Constrained Skyline Query Processing against Distributed Data Sites

Lijiang Chen, Bin Cui, Senior Member, IEEE, and Hua Lu, Member, IEEE,

Abstract—The skyline of a multi-dimensional point set is a subset of interesting points that are not dominated by others. In this paper, we investigate constrained skyline queries in a large-scale unstructured distributed environment where relevant data are distributed among geographically scattered sites. We first propose a partition algorithm that divides all data sites into incomparable groups, such that the skyline computations in all groups can be parallelized without changing the final result. We then develop a novel algorithm framework called PaDSkyline for parallel skyline query processing among partitioned site groups. We also employ intra-group optimization and multi-filtering technique to improve the skyline query processes within each group. In particular, multiple (local) skyline points are sent together with the query as filtering points which help identify unqualified local skyline points early on a data site. In this way, the amount of data to be transmitted via network connections is reduced, and thus the overall query response time is shortened further. Cost models and heuristics are proposed to guide the selection of a given number of filtering points from a superset. A cost-efficient model is developed to determine how many filtering points to use for a particular data site. The results of an extensive experimental study demonstrate that our proposals are effective and efficient.

Index Terms—Constrained Skyline Query, Filtering Point, Distributed Query Processing

1 INTRODUCTION

Given a multi-dimensional point set, a skyline query [1] returns all interesting points that are not dominated by any other points. A point \( pt_1 \) is said to dominate \( pt_2 \), if \( pt_1 \) is not worse than \( pt_2 \) in every single dimension but better than \( pt_2 \) in at least one dimension. The implication of “better” varies in different contexts. For example, “better” can mean “smaller” or “larger” in value comparison, and “earlier” or “later” in date comparison.

Because of their powerful capability of retrieving interesting points from a large multi-dimensional data set, skyline queries are well suitable for applications like multiple criteria optimization. Skyline queries play an important role in multi-criteria decision making and user preference applications. For example, a tourist can issue a skyline query on a hotel relation to get those hotels with high stars and cheap prices.

Most work on skyline queries [1], [2], [4], [5], [6], [7] so far has assumed a centralized data storage, and been focused on providing efficient skyline computation algorithms on a sole database. This assumption, however, fails to reflect the distributed computing environments consisting of different computers, which are located at geographically scattered sites and connected via Internet.

For example, a stock trader needs to know which stocks worldwide are worth investing, based on the trading records of the previous day. For this purpose, she needs to access multiple stock information databases available at different places like New York Stock Exchange, London Stock Exchange, Tokyo Stock Exchange, etc. For each single stock, the agent needs to take into consideration multiple attributes like last trade price, change, last close price, estimated price, volume, etc. Therefore, a skyline query against those distributed databases will help the agent get those interesting stocks.

Another example is online comparative shopping, in which a search engine needs to get good bargains from many different shopping sites according to multiple criteria like price, quality, guarantee, etc. Clearly, such multiple criteria are best captured by a skyline query.

For the aforementioned tourist looking for interesting hotels, she may issue her skyline query on a tourism web site which in turn accesses data from many other sites with hotel relations. In this example, efficient skyline queries against those distributed relations will give tourists good user experiences with the tourism web site and thus attract them to pay revisits.

In such example cases, however, directly applying existing centralized skyline approaches to process distributed skyline queries would incur large overheads. Such approaches assume a sole relation as input, and lack adaptations or optimizations specific to distributed computing environments.

In this work, we intend to efficiently process constrained skyline queries [6] in such widely distributed environments. A constrained skyline query is attached with constraints on specific dimensions. A constraint on a dimension is a range specifying the user’s interest. Refer again to the stock selection example. The agent may only be interested in those stocks whose last trade prices are between $15 to $20. Similar constraints are

- Lijiang Chen and Bin Cui are with the Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, China & School of EECS, Peking University, China. E-mail: {clj, bin.cui}@pku.edu.cn
- Hua Lu is with the Department of Computer Science, Aalborg University, Denmark. E-mail: lahua@cs.aau.dk
also applicable to other dimensions. Note that the range denoted by a constraint can be unclosed, like estimated price higher than $20. Also, constraints may be available in only part of the total $n$ dimensions.

Given a distributed environment without any overlay structures, our objective is efficient query processing strategies that shorten the overall query response time. We first speed up the overall query processing by achieving parallelism of distributed query execution. Given a skyline query with constraints, all relevant sites are partitioned into incomparable groups among which the query can be executed in parallel. The parallel execution also makes it possible to report skyline points progressively, which is usually desirable to users.

Within each group, specific plans are proposed to further improve the query processing involving all intra-group sites. On a processing site, multiple filtering points are deliberately picked based on their overall dominating potential from the local skyline. They are then sent to other sites with the query request, where they help identify more unqualified points that would otherwise be reported as false positives, and thus reducing the communication cost between data sites.

We make the following contributions in this paper.

- We propose a specific partition algorithm that divides all relevant sites into groups, such that a given query can be executed in parallel among all those site groups. We elaborate on the intra-group query execution strategies. We then give a parallel distributed skyline algorithm, together with a cost model to estimate the overall query response time.

- We detail heuristics for selecting a given number of multiple filtering points in distributed query processing, such that the amount of data to be transmitted via the network is reduced.

- We propose a cost-efficient model for dynamically determining the number of filtering points to be sent to a particular site, such that the benefit of using filtering points is maximized.

- We conduct an extensive experimental study on both synthetic and real datasets, and the results demonstrate the effectiveness, efficiency and robustness of our proposals.

This paper extends a preliminary work [3] in several substantial ways. First, we present detailed algorithm and enhance it for the intra-group query execution. Second, we analyze the problem of two basic heuristics for filtering points selection and propose the improvements accordingly. Third, we propose a cost-efficient model for dynamically determining the number of filtering points to be sent to a particular site. Fourth, we redesign the experimental study and conduct more extensive experiments in regards to technical extensions.

The remainder of this paper is organized as follows. Section 2 gives the problem definition and provides a brief review of related work on skyline queries. Sections 3 presents the parallel distributed query execution. Section 4 elaborates on the selection of a given number of multiple filtering points. Section 5 details the cost-efficient model for determining the beneficial number of filtering points to use. Section 6 presents the experimental results, followed by the conclusion in Section 7.

2 Preliminaries

2.1 Problem Definition

Given a set of $m$ sites $S = \{S_1, S_2, \ldots, S_m\}$ distributed at different geographic locations, each $S_i$ has a local relation $R_i$. Every tuple in any $R_i$ is an $n$-dimensional point, represented as $(p_1, p_2, \ldots, p_n)$. Different $R_i$’s may overlap, i.e., it is possible that $R_i \cap R_j \neq \emptyset$ for $i \neq j$.

Without loss of generality, we assume smaller values are preferred in the skyline operator. We use $p_1 \prec p_2$ to represent point $p_1$ dominates point $p_2$. In addition, we suppose any site $S_{org}$, able to directly communicate with any other site $S_i \in S$ through wired end-to-end connections, may initiate against all $R_i$:s a skyline query with a set of $n$ constraints $C = \{C_1, C_2, \ldots, C_n\}$. Each $C_i$ is either a range $[l_i, u_i]$, or a $\emptyset$ indicating no constraint in that dimension. Our goal is to get the result for the constrained skyline query efficiently, i.e., with short response time. We define the query response time as the time period from the moment a query is issued by a site $S_{org}$, to the moment $S_{org}$ receives the complete and correct answers after contacting other sites.

To shorten the response time of a query, we mainly endeavor on two aspects. On one hand, we propose effective ways to guide deciding the query forwarding and execution order between different sites, so as to obtain the query execution parallelism, and boost the result reporting progressiveness. On the other hand, we generalize the single filtering point idea [13] to use multiple filtering points, and thus enhancing the filtering power and reducing the amount of data transmitted between remote sites. We propose benefit measurements and heuristics to guide the determination of the number of filtering points to use and selection of powerful multiple filtering points.

In contrast to previous work [12], [10], our problem definition does not assume the availability of an overlay network where different nodes hold disjoint data partitions. Instead, different relations on different sites may overlap. Therefore, these previous methods are not applicable to our problem.

2.2 Related Work

Borzonyi et al. [1] introduced the skyline operator into database systems with algorithms Block Nested Loop (BNL) and Divide-and-Conquer (D&C). Chomicki et al. [2] proposed a Sort-Filter-Skyline (SFS) algorithm as a variant of BNL. Tan et al. [7] proposed two progressive algorithms: Bitmap and Index. The former represents points in bit vectors and employs bit-wise operations, while the latter utilizes data transformation and $B^+$-tree indexing. Kossmann et al. [5] proposed a Nearest Neighbor (NN)

Deviating from skyline queries in the centralized setting, Balke et al. [9] addressed skyline operation over web databases where different dimensions are stored in different data sites. Their algorithm first retrieves values in every dimension from remote data sites using sorted access in round-robin on all dimensions. This continues until all dimension values of an object, called the terminating object, have been retrieved. Then all non-skyline objects will be filtered from all those objects with at least one dimension value retrieved. Differently in this paper, our work deals with distribution of data horizontally partitioned.

Wu et al. [10] proposed a parallel execution of constrained skyline queries in a CAN [11] based distributed environment. By using the query range to recursively partition the data region on every data site involved, and encoding each involved (sub-)region dynamically, their method avoids accessing sites not containing potential skyline points and progressively reports correct skyline points. Wang et al. [12] developed Skyline Space Partitioning (SSP) approach to compute skylines on a tree-structured P2P platform BATON. SSP partitions the skyline space into regions and maps them in a single dimensional order, which allows regions to be distributed to different peer nodes according to BATON protocols. Our proposal in this paper differs from these two pieces of work in that we do not assume any overlay availability on top of the original network.

Huang et al. [13] proposed techniques for skyline query processing in MANETs. Lightweight devices in MANETs are able to issue spatially constrained skyline queries that involve data stored on many mobile devices. Queries are forwarded through the whole MANET without routing information. They proposed a filtering based data reduction technique that reduces the data transferred among devices. Our work, assuming a wired large-scale distributed environment, is also different from this work in a MANET setting. Zhu et al. [21] proposed a feedback-based distributed skyline (FDS) algorithm within a network setting similar as ours. However, FDS only deals with skyline queries without any constraints that are considered in our work. Also, FDS is focused on minimizing the network bandwidth use, and therefore it uses a multiple-round feedback mechanism to prune the unqualified skyline candidates. When the network becomes large, the delay (i.e., the query response time) of FDS will increase considerably. In contrast, our work applies a one-round filtering technique that is aimed to reduce the network delay.

3 SKYLINE PROCESSING AGAINST DISTRIBUTED DATA SITES

3.1 Motivation

In a previous work [13], a skyline query is forwarded among mobile peers via multiple hops in a MANET. Whereas in a wired environment, connections are end-to-end. Because of such wired connections between each pair of sites, a constrained skyline query can be sent out to all peer sites and then each site can execute the query on its own data set simultaneously. This naive execution plan might benefit from the parallelism among all peer sites. However, it is very sensitive to large local skyline results, usually of but not limited to anti-correlated datasets [1], which lead to heavy communication cost. For this reason, identifying unqualified candidates in local skylines early is favorable.

Following the data reduction principle in semijoin in distributed query processing [14], Huang et al. [13] proposed to transfer a single local skyline point with the query request among mobile peers, which acts as a filter to identify those unqualified ones in peers’ local skylines. In this work, we extend this single filtering point to multiple ones since wired connections are much faster and more reliable than wireless channels. How to choose multiple filtering points will be detailed in Section 4. In this section, we focus on finding a good distributed skyline query execution plan that minimizes the total execution time.

We argue that the order to execute a distributed skyline query among multiple sites really matters, because an appropriate execution order can filter more data points earlier and thus reducing not only communication cost but also the local processing cost in the subsequent execution. We intend to balance the parallelism and filtering when processing a distributed skyline query (DSQ for short) and get short overall response time.

3.2 Parallel Distributed Query Execution

To help determine execution order among different sites, the query originator first asks each site $S_i$ for its $MBR_i$, the $n$-dimensional minimum bounding box of the local relation $R_i$. If a site’s $MBR$ disjoins with the constraint set $C$ specified in the query, it will not be considered in the following query processing. For each site whose $MBR_i$ overlaps with $C$, we only need to consider the intersection $MBR_i \cap C$. We call this intersection reduced minimum bounding box and use $rMBR_i$ to represent it.

We proceed to partition all those sites left into several groups according to their $rMBRs$, such that the skyline computation in any one group does not depend on or affect the computation in any other group. Therefore, the given skyline query can be executed in parallel among those site groups. While within any individual site group the query is executed according to some local plans, which will be discussed in Section 3.2.2.
3.2.1 Incomparable Partition of Sites and Parallelism

Given two data sites $S_i$ and $S_j$ (with their reduced minimum bounding boxes $rMBR_i$ and $rMBR_j$ respectively), we need to determine if the skyline query can be executed against them in a parallel way such that two results do not affect each other. Intuitively, we cannot do this if they overlap, as in the overlapping region points from different boxes may be not incomparable. Whereas, the non-overlapping relationship does not necessarily permit parallelism. To handle this problem, we first give a definition of “incomparable” between two data sites.

**Definition 1:** Two data sites $S_i$ and $S_j$ are incomparable with respect to the constrained skyline query, if $rMBR_i.min.DR \cap rMBR_j = \emptyset$ and $rMBR_i.min.DR \cap rMBR_j = \emptyset$.

In Definition 1 above, $rMBR_i.min.DR$ stands for the minimum corner of $rMBR_i$’s minimum corner with respect to the constraints specified in the skyline query. It is obtained as the intersection of the original dominating region [13] and the set of constraints $\mathcal{C}$, which will be formalized in Section 4.1.1. When there is no confusion or ambiguity in the context, we also say that two reduced minimum bounding boxes $rMBR_i$ and $rMBR_j$ are incomparable, which actually means that their corresponding data sites are incomparable. With this definition, we have the following lemma.

**Lemma 1:** If two data sites $S_i$ and $S_j$ are incomparable with respect to the constrained skyline query, it holds that $\forall pt_i \in rMBR_i, \exists pt_j \in rMBR_j$ s.t. $pt_i < pt_j$ and $\forall pt_j \in rMBR_j, \exists pt_i \in rMBR_i$ s.t. $pt_j < pt_i$.

The correctness of this lemma is guaranteed by the property that the minimum corner of $rMBR$ has the strongest dominating capability among all possible points in $rMBR$. This lemma shows that a given skyline query can be executed against two incomparable sites in parallel without conflict between the results. It is easy to see that this parallelism can be generalized to multiple sites, any pair of which are incomparable. A by-product of this parallelism is the progressiveness of skyline reporting, since the result of any single site among those pair-wise incomparable sites definitely appears in the final skyline.

As we do not assume any specific partition among all sites, it is possible that all sites are not pair-wise incomparable. Thus, we partition the set of sites $S$ into non-empty groups that satisfy: (1) Any pair of sites from different groups are incomparable; (2) For any pair of sites $S_i$ and $S_j$ within the same non-singleton group, there exists a path $S_i, S_k, \ldots, S_j$ such that any adjacent pair along the path are not incomparable. Property 1 ensures that parallel execution can be carried out against groups of sites. While property 2 clusters together those sites whose local skyline results potentially conflict, and provides opportunity for alternative optimizations. We call the partition incomparable partition of sites.

The partition algorithm, $icmpPartition$ for short, is shown in Algorithm 1. Before the real partitioning, intersections of site $MBRs$ and constraint set $\mathcal{C}$ are obtained with unpromising sites being pruned (lines 1–4). The algorithm starts by assigning the first candidate site to a singleton group (line 5). Then, each remaining site $S_i$ is compared with every current group, and any group containing sites not incomparable with $S_i$ is removed from the partition into a temporary group $\mathcal{S}_T$ (lines 6–11). At the end of each loop $S_i$ is assigned to the adjusted partition, either in a new singleton group or in the temporary group with other relevant sites found in the loop (line 12). Given all site $MBRs$ and the constraint set, the group formation (group number and compositions) is deterministic accordingly. Our partitioning algorithm obtains the group formation exactly.

**Algorithm 1:** $icmpPartition(S, \mathcal{C})$

| Input: | $S$ is the set of data sites; $\mathcal{C}$ is the set of constraints in the skyline query; |
| Output: | an incomparable partition of $S$; |

// Adjust MBRs and prune unqualified sites
1. for each $S_i \in S$ do
2. $rMBR_i = MBR_i \cap \mathcal{C}$;
3. if $rMBR_i = \emptyset$ then
4. $S = S - \{S_i\}$;
// Compute the independent partition of all relevant sites
5. $\Pi_S = \{(S_i')\};$ // $S_i'$ is the current $1^{st}$ element in $S$;
6. for each $S_i \in S - \{S_i'\}$ do
7. $\mathcal{S}_T = \emptyset$;
8. for each $S_i \in \Pi_S$ do
9. if $\exists S_j \in S_i$ s.t. $S_i$ and $S_j$ are not incomparable then
10. $\Pi_S = \Pi_S - \{S_i\}$;
11. $\mathcal{S}_T = \mathcal{S}_T \cup S_i$;
12. $\Pi_S = \Pi_S \cup \{S_i\} \cup \mathcal{S}_T$;

**Fig. 1:** Incomparable Partition of Sites

Example 1: Refer to an example in 2-dimensional space shown in Figure 1, where $S = \{A, B, C, D, E, F, G\}$. Each site’s $MBR$ is shown in a corresponding rectangles. Based on the incomparability related properties above, $S$ is partitioned into $\{\{A\}, \{B, C, D, E\}, \{F, G\}\}$. A forms singleton group because it is incomparable with any other site. Though $B$ and $C$ are incomparable, they are assigned to the same group with $D$ and $E$, because either of them are not incomparable with $D$ (and $E$). $F$ and $G$ are incomparable with any other site but not with each other, therefore they constitute another group.

3.2.2 Intra-Group Query Execution

Within each group of the sites obtained above, we need to consider the execution order among those relevant
sites. Now effective filtering is our concern, in other words we use filtering points when forwarding a DSQ between sites in the same group. Given a group \( S_i = \{ S_{i1}, \ldots, S_{in} \} \), we have two alternatives for intra-group execution order.

A simple way is to decide a sequence of these sites and forward the DSQ with filtering points along the sequence. The sequence is gained by sorting all sites \( S_i \) in a non-descending order of the Euclidean distance between the minimum corners of \( rMBR_i \) and constraints set \( C \). The reason for this lies in the intuition that a point nearer to the minimum corners of \( C \) (or the origin when a skyline query without constraints is concerned) is more powerful in terms of dominating capability. The sorting can be integrated into the partition algorithm in Algorithm 1, on line 12 where \( \Pi \) and \( S_i \) \((\{S_i\})\) are united.

![Fig. 2. Intra-Group Execution](image)

**Algorithm 2: icmpPartition2(S,C)**

**Input:** \( S \) is the set of data sites; 
\( C \) is the set of constraints in the skyline query; 

**Output:** an incomparable partition of \( S \); 

\[
\text{for each } S_i \in S \text{ do} \\
\quad \text{rMBR}_i = MBR_i \cap C; \\
\quad \text{if } rMBR_i = \emptyset \text{ then} \\
\quad \quad S = S - \{S_i\}; \\
\quad \text{// Compute the independent partition of all relevant sites} \\
\quad \Pi = \{ \{S_i\} \}; \quad // \text{ } S_i \text{ is the current } 1^{\text{st}} \text{ element in } S; \\
\text{for each } S_i \in S - \{S_i\} \text{ do} \\
\quad \text{tree.Root} = S_i; \\
\quad \text{for each } S_i \in \Pi \text{ do} \\
\quad \quad \text{if } \exists S_j \in \Pi \text{ s.t. } S_j \text{ and } S_i \text{ are not incomparable then} \\
\quad \quad \quad \Pi = \Pi - \{S_j\}; \\
\quad \quad \text{tree.addSubTree} \text{(Root, } S_i); \\
\quad \Pi = \Pi \cup \{\text{tree}\}; \\
\]

**Example 3:** Refer to Example 1 and group \( \{B, C, D, E\} \) again. According to Algorithm 2, site B and C are first partitioned into two different groups. And then they are attached to D as subtrees because both of them are not incomparable with site D. At the end, the tree rooted at D is attached to E as a subtree because D and E are not incomparable. The tree structure of the result is shown in Figure 2(b). In this tree, site B and C are executed in parallel after E and D.

In either the linear approach or the tree based approach, every group needs an assembly to merge results from sites and remove false positives during the query processing. We install this procedure in the group head site. The head site is responsible for returning result to query originator. It is the first site in the linear approach, or the root site in the tree-base approach.

### 3.3 Parallel Distributed Skyline Algorithm

Based on the discussion so far, our overall parallel distributed skyline algorithm, called PaDSkyline, is presented in Algorithm 3. When a site \( S_{org} \) issues a distributed skyline query with constraints \( C \), it first gets the incomparable partition of all sites by calling icmpPartition (line 1). After that, the query request will be sent out to each group head in parallel (lines 2–3). The
query request includes constraints \( C \), network addressable query originator identifier \( S_{org} \), and the intra-group query execution plan \( g_i.plan \). Next, \( S_{org} \) enters a wait, during which it will be triggered by an incoming result reply from a group head, until all of them have replied (lines 4–7). When receiving the reply from the head of group \( g_i \), \( S_{org} \) directly reports the result \( g_i.result \) (lines 5–6) as it is incomparable with results from other groups.

For the sake of simplicity, we designate \( S_{org} \) as the head of its site group. The network communication between \( S_{org} \) and a group head degrades to an inter-process or inter-thread communication on a single host.

After a group \( g_i \)’s head receives the query request, it carries out the intra-group skyline computation as shown in Algorithm 4. First, it computes its local constrained skyline \( R_g \) and captures the initial filtering points set \( F_C \) during the local computation (line 1). How to select multiple filtering points will be detailed in Section 4. Next, it sends the query request further to downstream site(s) in the group query plan (line 2). Note here the query request includes the constraints \( C \), network addressable group head’s identifier \( S_g \), the reduced query plan \( plan' \), and the initial filtering points set \( F_C \). Each site \( S_i \) receiving the query request computes its local result \( S_i.R \), returns \( S_i.R \) to site \( S_g \), and sends the query request with updated filtering points and further reduced plan to its own downstream site(s). While \( S_g \) keeps receiving \( S_i.R \) and merging it with \( R_g \) until all group members have replied (lines 3–6).

### Algorithm 4: groupSkyline \((C, S_{org}, plan)\)

**Input:**
- \( S \) is the set of data sites;
- \( C \) is the set of constraints in the skyline query;
- \( plan \) is the query execution plan in the group;

**Output:**
- the constrained skyline \( R_g \) within the group;

1. compute local skyline \( R_g \) and get the initial filtering points set \( F_C \);
2. send \((C, S_g, plan', F_C)\) to next site(s) in \( plan \);
3. repeat
4. receive result reply from a group member \( S_i \);
5. merge \( S_i.R \) with \( R_g \);
6. removing duplicates and false positives;
7. until all group members have replied;
8. return \( R_g \) to \( S_{org} \);

During query execution within a site group, the filtering points set \( F_C \) can be dynamically changed. Also, the query plan can be dynamically reduced as follows. Each site, including the group head, removes itself from the plan. If the plan is a single sequence, the reduced plan is the sub-sequence left and its head is exactly the target downstream site. If the plan is a tree, the removal may result in more than one sub-trees. For that case, each of them will be sent to a corresponding target downstream site, which is exactly the root of a sub-tree.

### 3.4 Cost Model for Query Response Time

Suppose the time to compute the local skyline \( SK_i \) is \( T_i(R_i) \) on a local relation \( R_i \). We use \(|SK_i|\) to represent the size in bytes of a local skyline. The time to transfer \( SK_\text{between two sites is determined by its size, together with the relevant network conditions like bandwidth.} \)

In the naive approach without filtering (see Section 3.1), the time for query originator \( S_{org} \) to get result from a peer site \( S_i \) is the sum of the local computation time and network transmission time, as shown in the following Formula 1.

\[
T_i = T_i(R_i) + T_{\text{trans},org}(|SK_i|) \tag{1}
\]

Due to the parallelism of a naive query execution, its overall response time \( T_{\text{nav}} \) is determined by the peer site whose result reaches \( S_{org} \) latest and the local assembly time on \( S_{org} \), as shown in the following Formula 2.

\[
T_{\text{nav}} = \max \{T_i \mid 1 \leq i \leq m\} + T_{\text{asm}} \tag{2}
\]

Suppose an incomparable partition \( \Pi^g = \{S_1, \ldots, S_k\} \) is produced for a given distributed skyline query with constraints. Its overall response time \( T_{\text{pdq}} \) is determined by the group whose result reaches \( S_{org} \) latest, as shown in Formula 3. Note in the parallel distributed query processing, we do not need a global assembly on \( S_{org} \).

\[
T_{\text{pdq}} = \max \{T_{S_i} \mid 1 \leq i \leq k\} \tag{3}
\]

In the formula above \( T_{S_i} \) is the response time of group \( S_i \), i.e., the time lapse before \( S_{org} \) receives the result from \( S_i \’s \) group head \( h_i \). It is detailed in Formula 4. Here, \( h_i \) is the local response time, i.e., the time lapse before \( h_i \) gets all results from sites in the group, \( T_{\text{asm},h_i} \) is the local assembly time, and \( SK_{S_i} \) is the result sent to \( S_{org} \).

\[
T_{S_i} = T_{h_i} + T_{\text{asm},h_i} + T_{\text{trans},org}(|SK_{S_i}|) \tag{4}
\]

The naive approach has two main disadvantages. First, it has to transmit all unreduced local skylines. Second, it has to assemble all local skylines on \( S_{org} \). In contrast, our distributed approach sends less data among sites, and conducts assembly in parallel in multiple groups.

### 4 Data Reduction Using Multiple Filtering Points

One single filtering point is transferred and changed from peer to peer in [13]. During the query forwarding and processing, the single filtering point is dynamically changed once another point is found to be more powerful in filtering out unqualified candidates.

Data sites in this work, in contrast, are wired with considerably steady and high bandwidth compared to wireless MANETs. This allows us to use multiple filtering points among sites, as data transmission cost via wired connections is lower and multiple filtering points are expected to have higher filtering power. Consequently, we need to decide which and how many skyline points should be used as filtering points in the distributed skyline query processing, such that the benefit is maximized.

In this section, we formalize the dominating region of multiple skyline points with respect to constraints, and address how to select a given number of filtering points initially. In Section 5, we study how many filtering points are adequate for a particular data site.
4.1 Dominating Capability of $K$ Filtering Points

Filtering points are selected from the local skyline result initially obtained. Suppose the initial skyline result is $SK_{init} = \{s_1, s_2, \ldots, s_l\}$, we need to select $K$ ($< l$) points from it as the multiple filtering points. The concrete value of $K$ is constrained by $l$ and has effect on network communication cost and pruning capability obtained. The determination of $K$ value is addressed in Section 5.1.

4.1.1 Dominating Region with Constraints

When $K$ equals 1, we select the point with the maximum potential of dominating others, which is measured by the volume of a point’s dominating region, the hyper-rectangle determined by the point and the maximum corner of the data space [13].

In the presence of the constraints specified for relevant dimensions, the definition of dominating region needs amendments accordingly. As shown in Figure 3, suppose a constraint range $[l_1, u_1]$ is specified on the dimension $p_1$. Consequently, for a point $tp_j (<p_{j1}, \ldots, p_{jn}>)$, its dominating region shrinks to the smaller one determined by its own value, constraint upper bound $u_1$, and dimension $p_2$’s upper bound $b_2$.

Suppose the value range on dimension $p_k$ is $[s_k, b_k]$ in terms of all $R$.s. Then the volume of tuple $tp_j$’s dominating region with respect to constraints is

$$VDR_j = \prod_{k=1}^{n} (\tilde{b}_k - p_{jk})$$

where

$$\tilde{b}_k = \begin{cases} b_k, & \text{if } C_k = \emptyset \\ u_k, & \text{if } l_k \leq p_{jk} \leq u_k \\ p_{jk}, & \text{otherwise} \end{cases}$$

![Fig. 3. Dominating Region with Constraints](image)

Note that it does not make sense to define the dominating region for points out of the region specified by all constraints, because such points does not appear in the skyline result and will not be considered when we select filtering points. In the Formula 5, we ensure the $VDR$ value is zero for any point uncovered by given constraints, which will prevent it from being chosen as a filtering point.

Given a collection of skyline points $\{s_1, s_2, \ldots, s_k\}$, their dominating regions only differ on their own positions in the data space. Thus for any $s_i$ in that collection, its dominating region is represented by a hyper-rectangle $HR_i ([p_{i1}, b_i], \ldots, [p_{in}, b_n])$, where $b_i$ is determined as Formula 5 describes.

4.1.2 Fused Dominating Region

For multiple filtering points, we need to consider their overall filtering capability which is measured by the volume of their fused dominating region. Given any two distinct skyline points $s_i$ and $s_j$, the volume of their fused dominating region is

$$\tilde{VDR}_{i,j} = \tilde{VDR}_i + \tilde{VDR}_j - \prod_{k=1}^{n} (\tilde{b}_k - \max(p_{ik}, p_{jk}))$$

By applying the Inclusion-Exclusion principle further, we can compute the volume of fused dominating region of arbitrary number ($K$) of skyline points.

$$\tilde{VDR}_{1..K} = \sum_{i=1}^{K} \tilde{VDR}_i + \sum_{j=2}^{K} \prod_{k=1}^{n} (\tilde{b}_k - \max(p_{ik}, \ldots, p_{jk}))$$

We call it fused $\tilde{VDR}$ value for $K$ skyline points. Basically, the computation complexity of the Formula 7 is $O(2^K)$, which is quite high. What makes it even more complex is that the optimal selection of multiple filtering points is made by choosing points with the maximum volume of fused dominating region. For that purpose, we need to enumerate all $\binom{K}{K}$ $K$-combinations of $SK_{init}$, whose computation complexity is $O((\frac{n！}{K！})^K)$. Hence the total complexity of computing the fused $\tilde{VDR}$ values for all $K$-combinations is $O((\frac{2n！}{K})^K)$. This high complexity renders undesirable the optimal selection of $K$ filtering points with the maximum fused $\tilde{VDR}$ value. Therefore, we turn to alternatives that make more computationally efficient selections at the cost of the quality of the results.

4.2 Heuristics for Selecting $K$ Filtering Points

In this section, we study heuristics that guide the selection of $K$ filtering points from $l$ ($> K$) skyline points. Note that our goal here is to maximize the collective filtering power of $K$ filtering points, which differs from the criteria used in selecting $K$ most representative skyline points [17], [19].

4.2.1 Two Basic Heuristics

The first heuristic for selecting multiple filtering points maximizes the sum of the $K \tilde{VDR}$ values of all possible choices. To accomplish this, we need to sort points in $SK_{init}$ in a non-ascending order and then pick the top-$K$ ones. In an alternative way, the sorting can be integrated into the skyline computation, which produces a sorted
SK init for easy picking of top-K points. For both ways, the time complexity depends on the sorting method used, which usually does not go beyond O(l^2).

We call this heuristic MaxSum. It actually simplifies the computation by ignoring the overlapping between different skyline points’ dominating regions. In other words, the overlapping between dominating regions determines the accuracy of this approximation. The smaller the overlapping regions are, the more accurate the method will be.

In the second heuristic, we intend to take into account the topology between filtering points, to reduce the overlapping faced by the first heuristic. Distance is a simple metric to help consider this. Intuitively, the farther two skyline points are apart, the less their dominating regions overlap. Hence we propose a greedy heuristic, MaxDist for short, that maximizes the distance between filtering points.

**Algorithm 5: maxDist (SK init, K)**

Input: SK init is the initial skyline; K is the number of filtering points needed;
Output: a set of multiple filtering points;
1. Fflt = Ø ;
2. Pick si and sj from SK init satisfying |si, sj| ≥ |si’, sj’|, ∀ 1 ≤ si’, sj’ ≤ 1 ;
3. Fflt = {si, sj}; SK’ = SK init − {si, sj} ;
4. while |Fflt| < K do
   5. Pick si from SK’ satisfying \[\sum_{sj \in Fflt} |si, sj| \geq \sum_{sj \in Fflt} |si’, sj’, s|, \forall si’ \in SK’ ;\]
6. Fflt = Fflt ∪ {si}; SK’ = SK init − {si};
7. return Fflt

The algorithm of this heuristic is shown in Algorithm 5. Initially, it picks from SK init two points between which the distance is the largest among all pairs (line 2). Then, it incrementally selects points from SK init and add them to the filtering set, until K filtering points are obtained.

In every incremental step, the point with the maximal distance is the largest among all pairs (line 2). Then, we proceed to generalize the idea behind the example. Referring to Figure 5, the dominating regions of tp_i, tp_j and tp_k have the same area. This indicates that they have the same filtering power. Among them, tp_i and tp_j are very close to each other. Therefore, most of their dominating regions are overlapping. Whereas tp_k is far from tp_j, and thus its dominating region is less overlapping with that of tp_j than that of tp_i.

### 4.2.2 Improvement of Filtering Points Selection

Both MaxSum and MaxDist heuristics are intended to select the K filtering points with maximum filtering capability. However, the former suffers from the overlaps among K filtering points, which reduces their filtering power. The latter alleviates the problem by maximizing the distance between filtering points.

The idea behind MaxDist heuristic is good, but it is difficult to implement strictly due to its computational complexity. Therefore, we count the sum of distance between a point to all current filtering points in MaxDist, and then pick the one with the maximum sum as a new filtering point. Though this indeed helps choose K filtering points with more filtering power, there is still room for further improvement.

Refer to the example shown in Figure 4, where a total of 3 filtering points are needed, and ft_1 and ft_2 are two already selected. If we select the third filtering point according to heuristic MaxDist, it is clear that tuple tp_1 is selected because it has the larger sum of distances to the existing filtering points. However, most portion of the dominating region of tp_1 is overlapping with that of ft_1. In other words, selecting tp_1 as a new filtering point only gives very little additional filtering power.

Fig. 4. Example of Filtering Points Selection

Fig. 5. Distance of Filtering Points

In contrast, if we choose tp_2 to be the new filtering point, a much larger extra region will be included to enlarge the fused dominating region of the filtering points, as shown in gray rectangle in Figure 4. This will enhance the filtering power of the 3 filtering points.

We proceed to generalize the idea behind the example. Referring to Figure 5, the dominating regions of tp_i, tp_j and tp_k have the same area. This indicates that they have the same filtering power. Among them, tp_i and tp_j are very close to each other. Therefore, most of their dominating regions are overlapping. Whereas tp_k is far from tp_j, and thus its dominating region is less overlapping with that of tp_j than that of tp_i.
In order to evaluate the cost-efficiency of given filtering points, we simply fix the number of filtering points to $K$ at a site to site. A fixed number of filtering points for all data sites can cause some problems. For some individual data sites, the number of filtering points can be too large and become overkill. This unfortunately incurs unnecessary data transmission cost because a number of filtering points are transferred via the network.

On the other hand, some other sites that have large data spaces and large local skylines may need more filtering points to achieve good filtering effect. As a result, using a fixed number filtering points on all data sites may be too rigid to shorten the overall query response time.

In this section, we propose a cost-efficient model to estimate how many filtering points are adequate for an individual site in term of filtering power. Based on this model, we accordingly vary the number of filtering points for each site in the process of distributed skyline computation.

### 5.1 Cost Model

Principally, what we need to do is to ensure that more local skyline points are filtered by $K$ filtering points, such that sending all these $K$ filtering points does pay off. In order to evaluate the cost-efficiency of given $K$ filtering points, we first define the concept of filtering ratio as follows.

**Definition 2:** Given $K$ filtering points, the filtering ratio on site $S_i$, termed as $FR_i(K)$, is the percentage of local skyline points in $Sky_i$ that are filtered by $K$ filtering points:

$$ FR_i(K) = \frac{|FltedSky_i|}{|Sky_i|} $$

Here, $|Sky_i|$ is the cardinality of local skyline on Site $S_i$, and $|FltedSky_i|$ is the number of local skyline points filtered by those $K$ filtering points.

With the definition of filtering ratio, we are able to measure the benefit of the $K$ filtering points on a particular site according to the following formula:

$$ E_i(K) = FR_i(K) \cdot |Sky_i| - K $$

To make the $K$ filtering points cost-efficient for a particular data site, we need to ensure $E_i(K) > 0$. As a matter of fact, the larger $E_i(K)$ is, the more benefit we obtain from sending the $K$ filtering points. With this formula, we can find a minimum $K$ that maximizes the benefit of $K$ filtering points for a particular site $S_i$.

For this purpose, we need to predict $|Sky_i|$ and $FR_i(k)$ in advance for a particular site $S_i$. Assuming that each site provides data cardinality and distribution, in addition to its MBR, we can estimate the local skyline size $|Sky_i|$ on $S_i$ according to [15], [16], [18], [20]. The relevant result in [16] is used in the experiments.

Regarding $FR_i(K)$, however, it is difficult to directly estimate the number of skyline points on site $S_i$ that can be filtered by $K$ filtering points. Again, we can use the area of fused dominating region of the $K$ filtering points and the data space of site $S_i$ to estimate $FR_i(k)$. Refer to the example shown in Figure 6. Assuming a uniform data distribution on site $S_i$, the filtering point $ft$ dominates half of the data space of $S_i$. Thus, we estimate that $ft$ also dominates half of the local skyline points on $S_i$. As a result, the estimation of $FR_i(K)$ can be represented as the proportion of intersection area between the fused dominating region of $K$ filtering points and the data space of site $S_i$, to the whole space.

$$ FR_i(K) = \frac{\text{Area of fused dominating region of } K \text{ filtering points}}{\text{Data space of } S_i} $$

![Fig. 6. Example of Filtering Ratio](image-url)
of site $S_i$. The formula to estimate $FR_i(k)$ is given as follows:

$$FR_i(K) = f_d \cdot \frac{\text{Area}(FDR_i(K) \cap rMBR_i)}{\text{Area}(rMBR_i)}$$ (10)

Algorithm 7: maxFlt $(SK_{init}, f_d, n_i, rMBR_i)$

Input: $SK_{init}$ is the initial skyline;
$f_d$ is the density function of $S_i$;
$rMBR_i$ is the reduced MBR of $S_i$;
$n_i$ is the number of local skyline tuples of $S_i$;

Output: a set of multiple filtering points;

1. $K = 1; \text{max}E_i = 0; \text{max}F_{flt} = 0$;
2. while $K \leq |SK_{init}|$ do
3. 
4. if $K == 1$ then
5. 
6. else
7. 
8. if $\text{max}E_i < E_i$ then
9. 
10. return $\text{max}F_{flt}$

With the estimations of $|Sky_i|$ and $FR_i(k)$, we propose a greedy algorithm, MaxFlt for short. The pseudo code of MaxFlt is shown in Algorithm 7. It basically checks each possible $K$ value that is not larger than the initial skyline cardinality (lines 2–13), and returns the $K$ filtering points that maximizes the benefits for a given site $S_i$ (line 14). For each $K(> 1)$ value, filtering points are selected according to the improved heuristic MaxDist2 (line 7). Then the filtering ratio and the benefit of those $K$ filtering points are calculated according to formulae defined above (lines 8–9). If the benefit is increased since last iteration, the set of filtering points and the benefit are updated (lines 10–12).

5.2 Dynamic Update of Multiple Filtering Points

After a site $S_i$ receives a query request with a set of filtering points $F_{flt}$, it will execute a local query processing. To take advantage of the filtering points, the local processing can be implemented in two ways. In an integrated way, $F_{flt}$ is checked against every candidate point $pt$ met during the skyline computation, any dominated $pt$ is ignored, and any dominated $s_i$ in $F_{flt}$ is removed from the set.

In a separate way, a local skyline is computed first, and then it will be compared with $F_{flt}$ to filter out those unqualified candidates and dominated $s_i$s. The integrated way is seamlessly applicable to those centralized skyline algorithms that are not based on data transformation [1], [2], [4], [5], [6], whereas the separate way is applicable to all existing skyline algorithms.

For either way, we get a local skyline result $SK_i$ and the reduced set of filtering points $F_{flt}$. Note that $F_{flt} \subseteq F_{flt}$ as some filtering points may be dominated by some local skyline point(s). Before we send the query to further data sites, we need to update the $F_{flt}$ with points in $SK_i$, so that the dominating capability of the multiple filtering points is maintained or increased.

A simple yet efficient way is to treat all points in these two sets equally and select $K$ filtering points from scratch. The heuristics proposed in Section 4.2 can be used to pick fixed $K$ ones from $SK_i \cup F_{flt}$. To update $F_{flt}$ based on heuristic MaxSum, $SK_i$ is sorted in non-descending order of fused $VDR$ values, and merged with $F_{flt}$ to get the top-$K$ ones as those new filtering points. To update $F_{flt}$ based on heuristic MaxDist (MaxDist2), Algorithm 5 (Algorithm 6) is adapted to pick $K$ points from $F_{flt} \cup SK_i$.

Alternatively, we can decide the most appropriate $K$ value and pick $K$ ones from $SK_i \cup F_{flt}$. For this purpose, the cost model proposed in Section 5.1 can be used for a further site $S_i$ to which the query is sent further.

6 Experimental Study

In this section, we evaluate our distributed skyline query mechanism with extensive experiments. All the simulation experiments are conducted on a Linux Server with two Intel(R) Xeon(TM) 2.80GHz processors and 1.0GB RAM. We use two kinds of synthetic datasets, i.e., independent and anti-correlated datasets, and a real-life dataset of NBA players’ statistics (http://databasebasketball.com), which contains 16,644 records of 17 attributes and approximates a correlated data distribution.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimensionality</td>
<td>2, 3, 4, 5</td>
</tr>
<tr>
<td>Number of sites</td>
<td>1,000, 2,000, ..., 10,000</td>
</tr>
<tr>
<td>Filtering point percentage</td>
<td>10%, 20%, ..., 50%, ..., 90%</td>
</tr>
<tr>
<td>Distance threshold ratio $\delta$</td>
<td>10%, 20%, 30%, ..., 90%</td>
</tr>
<tr>
<td>Cardinality of each site</td>
<td>1,000</td>
</tr>
<tr>
<td>Number of queries</td>
<td>50</td>
</tr>
</tbody>
</table>

TABLE 1

Parameters Used in Experiments

In our preliminary work [3], we have studied the performance of algorithm PaDSkyline by comparing it with Naive and Random approaches. The experimental results demonstrate the superiority of PaDSkyline. Therefore, we do not include the results from the previous work.

In this section, we aim at investigating the technical extensions proposed and evaluating their performance within our PaDSkyline framework. For multiple filtering points selection, we apply the improvement (Section 4.2.2) to the two basic heuristics (Section 4.2.1), and name them IpvMaxSum and IpvMaxDist respectively. We further evaluate a new method named Predict, which dynamically selects a most beneficial number of filtering points by predicting $|Sky_i|$ and $FR_i(k)$ of the
cost-efficient model presented in Section 5. We compare these three methods with two heuristics MaxSum and MaxDist [3].

The parameters of the experiments are listed in Table 1, and the default parameter values are given in bold. Same as the results presented in previous experimental study [3], the proposed methods yield similar performance tendency on different datasets. We only detail the results on independent dataset for ease of presentation in this paper.

We consider the following four performance metrics.

- **Query Traffic**, which counts the number of data points sent in the network during the query processing.
- **Data Reduction Efficiency**, the efficiency of multiple filtering points in local skyline computation in terms of the data reduction rate DRR [13]. The DRR is the proportion of data points reduced by the filtering points to the points in the unprocessed local skyline. It is defined as

\[
DRR = \frac{\sum_{i=1,i\neq org}^m (|unredSK_i| - |redSK_i| - K_i)}{\sum_{i=1,i\neq org}^m |unredSK_i|}
\]

where \(K_i\) is the number of filtering points sent to a processing site, and \(m\) is the network size.

- **Response Time**, which records the overall query processing time, from the moment when a query is issued to the moment when the final result is obtained.

- **Precision**, which indicates how much data returned to the query originator is really useful in the final result. The Precision is defined as

\[
Precision = \frac{|finalSK|}{\sum_{i=1,i\neq org}^m |returnedSK_i|}
\]

where \(|returnedSK_i|\) is the number of skyline returned to the query originator site, \(m\) is the network size, and \(|finalSK|\) is the number of skyline retrieved from all returned results.

### 6.1 Analysis of Distance Threshold \(\Delta\)

In the first experiment, we analyze the parameter \(\Delta\) which is used as a distance threshold to select filtering points in Section 4.2.2. In our enhanced algorithms, IpvMaxSum and IpvMaxDist, the \(\Delta\) is calculated from the following formula:

\[
\Delta = \sum_{k=1}^{\text{candtFlt}_i} -1 \sum_{j=k+1, pt_k, pt_j \in \text{candtFlt}_i} \frac{\text{dist}(pt_k, pt_j)}{|\text{candtFlt}_i| \times (|\text{candtFlt}_i| - 1)/2}, \delta
\]

where \(\text{candtFlt}_i\) is the set of candidate filtering points in Site \(S_i\) and \(\delta\) (\(0 < \delta < 1\)) is parameter to control \(\Delta\) value. In other words, the distance threshold \(\Delta\) is calculated from a certain percentage \(\delta\) of average distance between every two points in candidate set.

In our algorithms, IpvMaxSum and IpvMaxDist, we utilize distance threshold \(\Delta\) as a constraint to select \(K\) filtering points. In the implementation of the algorithms, we first give an initial value of \(\delta\) to generate the threshold. However, a large threshold may not produce enough filtering points. Therefore, we may need to automatically adjust \(\delta\) to guarantee the pre-assigned number of filtering points can be selected during the query processing. Consequently, the practical \(\delta\) value used on the site may be equal to or smaller than the initial one. In this experiment we show the practical \(\delta\) value with varying initial distance thresholds.

In Figure 7, we pre-assign the percentage of filtering points to be 50% and vary \(\delta\) from 10 to 90. For Ipv-

![Fig. 7. Practical Value of \(\delta\) for 50% Filtering Points](image)

MaxSum, when \(\delta \leq 30\), the practical values of \(\delta\) are equal to the initial ones. When \(\delta > 30\), the practical value of \(\delta\) is smaller than the initial one, as we need to relax the constraint of distance threshold to generate more qualified filtering points. All practical values fall into a range of \([10\%, 42\%]\). That is to say, to satisfy the condition of selecting 50% filtering points, we must control the \(\delta\) value into this range. For IpvMaxDist, the practical values are much smaller than the initial ones, because it is difficult that a data point has maximum distance to all selected filtering points at the same time it has a distance larger than \(\delta\) to each one of them, if \(\delta\) is not small enough.

In the next experiment, we evaluate the effect of distance thresholds by varying the initial values, and the results are shown in Figure 8. The number of filtering points is another parameter for our scheme which has been studied in previous work [3], we simply set 50% as default value in this paper.

In Figure 8, both algorithms yield better performance when \(\delta\) is relatively small, e.g., \(\delta < 30\). While if \(\delta\) increases, the performance degrades accordingly. The reason is that when the initial \(\delta\) goes larger, more points with long-distance but small dominating region are added to the filtering points, which decreases the overall filtering capacities. Regarding to IpvMaxSum and IpvMaxDis, there is a tradeoff between the size of dominating regions and the overlapping of dominating regions when we select filtering points. From the results, we can see that IpvMaxSum has better overall filtering capability, which is consistent with the results on basic heuristics, i.e., MaxSum and MaxDis. More detailed
6.2 Comparison of Different Methods

In this section, we demonstrate the results of different methods for performance study, i.e., two basic methods MaxSum and MaxDist, their improved versions IpvMaxSum and ipvMaxDist, and Predict which can automatically select filtering points according to the cost-efficient model in Section 5. For IpvMaxSum and ipvMaxDist, we set the initial distance threshold as 30% according to our previous experiment, which yields near optimal performance.

6.2.1 Performance on Query Traffic

We first evaluate the performance of various methods on Query Traffic against two important factors, i.e., network size and dimensionality. In Figure 9, it is clear that Predict algorithm incurs much less query traffic than other algorithms, owing to its dynamical filtering points selection strategy. The Predict method can automatically select “optimal” number of filtering points using the proposed cost model. In our experiments, the “optimal” percentage of filtering points varies from 10% to 60% on different sites. We can also see that the improved version can yield better performance, e.g., IpvMaxSum is more efficient than MaxSum, due to the usage of distance threshold $\Delta$, which makes its filtering points more powerful in terms of filtering capability.

6.2.2 Performance on Data Reduction Rate

In this experiment, we measure the performance on data reduction rate (DRR). The higher DRR of an algorithm has, the more unqualified points it filters out. Figure 10 shows that Predict algorithm again outperforms other algorithms. In Predict, every site utilizes the cost model to select the right filtering points for their next sites,
which is expected to cover the next site’s data space quite well in an approximate way. This can produce a tailor-made filter set for a specific site with maximized filtering capability, compared with the other methods with pre-assigned number of filtering points. The sites generally have different data distributions, and therefore a fixed number cannot be optimal for all the sites in the distributed environment.

6.2.3 Performance on Response Time

We further study the response time of our five algorithms. The response time is calculated at the moment when the query is issued till the final results are retrieved. It includes query propagating time, skyline computing time, filtering point selection time and result transferring time, etc.

As shown in Figure 11, the performance differences among these algorithms are not significant as that on Query Traffic, because the costs of query propagation and local skyline computation are similar for different approaches, and the cost variations of filtering point selection and intermediate data transfer are the main reason of performance difference. The performance of Predict algorithm remains the best because of its effective communication cost reduction, although its automatical filtering point generation may incur more computational cost. However, such local computational cost overhead on sites is marginal compared with data transmission cost in the network. The IpvMaxSum and IpvMaxDist are comparable with the original ones, because the reduced data transmission cost can well offset the additional distance evaluating processes on filtering point selection.

6.2.4 Performance on Average Precision

In the last experiment, we study the performance of average precision of the returned results to the query issuers. Figure 12 shows that the Predict algorithm performs the best among all competitors, owing to its powerful filtering capability and high data reduction rate, which prevents more unqualified points from being transferred to the query issuer. The gap here is not significant as the result on DRR, because the metric Precision only counts the effectiveness of final results rather than the intermediate results transferred among sites. The improved versions yield better performance than original MaxDist and MaxSum respectively. Although they deploy the same number of filtering points for skyline processing, the filtering points are more effective with the enhancement of distance threshold.

7 CONCLUSION

In this paper, we have addressed the problem of constrained skyline query processing against distributed data sites. To accelerate the query processing, we partition all relevant sites into incomparable groups and
parallelize the query processing among all groups. We select local skyline points and send them as filtering points together with the query to relevant data sites, in order to prevent more data from being transmitted through the network. The distance threshold enhancement methods are discussed and employed to improve multi-filtering points selection. Furthermore, a dynamic filtering points selection strategy is proposed based on a novel cost-efficient model. Extensive experimental results demonstrate the efficiency and effectiveness of our proposals in a distributed network environment.

**ACKNOWLEDGMENT**

This research was supported by the National Natural Science foundation of China under Grant No.60603045, 60873063 and 60873051.

**REFERENCES**


Lijiang Chen is a PhD student in the Computer Networks and Distributed Systems Lab, Department of Computer Science, Peking University, China. He is supervised by Professor Bin Cui. His current research interests include P2P data management, P2P search and distributed systems. He is a student member of IEEE.

Bin Cui received the PhD degree in computer science from the National University of Singapore in 2004. Currently, he is a Professor in Department of Computer Science, Peking University. His major research interests include index techniques, multi/hih databases, multimedia retrieval, and database performance. He has served on program committees of several database conferences, including SIGMOD, VLDB and ICDE. Dr. Cui is a member of ACM, and a senior member of the IEEE.

Hua Lu received both his BSc and MSc degrees from Peking University, China, in 1998 and 2001, respectively. He received the PhD degree in computer science from National University of Singapore in 2007. He is currently an Assistant Professor in the department of Computer Science, Aalborg University, Denmark. His research interests include database query, spatio-temporal databases, geographic information systems, and mobile computing.