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 show that the anti-monotonic property in the well-known *Apriori* algorithm does not hold for skyline group pruning. We then identify *order-specific property* which applies to SUM, MIN, and MAX and *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 a baseline method.

Categories and Subject Descriptors

H.2 [**Database Management**]: Database Applications

Keywords

skyline queries, group recommendation, anti-monotonic properties

1. INTRODUCTION

In this paper we formulate and investigate the novel problem of computing the *skyline groups* of a dataset. Consider a database table of n tuples and m numeric attributes. 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. While the traditional skyline tuple problem has been extensively investigated in recent years [4–7, 9, 12, 13], the skyline group problem surprisingly has not been studied in prior work.

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 application-specific preference order on the domain of each attribute, 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. 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.

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. Another motivating application is to choose a group of experts to perform a task (e.g., develop a software) or to evaluate a work (e.g., review a grant proposal), based on the experts' collective strength on multiple desired skills.

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.

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., $O(\binom{n}{k})$

CIKM'12, October 29–November 2, 2012, Maui, HI, USA.

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 output is to list the distinct aggregate tuples, each representing possibly many skyline groups, but also provide 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 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 input tuples and significantly reduce 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 two 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 (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. 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. Due to space limitations, we do not further discuss the algorithms and refer interested readers to the extended version of this paper [11].

2. RELATED WORK

Skyline query has been intensively studied over the last decade. Kung et al. [8] first proposed in-memory algorithms to tackle the skyline problem. Börzsönyi et al. [4] 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 [6, 7], progressive skyline computation [9, 12, 13], query optimization [5], and many variants of skyline queries.

With regard to the concept of skyline groups, the most related previous works are [3] and [14]. In [3] 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. [14] studied set preferences where the preference relationships between k-subsets of tuples are based on features of k-subsets. The features are 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 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 5 shall demonstrate.

With regard to the problem of forming expert teams to solve tasks, the most related prior works are [10] and [2]. In [2] teams are ranked by a scoring function, while in our case groups are compared by skyline-based dominance relationship. In [10], 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 attribute value of v_1 is either larger than or equal to the corresponding value of v_2 according to the preference order and v_1 is larger than v_2 on at least one attribute.

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 in-

stance, 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

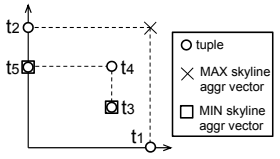


Figure 1: Running Example in 2-d Space

	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

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. 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. 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.

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 5.

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, in analogy to the Apriori algorithm for frequent itemset mining [1]. It is important to note that the 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.

4.3.1 Order-Specific Anti-Monotonic Property

Our first idea is to factor 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.

[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.

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.

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 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 $O((n-r)^{k-h})$.

Theorem 1 SUM, MIN and MAX satisfy the order-specific anti-monotonic 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, the order-specific pruning process may incur a significant overhead. To address this, we consider an order-free anti-monotonic property as follows.

4.3.2 Weak Candidate-Generation Property

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., $\mathcal{F}(G_{k-1} \cup \{t\}) = v$). ■

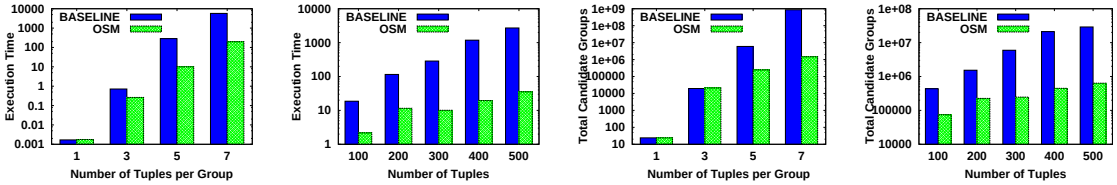
An intuitive way to understand the definition is to consider the case where every skyline group has a distinct aggregate vector. In this case, the weak anti-monotonic property holds when every skyline k-tuple group has at least one $(k-1)$-tuple subset being a skyline $(k-1)$-tuple group. The property 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.

Theorem 2 MIN and MAX satisfy the weak candidate-generation property, while SUM does not satisfy the property. ■

5. EXPERIMENTS

In this section we provide a partial presentation of our experimental results. We refer interested readers to the extended version [11] for more details and results.

Datasets: We collected 512 tuples of NBA players who had played in the 2009 regular season. The tuple of each player has



(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

5 performance statistics – points per game, rebounds per game, assists per game, steals per game, and blocks per game. 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 [4]. 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.

Aggregate Functions and Methods Compared: We investigated the performance of two algorithms based on order-specific property (OSM) and weak candidate-generation property (WCM), respectively. We also compared these methods with the baseline method (BASELINE), which is a direct adaptation of the general framework in [14] for our problem (cf. Section 2). We executed these methods for aggregate functions SUM, MIN, and MAX. Due to space limitations, we will not discuss the results from WCM.

Size of Output under Different Functions: Table 3 shows, for different n (number of tuples, i.e., dataset size), k (number of tuples per group, i.e., group size), 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. 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.

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.

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.

Comparison of Various Methods: Figure 2 shows the execution time and number of generated candidate groups, by BASELINE/OSM for SUM, over the NBA dataset. In sub-figure (a) and (c), we fix the size of dataset (n) to 300 tuples and vary group size (k). In sub-figure (b) and (d), we fix the group size (k=5) and vary dataset size. We observed that OSM performed substantially (often orders of magnitude in execution time) better than BASELINE. Without the order-specific pruning properties, BASELINE produced much more candidate groups than OSM.

Acknowledgments: The work of Li is partially supported by NSF grants 1018865, 1117369, and 2011, 2012 HP Labs Innovation Research Award. The work of Zhang is supported in part by NSF under grants 0852674, 0915834, and 1117297. The work of Das is partially supported by NSF grants 0812601, 0915834, 1018865, a NHARP grant from the Texas Higher Education Coordinating Board, and grants from Microsoft Research and Nokia Research. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the funding agencies.

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

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

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

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

[8] H.T.Kung, F.Luccio, and F.P.Preparata. On finding the maxima of a set of vectors. *JACM*, 22(4), 1975.

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

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

[11] C. Li, N. Zhang, N. Hassan, S. Rajasekaran, and G. Das. On skyline groups (extended version). Technical report, CSE Department, University of Texas at Arlington, August 2012.

[12] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *TODS*, 2005.

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

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