defined according to certain application-specific preferences. In particular, such preferences are captured 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. Hence, an aggregate vector v_1 dominates v_2 if and only if every at-

tribute value of v_1 is either larger than or equal to the corresponding value of v_2 according to the preference order.

Table 1 depicts a 5-tuple, 2-attribute 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. We consider the natural order of real numbers as the preference order for all attributes. For instance, t_2 dominates t_5 while neither t_2 nor t_3 dominates each other. Table 2 shows a sample case of comparing two 3-tuple groups for each aggregate function. Figure 1 also shows the symbols corresponding to MIN and MAX aggregate vectors of skyline 2-tuple groups in the running example. For instance, the skyline 2-tuple group under MAX function is $\{t_1, t_2\}$, with aggregate vector $\langle 3, 3\rangle$. The aggregate vectors of skyline 2-tuple groups under MIN are $\langle 2, 1\rangle$ (for group $\{t_3, t_4\}$) and $\langle 0, 2\rangle$ (for groups $\{t_2, t_4\}$, $\{t_2, t_5\}$, $\{t_4, t_5\}$).

	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

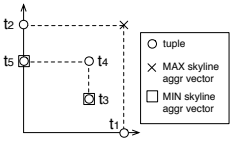


Figure 1: Running Example in 2-d Space

	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. Our order-specific property (OSM) (Section 4.3.1) can handle arbitrary mixture of SUM, MIN, and MAX, while the weak candidate-generation property (WCM) (Section 4.3.2) 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 6.

4. FINDING SKYLINE GROUPS

In this section, we develop our main ideas for finding skyline groups. 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.

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 nevertheless too large to explicitly compute and store. Consider an extreme example under MAX. If a tuple t dominates all other tuples, then every k-tuple combination that contains t is a MAX skyline group – leading to a total of $O(n^k)$ skyline groups. Such a large output size not only leads to significant overhead in computing and storing skyline groups, but also makes post-processing (e.g., ranking and browsing of skyline groups) costly.

Another idea is to consider skyline tuples only. While seemingly intuitive, this idea will not work correctly in general. 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 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 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.

To address these challenges, we develop several techniques, namely *output compression*, *input pruning*, and *search space pruning*. We start with developing an *output compression* technique that significantly reduces the output size when the number of skyline groups is large, thereby enabling more efficient downstream processes that consume the skyline groups. 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 from consideration a large number of combinations. Note that the two types of pruning techniques are transparent to each other and therefore can be readily integrated.

4.1 Output Compression for MIN and MAX

Main Idea: A key observation driving our design of output compression is that while the number of skyline groups may be 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 skyline aggregate vectors (in short *skyline vector*) as well as one skyline group for each skyline vector.

Among 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.

Reconstructing all Skyline Groups for a Skyline Vector: While the distinct skyline vectors and their accompanying (sample) skyline groups may suffice in many cases, a user may be willing to spend time on investigating all groups equivalent to a particular skyline vector, and to choose a group after factoring in her knowledge and preference. Thus, we now discuss how one can reconstruct all skyline groups from a given skyline vector, if required.

Consider MIN first. For a given MIN skyline vector v, the process is as simple as finding $\Omega(v)$, the set of all input tuples which dominate or are equal to v. The reason is as follows. Given any k-tuple subset of $\Omega(v)$, its aggregate vector either dominates or is equal to v, thus it must be a skyline group. On the other hand, any group which contains a tuple outside of $\Omega(v)$ must have an aggregate vector dominated by v, and therefore cannot be in the skyline. The time complexity of a linear scan process in finding $\Omega(v)$ is

$O(n)$. Given $\Omega(v)$, the only additional step needed is to enumerate all k-tuple subsets of $\Omega(v)$.

For MAX, interestingly, the problem is much harder. To understand why, consider each tuple as a set consisting of all attributes for which the tuple reaches the same value as a MAX skyline vector. 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 MAX skyline vector v is usually efficient in practice. This is mainly because the number of tuples that "hit" the MAX attribute values in v – i.e., the input size – is typically small. As such, even a brute-force enumeration can be efficient, as demonstrated by experimental results in Section 6.

Summary: In the rest of the paper, we shall focus on the problem of finding all skyline k-tuple groups for SUM, and finding all distinct skyline vectors and their accompanying (sample) skyline groups for MIN and MAX. We use the term "skyline search" to refer to the process in solving the problem.

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 vector in this case – 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.

4.2 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 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 skyline 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. 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 repeat this process until finding $t' \notin G$ that 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 possible that the aggregate vectors of the above G' and G are exactly the same. Even in this case, we can still safely exclude t from the input without influencing the distinct skyline vectors. 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 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 skyline 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). If $t' \in G$, we can remove t from G without changing the aggregate vector of G. In either way, t can be safely excluded from the input. By repeatedly replacing or removing non-skyline tuples in the above way, we will obtain a group of size at most k that is formed solely by skyline tuples.[1] Padding the group with arbitrary additional tuples to reach size k will result in a group of the same aggregate vector as G.

4.3 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 [1]. Nonetheless, it is important to note that the original anti-monotonic property in 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 3 can serve as proof by contradiction, to demonstrate the inapplicability for SUM and MIN. 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 is to find those anti-monotonic properties that hold for skyline search. We would like to stress that the main contribution here is not about *proving* these properties, 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 and discussing their effectiveness on improving the efficiency of skyline search.

4.3.1 Order-Specific Anti-Monotonic Property

Our first idea is to make a small revision to the classic property in the Apriori algorithm – specifically, by factoring in an order of all tuples. 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. 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 or equal $G_k = G_{k-1} \cup \{t_k\}$, 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

[1] Note that if the resulting group has size smaller than k, then it (and thus G) reaches the maximum values on all attributes. If there are fewer than k skyline tuples in the input, then we can immediately conclude that any skyline k-tuple group must reach the maximum values on all attributes.

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.

The following theorem shows that the order-specific property holds for all three aggregate functions we consider.

Theorem 1: 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. 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 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 6. To address this problem, we consider order-free anti-monotonic properties as follows.

4.3.2 Weak Candidate-Generation Property

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 $(k-1)$-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. The property is clearly "weaker" than the classic (Apriori) anti-monotonic property when being used for pruning, in the sense that it allows 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. However, its limitation is that it only holds for MIN and MAX, but not for SUM.

Theorem 2: MIN and MAX satisfy the weak candidate-generation property. ■

PROOF. We prove the case for MIN by contradiction. The case for MAX can be proved in analogy. Suppose that G_k is a skyline k-tuple group which satisfies $\mathcal{F}(G_k)$=v_k, but no v_{k-1} according to the definition exists. Consider an arbitrary $(k-1)$-tuple subset of G_k, denoted by G. Let t_1 be the other tuple not included in G, i.e., t_1=$G_k \backslash G$. Since G is not a skyline $(k-1)$-tuple group, there must exist another $(k-1)$-tuple group G' that dominates G. We consider the following two cases respectively: (1) $t_1 \notin G'$, and (2) $t_1 \in G'$.

In Case (1), either $G' \cup \{t_1\}$ dominates G_k, which leads to contradiction because G_k is on the skyline, or $\mathcal{F}(G' \cup \{t_1\})$=$\mathcal{F}(G_k)$, which leads to contradiction as well because G' and t_1 would exactly satisfy the requirement of weak anti-monotonic property.

For Case (2), note that since G' and G are of equal size, there must exist at least one tuple in G which is not in G'. Let t_2 be such a tuple. Consider $G' \cup \{t_2\}$. Since $t_2 \in G$, every attribute value in $\mathcal{F}(G' \cup \{t_2\})$ is still greater than or equal to the corresponding value in $\mathcal{F}(G)$, which is in turn greater than or equal to that in $\mathcal{F}(G)$. Thus, we again reach the conclusion that either $G' \cup \{t_2\}$ dominates G_k or $\mathcal{F}(G' \cup \{t_2\})$=$\mathcal{F}(G_k)$ – both lead to contradiction for the same reasons in Case (1).

Theorem 3: 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 study 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.

5. ALGORITHMS

In this section, we develop skyline group search algorithms based on the anti-monotonic properties derived in Section 4.

5.1 Dynamic Programming Algorithm Based on Order-Specific Property

Consider an arbitrary[2] order of the n tuples in the input table, denoted by $t_1, \ldots, t_n$. Let T_r be the set of the first r according to this order - i.e., T_r=$\{t_1, \ldots, t_r\}$. Let Sky_k^r be set of all skyline k-tuple groups with regard to T_r - i.e., each group in Sky_k^r is not dominated by any other k-tuple group consisting solely of tuples in T_r. One can see that our original problem can be considered as finding Sky_k^n. We now develop a dynamic programming algorithm which finds Sky_k^n by recursively solving the "smaller" problems of finding Sky_k^{n-1} and Sky_{k-1}^{n-1}, etc.

The algorithm is based on the following idea - All skyline k-tuple groups in Sky_k^n can be partitioned into two disjoint sets S_1 and S_2 ($Sky_k^n \equiv S1 \cup S2$ and $S1 \cap S2 = \phi$) according to whether a group contains t_n or not. In particular, $S1 = \{G | G \in Sky_k^n, t_n \notin G\}$ and $S2 = \{G | G \in Sky_k^n, t_n \in G\}$. One can that there must be $S1 \subseteq Sky_k^{n-1}$. On the other hand, $S2$ is subsumed by a set of groups that can be expanded from Sky_{k-1}^{n-1}, the skyline (k-1)-tuple groups with regard to T_{n-1}. More specifically, given a skyline k-tuple group that contains t_n, if we remove t_n from it, then the resulting group belongs to Sky_{k-1}^{n-1}. These two properties are formally presented as follows. We omit the fairly simple proof. Note that Proposition 2 can be directly derived from Theorem 1.

Proposition 1: Given $G \in Sky_k^n$, if $t_n \notin G$, then $G \in Sky_k^{n-1}$. ■

Proposition 2: Given $G \in Sky_k^n$, if $t_n \in G$, then $G \backslash \{t_n\} \in Sky_{k-1}^{n-1}$. ■

We further explain the dynamic programming algorithm by referring to the outline in Algorithm 1. The idea is also intuitively illustrated in Figure 2. The function $sky\_group(k, n)$ is for finding Sky_k^n. It first recursively computes Sky_{k-1}^{n-1} (Line 7). By adding t_n into each group in Sky_{k-1}^{n-1} (Line 8-10), the algorithm

[2]We consider a random order in the experimental studies of this paper and leave the problem of finding an optimal order (in terms of efficiency) to future work.

Algorithm 1: $sky\_group(k, n)$: Dynamic programming algorithm based on order-specific property

Input: n: input tuples T_n={$t_1, \ldots, t_n$}; k: group size; $k \leq n$
Output: Sky_k^n: skyline k-tuple groups among T_n

1 **if** Sky_k^n is computed **then**
2 **return** Sky_k^n;
3 **if** k == 1 **then**
4 $S2^+ \leftarrow \{\{t_n\}\}$;
5 **else**
6 $S2^+ \leftarrow \phi$;
7 $Sky_{k-1}^{n-1} \leftarrow sky\_group(k\text{-}1, n\text{-}1)$;
8 **foreach** group $G \in Sky_{k-1}^{n-1}$ **do**
9 $candidate\_group \leftarrow G \cup \{t_n\}$;
10 $S2^+ \leftarrow S2^+ \cup \{candidate\_group\}$;
11 **if** $k < n$ **then**
12 Sky_k^{n-1} (i.e., $S1^+$) $\leftarrow sky\_group(k, n\text{-}1)$;
13 **else**
14 $S1^+ \leftarrow \phi$;
15 $C_k^n \leftarrow S1^+ \cup S2^+$;
16 $Sky_k^n \leftarrow skyline(C_k^n)$;
17 **return** Sky_k^n;

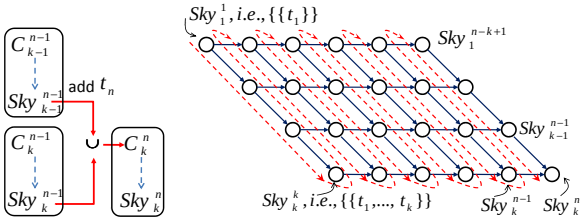


Figure 2: (a) Calculate Sky_k^n from Sky_{k-1}^{n-1} and Sky_k^{n-1}; (b) Overall dynamic programming algorithm for calculating Sky_k^n.

obtains a superset of the aforementioned $S2$, according to Proposition 2. We denote this superset $S2^+$. By recursively calling the sky_group function (Line 12), it further computes Sky_k^{n-1}, which is a superset of the aforementioned $S1$, according to Proposition 1. We also denote Sky_k^{n-1} by $S1^+$. $S1^+$ and $S2^+$ thus contain all necessary candidate groups for Sky_k^n. Thus, the skyline over candidate groups (C_k^n=$S1^+ \cup S2^+$, Line 15) is guaranteed to be equal to Sky_k^n. Existing skyline query algorithms (e.g., [5, 8, 10]) can be applied for this purpose. Hence we use $skyline()$ to refer to such algorithms (Line 16). These algorithms involve comparing groups based on the dominance relationship between groups by their aggregated vectors. The number of candidate groups considered ($|S1^+ \cup S2^+|$) can potentially be much smaller than the number of all possible groups formed by all tuples, i.e., $\binom{n}{k}$.

Note that Sky_k^n is needed in calculating both Sky_k^{n+1} and Sky_{k+1}^{n+1}. The algorithm recursively calls $sky\_group(k, n)$ inside sky_group $(k, n+1)$, to compute and memoize Sky_k^n. Later it will call $sky\_group(k, n)$ again inside $sky\_group(k+1, n+1)$. This time Sky_k^n is not recomputed. Instead, the memorized result is directly used (Line 1). Hence it is a dynamic programming algorithm. The sequence of real calculation of Sky_1^1, Sky_2^2, ..., Sky_k^n is shown by the dashed directed lines in Figure 2(b).

5.2 Iterative Algorithm Based on Weak Candidate-Generation Property

The idea of weak candidate-generation property (Definition 2) can be summarized as follows - Given a skyline group G and any i, at least one i-tuple sub-group of G must be a skyline i-tuple group. Based on this property, Algorithm 2 iteratively generates candidate i-tuple groups by adding new tuples into skyline $(i-1)$-tuple groups (Line 6-12) and applies skyline algorithm over these candidates to find skyline i-tuple groups (Line 14). At every step of iteration, the algorithm only needs to generate i-tuple candidates by extending skyline $(i-1)$-tuple groups instead of all $(i-1)$-tuple groups. Hence it effectively prunes candidate groups by generation.

Algorithm 2: $sky\_group(k, n)$: Iterative algorithm based on weak candidate-generation property

Input: n: input tuples T_n={$t_1, \ldots, t_n$}; k: group size; $k \leq n$
Output: Sky_k: skyline k-tuple groups among T_n

1 $C_1 \leftarrow T_n$;
2 $Sky_1 \leftarrow skyline(C_1)$;
3 **for** $i \leftarrow 2$ **to** k **do**
4 //generate candidate i-tuple groups C_i from skyline $i-1$-tuple groups Sky_{i-1}.
5 $C_i \leftarrow \phi$;
6 **foreach** $G \in Sky_{i-1}$ **do**
7 **foreach** $t \in T_n$ **do**
8 //generate candidate group
9 **if** $t \notin G$ **then**
10 $G' \leftarrow G \cup \{t\}$;
11 **if** $G' \notin C_i$ **then**
12 $C_i \leftarrow C_i \cup \{G'\}$;
13 //generate skyline i-tuple groups Sky_i based on candidates C_i
14 $Sky_i \leftarrow skyline(C_i)$;
15 **return** Sky_k

Algorithm 3: Finding skyline groups with identical aggregated vectors (MIN function)

Input: input tuples R; k: group size; $k < |R|$
Output: Sky: skyline k-tuple groups for R

1 $Sky \leftarrow \phi$;
2 $T \leftarrow$ remove k-dominator tuples from R;
3 $n \leftarrow |T|$; /* number the tuples in T as t_1, ..., t_n */
4 $Sky_k \leftarrow sky\_group(k, n)$; /* Algorithm 1 or Algorithm 2 */
5 **foreach** *skyline k-tuple group* $G \in Sky_k$ **do**
6 $R_G \leftarrow$ the set of tuples in R that dominate or are equivalent to the aggregated vector of G;
7 **foreach** *k-combination* G' *of tuples in* R_G **do**
8 $Sky \leftarrow Sky \cup \{G'\}$;
9 **return** Sky;

5.3 From Distinct Vectors to Equivalent Skyline Groups

For MIN and MAX, even the output size - i.e., the number of skyline groups produced - may be too large to explicitly compute and store. As discussed in Section 4.1, for output compression, we only need to retain one representative skyline group for each distinct aggregated vector. To be more specific, it is sufficient for Sky_k^n in Algorithm 1 and Sky_k in Algorithm 2 to contain one representative group for each distinct aggregated vector of k-tuple groups. It can be easily achieved by a simple modification of the skyline algorithm at Line 16 of Algorithm 1 and Line 14 of Algorithm 2. Whenever a candidate group is compared with current groups in the skyline, we prune it if it is equivalent to some existing group. This will further reduce the size of candidate groups and the number of group comparisons in succeeding iterations.

For input pruning, in the case of SUM and MIN, we remove all tuples dominated by at least k others. In the case of MAX, we remove all tuples not on the skyline. We showed in Section 4.2 that such input pruning techniques are safe - i.e., we will still obtain all distinct vectors and their representatives.

As discussed in Section 4.1, although in many cases distinct vectors and their representative groups suffice, a user may request all

Algorithm 4: Finding skyline groups with identical aggregated vectors (MAX function)

```
   Input: input tuples R; k: group size; k < |R|
   Output: Sky: skyline k-tuple groups among R
1  Sky ← φ;
2  T ← remove k-dominator tuples from R;
3  n ← |T|; /* number the tuples in T as t_1, ..., t_n */
4  Sky_k ← sky_group(k, n); /* Algorithm 1 or Algorithm 2 */
5  foreach skyline k-tuple group G ∈ Sky_k do
6      v ← the aggregated vector of G
7      candidate_group ← φ;
8      i ← 1;
9      p[1] ← null;
10     while i > 0 do
11         /* Note that it is fine to select a tuple multiple times because a tuple
           can get the same value as v on multiple dimensions. */
12         candidate_group ← candidate_group \ {p[i]};
13         p[i] ← get the next tuple in R that has v's value on the ith dimension;
14         if p[i] == null then
15             i ← i−1;
16             continue;
17         candidate_group ← candidate_group ∪ {p[i]};
18         if |candidate_group| > k then
19             continue;
20         if i==d then
21             /* d is the number of dimensions. */
               k' ← k − |candidate_group|;
22             if k'==0 then
23                 Sky ← Sky ∪ {candidate_group};
24             else
25                 R' ← R \ candidate_group;
26                 foreach k'-tuple combination G' among the tuples in R'
                   do
27                     Sky ← Sky ∪ {candidate_group ∪ G'};
28         else
29             i ← i + 1;
30             p[i] ← null;
31 return Sky;
```

skyline groups equivalent to a particular aggregated vector, for applying further criteria in choosing a group. To return such equivalent groups, various postprocessing steps are required, due to output compression and input pruning. Below we discuss such postprocessing for individual functions.

Note that the same Algorithm 1 and 2 work if we do not apply output compression and input pruning. However, even if our application is to ultimately find all skyline groups, it is still beneficial to apply these two techniques and use postprocessing steps to find all skyline groups. Output compression and input pruning together not only reduce the output size, but also save computational cost by allowing the algorithms to deal with smaller input and intermediate results. In Section 6 we present experimental results to compare the execution time of our methods with and without k-dominator tuple pruning. The results verify the benefit of applying this pruning technique regardless of the ultimate output– representative groups for all distinct aggregated vectors or all skyline groups.

SUM: No postprocessing is necessary for SUM. First, a k-dominator tuple cannot appear in any skyline k-tuple group, as discussed in Section 4.2. Thus, input pruning will not trigger postprocessing for SUM. Second, if the ultimate goal is to fetch all skyline groups, output compression should not be applied, because there is no effective way of reconstructing skyline groups from distinct aggregated vectors. In Line 16 of Algorithm 1, all skyline i-tuple groups should be retained, without applying the aforementioned simple modification that removes equivalent groups. Note that SUM only satisfies the order-specific property. Thus, only Algorithm 1 applies.

MIN: Two factors contribute to the need for postprocessing. First, the pruned k-dominator tuples may appear in skyline groups. Second, the aforementioned equivalent group removal performed at Line 16 of Algorithm 1 and Line 14 of Algorithm 2 will only keep one representative for each distinct aggregated vector. Note that both algorithms are applicable to MIN since MIN satisfies both order-specific and weak candidate-generation properties. At the end of both algorithms, we obtain Sky_k, which contains representatives of all distinct aggregated vectors, but not necessarily all skyline k-tuple groups. To generate all skyline groups from Sky_k for MIN, we follow Algorithm 3. For each representative group, we find all the tuples that dominate or are equal to its aggregated vector. Any $k-$combination of these tuples is a skyline k-tuple group. This is based on the results from Section 4.1.

MAX: Algorithms 1 and 2 are both applicable to MAX. Similar to MIN, MAX needs postprocessing due to both input pruning and output compression. We thus devise Algorithm 4 to produce all skyline groups from representative groups.

For each representative group G that is found by Algorithms 1 and 2, Algorithm 4 uses a backtracking process to find all skyline groups that are equivalent to G. Denote the aggregated vector for G as v. On each dimension, we maintain a list of tuples from R (all input tuples to be considered) that attain v's value on that dimension. We use the backtracking algorithm to enumerate all possible groups of the tuples from these lists, such that the groups have the same aggregated vector v and have less than or equal to k tuples. If a group has less than k tuples, it means there can be some "free" tuples. Any combination of other tuples will complement this group to form a skyline k-tuple group (Line 25-27).

A special case for MAX function is when there is only one distinct aggregated vector, i.e., all skyline k-tuple groups reach the highest possible value on every dimension. In Algorithms 1 and 2, whenever an i-tuple candidate group ($i \leq k$) is generated, we test if this group attains the highest possible value on every attribute. If so, we has already found the aggregated vector for all skyline groups. Using that vector, we either find one representative group or all skyline groups, by a backtracking process that is essentially the same as Algorithm 4. We omit the details.

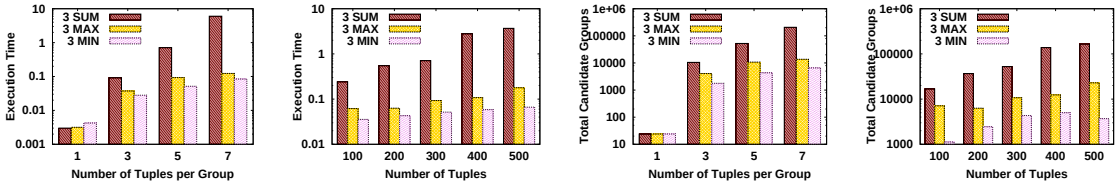
6. EXPERIMENTS

The algorithms were implemented in C++. We executed all experiments on a Dell PowerEdge 2900 III server running Linux kernel 2.6.27-7, with dual quad-core Xeon 2.0GHz processors, 2x6MB cache, 8GB RAM, and three 250GB SATA HDs in RAID5.

Datasets: We collected 512 tuples of NBA players who had played in the 2009 regular season. The tuple of each player 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). Players and groups of players are compared by these statistics and their aggregates.

To study the scalability of our methods, we also experimented with synthetic datasets produced by the data generator in [5]. The datasets have 1 to 10 million tuples, on 5 attributes. The data generator allows us to produce datasets where the attributes are correlated, independent, and anti-correlated. In independent datasets, the attribute values of a tuple were generated by a uniform distribution. In correlated datasets, attribute values were generated using normal distributions. Anti-correlated datasets were generated by a more complex procedure, which involves adding and subtracting values from otherwise uniformly distributed attribute values.

Aggregate Functions and Methods Compared: We investigated the performance of the two algorithms discussed in Section 5, namely the algorithms based on order-specific property (OSM) and weak candidate-generation property (WCM). We also compared



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

Figure 3: (a)-(b): Execution time (in seconds, log scale) and (c)-(d): number of candidate groups (log scale), mixture of SUM/MAX/MIN

these methods with the baseline method (BASELINE), which is a direct adaptation of the general framework in [24] for our skyline group problem. (The detailed discussion of [24] is in Section 2.) We executed these methods for the aggregate functions discussed in previous sections– SUM, MIN, and MAX.

Parameters: We ran our experiments under combinations of two parameter values, which are number of tuples, i.e., dataset size (n) and number of tuples per group, i.e., group size (k).

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	4×10^{11}	247	247	4×10^{22}	1654	1654	1×10^{33}	6146	6146
	MIN		187	141		1914	436		12816	870
	MAX		368	220		147	73		2.9 M	1
4 M	SUM	8×10^{12}	219	219	1×10^{25}	1610	1610	6×10^{36}	7482	7482
	MIN		179	131		2182	461		17784	1148
	MAX		396	274		164	78		11 M	1
7 M	SUM	2×10^{13}	221	221	1×10^{26}	1374	1374	2×10^{38}	5825	5825
	MIN		188	134		2193	455		16347	1002
	MAX		552	323		354	90		55 M	1
10 M	SUM	4×10^{13}	210	210	4×10^{26}	1300	1300	1×10^{39}	4487	4487
	MIN		183	133		2130	450		15442	913
	MAX		402	224		968	63		0.8 B	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.

6.1 Study of Different Aggregate Functions

Size of Output under Different Functions: Table 3 shows, for different n, k, and aggregate functions, the number of all possible groups (G), the number of all skyline groups (S), and the number of distinct aggregate vectors (V) for the skyline groups. The table is for correlated synthetic datasets. The observations made on the NBA dataset were similar. It can be seen that G quickly becomes very large, which indicates that any exhaustive method will suffer due to the large space of possible answers. We also want to point out that the number of skyline vectors (V) can be large (e.g., under k=6). As discussed in Section 1, these distinct vectors become the input to further post-processing such as filtering, ranking and browsing. When a particular skyline vector is chosen by a user, the equivalent skyline groups corresponding to the vector can be generated if requested.

Among the 3 functions, in general SUM has the largest number of skyline vectors and MAX results in the smallest output size. This is due to the intrinsic characteristics of these functions. In computing the aggregate 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. In contrast, MIN (MAX) chooses the lowest (highest) value among group members on each dimension. Hence skyline groups are formed by relatively small number of extremal tuples.

On the other hand, if we compare the sizes of all skyline groups including the equivalent ones, it is rare under SUM to have multiple skyline groups sharing the same aggregate 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 aggregate 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 different attributes. OSM can handle arbitrary mixture of SUM, MIN, and MAX, while WCM can handle any mixture of MIN and MAX. Figure 3 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 3 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.

6.2 Experiments on NBA Dataset

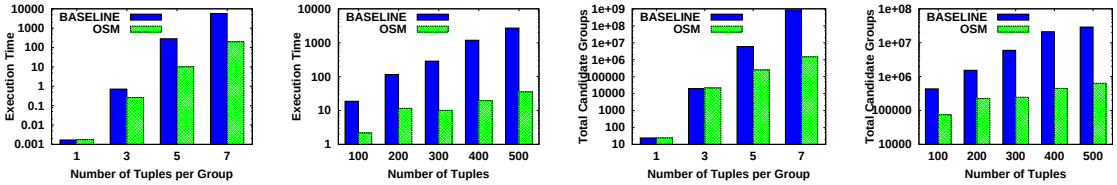
Sample Resultant Skyline Groups: Table 4 shows several sample skyline 5-tuple groups under aggregate function SUM, from the 512-player NBA dataset. We see from the sample groups that they are formed by elite players and have different strengths. For instance, G1 is excellent in scoring (PPG), G2 excels 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 4-6 show the execution time and number of generated candidate groups, by BASELINE/OSM/WCM under all applicable functions, over the NBA dataset. Figure 7 further shows the number of pairwise group (aggregate vector) comparisons performed by these algorithms under MIN and MAX. 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 (aggregate vector) comparisons inside skyline function invocations.

Effect of Input Pruning: Input pruning was applied in all the experiments for Figure 4-6. It had a good impact on the perfor-

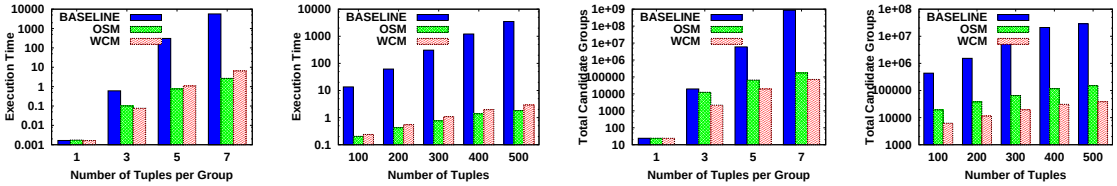
						PPG	RBG	APG	SPG	BPG
G1	Carmelo Anthony	Kobe Bryant	Kevin Durant	LeBron James	Dwyane Wade	283.2	63.4	52.2	15.2	7.6
G2	Andrew Bogut	Marcus Camby	Monta Ellis	Dwight Howard	Josh Smith	166.2	96.4	32.2	13.4	19.4
G3	Trevor Ariza	Monta Ellis	Dwyane Wade	Dwight Howard	Josh Smith	202	72.6	43.2	16.6	14
G4	Carlos 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



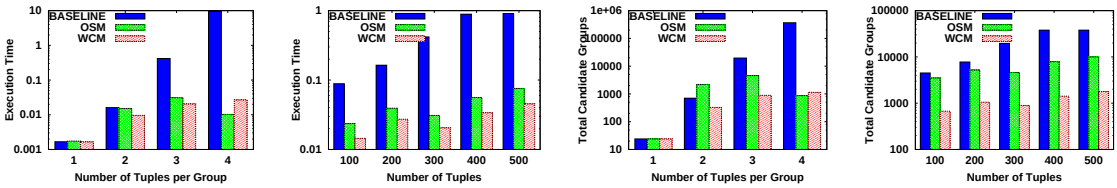
(a) n = 300 (b) k = 5 (c) n = 300 (d) k = 5

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



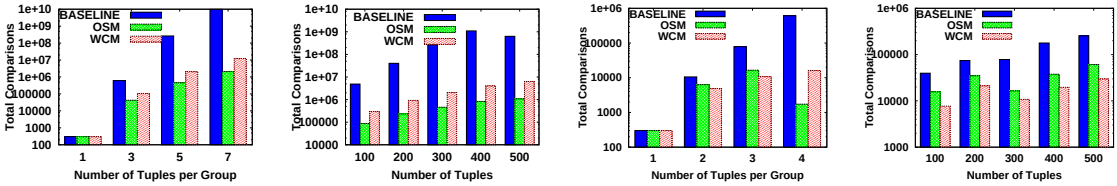
(a) n = 300 (b) k = 5 (c) n = 300 (d) k = 5

Figure 5: (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 6: (a)-(b): Execution time (in seconds, logarithmic scale) and (c)-(d): number of candidate groups (logarithmic scale), MAX

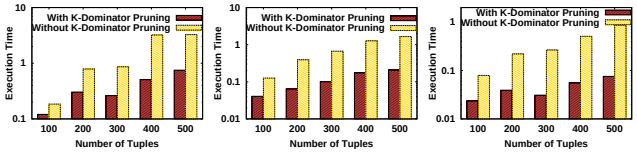


(a) n = 300 (b) k = 5 (c) n = 300 (d) k = 3

Figure 7: Number of pairwise group comparisons by different methods for MIN (a)-(b) and MAX (c)-(d)

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 8: Effect of input pruning on OSM, k = 3

mance 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 8 shows that substantial saving on execution time was achieved for all functions.

Search Space Pruning Power of OSM and WCM: Figure 5, 6 and 7 compare OSM and WCM, in terms of execution time, number of candidate groups produced, and number of pairwise group (aggregate 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 6(c) and 6(d)) and less pairwise group comparisons (Figure 7(c) and 7(d)). With regard to MIN, even though

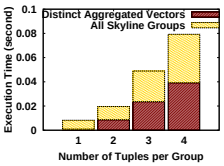
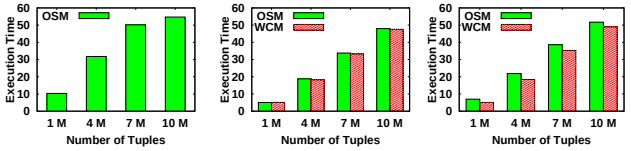
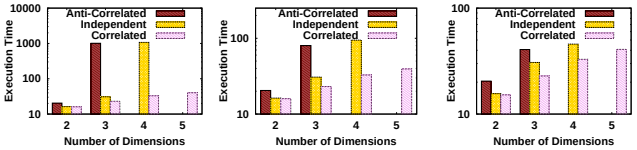


Figure 9: Finding all skyline groups for MAX, n= 100, OSM



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

Figure 10: Execution time (in seconds) of OSM/WCM on correlated synthetic dataset with 5 attributes, k = 4



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

Figure 11: Execution time (in seconds, logarithmic scale) of OSM on different synthetic datasets, k = 3, n = 10 million

WCM produced less candidate groups (Figure 5(c) and 5(d)), it required more group comparisons (Figure 7(a) and7(b)). Hence it lost in comparison with OSM under MIN for NBA dataset.

Effect of Output Compression: Figure 9 shows the cost (in execution time) of post-processing for obtaining all skyline groups from distinct skyline 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 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 4.1. As n increases, naturally the cost of post-processing will also increase. However, in reality we may only need to produce the equivalent groups for a skyline vector chosen by the user, instead of for all skyline vectors.

6.3 Experiments on Synthetic Datasets

To show the scalability of our methods, we experimented on the synthetic datasets with 1 to 10 million tuples. In Figure 10, we see that OSM/WCM can 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. Figure 11 shows the performance of OSM on three different datasets of equal cardinality, under different number of attributes. We see 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. 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 in such a dataset cannot reduce the input size effectively, and OSM/WCM cannot prune many candidates either. Attributes in real datasets 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.

7. CONCLUSION

We proposed the novel problem of finding skyline groups which lends itself to many real-world applications. We developed novel algorithmic techniques on output compression, input pruning, and search space pruning to address the problem. 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.

8. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, 1994.
[2] A. Anagnostopoulos, L. Becchetti, C. Castillo, A. Gionis, and S. Leonardi. Power in unity: forming teams in large-scale community systems. In *CIKM*, 2010.
[3] S. Antony, P. Wu, D. Agrawal, and A. El Abbadi. Moolap: Towards multi-objective olap. In *ICDE*, 2008.
[4] W.-T. Balke, U. Guntzer, and J. Zheng. Efficient distributed skylining for web information systems. In *EDBT*. 2004.
[5] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, 2001.
[6] S. Chaudhuri, N. Dalvi, and R. Kaushik. Robust cardinality and cost estimation for skyline operator. In *ICDE*, 2006.
[7] B.-C. Chen, K. LeFevre, and R. Ramakrishnan. Privacy skyline: privacy with multidimensional adversarial knowledge. In *VLDB*, 2007.
[8] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE*, 2003.
[9] E. Dellis and B. Seeger. Efficient computation of reverse skyline queries. In *VLDB*, 2007.
[10] P. Godfrey, R. Shipley, and J. Gryz. Maximal vector computation in large data sets. In *VLDB*, 2005.
[11] H.T.Kung, F.Luccio, and F.P.Preparata. On finding the maxima of a set of vectors. *JACM*, 22(4), 1975.
[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] X. Lian and L. Chen. Monochromatic and bichromatic reverse skyline search over uncertain databases. In *SIGMOD*, 2008.
[15] X. Lin, Y. Yuan, W. Wang, and H. Lu. Stabbing the sky: efficient skyline computation over sliding windows. In *ICDE*, 2005.
[16] M. Morse, J. M. Patel, and H. V. Jagadish. Efficient skyline computation over low-cardinality domains. In *VLDB*, 2007.
[17] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *TODS*, 2005.
[18] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *VLDB*, 2007.
[19] 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.
[20] M. Sharifzadeh and C. Shahabi. The spatial skyline queries. In *VLDB*, 2006.
[21] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *VLDB*, 2001.

[22] P. Wu, C. Zhang, Y. Feng, B. Zhao, D. Agrawal, and A. El Abbadi. Parallelizing skyline queries for scalable distribution. In *EDBT*. 2006.
[23] T. Xia and D. Zhang. Refreshing the sky: the compressed skycube with efficient support for frequent updates. In *SIGMOD*, 2006.
[24] X. Zhang and J. Chomicki. Preference queries over sets. In *ICDE*, 2011.