2017 IEEE 33rd International Conference on Data Engineering

K-Dominant Skyline Join Queries: Extending the Join Paradigm to K-Dominant Skylines

Anuradha Awasthi, Arnab Bhattacharya, Sanchit Gupta, Ujjwal Kumar Singh
Dept. of Computer Science and Engineering, Indian Institute of Technology, Kanpur, India.
{aawasthi, arnabb, sanchitg, ujjkumsi}@cse.iitk.ac.in

***Abstract*—Skyline queries enable multi-criteria optimization by filtering objects that are worse in all the attributes of interest than another object. To handle the large answer set of skyline queries in high-dimensional datasets, the concept of k-dominance was proposed where an object is said to dominate another object if it is better in at least k attributes. However, many practical applications, such as flights having multiple stopovers, require that the preferences are applied on a joined relation. In this paper, we extend the k-dominant skyline queries to work on joined relations. We name such queries *KSJQ (k-dominant skyline join queries)*. We show how pre-processing the base relations helps in making such queries efficient. We also extend the query to handle cases where the skyline preference is on aggregated values in the joined relation (such as total cost of the multiple legs of the flight). In addition, we devise efficient algorithms to choose the value of k based on the desired cardinality of the skyline set. Experiments demonstrate the efficiency and scalability of our algorithms.**

I. Introduction and Motivation

Skyline queries [1], [2], [3], [4] are widely used to enable multi-criteria decision making in databases. Consider a scenario where a person wants to buy a good house. Her preferences are low cost, proximity to market, quiet neighborhood, etc. It is almost impossible in a real scenario to find a single house that is best in all her preferences. The skyline query helps her narrow down the choices by filtering out houses that are *worse* (or equal) than some other house in *all* the preferences. For a rational person, the choices cannot lie outside the skyline set.

In high-dimensional spaces, however, the skyline set becomes less useful due to its impractically large size [5], [6]. This happens as it becomes harder for any object to dominate another object in *all* the attributes. An important way to handle this is k-dominant skylines where, instead of being better in all the d dimensions, an object need only be better in some $k < d$ dimensions to dominate another object [7]. As a result, it becomes easier for an object to be dominated which leads to lesser number of skylines.

The k-dominant skylines are quite useful in real scenarios. In the example of housing discussed above, if a smaller number of attributes, say $k = 2$, is specified, there are more chances of finding a house that has a lower cost and a quieter neighborhood (but may not be closer to a market) than another house. As a result, more houses can be filtered, and the retrieved skyline set becomes more manageable and useful.

However, to the best of our knowledge, the k-dominant skyline queries have not been explored for *multiple* relations. Two base relations having d_1 and d_2 skyline attributes are joined to form a bigger relation with $d_1 + d_2$ skyline attributes. A k-dominant skyline, where $k < d_1 + d_2$, is then sought on this joined relation.

A real-life example of this situation happens often in flight bookings. Suppose a person wants to fly from city A to city B. Her preferences are lower cost, lower duration, higher ratings and higher amenities. While the basic skyline works for direct flights, in many cases, a flight route from A to B includes one (or more) stopovers. Thus, a valid flight path contains the join of all flights from city A to other cities and from those cities to city B where the intermediate city is the same. The preferences a user would want now applies to the *entire* flight path and not a single leg of the journey. The skyline is, therefore, needed on the joined relation [8], [9]. Since it is even harder for a flight combination to dominate another flight combination in all the skyline attributes over the two relations, the k-dominant skyline query is a natural choice here.

A further practical consideration in the flight example is that a user is not really worried about the individual legs but rather the the *total* cost and *total* duration of the journey. Thus, the skyline preferences should be applied on the *aggregated* values of attributes from the base relations. Note that the aggregated values are available only after the join and are, therefore, harder to process efficiently.

In this paper, we explore the question of finding k-dominant skylines on joined relations, where the skyline preferences can be on both aggregated and individual values.

In addition to finding k-dominant skylines, an important question that often arises in practical applications is how to choose a "good" value of k? While there is no universal answer, one guiding principle is the number of skyline objects finally returned [7]. It is easier to specify a value of δ objects that she is interested in examining more thoroughly rather than a value of k. We devise a way to return the value of k that returns around δ objects.

II. Problem Statements

Consider a dataset R of objects. Each object u has a set of d *skyline attributes* $\{u_1, \dots, u_d\}$. For each skyline attribute, without loss of generality, the preference is be assumed to be less than ($<$). An object u *dominates* another object v, denoted by $u \succ v$, if and only if, for all the d skyline attributes, u_i is preferred over or equal to v_i, and there exists at least one skyline attribute where u_j is strictly preferred over v_j. The *skyline* set $S \subseteq R$ contains objects that are *not* dominated by any other object [1].

fno	destination	cost	dur	rtg	amn	category
11	C	448	3.2	40	40	SS_1
12	C	468	4.2	50	38	NN_1
13	D	456	3.8	60	34	SN_1
14	D	460	4.0	70	32	NN_1
15	E	450	3.4	30	42	SN_1
16	F	452	3.6	20	36	SS_1
17	G	472	4.6	80	46	SN_1
18	H	451	3.7	20	37	SS_1
19	E	451	3.7	40	37	NN_1

TABLE I: Flights from city A (f_1).

An object u *k-dominates* another object v, denoted by $u \succ_k v$, if and only if, for at least k of the d skyline attributes, u_i is preferred over or equal to v_i, and there exists at least one skyline attribute where u_j is strictly preferred over v_j [7]. The *k-dominant skyline* set contains objects that are not k-dominated by any other object. The k attributes are *not* fixed and can be *any* subset of the d skyline attributes.

Consider two datasets R_1 and R_2 with d_1 and d_2 skyline attributes respectively. The *join* of the two relations forms $R = R_1 \bowtie R_2$ with $d = d_1 + d_2$ skyline attributes. The k-dominant skyline is sought from R. In a significant variation, a number of skyline attributes in each relation can be marked for *aggregation*. Thus, if a attributes are aggregated, R contains $d_1 + d_2 - a$ skyline attributes. The skyline query is then asked over these attributes, including the aggregated ones. The aggregation function is assumed to be *monotonic*.

These two main variants of the K-DOMINANT SKYLINE JOIN QUERY (KSJQ) problem are next formally stated.

Problem 1. *Given two datasets R_1 and R_2 having d_1 and d_2 skyline attributes respectively, find k-dominant skylines from the joined relation $R_1 \bowtie R_2$ having $d_1 + d_2$ skyline attributes.*

Problem 2. *Given two datasets R_1 and R_2 having d_1 and d_2 skyline attributes respectively, of which a attributes are used for aggregation, find k-dominant skylines from the joined relation $R_1 \bowtie R_2$ having $d_1 + d_2 - a$ skyline attributes.*

The number of skyline attributes k is restricted to be at least one more than the dimensionality in the base relations, i.e., $\max\{d_1, d_2\} < k < d$. This restriction constrains at least some skyline attributes from each relation to satisfy the preferences.

The third problem addresses the tuning of the value of k.

Problem 3. *Given a KSJQ query framework and a threshold number of skyline objects δ, determine the* smallest *value of k that returns at least δ skyline objects.*

III. Optimizations

We now describe the various optimization schemes to speed-up finding k-dominant skylines in joined relations.

We assume an *equality* join condition. Two base tuples $u \in R_1$ and $v \in R_2$ can be joined to form $t = u \bowtie v \in R$ if and only if all their join attributes match.

All the tuples in a base relation are, thus, divided into *groups* according to the value of the join attributes. In every group, the values of the join attributes are the same.

A tuple may be a k-dominant tuple for the entire base relation. However, even if it is not, it may not be k-dominated by any other tuple in its group. It is then a k-dominant tuple within its group. Based on this notion, each base relation R_i is divided into 3 *mutually exclusive* and *exhaustive* sets SS_i, SN_i, and NN_i: $R_i = SS_i \cup SN_i \cup NN_i$.

fno	source	cost	dur	rtg	amn	category
21	D	348	2.2	40	36	SS_2
22	D	368	3.2	50	34	NN_2
23	C	356	2.8	60	30	SN_2
24	C	360	3.0	70	28	NN_2
25	E	350	2.4	30	38	SN_2
26	F	352	2.6	20	32	SS_2
27	G	372	3.6	80	42	SN_2
28	H	350	2.4	35	37	SN_2

TABLE II: Flights to city B (f_2).

Definition 1 (SS)**.** *A tuple u is in SS if u is a k-dominant skyline in the* overall *relation; consequently, it is a k-dominant skyline in its group as well.*

Definition 2 (SN)**.** *A tuple u is in SN if u is a k-dominant skyline* only *in its* group *but not in the overall relation.*

Definition 3 (NN)**.** *A tuple u is in NN if u is* not *a k-dominant skyline in its group; consequently, it is not a k-dominant skyline in the overall relation as well.*

Consider the examples in Table I and Table II. We assume that all the attributes have lower preferences. The categorization of the tuples into the three sets (3-dominant) are shown in the last column. Table III shows the joined relation.

The important outcome of this division is that it allows automatically designating certain *joined* tuples as k-dominant skylines (or not) *without* computing the join.

Since the skyline attributes can be from any of the base relations, a brute-force way is to generate all k_1- and k_2-dominant skylines such that $k_1 + k_2 = k$ and, then, combine the answer sets. However, it can be done more efficiently.

We consider the following cases:

$$k'_1 = k - d_2 \qquad k'_2 = k - d_1 \tag{1}$$

For all combinations of k_1 and k_2 such that $k_1 + k_2 = k$, the inequalities, $1 \leq k'_1 \leq k_1 \leq d_1$ and $1 \leq k'_2 \leq k_2 \leq d_2$, hold.

Consider Table III. If $k = 7$, then $k'_1 = k'_2 = 3$.

Using the monotonicity of the *number of skyline attributes*, the following theorems and observations can be proved.[1]

The first base relation is divided into SS_1, SN_1, NN_1 according to k_1-domination while the second base relation is divided into SS_2, SN_2, NN_2 using k_2-domination.

Theorem 1. *The tuples in the set $(SS_1 \bowtie SS_2)$ are k-dominant skylines.*

Theorem 2. *The tuples in the sets $(SS_1 \bowtie NN_2)$, $(SN_1 \bowtie NN_2)$, $(NN_1 \bowtie SS_2)$, $(NN_1 \bowtie SN_2)$, and $(NN_1 \bowtie NN_2)$ are not k-dominant skylines.*

The flight combination (16,26) in Table III is a k-dominant skyline, while (11,24) is dominated by (11,23).

Observation 1. *The tuples in the sets $(SS_1 \bowtie SN_2)$ and $(SN_1 \bowtie SS_2)$ are most likely to be k-dominant skylines, although that is not guaranteed.*

[1]The proofs of the theorems and observations are in [10].

fno	stop-over	f1.cost	f1.dur	f1.rtg	f1.amn	f2.cost	f2.dur	f2.rtg	f2.amn	categorization	skyline
(11,23)	C	448	3.2	40	40	356	2.8	60	30	$SS_1 \bowtie SN_2$	yes
(11,24)	C	448	3.2	40	40	360	3.0	70	28	$SS_1 \bowtie NN_2$	no
(12,23)	C	468	4.2	50	38	356	2.8	60	30	$NN_1 \bowtie SN_2$	no
(12,24)	C	468	4.2	50	38	360	3.0	70	28	$NN_1 \bowtie NN_2$	no
(13,21)	D	456	3.8	60	34	348	2.2	40	36	$SN_1 \bowtie SS_2$	yes
(13,22)	D	456	3.8	60	34	368	3.2	50	34	$SN_1 \bowtie NN_2$	no
(14,21)	D	460	4.0	70	32	348	2.2	40	36	$NN_1 \bowtie SS_2$	no
(14,22)	D	460	4.0	70	32	368	3.2	50	34	$NN_1 \bowtie NN_2$	no
(15,25)	E	450	3.4	30	42	350	2.4	30	38	$SN_1 \bowtie SN_2$	yes
(16,26)	F	452	3.6	20	36	352	2.6	20	32	$SS_1 \bowtie SS_2$	yes
(17,27)	G	472	4.6	80	46	372	3.6	80	42	$SN_1 \bowtie SN_2$	no
(18,28)	H	451	3.7	20	37	350	2.4	35	39	$SS_1 \bowtie SN_2$	no
(19,25)	E	451	3.7	40	37	350	2.4	30	38	$NN_1 \bowtie SN_2$	no

TABLE III: Joined relation ($f_1 \bowtie f_2$).

	SS_2	SN_2	NN_2
SS_1	yes (Th. 1)	likely (Obs. 1)	no (Th. 2)
SN_1	likely (Obs. 1)	may be (Obs. 2)	no (Th. 2)
NN_1	no (Th. 2)	no (Th. 2)	no (Th. 2)

TABLE IV: Fate of k-dominant skylines.

While (11,23) is a skyline, (18,28) is not. Flight 18 has same k_1 values as 19 while 28 is dominated by 25. Therefore, (18,28) is k-dominated by (19,25).

Observation 2. *A tuple in the set $(SN_1 \bowtie SN_2)$ may or may not be a k-dominant skyline.*

Consider (15,25). Since its dominators, flights 11 and 21 respectively, are not join compatible, (11,21) is invalid. Consequently, (15,25) becomes a skyline. On the other hand, for (17,27), the dominators 16 and 26 join, and it is not a skyline.

The overall situation is summed up in Table IV.

We next consider the case of aggregation. There are l_1 local and a aggregate skyline attributes in R_1 and l_2 local and a aggregate skyline attributes in R_2. The total number of skyline attributes in R is, therefore, $l_1 + l_2 + a$ attributes.

The groups in the base relations are partitioned based on both the local and aggregate attributes. Out of the final number of skyline attributes, k, a can be aggregate. Thus, the minimum number of local attributes that must be dominated in each base relation is $k_1'' = k - a - l_2$ and $k_2'' = k - a - l_1$. The categorization of the base relations are done on the basis of $k_1' = k_1'' + a$ and $k_2' = k_2'' + a$. Since $d_1 = a + l_1$ and $d_2 = a + l_2$, these definitions remain the same as earlier. Hence, the summarization also remains the same (as in Table IV).

IV. Algorithms

The naïve algorithm computes the join of the two relations first and then computes the k-dominant skylines from the joined relation. It suffers from two major disadvantages: inefficiency and non-progressiveness. The user has to wait a long time before even the first skyline is presented.

A. Target Set based Algorithms

To alleviate the problems of the naïve algorithms, we next propose two algorithms, *grouping* and *dominator-based*.

We first explain the concept of *target sets*. A target set for a tuple u' in a base relation is the set of tuples $\tau(u')$ that can potentially combine with other tuples from the other base relation and k-dominate a joined tuple formed with u'. In other words, for a joined tuple $t' = u' \bowtie v'$, there *may* exist v such that $t = u \bowtie v$ where $u \in \tau(u')$ k-dominates t'. No tuple outside the target set $\tau(u)$ of u may combine with any other tuple and dominate t.

Definition 4 (Target Set)**.** *The target set for a tuple $u' \in R_i$ is the set of tuples $\tau(u') \subseteq R_i$ such that $\forall u \notin \tau(u')$, $\nexists v, v'$, $t = u \bowtie v \succ t' = u' \bowtie v'$.*

The definition is one-sided: a tuple in the target set may or may not join and dominate t', but no tuple outside the target set can join and dominate t'. To check whether t' is a k-dominant skyline, u' needs to be checked only against its target set and nothing outside it. The *join* of target sets for the base tuples produces the potential dominating set for the joined tuple.

The target set for a tuple $u' \in SS_1$ constitutes itself and the set of tuples $\{u\}$ that has *at least* k_1' attributes same as u'. The augmentation is required to guarantee correctness. The tuple u' must be included in the target set of u' for the same reason. For each tuple in SS_1, the number of such tuples sharing at least k_1' attributes is typically low. Thus, maintenance of target sets is quite feasible and practical. However, the target set for a tuple in SN_1 can be any tuple *outside* its group that dominates it. For simplicity, we consider it as the entire dataset R_i. Similarly, the target set for NN_1 is R_i.

Our first algorithm is called the *grouping* algorithm. A tuple in a set marked by "may be" or "likely" is checked against the join of the *target* sets of the base tuples. Thus, only the tuples in $SN_1 \bowtie SN_2$ need to be compared against the entire $R_1 \bowtie R_2$. For other tuples, the decision can be taken without joining (the "yes" and "no" cases), or the comparison set is small (for the tuples in $SS_1 \bowtie SN_2$ and $SS_2 \bowtie SN_1$).

The grouping algorithm has the problem that for tuples in SN_i, the target set is the entire relation R_i. The next algorithm, *dominator-based* algorithm rectifies this by *explicitly* storing the set of dominators. For SS_i, this set is empty. The dominator sets are augmented by the tuples themselves and those tuples having the same values in the required number of skyline attributes. The target sets for the joined tuples are composed of the joins of the dominating tuples. The joins of dominating sets are substantially less in size than the target sets for the grouping algorithm (which is the join of the entire target sets). The saving, however, comes at a cost. For SN_i, *all* the dominators need to found out. In addition, the entire dominator set needs to be stored explicitly.

The algorithms that consider aggregation are essentially the same. The only difference is that when the join is performed, the aggregation of the attributes are done as an additional step.

B. Algorithms for Finding k

The naïve algorithm to search for the lowest k that produces at least δ skylines starts from the least possible value of k, i.e., $\max\{d_1, d_2\}+1$, and keeps incrementing it till the number of k-dominant skylines is at least δ. The largest possible value of k, i.e., d, is otherwise returned by default, even if it does not satisfy the δ criterion. The algorithm is inefficient as for each case, it computes the actual k-dominant skyline set.

For a particular value of k, the actual number of k-dominant skylines, Δ_k, is *at least* the size of the "yes" sets, denoted by $\Delta_{k,lb}$, and *at most* the sum of sizes of the "yes", "likely" and "may be" sets, denoted by $\Delta_{k,ub}$. These, thus, denote the lower and upper bounds respectively: $\Delta_{k,lb} < \Delta_k < \Delta_{k,ub}$.

The *range-based* algorithm uses these bounds to speed-up the process. It starts from minimum k. If $\Delta_{k,lb} > \delta$, k is returned. Otherwise, if $\Delta_{k,ub} < \delta$, the next k is searched. Else, the actual number of skylines is found. If $\Delta_k \leq \delta$, k is returned; else, the search is continued.

The range-based algorithm still suffers from the shortcoming that it examines a large number of k's one by one.

The next algorithm uses binary search and starts from the middle of the possible values of k. If $\Delta_{k,lb} \geq \delta$, then k is a potential answer. No value in the higher range can be the answer as the current k already satisfies the condition. Hence, the search is continued in the lower range to try and find a better (i.e., lesser) k. If, on the other hand, $\Delta_{k,ub} < \delta$, then the current estimate of k is too low. The search is, therefore, continued in the higher range. If none of these bounds help, the actual number of k-dominant skylines, Δ_k, is found. The same procedure as earlier is then followed.

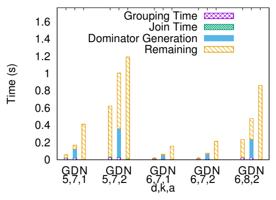
The full description of all the algorithms are in [10].

V. Experimental Results

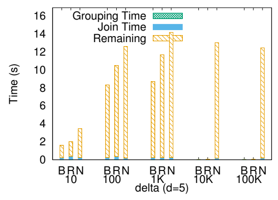
We experimented with data synthetically generated using http://randdataset.projects.pgfoundry.org/ on an Intel i7-4770 @3.40 GHz Octacore machine with 16 GB RAM using code written in Java. Here, we only present representative results. The full set of results can be found in [10].

Fig. 1a shows a medley of results across different d, k and a. The algorithms are denoted as: G for grouping, D for dominator-based, and N for naïve. The size of each base relation is $n = 3300$ with $g = 10$ join groups each, leading to a total size of $n^2/g = 1089000$ for the joined relation. When a or k increase, the running time increases as well. However, it seems that the reverse happens with d. Comparing the case of $d = 5, k = 7, a = 1$ with $d = 6, k = 7, a = 1$, in the first case, $k'_1 = k'_2 = 3$ while in the second case, $k'_1 = k'_2 = 2$. Thus, in the second case, it is easier to find the groups and perform the joins. Consequently, it runs faster. The same reasoning holds true for $d = 5, k = 7, a = 2$ against $d = 6, k = 7, a = 2$. We next compare $d = 5, k = 7, a = 2$ against $d = 6, k = 8, a = 2$. The values of $k'_1 = k'_2 = 4$ are same in both the cases. However, the sizes of dominator sets are larger in the first case. The time required to divide the base relations into the three sets is also higher. This leads to an overall higher running time.

Overall, the grouping algorithm is the fastest. The dominator-based algorithm spends a substantial amount of



(a) KSJQ.



(b) Finding k.

Fig. 1: Experimental evaluation.

time in finding the dominators for each tuple. This overhead is not compensated enough in the final checking stage. This is due to the fact that the average dominator set sizes are quite large and, hence, joining large dominator sets requires a substantial amount of time. With increasing dimensionality, the time required to find the dominator sets increases as well. As expected, the naïve algorithm performs the worst as it does not attempt any optimization at all. It is slower than the grouping algorithm by about about 2-4 times.

Fig. 1b shows the performance of the various algorithms to tune k on the same dataset with varying δ. The three algorithms are depicted as: B for binary search, R for range-based, and N for naïve. With increasing δ, the naïve algorithm requires larger running times since it keeps iterating over k. The range-based search also iterates over the values of k. When δ is very large, it falls outside the upper bounds for most values of k and, hence, the algorithm runs very fast. For low values of δ, the iterations of both naïve and range-based algorithms stop early. The answer for $\delta = 10$ is $k = 8$ while that for $\delta = 100$ and $\delta = 1,000$ are both $k = 9$. The binary search method is the fastest for all values of δ.

VI. Conclusions

In this paper, we proposed a novel query, k-dominant skyline join query (KSJQ), that incorporates finding k-dominant skylines over joined relations where the attributes may be aggregated as well. We analyzed certain optimizations for the query and used them to design efficient algorithms. We also proposed efficient algorithms to find the right value of k.

In future, we would like to extend the algorithms to work in parallel, distributed and probabilistic settings.

References

[1] S. Börzsönyi, D. Kossmann, and K. Stocker, "The skyline operator," in *ICDE*, 2001, pp. 421–430.
[2] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting," in *ICDE*, 2003, pp. 717–719.
[3] D. Kossmann, F. Ramsk, and S. Rost, "Shooting stars in the sky: an online algorithm for skyline queries," in *VLDB*, 2002, pp. 275–286.
[4] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An optimal and progressive algorithm for skyline queries," in *SIGMOD*, 2003, pp. 467–478.
[5] P. Godfrey, "Skyline cardinality for relational processing," in *FoIKS*, 2004, pp. 78–97.
[6] Z. Zhang, Y. Yang, R. Cai, D. Papadias, and A. Tung, "Kernel-based skyline cardinality estimation," in *SIGMOD*, 2009, pp. 509–521.
[7] C.-Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang, "Finding k-dominant skylines in high dimensional space," in *SIGMOD*, 2006, pp. 503–514.
[8] D. Sun, S. Wu, J. Li, and A. K. H. Tung, "Skyline-join in distributed databases," in *ICDE Workshop*, 2008, pp. 176–181.
[9] A. Bhattacharya and B. P. Teja, "Aggregate skyline join queries: Skylines with aggregate operations over multiple relations," in *COMAD*, 2010, pp. 15–26.
[10] A. Awasthi, A. Bhattacharya, S. Gupta, and U. K. Singh, "K-dominant skyline join queries: Extending the join paradigm to k-dominant skylines," arXiv:1702.03390 [cs.DB], 2017.