Communication-Efficient Distributed Skyline Computation*

Haoyu Zhang
Indiana University Bloomington
Bloomington, IN 47408, USA
hz30@umail.iu.edu

Qin Zhang
Indiana University Bloomington
Bloomington, IN 47408, USA
qzhangcs@indiana.edu

ABSTRACT

In this paper we study skyline queries in the distributed computational model, where we have s remote sites and a central coordinator; each site holds a piece of data, and the coordinator wants to compute the skyline of the union of the s datasets. The computation is in terms of rounds, and the goal is to minimize both the total communication cost and the round cost.

We first give an algorithm with a small communication cost but potentially a large round cost; we show information-theoretically that the communication cost is optimal even if we allow an infinite number of communication rounds. We next give algorithms with smooth communication-round tradeoffs. We also show a strong lower bound for the communication cost if we can only use one round of communication. Finally, we demonstrate the superiority of our algorithms over existing ones by an extensive set of experiments on both synthetic and real world datasets.

1 INTRODUCTION

Skyline computation, also known as the *maximal vector problem*, is a useful database query for multi-criteria decision making. If we view data objects as points in the d-dimensional Euclidean space, then the skyline is defined to be the subset of points that cannot be dominated by others. Formally, given two distinct points $x = (x_1, x_2, \ldots, x_d)$ and $y = (y_1, y_2, \ldots, y_d)$, we say y *dominates* x, denoted by $y \succeq x$, if $y_i \geq x_i$ for every $i = 1, 2, \ldots, d$ and $y_i > x_i$ for at least one $i \in \{1, 2, \ldots, d\}$. For a set of distinct points S, the skyline of S is defined to be

$$sk(S) = \{u \in S \mid \text{no other } v \in S \text{ s.t. } v \succeq u\}.$$

The skyline problem was first studied in computational geometry in the mid-1970's [14], and was later introduced into databases as a query operator [3]. Most work on skyline computation in the literature was conducted in the RAM (single machine) model [9, 13, 14, 16]. In recent years, due to the large size of the datasets and the popularity of the *map-reduce* type of computation, a number of parallel skyline algorithms have been proposed [1, 17, 18, 20, 24, 26]. A common feature of those parallel algorithms is that they use the *divide-and-conquer* approach: They first partition the whole point set into a number of subsets, and then assign each subset to a machine for a local processing; finally the local results are merged to form the global skyline. The art of the algorithm design in this line of work lies in how to choose the partition mechanism.

In this paper we consider the skyline computation on distributed data, which is different from parallel computation in that the data is inherently distributed in different locations, and we *cannot* afford to repartition the whole dataset since data repartition is communication prohibited over networks, and may also cause local storage issues which the query node cannot control.

Consider a global hotel search engine, where each hotel is represented as a point in the 2-dimensional Euclidean space with the x-coordinate standing for the price and the y-coordinate standing for the rate of the location. A user naturally wants to find a hotel with the best location *and* the best price, although in reality hotels in good locations typically have higher prices. Thus a good search engine should recommend the user with a list of candidates such that no other hotel has both cheaper price and better location. This list is exactly the skyline of the point set. Given a query, the search engine needs to contact servers/providers in different locations worldwide. The total bits of communication between the query node and servers and the communication rounds typically dominate the engine's response time, since sending messages through network is much slower than local computation, and the initialization of a new communication round takes quite some system overhead.

In this paper we study the skyline problem in the *coordinator model* which captures the type of distributed computation mentioned above. In this model we have s remote sites each holding a set of points S_i in the Euclidean space, and a central coordinator which acts as the query node. We assume there is a two-way communication channel between each site and the coordinator. The computation is in terms of rounds: at the beginning of each round the coordinator sends a message to some of the sites, and then each of the contacted site sends a response back to the coordinator. The goal is to compute $sk(S)$ while minimizing the total communication cost and the number of rounds of the computation. See Figure 1 for a visualization of the model.

Optimizing the communication cost in the skyline computation on distributed data has been studied previously [5, 18, 19, 27]. A notable difference between our work and the previous ones is that in our algorithms we allow to take a round budget as an input (to allow a tradeoff between round cost and communication cost), while all the existing algorithms use fixed numbers of rounds. Moreover, different from previous heuristics we also provide rigorous theoretical analysis on the communication cost given a round budget for the 2-dimensional Euclidean space.

*Both authors are supported in part by NSF CCF-1525024 and IIS-1633215.

CIKM'17, November 6–10, 2017, Singapore.

ISBN 978-1-4503-4918-5/17/11…$15.00
DOI: https://doi.org/10.1145/3132847.3132927

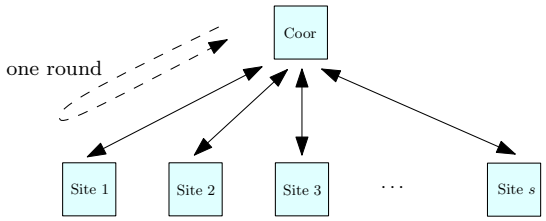


Figure 1: The Coordinator Model

We have made the following contributions. Let n be the total number of points, k be the number of skyline points (i.e., the output size), s be the number of sites, and d be the dimension of the Euclidean space.

(1) We have proposed an algorithm that achieves $O(ks)$ words of communication and at most k rounds. We complement this upper bound by proving a lower bound stating that the communication cost is in fact optimal for $d = O(1)$, even if we allow an infinite number of communication rounds.
(2) We have shown that if we want to finish the computation in one round then the communication cost has to be $\Omega(n)$.
(3) We have proposed an algorithm that gives smooth tradeoffs between the communication cost and the round cost.
(4) We have implemented our algorithms and relevant ones in the literature, and run them on both synthetic and real-world datasets. Our experiments have demonstrated the superiority of our algorithms over the existing ones in various aspects.

1.1 Related Work

Skyline computation has been studied extensively in various settings, including web information systems [2], peer-to-peer system[6, 8, 10, 21, 22, 24–26], mobile ad-hoc network [12, 23] and distributed systems [5, 18, 19, 27]. There is a vast literature on skyline computation and its variants, and we refer readers to [7] and [11] for excellent surveys. Most of the existing works consider specific network communication topology and are thus different from us, except *FDS* [27], *AGiDS* [18], *PaDSkyline* [5], *SkyPlan* [19] which we will describe below.

PaDSkyline and *SkyPlan* were proposed for the clique communication topology where each site can directly communicate with every other site. Both of them use the *minimum bounding rectangular* (MBR) to summarize data at each site. In *PaDSkyline*, the query node collects MBRs from all sites and partitions them into incomparable groups, such that points at a site can only be dominated by points at other sites in the same group. For each group, a specific query plan is determined and represented as a tree structure. The query is then forwarded from the root to the leaves of the tree, and each site sends points to be used for filtering to the next site in the plan. Finally the remaining points in each site are collected by the root, and the skyline points among them are sent back to the query node. *SkyPlan* is similar to *PaDSkyline*, where each site also sends its MBR to the query node as a summarization of data. *SkyPlan* improves the selection of query plan in *PaDSkyline* by building the *SD-graph*, where each node corresponds to a site and each directed edge is assigned with a weight to represent the dominance relationship between sites. A query plan is computed based on the graph to maximize a quality function, which is the sum of weights of all edges in a plan. We note that the tree-based computation is inherent sequential and will incur a lot of communication rounds if implemented in our coordinator model.

FDS [27] and *AGiDS* [18] are designed for the coordinator model, and will be used as competitors in our experiments. We will describe them in details in Section 4. Both *FDS* and *AGiDS* cannot take a round budget as input; *FDS* may use many rounds, while *AGiDS* is a fixed round (2-round) algorithm. We will see in our experiments that our algorithms outperform both *FDS* and *AGiDS*.

2 ALGORITHMS

In this section we give a set of algorithms for skyline computation on distributed data. The first algorithm has a small communication cost, but needs a number of communication rounds that is proportional to the size of the skyline. In Section 3.1 we will show that the communication cost of this algorithm is in fact *optimal*[1] even if we allow an *infinite* number of communication rounds.

We note that there is a simple algorithm for computing the skyline points in the coordinator model in *one* round: Each site computes the skyline of its local data points and sends it to the coordinator, and then the coordinator computes the global skyline by merging these local skylines. Unfortunately this algorithm has communication cost $\Omega(n)$; in other words, in the worse case almost all points in all sites need to be sent to the coordinator. We enclose a proof for this statement in Section 3.2.

We next try to explore if more rounds can help to reduce the communication cost, and propose algorithms with tradeoffs on communication cost and rounds.

For convenience, let $[n]$ denote $\{1, 2, \ldots, n\}$.

2.1 Optimal Communication Cost

We described our algorithm in Algorithm 1. Let us explain it in words. At the beginning, each site computes its local skyline points since only these points can possibly be the global skyline points. The rest of the algorithm works as follows. At each round, the coordinator tries to find the point with the largest first coordinate in the remaining points held by all sites. This is done by asking all sites to report the local maximums of their remaining points (Line 3-5). Next, the coordinator computes a new global skyline point from the received local maximums, and then sends the new global skyline point to all sites for another local pruning step (Line 7-9).

We now show the correctness of Algorithm 1 and analyze its costs. First, it is clear that the point with the largest first coordinate must be on the skyline. After the pruning, the points with the largest first coordinate in the remaining points must also be on the skyline since they cannot be dominated by the other remaining points as well as the previous skyline points. The correctness follows by induction.

The algorithm will terminate after k rounds since there are k skyline points. The running time at each site consists of two parts: the computation of the local skylines and the prunings. The computation of local skyline at the Site i needs $O(n_i \log^{d-2} n_i + n_i \log n_i)$

[1] Up to a logarithmic factor which counts the number of bits used to represent a point.

Algorithm 1 Optimal($S_1, \ldots, S_s$)

Input: S_i ($i \in [s]$): the point set held by Site i
Output: the global skyline

1: Site i computes its local skyline points and discards the other points in S_i
2: **while** $\exists i \in [s]$ $s.t.$ $S_i \neq \emptyset$ **do**
3: **for** each i $s.t.$ $S_i \neq \emptyset$ **do**
4: Site i sends the point with the largest first coordinate to the coordinator
5: **end for**
6: The coordinator picks the point with the maximum first coordinate among the points received from all sites, and sends the new global skyline point to each site
7: **for** each i $s.t.$ $S_i \neq \emptyset$ **do**
8: Site i prunes S_i by the new global skyline point received from the coordinator
9: **end for**
10: **end while**

time [14], where $n_i = |S_i|$ is the number of points at the Site i and $d \geq 2$ is the dimension of dataset. The time used for pruning is $O(dkk_i)$ at the Site i where k_i is the number of points in the local skyline of Site i. At the coordinator, for each round we only need to compute the maximums over at most s points. Thus the total running time is bounded by $O(ks)$.

Theorem 2.1. *There exists an algorithm for computing the skyline on n points in the d-dimensional Euclidean space in the coordinator model with s sites that uses $O(ks)$ communication and k rounds, where k is the output size, that is, the number of points in the skyline. The total running time at the Site i is $O(n_i \log^{d-2} n_i + n_i \log n_i + dkk_i)$ where n_i and k_i are the number of points and the size of the local skyline at the Site i respectively, and that at the coordinator is $O(ks)$.*

2.2 Communication-Round Tradeoff for $d = 2$

In the previous section we have shown an algorithm with the optimal communication cost but using up to k rounds. On the other hand, there is a naive one-round algorithm but in the worst case it needs $\Omega(n)$ words of communication. The natural questions is:

> *Can we obtain a communication-round tradeoff to bridge the two extremes?*

We try to address this question by proposing an algorithm that allows the users to choose the number of the communication rounds in the computation. In this section we first show such a tradeoff result for 2-dimensional Euclidean space.

Theorem 2.2. *There exists an algorithm for computing the skyline on n points in the 2-dimensional Euclidean space in the coordinator model with s sites that uses r (≥ 3) rounds and $C = sk(n/s)^{1/\lceil r/2 \rceil}$ communication, where k is the output size, that is, the number of points in the skyline. The total running time at the Site i is $O(C/s + n_i \log n_i)$ where n_i is the number of points at the Site i, and the total running time at the coordinator is $O(C)$.*

We define the ϕ-quantile of a set S to be an element a such that at most $\phi |S|$ elements of S are smaller than a and at most $(1-\phi)|S|$ elements of S are greater than a. If an ϵ-approximation is allowed (denoted by (ϕ, ϵ)-quantile), then we can return any ϕ'-quantile of S such that $\phi - \epsilon \leq \phi' \leq \phi$.

Algorithm 2 Tradeoff-2D($S_1, \ldots, S_s, r$)

Input: S_i ($i \in [s]$): the point set held by Site i; r: user-chosen round budget
Output: the global skyline

1: Site i computes its local skyline points and discards the other points in S_i
2: $\ell \leftarrow 1$; $t \leftarrow \lceil r/2 \rceil$
3: **while** $(\ell \leq t-1) \wedge (\exists i \in [s]$ $s.t.$ $S_i \neq \emptyset)$ **do**
4: Let λ_ℓ be a parameter whose value is given in the analysis (Equation (5)). All sites and the coordinator jointly compute $(1/\lambda_\ell, 1/(2\lambda_\ell))$-quantiles according to the x-coordinates of points in $\bigcup_{i \in [s]} S_i$ ▷ the quantile points naturally partition the Euclidean plane to λ_ℓ vertical strips
5: **for** each i $s.t.$ $S_i \neq \emptyset$ **do**
6: Site i, for each non-empty strip, sends the point with the largest y-coordinate to the coordinator
7: **end for**
8: The coordinator, for each strip, finds the point with the largest y-coordinate among points received from sites; let Y_ℓ denote the set of these points among all strips
9: The coordinator computes new skyline points from Y_ℓ and sends them to each site
10: **for** each i $s.t.$ $S_i \neq \emptyset$ **do**
11: Site i prunes S_i by new global skyline points received from the coordinator
12: **end for**
13: $\ell \leftarrow \ell + 1$
14: **end while**
15: $\forall i \in [s]$, Site i sends S_i to the coordinator
16: The coordinator updates the global skyline using the new points received from sites

We call the first coordinate of a point in the 2-dimensional Euclidean space the x-coordinate, and the second the y-coordinate. Let $t = \lceil r/2 \rceil$. We describe our tradeoff algorithm in Algorithm 2. The algorithm again starts with a local skyline computation at each site. Similar to Algorithm 1, the rest of the tradeoff algorithm still proceeds in rounds. The main difference is that at each round, the parties (sites and the coordinator) first jointly compute $(1/\lambda, 1/(2\lambda))$-quantiles to partition the Euclidean plane to a set of at most λ vertical strips, and then instead of computing the point with the global maximum x-coordinate, the coordinator computes for each non-empty strip the point with the maximum y-coordinate by collecting information from the sites (Line 5-8); after that the parties use these points to compute new skyline points and prune each strip. We call the combination of computing the quantiles and maximum y-coordinates, and finding new skyline points and performing local pruning, one *step* of the computation. The algorithm runs for $(t-1)$ steps, and after that the sites simply send all the remaining points to the coordinator.

We now show the correctness of Algorithm 2, and analyze its costs. The high level intuition on the round efficiency of Algorithm 2 is that at each round, the point with the maximum y-coordinate

in each strip will either contribute to the global skyline or help to prune all the points in that strip. Compared with Algorithm 1, one can think that we are trying to prune the whole data set *in parallel*, that is, in each strip of the plane. This will reduce the round complexity at the cost of mildly increasing the total communication cost.

Correctness. The correctness of Algorithm 2 is straightforward: our skyline computation does not prune any point that is not dominated by others. Indeed, up to the $(t-1)$-th step (or, $(2t-2)$-th round), what Algorithm 2 does can be summarized as "sites send candidate global skyline points $\rightarrow$ the coordinator computes new global skyline points from these candidates $\rightarrow$ sites use new skyline points to prune their local datasets". At the $(2t-1)$-th round, sites just send all the remaining unpruned points to the coordinator so that we will not miss any skyline points.

Communication cost. We count the communication cost in two parts. The first part is the communication needed at the first $(t-1)$ steps, and the second part is the total number of remaining points at all sites after the $(t-1)$-th step, which will be sent to the coordinator all at once at the final round.

We first analyze the cost of computing quantiles at each step. We can compute $(\epsilon, \epsilon/2)$-quantiles using the following folklore algorithm: Site i (for all $i \in [s]$) sends the coordinator the exact $\epsilon/2$-quantiles Q_i of its local point set S_i. Using $\{Q_1, \ldots, Q_s\}$, the coordinator can answer quantile queries as follows: Given a query rank β, it returns the largest v satisfying

$$\beta - \textstyle\sum_{i\in[s]} rank_i(v) \geq 0,$$

where $rank_i(v) = n_i(v) \cdot (\epsilon/2 \cdot |S_i|)$, and $n_i(v)$ is the number of $\epsilon/2$-quantiles in Q_i that is smaller than v. It is easy to see that $0 \leq \beta - v \leq \epsilon/2 \cdot \sum_{i\in[s]} |S_i|$. The following lemma summarizes the communication cost of this algorithm.

Lemma 2.3. *There is an algorithm that computes $(\epsilon, \epsilon/2)$-quantiles in the coordinator model with s sites using one round and $O(s/\epsilon)$ communication.*

Thus the communication used for quantile computation can be bounded by $O(s\lambda_\ell)$ at step ℓ. The rest communication at each step includes sending local maximums at all strips and new skyline points, which can be bounded by $O(s\lambda_\ell)$ as well. To sum up the total communication in the first part is bounded by $O(s\sum_{\ell\in[t-1]}\lambda_\ell)$.

The rest of the analysis is devoted to the second part, that is, to bound the number of the remaining points after the $(t-1)$-th step.

We first assume that the output size k is known, and $\lambda_\ell = \lambda$ for all $\ell \in [t-1]$ will be chosen as a function of k (see Equation (4)). We will then show how to remove this assumption.

Let $Y_\ell \in [1, \lambda]$ denote the number of new skyline points we find at the ℓ-th step. Observe that in each strip, if the point with the largest y-coordinate is not a skyline point, then the rest of the points in that strip cannot be skyline points and thus are pruned. After the first step, there are at most Y_1 strips having point and each strip has at most $2n/\lambda$ points, so there are at most

$$Y_1 \cdot 2n/\lambda = 2nY_1/\lambda$$

points left. After the second step, there are at most Y_2 strips having point and each strip has at most $2(2nY_1/\lambda)/\lambda$ points, so there are at most

$$Y_2 \cdot 2\left(2nY_1/\lambda\right)/\lambda = 4nY_1Y_2/\lambda^2 \qquad (Y_1 + Y_2 \leq k)$$

points left. After the $(t-1)$-th step, there are at most

$$2^{t-1}n \prod_{\ell\in[t-1]} Y_\ell \Big/ \lambda^{t-1} \qquad \left(\sum_{\ell\in[t-1]} Y_\ell \leq k\right) \tag{1}$$

$$\leq \quad n\left(\frac{2k}{(t-1)\lambda}\right)^{t-1} \tag{2}$$

points left, where from (1) to (2) we have used the AM-GM inequality and the equality holds when all $Y_\ell = k/(t-1)$ $(\ell = 1, \ldots, t-1)$. We thus have at most $n(2k/((t-1)\lambda))^{t-1}$ points left at sites after $(t-1)$-th step, and the sites will send all of them in the final (i.e., $(2t-1)$-th) round. Adding two parts together, the total communication cost is bounded by

$$O(s\lambda(t-1)) + n\left(\frac{2k}{(t-1)\lambda}\right)^{t-1}. \tag{3}$$

When

$$\lambda = \frac{2k}{t-1} \cdot \left(\frac{n(t-1)}{2sk}\right)^{1/t}, \tag{4}$$

Expression (3) simplifies to be $O(sk^{(t-1)/t}(n/s)^{1/t})$.

Dealing with unknown k. We now show how to deal with the case that we do not know k at the beginning. A simple idea is to guess k as powers of 2 (i.e., $1, 2, 4, 8, \ldots$), and for each guess, we run our algorithm, and report error if $\sum Y_\ell > k$ at some point, in which case we double the value of k and rerun the algorithm. The correctness of the algorithm still holds. The round complexity, however, may blow up by a factor of $\log k$ in the worst case. We will show that there is a way to preserve the round complexity even when we do not know k at the beginning.

The new idea is to guess k progressively, based on the number of new skyline points found in the previous step. More precisely, we set the guess of k at the ℓ-th step $(\ell \geq 2)$ to be

$$k_\ell = Y_{\ell-1} \cdot (t-1),$$

and we set $k_1 = t-1$ to begin with. Now at the ℓ-th step we use

$$\lambda_\ell = \frac{2k_\ell}{t-1} \cdot \left(\frac{n(t-1)}{2s}\right)^{1/t} \tag{5}$$

strips for the pruning. Note that (5) is very similar to (4), where we have replaced the first k in (4) by k_ℓ and removed the second k in (4).

Similar to (2), after $(t-1)$-th step, there are at most

$$2^{t-1}n \prod_{\ell\in[t-1]} (Y_\ell/\lambda_\ell)$$

points left, and consequently the total communication cost is bounded by

$$s\sum_{\ell\in[t-1]} \lambda_\ell + 2^{t-1}n \prod_{\ell\in[t-1]} (Y_\ell/\lambda_\ell). \tag{6}$$

We now bound the two terms in (6) separately. We first have

$$s\sum_{\ell\in[t-1]} \lambda_\ell \quad = \quad \frac{2s\sum_{\ell\in[t-1]} k_\ell}{(t-1)} \cdot \left(\frac{n(t-1)}{2s}\right)^{1/t}$$

$$\leq \quad 2sk \cdot \left(\frac{n(t-1)}{2s}\right)^{1/t}, \tag{7}$$

where we have used the inequality

$$\sum_{\ell \in [t-1]} k_\ell = (t-1)\left(1 + \sum_{\ell \in [t-2]} Y_\ell\right) \leq (t-1)k.$$

For the second term, we have

$$\begin{aligned} 2^{t-1} n \prod_{\ell \in [t-1]} (Y_\ell / \lambda_\ell) &= n \frac{Y_1 Y_2 \cdots Y_{t-1}}{Y_1 Y_2 \cdots Y_{t-2}} \left(\frac{n(t-1)}{2s}\right)^{t/(t-1)} \\ &= n Y_{t-1} \cdot \frac{2s}{n(t-1)} \left(\frac{n(t-1)}{2s}\right)^{1/t} \\ &\leq \frac{2sk}{(t-1)} \left(\frac{n(t-1)}{2s}\right)^{1/t}. \end{aligned} \tag{8}$$

By (7) and (8), the total cost is bounded by

$$O(sk(n/s)^{1/t}) = O\left(sk(n/s)^{1/\lceil r/2 \rceil}\right),$$

as claimed.

Running time. The time cost at each site involves three parts: the computation of the local skyline, the computation of local quantiles, and the point prunings. Computing local skyline again cost $O(n_i \log n_i)$ where n_i is the number of points at Site i. The cost of point prunings can again be made linear in n_i for the same reason as that in Algorithm 1. Now we analyze the time cost of computing the local quantiles. Since points are sorted after the local skyline computation, computing (exact) local quantiles needs $O(\lambda_\ell)$ time at the ℓ-th step. Thus the total time is bounded by $\sum_{\ell \in [t-1]} \lambda_\ell = O(k(n/s)^{1/t}) = O(k(n/s)^{1/\lceil r/2 \rceil})$.

The running time at the coordinator also consists of three parts: the computation of approximate global quantiles at each step, the computation of new skyline points from the first step to the $(t-1)$-th step, and the computation of skyline points (output) at the end. The observation is that at each step, for each of the three tasks, the running at the coordinator can be asymptotically bounded by the number of points it receives from all sites in that step, and thus the total running time at the coordinator is asymptotically upper bounded by the total communication cost.

2.3 Communication-Round Tradeoff for All Dimensions

In this section we consider general dimension d. We first note that the approach in Section 2.2 does not work any more. For $d = 2$ we can partition the space into strips according to the x-coordinate (1-dimensional subspace) such that each strip has same amount of points. As a result, once a strip is pruned, a certain number of points are guaranteed to be pruned. However, for general d, we can not find such a geometric partition for the $(d-1)$-dimensional subspace such that each part has the same amount of points.

We thus propose a simple algorithm which can be thought as a modified version of Algorithm 1. The algorithm is described in Algorithm 3. At each of the first $r-1$ rounds, each site sends the (at most) d points with the maximum coordinates at each of the d dimensions. The coordinator then computes new skyline points from the points received from all sites, and sends new skyline points to each site for a local pruning. Finally at the r-th round, each site simply sends all the remaining unpruned points to the coordinator for finalizing the global skyline.

Algorithm 3 Tradeoff-general($S_1, S_2, \ldots, S_s, r$)

Input: S_i ($i \in [s]$): the point set held by Site i; r: user-chosen round budget
Output: the global skyline

1: Site i computes its local skyline points and discards the other points in S_i
2: $\ell \leftarrow 1$
3: **while** $(\ell \leq r-1) \wedge (\exists i \in [s]\ s.t.\ S_i \neq \emptyset)$ **do**
4: **for** each $i\ s.t.\ S_i \neq \emptyset$ **do**
5: Site i sends the points with the maximum coordinate in each dimension to the coordinator
6: **end for**
7: Among the points received from sites, the coordinator finds the points with global maximum coordinates in each dimension, adds them to the global skyline, and sends them to each site
8: **for** each $i\ s.t.\ S_i \neq \emptyset$ **do**
9: Site i prunes S_i by new global skyline points received from the coordinator
10: **end for**
11: $\ell \leftarrow \ell + 1$
12: **end while**
13: $\forall i \in [s]$, Site i sends S_i to the coordinator
14: The coordinator updates the global skyline using the new points received from sites

Note that the main difference between Algorithm 3 and Algorithm 1 is that in the first $r-1$ rounds, in the former each site may send up to d distinct points, while in the later each site only sends at most one point. On the other hand, in the worst case in both algorithms the coordinator only obtains one new skyline point. Therefore in the worst case Algorithm 3 may waste a factor of d in the communication cost. However, in practice, the worse case rarely happens and computing skyline from all dimensions makes the pruning faster.

Algorithm 3 also gives a communication-round tradeoff, and it works for any dimension $d \geq 2$. The correctness of Algorithm 3 is obvious: at the first $r-1$ rounds, it does the same thing as Algorithm 1 for each coordinate, and every pruned point is guaranteed to be dominated by the global skyline. In the last round sites just send all the remaining unpruned points to the coordinator. Regarding the communication cost, in the worst case the sites need to sent $\Omega(n)$ points to the coordinator for small round budget r; more precisely, the number of remaining points at sites could be $\Omega(n)$ after round $r-1$. But we will show in Section 4 that in practice Algorithm 3 is always communication efficient.

2.4 Discussions

In this section we propose some strategies to further improve the communication costs of our algorithms.

Avoid sending duplicated points. We observe that in Algorithm 1 the same point may be sent to the coordinator multiple times: If a point is not pruned at the current round, then it will still be a local

Algorithm 4 Sorted-2D($S_1, \ldots, S_s$)

Input: S_i: the point set held by Site i
Output: the global skyline

1: Site i computes its local skyline points and discards the other points in S_i
2: Site i sends the point with the largest y-coordinate, denoted by y_i, to the coordinator
3: The coordinator computes for $\forall i \in [s], z_i = \max\{y_{i+1}, \ldots, y_s\}$, and sends z_i to Site i
4: Site i prunes S_i using z_i, and sends the rest points to the coordinator
5: The coordinator updates the global skyline using the new points received from sites

maximum of the site in the next round. To avoid communicating duplicated points, the coordinator requests a new local maximum point from a site only when the previous point is pruned by the global skyline.

More efficient pruning. We notice that in many cases the coordinator does not need to send *all* the global skyline points to sites. Consider the following example in the 3-dimensional Euclidean space. Let $a = (0.6, 0.2, 0.5)$ be the local maximum point in the first dimension at Site 1. Suppose that after the first few rounds, the global skyline becomes $b = (0.9, 0.1, 0.1), c = (0.8, 0.4, 0.4), d = (0.7, 0.6, 0.6)$. Since a is dominated by d, we should send the global skyline to Site 1 for the pruning. The observation is that we only need to send d to Site 1 but not b and c, since d dominates b, c in the second and third dimensions.

Reducing the cost of quantile computation. In our experiments we found that the quantile computation is relatively expensive when the size of the skyline is small. In this case we just compute quantiles once, and then use the same quantiles at each round.

2.5 A 2-round algorithm for sorted datasets

Finally we would like to mention that in the 2-dimensional Euclidean space, if data points are partitioned to the s sites in a *sorted* order with respect to one of the two coordinates, then we can do much better.

Theorem 2.4. *For n points in the 2-dimensional Euclidean space partitioned among the s sites in the coordinator model in the sorted manner according to their x-coordinates or y-coordinates, there exists an algorithm for computing the skyline that uses $O(k + s)$ communication and 2 rounds, where k is the output size, that is, the number of points in the skyline. The total running time at Site i is $O(n_i \log n_i)$ where n_i is the number of points at Site i, and that at the coordinator is $O(k + s)$.*

Let us assume that points are sorted according to the x-coordinates. The algorithm is described in Algorithm 4. Each site first does a local pruning and computes its local skyline. In the first round, Site i sends the coordinator the point which has the largest y-coordinate, denoted by y_i. In the second round, the coordinator for each $i \in [s]$ computes the value z_i which is the maximum value among $\{y_{i+1}, \ldots, y_s\}$, and sends z_i to Site i. Site i then prunes all its local points with y-coordinate smaller than z_i, and sends the rest of its points to the coordinator.

To show the correctness of this algorithm, the claim is that the points in S_i with y-coordinate larger than z_i, denoted by P_i, must on the global skyline. Indeed, points in P_i cannot be dominated by any point in $S_1, \ldots, S_{i-1}$ since all points in S_i have x-coordinates larger than those points in $S_1, \ldots, S_{i-1}$. On the other hand, points in P_i cannot be dominated by any point in $S_{i+1}, \ldots, S_s$ since all points in P_i have y-coordinates larger than the those points in $S_{i+1}, \ldots, S_s$.

The communication of the algorithm includes sending y_i and z_i (costs $2s$) plus sending skyline points (costs k). The running time at each site is dominated by the local skyline computation, and the time cost at the coordinator is clearly $O(k + s)$ ($O(s)$ for the first round and $O(k)$ for the second round).

3 LOWER BOUNDS

In this section we provide two lower bound results to complement our proposed algorithms.

3.1 Infinite Rounds

We prove a lower bound for the infinite-round case by a reduction from a communication problem called s-DISJ. The lower bound matches the upper bound by Algorithm 1 up to a logarithmic factor which counts the number of bits used to represent a point in the Euclidean plane.

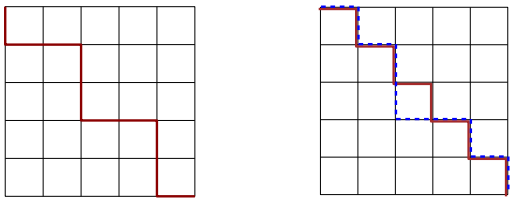
In s-DISJ, each of the s sites gets an m-bit vector. Let $X_i = (X_i^1, \ldots, X_i^m)$ be the vector the the i-th site gets. We can view the whole input as an $s \times m$ matrix X with X_i ($i \in [s]$) as rows. The s-DISJ problem is defined as follows:

$$s\text{-DISJ}(X_1, \ldots, X_s) = \begin{cases} 1, & \text{if there exists a } j \in [m] \text{ s.t.} \\ & \forall i \in [s], X_i^j = 1, \\ 0, & \text{otherwise.} \end{cases}$$

Lemma 3.1 ([4]). *Any randomized algorithm for s-DISJ that succeeds with probability 0.51 has communication cost $\Omega(sm)$. The lower bound holds even when we allow an infinite number of communication rounds.*

The Reduction. Given the m-bit vector X_i for s-DISJ, the i-th site first converts it to a $2m$-bit vector X_i' as follows: each 0 bit will be converted to 01, and each 1 bit will be converted to 10. For example, when $m = 5$ and $X_i = 10101$, X_i' should be 1001100110. The next step is to convert X_i' to a staircase. This step is illustrated in Figure 2 (a). We can "embed" the staircase into an $m \times m$ grid. The staircase starts from the top-left point of the grid, and grows in $2m$ steps. In the ℓ-th step, if the ℓ-th coordinate of X_i' is 0, then the staircase grows one step horizontally rightwards; otherwise if the ℓ-th coordinate is 1, then the staircase grows one step vertically downwards.

The observation is that if we create s staircases using $X_1', \ldots, X_s'$, then the skyline of the union of these s staircases is closely related to the value of s-DISJ$(X_1, \ldots, X_s)$: If s-DISJ$(X_1, \ldots, X_s) = 0$, then the skyline will be in the form of the red curve in Figure 2 (b); otherwise, the skyline will be different from the red curve (e.g., be the blue curve in Figure 2 (b) if the 3rd coordinates of $X_1, \ldots, X_s$ are all 1). This is because for each column $j \in [m]$ in the grid, as long as there is one $i \in [s]$ such that the j-th coordinate of X_i is 0, or the $(2j-1)$-th and $(2j)$-th coordinates of X_i' is 01, the skyline within the j-th column of the grid will be like "$\urcorner$"; otherwise if for



(a) (b)

Figure 2: (a)Translating vector $X_1' = 1001100110$ ($m = 5$) to a staircase. (b) The solid red skyline corresponds to the case that s-DISJ$(X_1, \ldots, X_s) = 0$, and the dash blue skyline corresponds to the case that s-DISJ$(X_1, \ldots, X_s) = 1$.

all $i \in [s]$ the j-th coordinate of X_i is 0, then the skyline within the j-th column of the grid will be like "$\llcorner$". The other direction also holds, that is, if the skyline is in the form of the red curve, then s-DISJ$(X_1, \ldots, X_s) = 0$; otherwise, s-DISJ$(X_1, \ldots, X_s) = 1$.

When the size of the skyline is k, we set $m = k$ according to our reduction, and obtain the following theorem.

Theorem 3.2. *Any randomized protocol for computing skyline in the coordinator model that succeeds with probability* 0.51 *has communication cost* $\Omega(sk)$ *bits, where* k *is number of points in the skyline. The lower bound holds even when we allow an infinite number of communication rounds.*

3.2 One Round

We now prove an $\Omega(n)$ communication lower bound for the case that the algorithm needs to finish in one round. This proof is simpler than the infinite-round case – we only need two sites to participate in the game. We assign Site 1 an m-bit vector u, which can be converted to a staircase in the $m \times m$ grid just like the infinite-round case, and assign Site 2 one bit v, which is translated to the upper-right corner point of the grid if $v = 1$ and the lower-left corner point of the grid if $v = 0$. We have the following simple observation, which holds because if we change one bit of the vector m from 0 to 1, we will change a "$\urcorner$" to "$\llcorner$" in the staircase, and thus change the skyline; same for replacing a bit 1 to 0.

Observation 1. *If the coordinator needs to compute the global skyline, and* $v = 0$*, then it needs to learn the vector* u *exactly.*

We immediately have the following lemma.

Lemma 3.3. *Any one round randomized algorithm for the coordinator to learn* u *exactly with probability* 0.51 *has communication cost* $\Omega(m)$.

Note that when $v = 1$, the skyline only consists of a single point in the upper right corner. Therefore the output size k is not directly related to the value m; we thus can set $m = \Omega(n)$.

Theorem 3.4. *Any one round randomized algorithm for computing skyline in the coordinator model that succeeds with probability* 0.51 *has communication cost* $\Omega(n)$ *bits, where* n *is the total number of points held by all sites.*

4 EXPERIMENTS

4.1 The setup

In this section we present the experimental studies of our proposed algorithms. We have implemented all algorithms in C++. All experiments were conducted on a Dell PowerEdge T630 server with 2 Intel Xeon E5-2667 v4 3.2GHz CPU with 8 cores each, and 256GB memory.

The Datasets and Partition. We generated three synthetic datasets following the standard literature [3]: Anti-correlated (`ANTI`), Independent (`INDE`), Correlated (`CORR`). As recommended in [27], in order to simulate the distribution differences among sites, in each site we distribute points within a bounding rectangle whose projection on the i-th dimension ($i \in [1, d]$) is an interval $[\alpha_i, \beta_i]$, where α_i, β_i are randomly chosen from $[0, 1]$, adhering to the constraint $\beta_i - \alpha_i \geq 0.95$. The parameters *number of points in each site*, *number of sites*, and *dimension* used in our experiments are described in the following table. We will use the parameters with bold font by default.

Parameter	Values
#Points in each site	$\{20k, 40k, \mathbf{60k}, 80k, 100k\}$
#Sites	$\{20, 40, \mathbf{60}, 80, 100\}$
Dimension	$\{2, \mathbf{3}, 4, 5\}$

Table 1: Parameters of synthetic datasets

We make use of the following real-world datasets in experiments for $d = 2, 3$.

- `Household` [15]: this dataset contains 2 million household electric power consumption records gathered between December 2006 and November 2010. We choose *voltage* and *intensity* as the two attributes for $d = 2$, and add *global active power* as the third attribute for $d = 3$. The skyline points represent those households that are recommended to pay attention to the energy efficiency. We partition the data collected in every two consecutive months to the same site; we thus have 24 sites.
- `Airline`: this dataset contains 1.2 million airline on-time performance records between 30 U.S. major cities in the 2016 fourth quarter.[2] We choose (minus) *departure delay time* and (minus) *arrival delay time* as the two attributes for $d = 2$, and add (minus) *taxi out time* (time between a flight leaving the departure gate and taking off) as the third attribute for $d = 3$. The skyline points are considered to be on-time flights. We partition the flights departing from the same city to the same site; we thus have 30 sites.
- `Covertype` [15]: this dataset contains 500 thousand natural statistics from four wilderness areas located in the Roosevelt National Forest of northern Colorado. We choose *elevation* and *slope* as the two attributes for $d = 2$, and add *horizontal distance to hydrology* as the third attribute for $d = 3$. The skyline points represent areas that may have interesting geological behaviors. We randomly partition the data to 20 sites.

[2] Available at http://www.transtats.bts.gov.

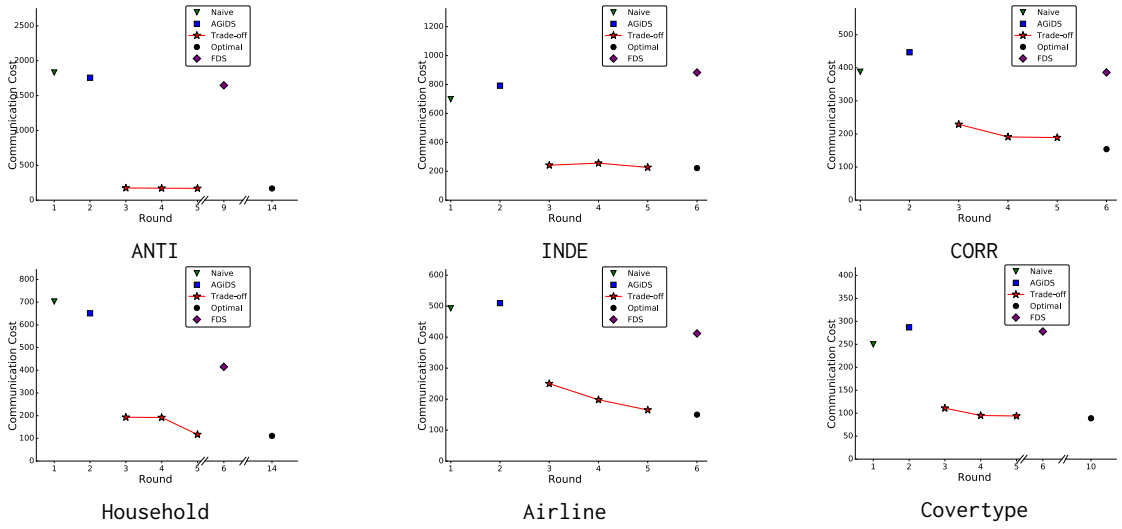


Figure 3: Communication cost vs round in 2-dimension

Algorithms. We compare the following algorithms in our experiments.

- `Naive`: The single round algorithm: Each site computes and sends its local skyline to the coordinator for a merge.
- `Optimal`: Algorithm 1 in Section 2.1. This algorithm achieves the optimal communication cost.
- `Tradeoff`: Algorithm 2 in Section 2.2 for $d = 2$, and Algorithm 3 in Section 2.3 for $d > 2$. This algorithm gives a smooth communication-round tradeoff. We note that if the round budget r is too large, then `Tradeoff` may terminate before using up the round budget.
- `AGiDS`: We use the `AGiDS` algorithm proposed in [18] as a comparison. To make `AGiDS` fit in our model, we use the following version of the original algorithm: At the beginning, the coordinator and sites share the information of a grid in which each cell represents a range in all axes (equal width partition). In the first round (the *planing phase*), sites send the information of non-empty cells to the coordinator. In the second round (the *execution phase*), the coordinator finds the cells that may contribute to global skyline and sends the information to sites, and then sites send points in these cells to the coordinator. Finally the coordinator computes the skyline of received points as the output. We choose the parameter *number of cells* to be 1024 in our experiments as recommended in [18].
- `FDS`: We use the `FDS` algorithm proposed in [27] as a comparison. The original algorithm proceeds in *iterations*. To make it fit in our model, we use three rounds for each iteration: In the first round (the *voluntary phase*), each site sends the top κ points with the largest scores (the sum of all coordinate values) to the coordinator. In the second round (the *compulsory and computation phase*), the coordinator calculates the minimum score (denoted by $F_{\min}$) from received points and sends it to each site. Each site then sends all its local points that have larger scores than $F_{\min}$ to the coordinator. The coordinator updates the global skyline with points received in the first two rounds. In the third round (the *feedback phase*), the coordinator calculates and sends each site a feedback, which consists of points that are guaranteed to dominate at least ℓ points in that site. And then each site does a local pruning. κ and ℓ are two parameters in `FDS`; we choose the optimal values $\kappa = 1$ and $\ell = 1$ as reported in [27].

Measurements. We make the following two measurements in the experiments: *communication cost* and *time*. We use the number of data points transmitted between coordinator and sites to represent *communication cost*. If an algorithm requires to send additional information other than data points (such as the quantiles in Algorithm 2; minimum score and ℓ-NN distance of points in `FDS`), for each number sent, we add $1/d$ to the communication cost. The *time* is the total time usage from the moment that the coordinator initializes a query to the end of the query, which is also referred as *response time* or *network delay* in other literatures. We assume that all sites have already computed the local skyline before the query start, since the local skyline computing needs same amount of time for all algorithms.

4.2 Results and Discussions

Figure 3 and Figure 4 show the communication and round costs of the algorithms on the three synthetic datasets and three real-world datasets for $d = 2, 3$.

Compared with `Optimal`, `Tradeoff` achieves the almost same communication cost with a much smaller round cost. For $d = 2$, we observed that by using five rounds, the communication cost of `Tradeoff` is 10%-48% of that of `Naive` on synthetic datasets and 16%-37% on real-world datasets. For $d = 3$, we observed that by using ten to twenty rounds, the communication cost of `Tradeoff` is 15%-44% of that of `Naive` on synthetic datasets and 12%-46% on real-world datasets. We observed that the advantage of `Tradeoff` against `Naive` is larger on `ANTI` and `INDE` datasets.

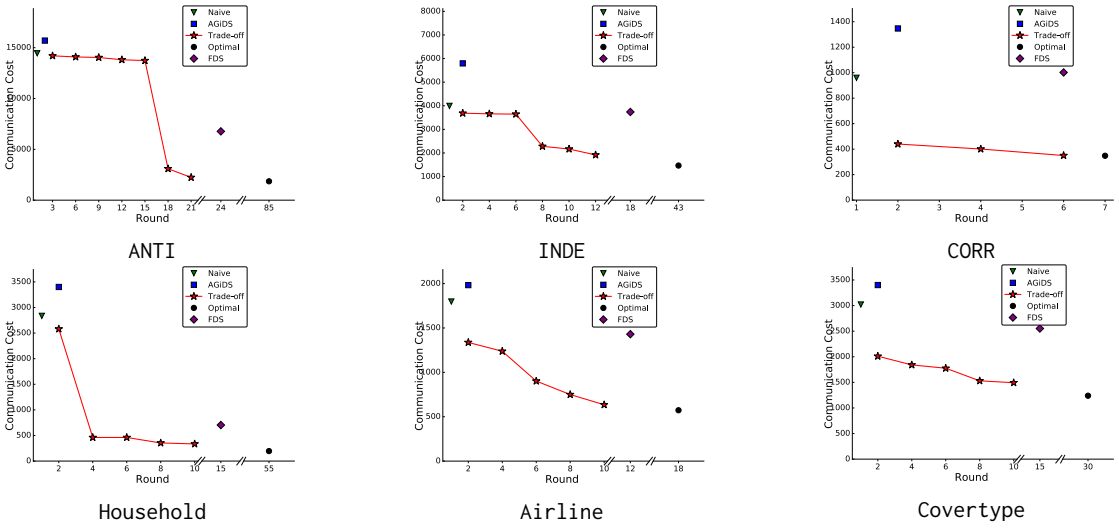


Figure 4: Communication cost vs round in 3-dimension

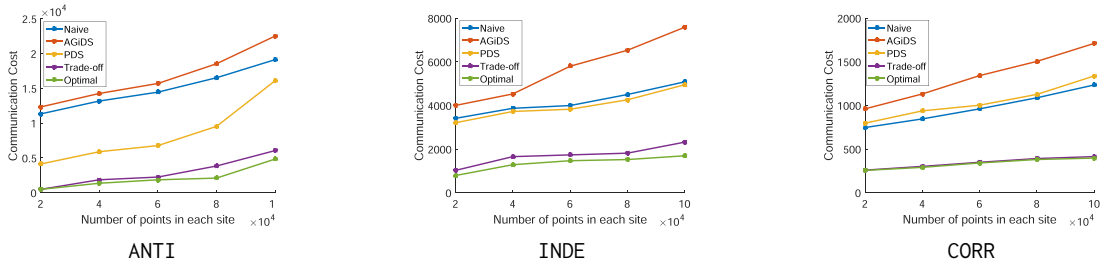


Figure 5: Communication cost vs number of points in each site

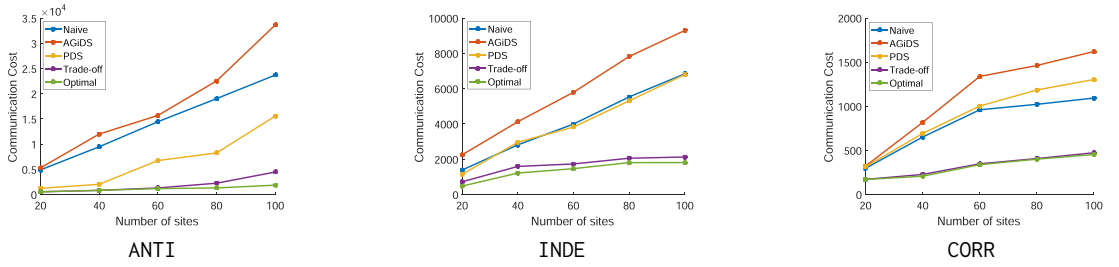


Figure 6: Communication cost vs number of sites

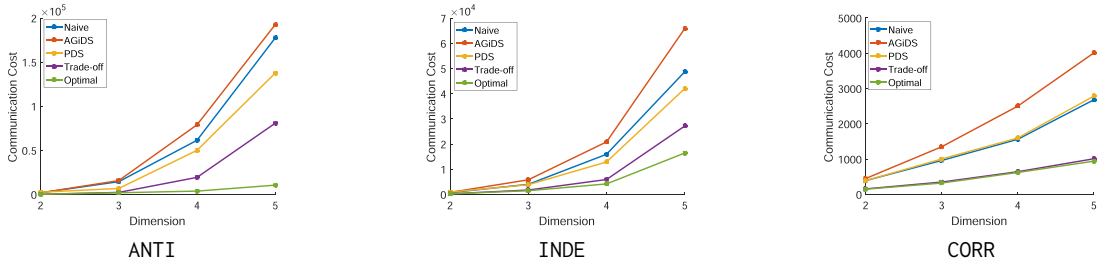


Figure 7: Communication cost vs data dimension

The communication cost of AGiDS is much larger than Naive in the most cases. One reason may be that the exchange of the information of cells is communication expensive.

FDS falls short on both communication and round costs compared with Tradeoff; sometimes its communication cost is even

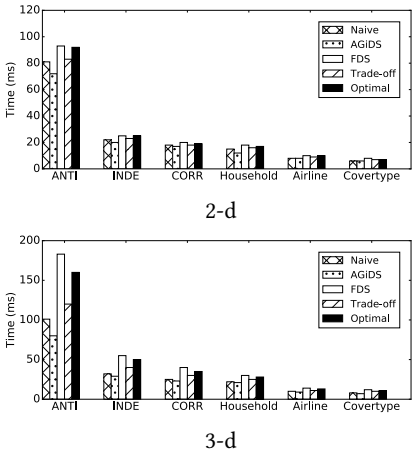


Figure 8: Running time

worse than Naive. It is clear that using 3 rounds for an iteration is not round efficient. Moreover, FDS exchanges a lot of additional information such as minimum score and ℓ-NN distance of points which makes it communication inefficient. We note that these additional information were *not* counted as communication cost in the experiments of [27], and this is why our results are a bit different from the results in [27].

Figure 5, Figure 6 and Figure 7 show how the communication cost changes with respect to the *number of points at each site*, *number of sites* and *dimension*. For Tradeoff, we set the round budget to be $r = 5$ when $d = 2$, and $r = 20$ when $d > 2$. We observed that the communication costs of all algorithms increase when these parameters increase. Tradeoff scales well with all the parameters and outperforms FDS and AGiDS in all circumstances. We also observed that the communication cost increases rapidly when the dimension of the data increases; this is because the number of skyline points increases significantly when the dimension increases.

Figure 8 shows the running time of the tested algorithms (excluding the cost of local skyline computation) for $d = 2, 3$. Generally speaking the running time of all algorithms are similar.

In summary, Tradeoff achieves noticeable communication cost reductions than AGiDS, FDS and Naive by using a small number of rounds; its performance is very close to the theoretically optimal algorithm Optimal in the communication cost but is much more efficient in rounds. On the other hand, the performance of AGiDS and FDS are clearly dominated by other algorithms. All algorithms have similar time costs.

5 CONCLUSION

In this paper we propose a set of algorithms for computing skylines on distributed data. We first give an algorithm that achieves the optimal communication cost. We also propose two algorithms with communication-round tradeoffs. We show experimentally that our algorithms significantly outperform existing heuristics in the communication cost and/or round cost.

REFERENCES

[1] F. N. Afrati, P. Koutris, D. Suciu, and J. D. Ullman. Parallel skyline queries. *Theory Comput. Syst.*, 57(4):1008–1037, 2015.

[2] W. Balke, U. Güntzer, and J. X. Zheng. Efficient distributed skylining for web information systems. In *EDBT*, pages 256–273, 2004.

[3] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[4] M. Braverman, F. Ellen, R. Oshman, T. Pitassi, and V. Vaikuntanathan. A tight bound for set disjointness in the message-passing model. In *FOCS*, pages 668–677, 2013.

[5] L. Chen, B. Cui, and H. Lu. Constrained skyline query processing against distributed data sites. *IEEE Trans. Knowl. Data Eng.*, 23(2):204–217, 2011.

[6] L. Chen, B. Cui, H. Lu, L. Xu, and Q. Xu. isky: Efficient and progressive skyline computing in a structured P2P network. In *28th IEEE International Conference on Distributed Computing Systems (ICDCS 2008), 17-20 June 2008, Beijing, China*, pages 160–167, 2008.

[7] J. Chomicki, P. Ciaccia, and N. Meneghetti. Skyline queries, front and back. *SIGMOD Record*, 42(3):6–18, 2013.

[8] K. Fotiadou and E. Pitoura. BITPEER: continuous subspace skyline computation with distributed bitmap indexes. In *Proceedings of the 2008 International Workshop on Data Management in Peer-to-Peer Systems, DaMaP 2008, Nantes, France, March 25, 2008*, pages 35–42, 2008.

[9] H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. In *STOC*, pages 135–143, 1984.

[10] K. Hose, C. Lemke, and K. Sattler. Processing relaxed skylines in PDMS using distributed data summaries. In *Proceedings of the 2006 ACM CIKM International Conference on Information and Knowledge Management, Arlington, Virginia, USA, November 6-11, 2006*, pages 425–434, 2006.

[11] K. Hose and A. Vlachou. A survey of skyline processing in highly distributed environments. *VLDB J.*, 21(3):359–384, 2012.

[12] Z. Huang, C. S. Jensen, H. Lu, and B. C. Ooi. Skyline queries against mobile lightweight devices in manets. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, page 66, 2006.

[13] D. G. Kirkpatrick and R. Seidel. Output-size sensitive algorithms for finding maximal vectors. In *SoCG*, pages 89–96, 1985.

[14] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *J. ACM*, 22(4):469–476, 1975.

[15] M. Lichman. UCI machine learning repository, 2013.

[16] J. Liu, L. Xiong, and X. Xu. Faster output-sensitive skyline computation algorithm. *Inf. Process. Lett.*, 114(12):710–713, 2014.

[17] K. Mullesgaard, J. L. Pederseny, H. Lu, and Y. Zhou. Efficient skyline computation in mapreduce. In *EDBT*, pages 37–48, 2014.

[18] J. B. Rocha-Junior, A. Vlachou, C. Doulkeridis, and K. Nørvåg. Agids: A grid-based strategy for distributed skyline query processing. In *Data Management in Grid and Peer-to-Peer Systems, Second International Conference*, pages 12–23, 2009.

[19] J. B. Rocha-Junior, A. Vlachou, C. Doulkeridis, and K. Nørvåg. Efficient execution plans for distributed skyline query processing. In *EDBT 2011, 14th International Conference on Extending Database Technology, Uppsala, Sweden, March 21-24, 2011, Proceedings*, pages 271–282, 2011.

[20] A. Vlachou, C. Doulkeridis, and Y. Kotidis. Angle-based space partitioning for efficient parallel skyline computation. In *SIGMOD*, pages 227–238, 2008.

[21] A. Vlachou, C. Doulkeridis, Y. Kotidis, and M. Vazirgiannis. SKYPEER: efficient subspace skyline computation over distributed data. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 416–425, 2007.

[22] A. Vlachou, C. Doulkeridis, Y. Kotidis, and M. Vazirgiannis. Efficient routing of subspace skyline queries over highly distributed data. *IEEE Trans. Knowl. Data Eng.*, 22(12):1694–1708, 2010.

[23] A. Vlachou and K. Nørvåg. Bandwidth-constrained distributed skyline computation. In *Eighth ACM International Workshop on Data Engineering for Wireless and Mobile Access, Mobide 2009, June 29, 2009, Providence, Rhode Island, USA, Proceedings*, pages 17–24, 2009.

[24] S. Wang, B. C. Ooi, A. K. H. Tung, and L. Xu. Efficient skyline query processing on peer-to-peer networks. In *ICDE*, pages 1126–1135, 2007.

[25] S. Wang, Q. H. Vu, B. C. Ooi, A. K. H. Tung, and L. Xu. Skyframe: a framework for skyline query processing in peer-to-peer systems. *VLDB J.*, 18(1):345–362, 2009.

[26] P. Wu, C. Zhang, Y. Feng, B. Y. Zhao, D. Agrawal, and A. El Abbadi. Parallelizing skyline queries for scalable distribution. In *EDBT*, pages 112–130, 2006.

[27] L. Zhu, Y. Tao, and S. Zhou. Distributed skyline retrieval with low bandwidth consumption. *IEEE Trans. Knowl. Data Eng.*, 21(3):384–400, 2009.