On Skyline Groups

Nan Zhang *Member, IEEE*, Chengkai Li *Member, IEEE*, Naeemul Hassan, Sundaresan Rajasekaran, Gautam Das *Member, IEEE*

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.

Index Terms—Skyline Queries, Skyline Groups, Anti-Monotonic Properties

1 Introduction

The traditional *skyline tuple* problem has been extensively investigated in recent years [5], [10], [12], [26], [14], [21], [8]. 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. 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 table.

In this paper, we formulate and investigate the novel problem of computing *skyline groups*. In contrast to the skyline tuple problem which has been extensively investigated, the skyline group problem surprisingly has not been studied in prior work. In this problem, 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 relation between tuples in skyline analysis. The dominance relation between two groups of k tuples 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, 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. Note that throughout the paper, we assume the larger the SUM/MIN/MAX values are, the better a group is. As an simple example, consider two 3-tuple groups—G=$\{\langle 0,3\rangle,\langle 2,1\rangle,\langle 2,2\rangle\}$ and G'=$\{\langle 2,1\rangle,\langle 2,2\rangle,\langle 0,2\rangle\}$. Their aggregate tuples under the function SUM are SUM(G)=$\langle 4,6\rangle$ and SUM(G')=$\langle 4,5\rangle$. Hence G dominates G'.

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 who may or may not be in the same real-world team, aiming at outperforming other gamers' teams. They select teams, which are of equal size, 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 a certain number 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.

The capability of recommending groups is valuable in the above-mentioned applications. An attractive property of

- *N. Zhang and S. Rajasekaran are with the Department of Computer Science, George Washington University.*
  E-mail: nzhang10@gwu.edu, sundarcs@gwmail.gwu.edu
- *C. Li, N. Hassan, G. Das are with the Department of Computer Science and Engineering, The University of Texas at Arlington, Arlington, TX 76019. G. Das is also with Qatar Computing Research Institute.*
  E-mail: cli@uta.edu, naeemul.hassan@mavs.uta.edu, gdas@uta.edu

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 process that ultimately recommends one group.

Recommending a few groups becomes non-trivial when there are many skyline groups. In addition to eyeballing skyline groups by browsing and visualization interface, such post-processing can also be automatic. One approach is to filter and rank skyline 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. Another automatic approach is to return a small number of representative skyline groups, by criteria proposed for skyline tuples [7], [18], [27], [6], since each group corresponds to an aggregate tuple. We do not further investigate such post-processing in this paper. In Section 2, we provide a more detailed discussion of previous work on choosing from a large number skyline tuples.

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.

A simple solution 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 in creating this huge intermediate input for the traditional skyline tuple algorithm (i.e., $O(\binom{n}{k})$ aggregate tuples). 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—while significantly smaller that $\binom{n}{k}$, may be nevertheless too large to explicitly compute. 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, and also provide enough additional information so that actual skyline groups can be reconstructed if required. Interestingly, there is a difference between MIN and MAX in this regard: while the compression for MIN is relatively efficient, the compression for MAX requires solving the NP-Hard Set Cover Problem (which fortunately is not a real issue in practice, as we shall show in the paper). Note that both output compression and the aforementioned post-processing are for tackling large output. Output compression is incorporated into the process of finding skyline groups and is performed before post-processing.

Our approach to input pruning is to filter input tuples and significantly reduce the input size to the search of skyline groups. The idea is that if a tuple t is dominated by k or more tuples, 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 without influencing the results.

Our final ideas (perhaps, technically the most sophisticated of the paper) are on search space pruning. Instead of enumerating every k-tuple combination, we exclude from consideration many 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]. 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. We identify two anti-monotonic properties with different applicability—while the *Order-Specific Anti-Monotonic Property* (OSM) applies to SUM, MIN and MAX, the *Weak Candidate-Generation Property* (WCM) applies to MIN and MAX but not SUM. We develop a dynamic programming algorithm and an iterative algorithm to compute skyline groups, based on OSM and WCM, respectively. Our algorithms iteratively generate larger candidate groups from smaller ones and prune candidate groups by these properties.

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.
- 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. [13] 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 [10], [12], progressive skyline computation [26], [14], [21], query optimization [8], and the investigation of many variants of skyline queries [23], [30], [17], [22], [19], [11], [16], [24], [9], [29], [4].

With regard to the concept of skyline groups, the most related previous works are [3] and [31]. 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. [31] 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 (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 [15] and [2]. In [2] teams are ranked by a scoring function, while in our case groups are compared by skyline-based dominance relation. Hence the techniques proposed in [2] are not applicable to our setting. In [15], 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.

A large number of skyline points may exist in a given dataset, due to various reasons such as high dimensionality. Such large size of skyline hinders the usefulness of skyline to end users. Researchers have noticed this issue and various approaches are proposed to alleviate the problem. One direction is to perform skyline analysis in subspaces instead of the original full space [23], [28]. Another direction is to choose a small number of representative skyline points. The semantics and methods proposed in various works on this line can be directly adopted for post-processing when there are many skyline groups, since each group corresponds to an aggregate tuple. Specifically, Chan et al. [7] propose to return only frequent points and they measure the frequency of a point by how often it is in the skyline of different subspaces. Lin et al. [18] select k most representative points such that the total number of data points dominated by the k points is maximized. Tao et al. [27] define representative skyline points differently, aiming at minimizing the maximal distance between non-representative skyline points and their closest representatives. Chan et al. [6] define k-dominant skyline as the points that are not dominated by any other points in any k-attribute subspace.

3 Skyline Group Problem

In this section we formulate the skyline group problem. Table 1 lists the major notations used in the paper. Consider a database table D of n tuples $t_1, \ldots, t_n$ and m attributes $A_1, \ldots, A_m$. We refer to any subset of k tuples in the table, i.e., $G : \{t_{i1}, \ldots, t_{ik}\} \subseteq D$, as a *k-tuple group*. Our objective is to find the skyline of k-tuple groups. Whether a k-tuple group belongs to the skyline or not is determined by the dominance relation 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, if and only if it is not dominated by any other k-tuple group in D. Note that a tuple t_i may be in multiple skyline groups.

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. While this definition allows general aggregate functions, we focus on three commonly used functions—SUM (i.e, AVG, since groups are of equal size), MIN, and MAX. Functions such as MEDIAN and VARIANCE do not satisfy the properties in Section 4 and thus do not lend themselves to efficient algorithms proposed in the paper. The aggregate vectors for two groups are compared according to the traditional tuple dominance relation which is defined according to certain application-specific preferences. 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 preferred over "smaller" values. Hence, an aggregate vector v_1 dominates v_2 if and only if every attribute value of v_1 is 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.

t	a tuple with m attributes $\{A_1, \ldots, A_m\}$
D	a database table of n tuples $\{t_1, \ldots, t_n\}$
G	$\{t_{i1}, \ldots, t_{ik}\} \subseteq D$, a *k-tuple group*
$G \succ G'$	G dominates G'
v	an aggregate vector
$\mathcal{F}(G)$	aggregate vector of group G under function $\mathcal{F}$
T_n	the first n tuples
Sky_k^n	all k-tuple skyline groups in T_n
Sky_k	all k-tuple skyline groups in D
OSM	order-specific property
WCM	weak candidate-generation property

TABLE 1: Notations

	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 2: Running example

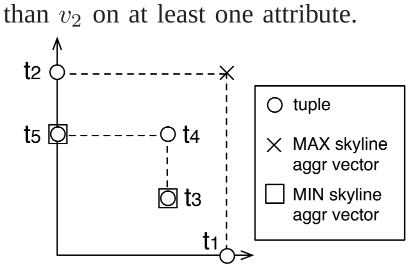


Fig. 1: Running example in 2-d space

	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 Relation		$G \succ G'$	$G \succ G'$	$G = G'$

TABLE 3: Examples of aggregate-based comparison

Table 2 depicts a 5-tuple, 2-attribute table which we shall use as a running example. Figure 1 depicts the tuples on a 2-dimensional plane defined by the two attributes. We consider the natural order of real numbers as the preference order. For instance, t_2 dominates t_5 while neither t_2 nor t_3 dominates each other. Table 3 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 $\{t_2, t_4\}$, $\{t_2, t_5\}$, $\{t_4, t_5\}$).

Our methods allow a mixture of different aggregate functions 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 3, the aggregated vectors for G and G' are $\langle 4, 3\rangle$ and $\langle 4, 2\rangle$, respectively. Our order-specific property (Section 4.3.1) can handle arbitrary mixture of SUM, MIN, and MAX, while the weak candidate-generation property (Section 4.3.2) handles any mixture of MIN and MAX. Section 6 presents the experimental results on such mixed functions.

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 on the seemingly natural strategy of listing all skyline groups as the output. 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 $O(n^{k-1})$ skyline groups. Such a large output size not only leads to significant overhead in computing 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 G=$\{t_1, t_2\}$ in the running example. Note that both t_1 and t_2 are skyline tuples. Nonetheless, under SUM, 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. 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*—filtering the input tuples to significantly reduce the input size to the search of skyline groups. The other is *search space pruning*—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 them 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, SUM 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. In the rest of the paper, we shall focus on 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 the skyline groups corresponding to 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 v' must be equal to v, because (1) v' cannot dominate v (otherwise v is not a skyline vector) and (2) v' does not contain smaller value than v on any attribute, by definition of $\Omega(v)$. On the other hand, if a group contains a tuple outside of $\Omega(v)$, its aggregate vector must have smaller values than v on some attributes, and therefore cannot be in the skyline. The time complexity of a linear scan in finding $\Omega(v)$ is $O(n)$. Given $\Omega(v)$, the only additional step 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. For instance, in finding the 2-tuple skyline groups for a skyline vector v=$\langle 4, 5, 6\rangle$, consider two tuples t_1=$\langle 3, 5, 2\rangle$ and t_2=$\langle 4, 1, 6\rangle$. The set for t_1 is $\{A_2\}$, because t_1 has the same value as v on attribute A_2. Similarly the set for t_2 is $\{A_1, A_3\}$. Since the union of the two sets is $\{A_1, A_2, A_3\}$, the two tuples together is a set cover of size 2. 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 is typically small. As such, even a brute-force enumeration can be efficient, as demonstrated by experimental results in Section 6. Nonetheless, when a large number of tuples "hit" the MAX attribute values, it is unclear how one can efficiently find and store all skyline groups - e.g., by using certain efficient indexing schemes. We leave the design of such indexing schemes as an open problem for future research.

Before algorithmic discussions, we 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—the vector that takes the MAX value on every attribute. In reconstructing skyline groups, for each skyline group, after finding tuples that cover the MAX values, the remaining tuples can be arbitrary.

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 that is dominated by h ($h \geq k$) tuples. There is always an input tuple t' that dominates t and is not in G. Since t' dominates t, there must be less than h tuples that dominate t'. 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, 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 other tuples in G are dominated by k or more tuples, we can use the same process to remove them 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 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 is as follows. Suppose a skyline group G contains a non-skyline tuple t that is dominated by a 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 two *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 Apriori—every subset of a group "of interest" (e.g., a group of frequent items) 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, 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 anti-monotonic properties that hold for skyline search. We stress that the main contribution here is not about *proving* these properties, but rather about *finding* the right ones that can effectively prune the search space.

4.3.1 Order-Specific Anti-Monotonic Property

Our first idea is to make a revision to the classic property in the Apriori algorithm, by factoring in an order of all tuples. To understand how, consider aggregate function SUM and a skyline k-tuple group G_k which violates the Apriori property, i.e., a $(k-1)$-tuple subset of it, $G_{k-1} \subset G_k$, 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 $t_k = G_k \setminus G_{k-1}$. To understand why, suppose that there exists a $(k-1)$-tuple group G' which dominates G_{k-1} but does not contain t_k. Then, $G' \cup \{t_k\}$ would always dominate $G_k = G_{k-1} \cup \{t_k\}$ under SUM, 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, $D \setminus \{t_k\}$ in the above example. This observation can be extended to MIN and MAX, with a small tweak. That is, although G_{k-1} might be dominated by a $(k-1)$-tuple group G' not containing t_k, the aggregate vectors of $G' \cup \{t_k\}$ and G_k must be equal. Therefore, considering G' and ignoring G_{k-1} will still lead us to the same skyline vector. If we require every subset G_{k-1} of a skyline group G_k to be a skyline

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.

group over table $D\backslash\{t_k\}$, where $t_k{=}G_k\backslash G_{k-1}$, we will not miss any skyline vector. This leads to the following property:

Definition 1 Order-Specific Property An aggregate function $\mathcal{F}$ satisfies the *order-specific anti-monotonic property* if $\forall k$, given a skyline k-tuple group G_k with aggregate vector v (i.e., $v = \mathcal{F}(G_k)$), 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 in the way search-space pruning can be done according to this property. Consider an arbitrary order of all tuples, 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 $O((n-r)^{k-h})$. The reason is that, if the aggregate function satisfies the property in Definition 1, either (1) such groups are not skyline k-tuple groups or (2) the aggregate vectors of such groups are unchanged if we replace G_h by h-tuple groups that are subsets of $\{t_1, \ldots, t_r\}$ and dominate G_h. Such a pruning technique considers all tuples in a specific order—hence the name of order-specific anti-monotonic property.

Theorem 1 SUM, MIN and MAX satisfy the order-specific anti-monotonic property. ■

Proof: Suppose G_k is a k-tuple skyline group with aggregate vector v (i.e., $v = \mathcal{F}(G_k)$) and $t \in G_k$. Consider $G_{k-1} = G_k\backslash\{t\}$. (A) If G_{k-1} is a skyline $(k-1)$-tuple group over $D\backslash\{t\}$, then G_{k-1} itself satisfies the two conditions in Definition 1; (B) Otherwise, by definition of skyline group, there must exist a skyline $(k-1)$-tuple group over $D\backslash\{t\}$, G', such that $G' \succ G_{k-1}$. For SUM, only the above (A) is possible, i.e., G_{k-1} must be a skyline $(k-1)$-tuple group over $D\backslash\{t\}$. If (B) is possible, then $G'{\cup}\{t\} \succ G_{k-1}{\cup}\{t\}$ = G_k by the concept of SUM, which contradicts with the assumption that G_k is a k-tuple skyline group. For MIN and MAX, for the above case (B), we prove that $\mathcal{F}(G' \cup \{t\})$=$\mathcal{F}(G_k)$, i.e., G' satisfies both condition (1) and (2) in Definition 1. According to the semantics of MIN (MAX), if $G' \succ G_{k-1}$, then either $G'{\cup}\{t\} \succ G_{k-1}{\cup}\{t\}$ = G_k or $G'{\cup}\{t\} = G_{k-1}{\cup}\{t\}$ = G_k. $G'{\cup}\{t\} \succ G_k$ would contradict with the assumption that G_k is a skyline group. Therefore $G' \cup \{t\} = G_{k-1} \cup \{t\}$ = G_k and thus $\mathcal{F}(G' \cup \{t\})$=$\mathcal{F}(G_k)$. □

We note a limitation of the order-specific property. To prune based on it, one has to compute for every $h{\in}[k, n-k]$ the aggregate vectors of skyline 1, 2, …, $\min(k, h)$-tuple groups over the first h tuples (by the order), because any of these groups may grow into a skyline k-tuple group when latter tuples (again, by the order) are brought into consideration. Given a large n, the order-specific pruning process may incur a significant overhead, as we shall show in Section 6.

4.3.2 Weak Candidate-Generation Property

We now describe an order-free anti-monotonic property which "loosens" the classic Apriori property to one that 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, 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, $\forall k$ 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_k (i.e., $\mathcal{F}(G_{k-1} \cup \{t\}) = v_k$). ■

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 theorem for MAX. The proof for MIN is similar. Suppose G_k is a skyline k-tuple group with $\mathcal{F}(G_k)$=v_k. Consider an arbitrary tuple $t_1{\in}G_k$ and the corresponding $(k{-}1)$-tuple subset of G_k, G=$G_k\backslash\{t_1\}$.

If G is a skyline $(k{-}1)$-tuple group in D, then for any G' (including G itself) such that $\mathcal{F}(G')$=$\mathcal{F}(G)$, there are two possible cases to consider: (A) $t_1{\notin}\, G'$ and (B) $t_1{\in}G'$. In Case (A), $\mathcal{F}(G' \cup \{t_1\})$=$\mathcal{F}(G \cup \{t_1\})$=$\mathcal{F}(G_k)$. In Case (B), note that since G' and G are of equal size, there must exist at least one tuple $t_2 \in G$ and $t_2 \notin G'$. Consider $G' \cup \{t_2\}$. Since $t_2{\in}G$ and $\mathcal{F}(G')$=$\mathcal{F}(G)$, we have that $\mathcal{F}(G' \cup \{t_2\})$=$\mathcal{F}(G \cup \{t_2\})$=$\mathcal{F}(G)$. Furthermore, since $t_1{\in}G'$, under MAX, $\mathcal{F}(G' \cup \{t_2\})$=$\mathcal{F}(G{\cup}\{t_1\})$=$\mathcal{F}(G_k)$. If G is not a skyline $(k{-}1)$-tuple group in D, consider a skyline $(k{-}1)$-group $G'' \succ G$. The same analysis above applies to G' instead of G.

In all cases, we always find a skyline $(k{-}1)$-tuple group and an extra tuple such that the aggregate vector of their union equals the original v_k under MAX. Therefore MAX satisfies the weak candidate-generation property in Definition 2. □

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, as shown in

t_1: ⟨ -131,-40,-4,-4,-98,-20,4,4,-69,-49,-9,-49,-9,54,-59,16,20,20,-107,-22,27,-22,27,61,-39,13,17,13,17,68,-12,-12,89,59,82,35,29,29,46,51,40,51,40,55,27,56,20,56,20,40,37,37,103,44,104,53,47,53,47,42,85,85,78,76,64,64,90,50,106⟩
t_2: ⟨ -40,-79,-38,-38,-80,-66,-52,-52,-85,-59,-67,-59,-67,-54,14,-47,-15,-15,-56,0,-41,0,-41,1,-76,-18,-52,-18,-52,-22,-63,-63,18,-52,3,-50,-32,-32,-60,-11,-47,-11,-47,-26,-67,-34,-51,-34,-51,-38,-59,-59,-22,-51,-18,-4,-32,-4,-32,-21,-17,-17,7,-27,-39,-39,-10,-39,-31⟩
t_3: ⟨ -49,50,-28,-28,51,33,10,10,64,15,35,15,35,20,-102,39,-44,-44,39,-79,14,-79,14,-65,81,-22,28,-22,28,-13,58,58,-51,44,-63,15,-24,-24,62,-52,8,-52,8,-31,57,-1,12,-1,12,-8,45,45,-7,19,6,-56,-8,-56,-8,-35,-9,-9,-68,-10,22,22,-30,5,25⟩
t_4: ⟨ 15,-23,-34,-34,-9,-42,-49,-49,-15,-16,-39,-16,-39,-52,-24,-58,-55,-55,13,-27,-47,-27,-47,-57,-28,-46,-54,-46,-54,-71,-29,-29,-48,-59,-67,-60,-57,-57,-41,-52,-55,-52,-55,-59,-53,-62,-54,-62,-54,-61,-50,-50,-68,-57,-75,-62,-63,-62,-63,-61,-63,-63,-63,-67,-64,-64,-72,-64,-70⟩
t_5: ⟨ 67,39,75,-94,68,22,52,-62,58,145,57,-97,-32,-42,22,11,39,-84,86,94,82,-106,-107,-58,50,111,47,-144,-53,-50,130,-87,-77,-29,-42,-8,13,-54,8,51,28,-129,-66,-41,7,39,20,-105,-33,-27,58,-75,-69,-22,-34,18,14,-95,-62,-32,51,-139,-61,-45,35,-89,-60,-27,-55⟩
t_6: ⟨ 67,39,-94,75,68,22,-62,52,58,-97,-32,145,57,-42,22,11,-84,39,86,-106,-107,94,82,-58,50,-144,-53,111,47,-50,-87,130,-77,-29,-42,-8,-54,13,8,-129,-66,51,28,-41,7,-105,-33,39,20,-27,-75,58,-69,-22,-34,-95,-62,18,14,-32,-139,51,-61,-45,-89,35,-60,-27,-55⟩
t_7: ⟨ 94,-82,44,44,-25,-47,-3,-3,-83,12,-50,12,-50,-11,147,-90,56,56,-66,119,-40,119,-40,84,-122,46,-46,46,-46,20,-86,-86,90,-75,72,-40,30,30,-124,69,-26,69,-26,38,-91,7,-22,7,-22,12,-71,-71,17,-34,-57,70,-5,70,-5,43,-13,-13,87,-5,-47,-47,38,-17,-54⟩
t_8: ⟨ -28,93,75,75,21,95,95,95,68,46,101,46,101,123,-23,115,79,79,1,19,107,19,107,87,80,55,110,55,110,114,84,84,51,136,52,112,91,91,97,69,112,69,112,101,109,96,104,96,104,104,111,111,111,119,104,73,107,73,107,93,100,100,77,119,114,114,101,115,130⟩

TABLE 4: Counter-example for proving Theorem 3

Table 4. With this counter-example, $\{t_1, t_2, t_3, t_4\}$ is a skyline group for SUM, whereas none of $\{t_1, t_2, t_3\}$, $\{t_1, t_2, t_4\}$, $\{t_1, t_3, t_4\}$, or $\{t_2, t_3, t_4\}$ is on the 3-tuple skyline.

5 Algorithms

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.

For ease of presentation, we assume aggregate function SUM in all the propositions, algorithms, and explanations in this section. At the end of the section, we shall explain why the idea is also applicable for MIN and MAX. 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 ($Sky_k^n \equiv S_1 \cup S_2$ and $S_1 \cap S_2 = \emptyset$) according to whether a group contains t_n or not. In particular, $S_1 = \{G | G \in Sky_k^n, t_n \notin G\}$ and $S_2 = \{G | G \in Sky_k^n, t_n \in G\}$. One can see that $S_1 \subseteq Sky_k^{n-1}$. On the other hand, S_2 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. 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 $G \in Sky_k^{n-1}$.■

Proof: We prove this by contradiction. Assume $G \notin Sky_k^{n-1}$. Then, there must be a k-tuple group $G' \in Sky_k^{n-1}$ such that $G' \succ G$. There are two possible cases. (A) $G' \in Sky_k^n$: It contradicts with $G \in Sky_k^n$. (B) $G' \notin Sky_k^n$: There must exist a k-tuple group $G'' \in Sky_k^n$ such that $G'' \succ G'$. By transitivity of dominance relationship, $G'' \succ G$. This also contradicts with $G \in Sky_k^n$. Hence $G \in Sky_k^{n-1}$. □

Proposition 2 Under aggregate function SUM, given $G \in Sky_k^n$, if $t_n \in G$, then $G \backslash \{t_n\} \in Sky_{k-1}^{n-1}$. ■

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  |  S_2^+ ← {{t_n}};
5  else
6  |  S_2^+ ← ∅;
7  |  Sky_{k-1}^{n-1} ← sky_group(k-1, n-1);
8  |  foreach group G ∈ Sky_{k-1}^{n-1} do
9  |  |  candidate_group ← G ∪ {t_n};
10 |  |  S_2^+ ← S_2^+ ∪ {candidate_group};
11 if k < n then
12 |  Sky_k^{n-1} (i.e., S_1^+) ← sky_group(k, n-1);
13 else
14 |  S_1^+ ← ∅;
15 C_k^n ← S_1^+ ∪ S_2^+;
16 Sky_k^n ← skyline(C_k^n);
17 return Sky_k^n;
```

We further explain the dynamic programming algorithm by referring to the outline in Algorithm 1. The idea is also 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 obtains a superset of the aforementioned S_2, according to Proposition 2. We denote this superset S_2^+. By recursively calling the sky_group function (Line 12), it further computes Sky_k^{n-1}, which is a superset of the aforementioned S_1, according to Proposition 1. We also denote Sky_k^{n-1} by S_1^+. S_1^+ and S_2^+ thus contain all necessary candidate groups for Sky_k^n. Thus, the skyline over candidate groups (C_k^n=S_1^+ $\cup$ S_2^+, Line 15) is guaranteed to be equal to Sky_k^n. Existing skyline query algorithms (e.g., [5], [10], [12]) can be applied over C_k^n. We use $skyline()$ to refer to such algorithms (Line 16). The number of candidate groups considered ($|S_1^+ \cup S_2^+|$) can potentially be much smaller than the number of all possible

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

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 calls $sky\_group(k,n)$ again inside $sky\_group(k+1,n{+}1)$. This time Sky_k^n is not recomputed. Instead, the stored result is directly used (Line 1). Hence it is a dynamic programming algorithm. The sequence of calculating Sky_1^1, ..., Sky_k^n is shown by the dashed directed lines in Figure 2(b).

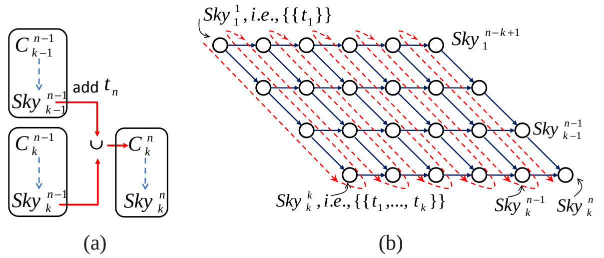


Fig. 2: (a) Calculate Sky_k^n from Sky_{k-1}^{n-1} and Sky_k^{n-1}; (b) Dynamic programming algorithm for calculating Sky_k^n

Our discussion in this section so far assumed SUM. For MIN and MAX, Proposition 2 requires a small modification, as shown in the following Proposition 3.

Proposition 3 Under aggregate function MIN and MAX, given $G{\in}Sky_k^n$, if $t_n{\in}G$, then there exists a group $G' \in Sky_{k-1}^{n-1}$ such that $F(G' \cup \{t_n\}){=}F(G)$. ∎

The implication of the applicability of Proposition 3 (instead of Proposition 2) for MIN and MAX is that, if we still apply Algorithm 1, the S_2^+ produced by Line 8-10 is not guaranteed to be a superset of the aforementioned S_2. In other words, Line 16, which applies the skyline operation over candidate groups, cannot guarantee to produce Sky_k^n. However, the algorithm can still guarantee that the result of it contains all distinct aggregate vectors in Sky_k^n, based on Proposition 3. Note that our goal is to find all distinct skyline vectors and their accompanying (sample) skyline groups for MIN and MAX. Hence the algorithm suffices for our goal without change.

5.2 Iterative Algorithm Based on Weak Candidate-Generation Property

The weak candidate-generation property (Definition 2) can be summarized as follows. Consider the scenario when every skyline group has a distinct aggregate vector. 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.

In reality, multiple skyline groups can have the same aggregate vector. The aforementioned statement is not true anymore. That is, given a skyline group G and any i, it is possible that none of its i-tuple sub-groups is a skyline i-tuple group. However, by Definition 2 and Theorem 2, a slightly different statement can be made for MIN and MAX—Given a skyline k-tuple group G_k and any i, there exists at least a skyline i-tuple group G_i that, when padded with other $k{-}i$ tuples, will result in a skyline k-tuple group G'_k such that $F(G'_k){=}F(G_k)$. Furthermore, given any skyline i-tuple group G'_i such that $F(G'_i){=}F(G_i)$, we can pad G'_i with $k{-}i$ other tuples to get a skyline k-tuple group that has the same aggregate vector as G_k. Therefore, although Algorithm 2 does not produce all skyline groups, it guarantees to find all distinct skyline vectors.

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; $k \le n$
Output: Sky_k: skyline k-tuple groups among T_n

```
1  C_1 ← T_n;
2  Sky_1 ← skyline(C_1);
3  for i ← 2 to k do
4      //generate candidate i-tuple groups C_i from skyline
       i−1-tuple groups Sky_{i−1}.
5      C_i ← ∅;
6      foreach G ∈ Sky_{i−1} do
7          foreach t ∈ T_n do
8              //generate candidate group
9              if t ∉ G then
10                 G' ← G ∪ {t};
11             if G' ∉ C_i then
12                 C_i ← C_i ∪ {G'};
13     //generate skyline i-tuple groups Sky_i based on candidates
       C_i
14     Sky_i ← skyline(C_i);
15 return Sky_k
```

On Feasibility of Combining Order-Specific and Weak Candidate-Generation Properties: The order-specific and weak candidate-generation properties cannot be meaningfully combined. The candidate and skyline groups generated in Algorithm 2 are with respect to all n tuples, for different group size k. However, the candidate and skyline groups in Algorithm 1 are with respect to the first i ($i{=}1..n$) tuples by a particular order. The combination is possible at the last step of Algorithm 1. We can take the intersection of C_k^n from Algorithm 1 and C_k from Algorithm 2 and then invoke $skyline(C_k^n \cap C_k)$. Even for this last step, the cost saving in $skyline()$ due to less candidates may not make up for the extra cost in producing both candidate sets C_k^n and C_k.

Complexity Analysis: The worst-case complexity of both Algorithms 1 and 2 is $O(\binom{n}{k})$, which is as poor as the complexity of the brute-force approach of enumerating all possible groups as candidates. We note that similarly the worst-case complexity of frequent itemset mining algorithms [1] is also exponential and equally poor as that of a brute-force approach. For both problems, it is the characteristics of real datasets that enables the algorithms to prune many candidates and thus to achieve better efficiency in reality. Specifically, one can see from Algorithms 1 and 2 that a critical factor determining the average-case complexity of these algorithms is the number of unique i-tuple skyline vectors in a j-tuple subset of the database (where $i \in [1,k]$ and $j \in [1,n]$) - which in turn depends on the underlying data distribution. For example, the number of unique skyline vectors tends to be small when values of different attributes are positively correlated: In the extreme-case scenario where all attributes share the same value, the number of unique skyline vector is always 1 for all i and j. On the other hand, there tends to be a large number of unique skyline vectors when the attributes are independently distributed. We shall evaluate the efficiency of Algorithms 1 and 2 over real-world datasets in the experiments section.

5.3 From Distinct Vectors to Equivalent 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 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.

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

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

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

$Sky \leftarrow \emptyset$;
$T \leftarrow$ remove k-dominator tuples from R;
$n \leftarrow |T|$; /* number the tuples in T as t_1, ..., t_n */
$Sky_k \leftarrow sky\_group(k, n)$; /* Algorithm 1 or Algorithm 2 */
foreach *skyline k-tuple group $G \in Sky_k$* **do**
  $v \leftarrow$ the aggregated vector of G
  $candidate\_group \leftarrow \emptyset$;
  $i \leftarrow 1$;
  $p[1] \leftarrow null$;
  while $i > 0$ **do**
    /* Note that it is fine to select a tuple multiple times because a tuple can get the same value as v on multiple dimensions. */
    $candidate\_group \leftarrow candidate\_group \setminus \{p[i]\}$;
    $p[i] \leftarrow$ get the next tuple in R that has v's value on the ith dimension;
    if p[i] == null **then** $i \leftarrow i-1$; **continue**;
    $candidate\_group \leftarrow candidate\_group \cup \{p[i]\}$;
    if $|candidate\_group| > k$ **then continue**;
    if i==d **then**
      /* d is the number of dimensions. */
      $k' \leftarrow k - |candidate\_group|$;
      if k'==0 **then**
        $Sky \leftarrow Sky \cup \{candidate\_group\}$;
      else
        $R' \leftarrow R \setminus candidate\_group$;
        foreach *k'-tuple combination G' among the tuples in R'* **do**
          $Sky \leftarrow Sky \cup \{candidate\_group \cup G'\}$;
    else
      $i \leftarrow i + 1$;
      $p[i] \leftarrow null$;
return Sky;

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 have 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 played in the 2009 regular season [20]. 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.

Another dataset is a collection of 35000 tuples that represent stocks for all the publicly traded firms as of December 31^{st}, 2009 in several international markets [25]. Each tuple has 4 attributes, which are market capital (MC), stock price (SP), interest coverage ratio (ICR) and net income (NI). All the values were converted to US dollars.

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 these methods with the baseline method (BASELINE), which is a direct adaptation of the general framework in [31] for our skyline group problem. (The detailed discussion of [31] 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).

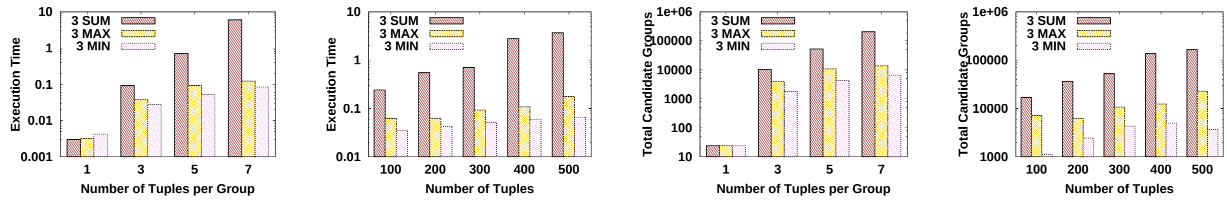
Values Measured: For each applicable combination of aggregate function, method, and parameter values, we measured the execution time needed to find all distinct aggregate vectors and their representative groups, as well as the time to find all skyline groups. Besides execution time, we also measured 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 5: Number of all groups (G), skyline groups (S), distinct skyline group vectors (V), under various 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 5 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



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

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

						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

PPG:Point Per Game, RBG: ReBound per Game, APG: Assist Per Game, SPG: Steal Per Game, BPG: Block Per Game

TABLE 6: Sample skyline groups from 512 players, 5 players per group

indicates that any exhaustive method will suffer due to the large space of possible answers. We 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 corresponding equivalent skyline groups are generated upon request.

Among the three functions, in general SUM has the largest number of skyline vectors and MAX results in the smallest output size (V). 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 6 shows several sample skyline 5-tuple groups under aggregate function SUM, from the NBA dataset. We see the sample groups are formed by elite players with 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.

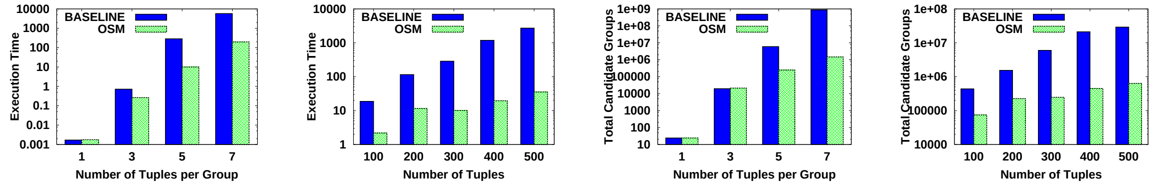
Comparison of Various Methods: Figure 4-6 show the execution time and number of generated candidate groups, by BASELINE/OSM/WCM under all applicable functions, over the NBA dataset. Figure 7 further shows the number of pairwise group (aggregate vector) comparisons performed by these algorithms under 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 properties, BASELINE produced much more candidate groups than OSM/WCM 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 performance of all algorithms, since it significantly reduced the size of input. Table 7 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.

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

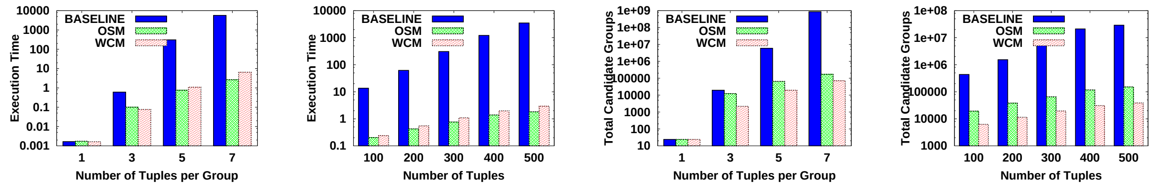
TABLE 7: Number of tuples dominated by $< k$ tuples in NBA

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, in terms of execution time, OSM performed better than WCM on the NBA dataset under both MIN and MAX. Although WCM demonstrated better pruning power in most cases as it resulted in less candidate groups



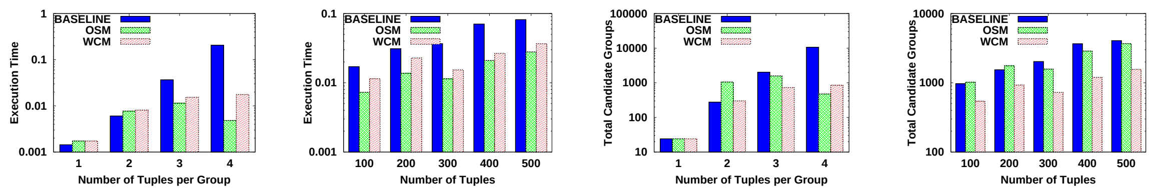
(a) n = 300 (b) k = 5 (c) n = 300 (d) k = 5

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



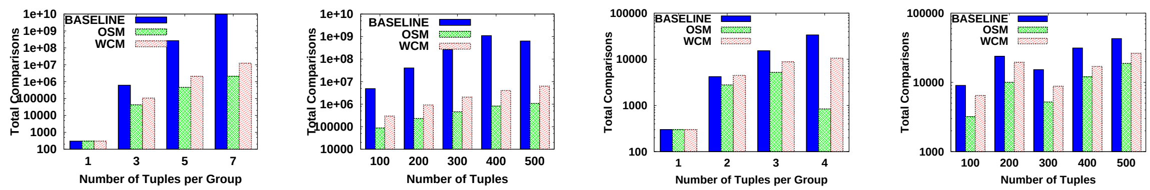
(a) n = 300 (b) k = 5 (c) n = 300 (d) k = 5

Fig. 5: (a)-(b): Execution time (seconds, logarithmic scale) and (c)-(d): number of candidate groups (logarithmic scale), MIN



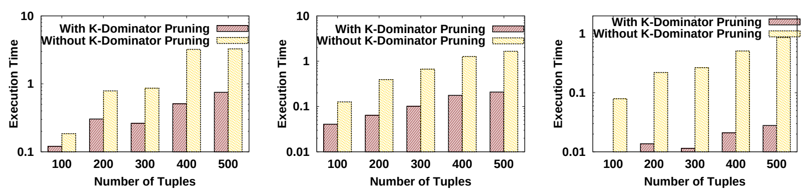
(a) n = 300 (b) k = 3 (c) n = 300 (d) k = 3

Fig. 6: (a)-(b): Execution time (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

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



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

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

(Figures 5(c), 5(d), 6(c), and 6(d)), WCM required more pairwise group comparisons than OSM (Figure 7). Hence it lost in comparison with OSM.

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.

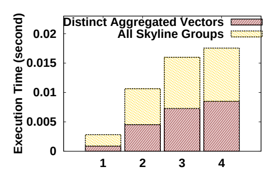
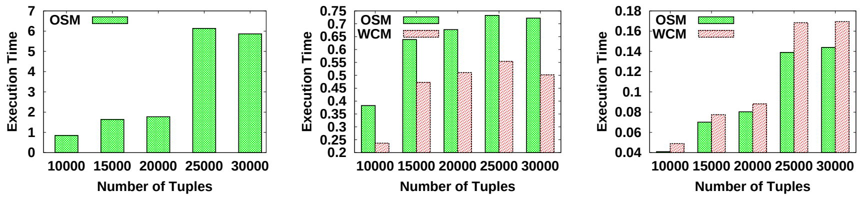


Fig. 9: Finding all skyline groups for MAX, n= 100, OSM

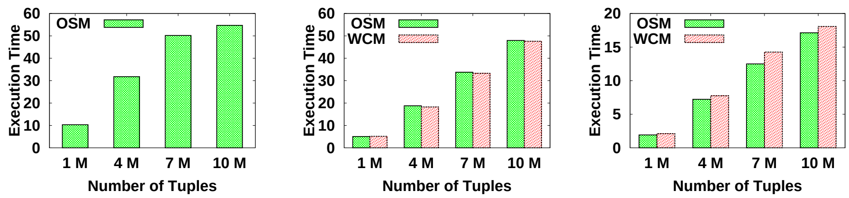
6.3 Experiments on Stock Dataset

We also experimented on the Stock dataset. As the behavior of our algorithms on this dataset is mostly similar to that on the NBA dataset, we do not present extensive results. Figure 10 shows the performance of OSM and WCM for group size k = 3 under various input sizes. It is observed that, although the stock dataset is much bigger than the NBA dataset, the execution time is still considerably small. This is



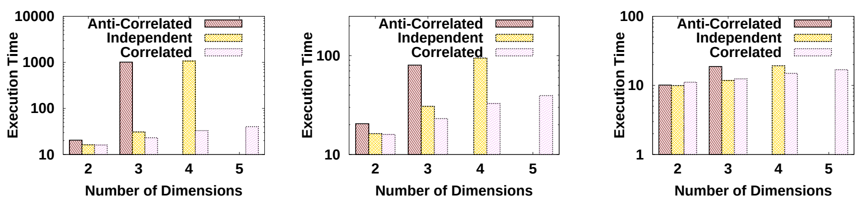
(a) SUM (b) MIN (c) MAX

Fig. 10: Execution time (seconds, logarithmic scale) on stock dataset, k = 3



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

Fig. 11: Execution time (seconds) of OSM/WCM on correlated synthetic dataset with 5 attributes, k = 4



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

Fig. 12: Execution time (seconds, logarithmic scale) of OSM on different synthetic datasets, k = 3, n = 10 million

due to the effective input pruning. Table 8 shows that only less than 300 tuples remained after k-dominator tuple pruning was applied. We also see that, in this dataset, WCM took less execution time than OSM for MIN function. This is partly due to the overhead of OSM in performing candidate generation and skyline comparison for multiple (group size, table size) combinations, as mentioned in Section 4.3.1.

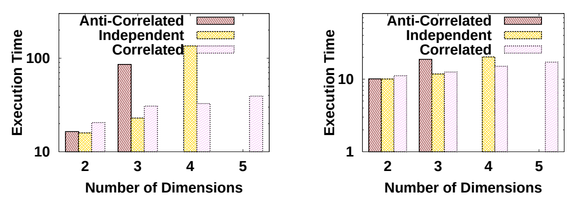
6.4 Experiments on Synthetic Datasets

To show the scalability of our methods, we experimented on the synthetic datasets with 1 to 10 million tuples. In Figure 11, 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. Figures 12 and 13 show the performance of OSM/WCM 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. A direction for future

n	k = 3	k = 5	k = 7
10000	99	139	178
15000	126	172	232
20000	130	180	239
25000	148	205	263
30000	143	217	281

TABLE 8: Number of tuples dominated by less than k tuples in stock dataset



(a) MIN (b) MAX

Fig. 13: Execution time (seconds, logarithmic scale) of WCM on different synthetic datasets, k = 3, n = 10 million

study is to investigate the performance of our methods on synthetic data with 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.

ACKNOWLEDGMENT

The work of Zhang is supported in part by NSF under grants 0852674, 0915834, and 1117297. The work of Li is partially supported by NSF grants 1018865, 1117369, and 2011, 2012 HP Labs Innovation Research Award. 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.

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] C. Chan, H. Jagadish, K. Tan, A. Tung, and Z. Zhang. Finding k-dominant skylines in high dimensional space. In *SIGMOD*, 2006.
[7] C.-Y. Chan, H. Jagadish, K.-L. Tan, A. Tung, and Z. Zhang. On high dimensional skylines. In *EDBT*. 2006.
[8] S. Chaudhuri, N. Dalvi, and R. Kaushik. Robust cardinality and cost estimation for skyline operator. In *ICDE*, 2006.
[9] B.-C. Chen, K. LeFevre, and R. Ramakrishnan. Privacy skyline: privacy with multidimensional adversarial knowledge. In *VLDB*, 2007.
[10] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE*, 2003.
[11] E. Dellis and B. Seeger. Efficient computation of reverse skyline queries. In *VLDB*, 2007.
[12] P. Godfrey, R. Shipley, and J. Gryz. Maximal vector computation in large data sets. In *VLDB*, 2005.
[13] H.T.Kung, F.Luccio, and F.P.Preparata. On finding the maxima of a set of vectors. *JACM*, 22(4), 1975.
[14] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: an online algorithm for skyline queries. In *VLDB*, 2002.
[15] T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *KDD*, 2009.
[16] X. Lian and L. Chen. Monochromatic and bichromatic reverse skyline search over uncertain databases. In *SIGMOD*, 2008.
[17] X. Lin, Y. Yuan, W. Wang, and H. Lu. Stabbing the sky: efficient skyline computation over sliding windows. In *ICDE*, 2005.
[18] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *ICDE*, 2007.
[19] M. Morse, J. M. Patel, and H. V. Jagadish. Efficient skyline computation over low-cardinality domains. In *VLDB*, 2007.
[20] NBA Dataset. from http://www.databasebasketball.com/, 2012.
[21] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *TODS*, 2005.
[22] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *VLDB*, 2007.
[23] 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.
[24] M. Sharifzadeh and C. Shahabi. The spatial skyline queries. In *VLDB*, 2006.
[25] Stock Dataset. from http://pages.stern.nyu.edu/~adamodar/New_Home_Page/data.html, 2012.
[26] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *VLDB*, 2001.
[27] Y. Tao, L. Ding, X. Lin, and J. Pei. Distance-based representative skyline. In *ICDE*, 2009.
[28] Y. Tao, X. Xiao, and J. Pei. Subsky: Efficient computation of skylines in subspaces. In *ICDE*, 2006.
[29] P. Wu, C. Zhang, Y. Feng, B. Zhao, D. Agrawal, and A. El Abbadi. Parallelizing skyline queries for scalable distribution. In *EDBT*. 2006.
[30] T. Xia and D. Zhang. Refreshing the sky: the compressed skycube with efficient support for frequent updates. In *SIGMOD*, 2006.
[31] X. Zhang and J. Chomicki. Preference queries over sets. In *ICDE*, 2011.

Nan Zhang received the BS degree in Computer Science from Peking University in 2001 and the PhD degree in Computer Science from Texas A&M University in 2006. He is an associate professor of Computer Science at the George Washington University. His current research interests include databases and information security/privacy. He is a member of the IEEE.

Chengkai Li is an Assistant Professor in the Department of Computer Science and Engineering at the University of Texas at Arlington. His research interests include databases, Web data management and data mining. In particular, he works on computational journalism, database exploration, database testing, entity search and query, and ranking and skyline queries. He received his Ph.D. degree in Computer Science from the University of Illinois at Urbana-Champaign in 2007, and an M.E. and a B.S. degree in Computer Science from Nanjing University, China, in 2000 and 1997, respectively. He is a member of the IEEE.

Naeemul Hassan received his B.S. degree in Computer Science from Bangladesh University of Engineering and Technology. He is currently a Ph.D. candidate in the Department of Computer Science and Engineering at the University of Texas at Arlington. His research interests include skyline analysis, computational journalism, and social media analytics.

Sundaresan Rajasekaran received an M.S. degree in Computer Science from the George Washington University and a B.E. degree in Computer Science from Anna University, India. He is currently a Ph.D. candidate in the Department of Computer Science at the George Washington University. His research interests include data privacy, network security and cloud computing.

Gautam Das received the BTech degree in computer science from IIT Kanpur, India, and the PhD degree in computer science from the University of Wisconsin, Madison. He is a professor in the Computer Science and Engineering Department at the University of Texas, Arlington. Prior to joining the University of Texas, Arlington in Fall 2004, he held positions at Microsoft Research, Compaq Corporation, and the University of Memphis. His research interests include data mining, information retrieval, databases, algorithms, and computational geometry. He is currently interested in ranking, top-k query processing, and sampling problems in databases as well as data management problems in the deep web, P2P and sensor networks, social networks, blogs, and web communities. His research has resulted in more than 130 papers which received several awards, including the IEEE ICDE 2012 Influential Paper award, VLDB journal special issue on best papers of VLDB 2007, best paper of ECML/PKDD 2006, and the best paper (runner up) of ACM SIGKDD 1998. He is on the editorial board of the journals the ACM Transactions on Database Systems and the IEEE Transactions on Knowledge and Data Engineering. He has served as the general chair of ICIT 2009, program chair of COMAD 2008, CIT 2004, and SIGMOD-DMKD 2004, and best paper awards chair of ACM SIGKDD 2006. His research has been supported by grants from US National Science Foundation, US Office of Naval Research, Department of Education, Texas Higher Education Coordinating Board, Microsoft Research, Nokia Research, Cadence Design Systems, and Apollo Data Technologies. He is a member of the IEEE.