Efficient Algorithms for Constrained Subspace Skyline Query in Structured Peer-to-Peer Systems

Khaled M. Banafaa and Ruixuan Li

School of Computer Science and Technology,
Huazhong University of Science and Technology,
Wuhan, Hubei 430074, P.R. China
kbanafaa@smail.hust.edu.cn, rxli@hust.edu.cn

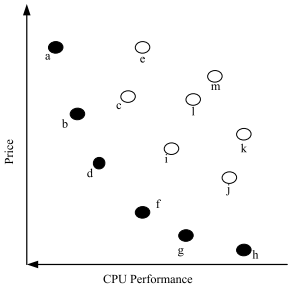
Abstract. To avoid complex P2P architectures, some previous research studies on skyline queries have converted multi-dimensional data into a single index. Their indexing, however, does not solve constrained subspace queries efficiently. In this paper, we have devised algorithms and techniques to solve *constrained subspace skyline queries* efficiently in *a structured peer-to-peer architecture* using a single index. Dataspace is horizontally partitioned; and peers are given Z-order addresses. Each subspace query traverses peers using the *subspace Z-order filling curve*. Such partitioning and traversal approaches allow parallelism for incomparable data as well as the use of some techniques to reduce the data traveled in the network. The order of traversal also preserves progressiveness. Our experiments, applied on Chord [1], have shown a reduction in the number of traversed peers and an efficient usage of bandwidth.

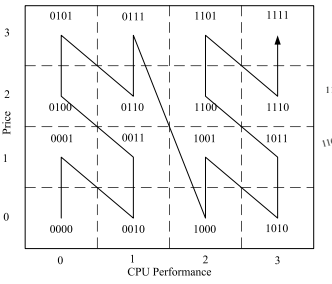
Keywords: Skyline query, peer-to-peer system, subspace skyline, constrained.

1 Introduction

Due to the huge available distributed data, advanced queries, such as skyline queries, identify some interesting data objects for the users. Peer-to-peer (P2P) networks have also become very popular for storing, sharing and querying data. Due to some application requirements for multi-dimensional data, P2P had to be adapted to host such kind of data. To cope with this requirements, complex network overlay architectures have been suggested. To maintain simple P2P architectures, researchers have looked for ways to convert multi-dimensional data to a single dimensional index (1D) for different applications. Skyline operator [2] has also attracted considerable attention in the database research community recently. A user may be interested in purchasing a computer on the web. Agents (peers) may have different specifications with different prices. As an example of CPU performance versus price of computers is shown Fig. 1. Other attributes will also affect the price (eg. HDD and RAM). A purchaser will definitely not be interested in a computer while there is a same or a better one with a better or

H. Gao et al. (Eds.): WAIM 2012, LNCS 7418, pp. 334–345, 2012.





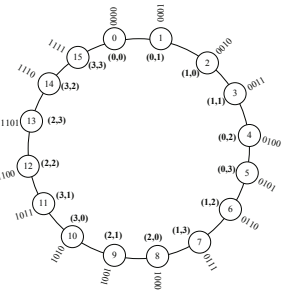


Fig. 1. Skyline **Fig. 2.** Space partitioning **Fig. 3.** Chord: Z-order

the same price. Purchasers may, however, have different preferences or requirements. A purchaser may be interested in a high capacity HDD; another may be interested in CPU performance. Answers to such preferences are called skyline [2] and [3]. Black filled points are called skyline in Fig. 1.

A purchaser may have ranges or constraints on the attributes. Price ranges and RAM sizes are some examples of constraints. Researches (eg. DSL [4], SkyPlan [5] and PaDSkyline [6]) have studied such constraints. These approaches, however, requires the query node to contact all nodes for their minimum bounding rectangles (MBRs) before building a plan. Other studies (e.g. SUBSKY [7] and Skypeer [8]) considered subspace skylines. Those studies, however, do not consider constraints due to their subspace pre-processing.

Our motivation is to efficiently process subspace queries [8] with constraints [5]. In high dimensional databases, a user may be interested only in some attributes with some constraints. Considering all attributes might return a huge number of data. A purchaser may be interested in a price within a range ($500-$800) and a minimum CPU performance (1.5GHz). They may not be interested in other attributes (eg. HDD, memory, video cards, ... etc). Such queries have been considered in centralized systems [3] using R-trees. Those methods are, however, not applicable in distributed systems for their requirements of having all data in one place.

In this paper, we aim at solving constrained subspace skyline queries in structured P2P systems using a single index. By partitioning the dataspace horizontally, Z-address scheme is used to address partitions as well as peers. Depending on query subspace and constraints, peers are traversed. Our contributions are:

- By Z-order addressing a horizontally partitioned data space on a simple, stable overlay structure such as Chord [1], progressive algorithms for *constrained subspace query* traversal are suggested. They increase the pruning ability.
- Incomparable partitions are exploited by parallelizing our algorithms.
- Experiments have shown that such approaches resulted in reduction in traversed peers and bandwidth usage while preserving progressiveness.

This paper is organized as follows. Related work is first discussed in Section 2. The main part of our work (partitioning data space, traversal algorithms and the used techniques) is discussed in Section 3. Section 4 discusses experiments and findings of this work. We end up our paper by conclusion in Section 5.

2 Related Work

Skyline was first introduced to database by Börzsönyi in [2]. All points are compared using a window in Block Nested Loop(BNL). BBS[9] is an optimal and progressive algorithm for skyline queries. It is based on nearest neighbor search. It used a heap to gets rid of duplicates. In [10], assuming integers, RZ-regions and ZB-tree are used. The above algorithms are not efficient for distributed and P2P systems for their centralized requirements.

In distributed systems, such as [11], data are distributed vertically. A round-robin is used to get the traditional skyline on the presorted attributes. Constrained queries are neither supported for fullspace nor for subspace. In DSL [4], data space is partitioned horizontally. It does not, however, support subspace and constrained subspace skyline. SkyFrame uses greedy and relaxed skyline search on a balanced tree structure for peer-to-peer networks (Baton) [12]. Even though it supports constrained skyline, it does not support subspace and constrained subspace queries. In PaDSkyline [6], the querying peer collects MBRs from other peers. A plan is made for incomparable peers to work in parallel. In SkyPlan [5], using weighted edges, plans are also mapped. The maximum spanning trees are used to maximize the pruning ability. In FDS [13], iterations and feedback are carried out between the coordinator and the other nodes. The above studies [5], [6], and [13] require the querying peer to contact all nodes. Some algorithms [14] have converted multi-dimensional data into a single-data index and adapted it into P2P. However, they support neither constrained queries nor subspace queries.

Subspaces have also been studied in both centralized and distributed systems. Centralized algorithms [7], [15], [16], and [17] and use different approaches. Sky-Cube [15], [16] precomputes all possible subspaces skylines exploiting various sharing strategies using bottom-up and top-down methods. SUBSKY [7] builds B+ tree by converting the multi-dimensional data into one-dimensional anchors. Authors in[17] used materialization and proposed the maximal space index to answer subspaces in a high dimensional data. Due to their pre-computation, updating any point may result in rebuilding the solutions. They are centralized and do not support constrained subspace queries.

Skypeer [8] and DCM [18] are meant for distributed systems. Skypeer uses *extended skylines* on a super-peer architecture. A querying peer submits its subspace query to its super-peer. The super-peer, then, contacts the other super-peers for subspace skyline. Skypeer is nonprogressive and does not support constrained subspace query. In DCM, the results of subspaces queries are cached in peers and indexed using a distributed cache index (DCI) on Baton or Chord. Subspace queries are then forwarded to cached skylines using the DCI. Even though it does not consider constrained subspace skylines, this work is orthogonal to our work for unconstrained subspaces.

Even though work in [19] (we call Chordsky) supports constrained full-dimension skyline in a Chord structure. ChordSky's usage of sum for the the monotonic functions makes it inefficient for subspace skyline queries. This is due to the disturbance of the monotonic order when a subspace is considered. The authors' objectives, however, were having a simple stable structure like Chord to answer constrained skyline queries.

We have the same objectives as ChordSky but for a more general skyline query (i.e. constrained subspace skyline). Thus, we consider ChordSky as a baseline for our work. We first use it with no modification (i.e. for constrained full-dimension skyline). We, then, modified it to support subspace. Our work has shown to be more efficient. Even though ChordSky is progressive for full-dimension queries, it is not for subspace skyline. Our work is, however, progressive for both constrained full-space as well as constrained subspace skyline. A Z-order structure [10] inspired our work. Our system uses the idea of Z-order on P2P architecture. It uses a one-dimensional index. We have applied our work on Chord [1] but it can also be applied on other architecture like Baton.

To the best of our knowledge, constrained subspace skyline has not been considered in distributed and P2P systems. Our aim is to minimize the visited peers and data transferred in the network while preserving progressiveness.

3 Constrained Subspace Skyline

In this section, we first give some formalization to constrained subspace skyline in Section 3.1. Data space partitioning and traversal techniques and algorithms are discussed in Section 3.2 and Section 3.3 respectively. Parallelism of our algorithms and load balancing are explored in Section 3.4.

3.1 Preliminaries

Without loss of generality, we assume minimum values of attributes are preferred (e.g. cheaper is preferred to expensive, near is preferred to far, etc). For maximum values preferences, the inverse of the values can be used. Let $S = \{d_1, d_2, ..., d_d\}$ be a d-dimensional space and PS be a set of points in S. A point $p \in PS$ can be represented as $p = \{p_1, p_2, ..., p_d\}$ where every p_i is a value on dimension d_i. Each non-empty subset S' of S ($S' \subseteq S$) is called a subspace. A point $p \in PS$ is said to dominate another point $q \in PS$ on subspace S' (denoted as $p \prec_{S'} q$) if (1) on every dimension $d_i \in S', p_i \leq q_i$; and (2) on at least one dimension $d_j \in S', p_j < q_j$. The skyline of a space $S' \subseteq S$ is a set $PS' \subseteq PS$ of so-called skyline points which are not dominated by any other point of space S'. That is, $PS' = \{p \in PS | \nexists q \in PS : q \prec_{S'} p\}$.

Let $C = \{c_1, c_2, ..., c_k\}$ be a set of range constraints on a subspace $S' = \{d'_1, d'_2, ..., d'_k\}$ where $k \leq d$. Each c_i is expressed by $[c_{i,min}, c_{i,max}]$, where $c_{i,min} \leq c_{i,max}$, representing the min and max value of d'_i. A constrained subspace skyline of a space $S' \subseteq S$ refers to the set of points $PS'_c = \{p \in PS_c | \nexists q \in PS_c : q \prec_{S'} p\}$, where $PS_c \subseteq PS$ and $PS_c = \{p \in PS | \forall d_i \in S' : c_{i,min} \leq p_i \leq c_{i,max}\}$.

3.2 Partitioning and Assignment

We use a shared-nothing architecture (SN) where the data space is horizontally partitioned; and each peer is assigned a partition. Such architecture has also been used in other previous works like [19].

As in [14], the range of each attribute is assumed to be normalized into [0,1]. Each dimension i (i.e. $1 \leq i \leq d$ of the data space is divided into k_i equal partitions. Each partition is assigned an integer number within $[0, k_i - 1]$ in an ascending order starting from 0. The whole space is, thus, divided into a grid of $\prod k_i$ cells. The lower left cell is $Cell(0, 0, 0, 0)$ while the upper right cell is $Cell(k_0 - 1, k_1 - 1, k_2 - 1, ..., k_d - 1)$ as shown if Fig. 2.

A cells' Z-addresses are obtained by interleaving their dimensions' values. Peers are also assigned a Z-order address using the same grid above. In Chord, each peer is assigned the next Z-order address starting with '0' as in Fig. 3.

A data point obtains its value for each dimension using Equation 1. The Z-address of a peer responsible of a point is found by interleaving these values.

$$IntValue(d_i) = \lfloor p(d_i) * k \rfloor \tag{1}$$

3.3 Skyline Query Traversal

Fig. 2 and Fig. 3 demonstrate the previous running (computer) example of a 2-dimension data space where $k_i = 4$.

Since dataspace has been partitioned and assigned to peers, query traversal needs exploit such partitioning. Before explaining our traversal and prunability, some definitions are introduced to show the domination relation between cells and points using lower left point (LLP) and upper right point(URP):

Definition 1. *A point p completely(partially) dominates a cell α if p dominates α's LLP(URP).*

Definition 2. *Cell Domination. A cell α completely or partially dominates another cell β if α's URP or α's LLP dominates β's LLP respectively.*

Property 1. Monotonic Ordering. Cells ordered by non-descending Z-addresses are monotonic such that cells are always placed before their (completely and partially) dominated cells.

Constrained Fullspace Skyline Queries. For a constrained fullspace skyline queries, the querying peer sends its query to the peer responsible of the minimum constraints C_{min}. By Property 1, a traversal is straight forward. Each peer calculates its and previous peers' constrained skyline and sends it to the next unpruned peer. An unpruned peer is a peer within the constraints and is not completely dominated by any discovered constrained skyline point. Chord nodes can also prune empty nodes by maintaining an empty nodes list in each node.

Algorithm_1-(Fullspace)

```
1: Input:
2: RS: Received Skyline
3: Output:
4: DS: Discovered Skyline
5: BEGIN
6: LS: Constrained Local
   Skyline Points
7: DS = φ
8: for all P ∈ LS do
9:    if ∄Q ∈ RS : Q ≺ P then
10:      DS = DS ∪ P;
11:   end if
12: end for
13: Send DS to querying
    peer as final skyline points
14: RS = RS ∪ DS
15: Send RS to next
    unpruned peer
16: END
```

Algorithm_2-(CSSA)

```
1: Input:
2: RFS: Rcvd Final Sky Pts
3: RGS: Rcvd Group Sky Pts
4: Output:
5: DS: Discovered Skyline
6: BEGIN
7: LS: Constrained Local Sky Pts
8: LS = findSkyline(LS ∪ RGS)
9: DS = φ
10: for all P ∈ LS do
11:    if ∄Q ∈ RFS : Q ≺ P then
12:       DS = DS ∪ P;
13:    end if
14: end for
15: if (Last-peer-of-subspace-group) then
16:    Send DS to query peer as a final sky
17:    RFS = RFS ∪ DS
18:    Send RFS to first unpruned peer in SG
19: else
20:    RGS = DS
21:    Send RFS and RGS to next peer in SG
22: end if
23: END
```

Fig. 4. Constrained full space vs. constrained subspace skyline algorithms

Constrained Subspace Skyline Queries (CSSA). A more general skyline query type of the above query is *constrained subspace skyline queries* . Some issues are, therefore, reconsidered. First, the starting and ending peers need be calculated. Second, some peers partially dominate each other with respect to the query subspace. This disturbs progressiveness. A way to preserve progressiveness as much as possible needs be used. Third, in each step, the next peers needs be clear and deterministic. Last, peer's pruning must be reconfigured for subspaces.

Starting and ending peers: To get their Z-addresses, the minimum and maximum values are placed in Equation 1 for the missing dimensions respectively.

Peers partial domination: Peers partially dominate each other with respect to the queried subspace if they have the same subspace Z-address. Thus, grouping those peers with the same subspace Z-address value creates subspace-groups.

Definition 3. *A subspace-group (SG) is all peers with the same IntValues (Equation 1) in all dimensions of the subspace.*

Peers in a subspace-group partially dominate each other. Thus the data transferred within the subspace-group may have false skyline points. Once all members of the subspace-group are visited, the final skyline points of a subspace-group are reported to the querying peer as a final skyline points. Thus progressiveness can

be preserved between subspace-groups. From Lemma 1, the traversal between subspace-groups is deterministic and progressive.

Lemma 1. *Subspace Group Traversal. Traversing the subspace-groups in a non-descending order of their subspace represented by peers' subspace Z-address ensures traversing a subspace-group before their dominated subspace-groups.*

Proof. Suppose a subspace-group β with a subspace address ω comes after subspace-group α with a subspace address υ. Let $\beta \prec \alpha$. Therefore, ω has less than or equal values to α's values in all considered subspace dimensions. Thus $\omega < \upsilon$ which contradicts our assumption. This means β can not dominate α.

Subspace pruning: A subspace-group is pruned if it is dominated by any discovered point with respect to the query's subspace.

Algorithm 2 in Fig. 4 is used for computing constrained subspace skyline. The traversal order depends on the bits of dimensions in favor. Using the subspace bits in favor as a prefix, the list of peers can be sorted. Thus traversal order takes the sorted peers order. For example, in Fig. 2 and Fig. 3, the traversal order for fullspace skyline queries is {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15} because all bits are considered. For the CPU performance attribute subspace only or price subspace query traversals, the order would be {0, 1, 4, 5, 2, 3, 6, 7, 8, 9, 12, 13, 10, 11, 14, 15} and {0, 2, 8, 10, 1, 3, 9, 11, 4, 6, 12, 14, 5, 7, 13, 15} respectively. As seen in Fig. 2, peers {0,1,4,5} are in column 1. Their CPU performance attribute value (IntValue =0). while peers {2,3,6,7} are in the second column with their CPU performance attribute value (IntValue =1). Column 1 precedes column 2 because it partially dominates column 2. The same is done for the other groups. Notice that for price attribute, rows are taken as subspace groups. The peers within a subspace group can be ordered in any order because they partially dominated each other. We, however, used order of the fullspace within the subspace groups. Once a query is triggered, the starting and ending nodes' Z-addresses are calculated using Equation 1. The query traverses nodes using the order determined by the subspace of the query between the starting node and the ending node excluding pruned peers. Each node can calculate its next peer to send the query and its results to.

3.4 Parallelization of Our Approach

In P2P systems, total parallelism can be achieved by having all peers involved in the query work concurrently. But it results in an increase in transferred data and it will disable prunability and progressiveness. We, however, exploit the incomparable cells' features in skyline. The transferred data reduction is also explored. One-dimension neighbors are used to find the next incomparable peers.

Definition 4. *One-dimension Neighbor. A one-dimension neighbor to a cell α is any cell β that can be converted to α by subtracting 1 from only one dimension.*

Property 2. All one-dimension neighbors of a cell are incomparable.

Constrained Subspace Skyline Parallel Algorithms (CSSPA). Using Property 2, a peer sends the query to all unpruned one-dimension neighbors. This can be achieved by having the query traverses from a peer to all peers with Z-values higher by one in one dimension and keeping the other values of the other dimensions. An example of fullspace queries in Fig. 2, a node 0 sends the query to its one-dimension neighbor (i.e 1 and 2). The same is done for subspaces queries. The difference is that dimensions of the query's subspace are only considered for the one-dimension neighbors. For example, for the CPU performance attribute subspace query, node 0 sends to node 2 only because it is the only one-dimension neighbor to node 0. Node 1 is in the same subspace group of node 0. The one-dimension neighbors here are subspace-groups. All peers within a group can work concurrently for total concurrency. But as mentioned earlier, it resulted in more data to transfer. A serial traversal can be used for nodes within a subspace group. The first peer of each group is determined as explained above.

Constrained Subspace Skyline Parallel Algorithms with Data Reduction (CSSPA-DR). Parallelism of the previous section implies that queries travel from lower values in each dimension to a higher value in that dimension. The skyline points received from previous peer are used to prune points. Since the parallel traversal comes from low values to higher values in a dimension, that dimension can be excluded and only the subspace skyline of the rest of queried subspace can be sent. Thus, the points sent (PS_i) through dimension D_i are PS_i = subspace-skyline-of(Query-dimensions - D_i). Thus, some of the points may only be sent. In our running example, if CPU performance attribute is used, a maximum of one point is traveled between the subspace groups.

Load Balancing. Load imbalance can be introduced by query imbalance as well as data in each cell [4]. To overcome this problem, we adopt the method in [4]. We use probing and replication. Each peer randomly chooses m points in the d-dimensional space. Each peer responsible for a point is probed for its query load balance. By sampling replies, a peer whose load exceeds a threshold will make a copy of its contents to a peer with minimum load. Then, queries arrived to such peer are distributed among the replicas in a round-robin fashion.

4 Performance Evaluation

The work in [19], we called ChordSky, is used as a baseline. Our algorithms (CSSA, CSSPA and CSSPA-DR) as well as the modified ChordSky were built using Peersim 1.0.5 simulator on Intel Pentium 4 CPU 2.00GHz, 512M memory. A uniform data with cardinality of 1 million and network sizes of 100-4000 nodes. Different numbers of dimensions are used {2,3,4,5,6,7}. Subspace queries are also randomly chosen with random dimensions. Extensive skyline queries are randomly produced and results were reported as shown later. Our aim is at minimizing the visited peers and transmitted data while preserving progressiveness.

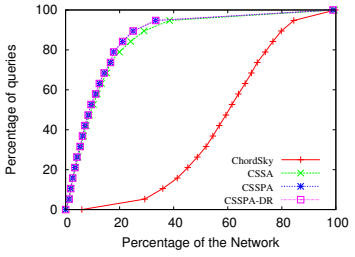


Fig. 5. Queries vs. traversed peers

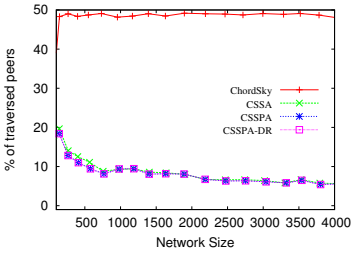


Fig. 6. Percentage of traversed peers

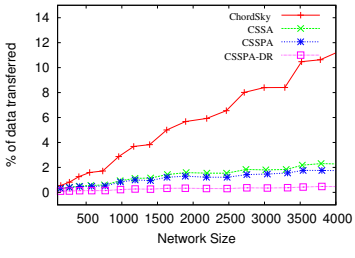


Fig. 7. Transferred data

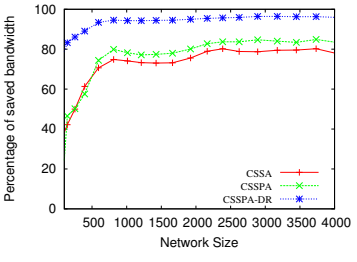


Fig. 8. Percentage of saved data

4.1 Accessed Nodes

Fig. 5 demonstrates the relationship between the percentage of queries and the needed traversed peers to answer queries. For the baseline algorithms, the slope is around 1.7 as opposed to our new algorithms whose slope are around 5.3. The baseline algorithm's traversed peers size increases by only 20% the answered queries percentage increase while it is 60% for the new algorithms. Thus, the new algorithms results in fewer number of traversed peers to answer the same percentage of queries. For example, less than 20% of peers can answer upto 80% queris in the new algorithms while the modified ChordSky (i.e. baseline) needs around 80% peers for same number of queries. In general, the reduction of traversed peers by the new algorithms(i.e. CSSA, CSSPA, and CSSPA-DR) as compared to ChordSky is between 40% to 50% of the network size to answer between 20% to 90% of the queries. This big reduction is due to pruning ability obtained by our partitioning and traversal algorithms. Comparing our new algorithms (i.e. CSSA, CSSPA, and CSSPA-DR) with each other, they have similar results to each other because they are using the same pruning method.

Fig. 6 shows traversed peers percentage average for different network sizes. The average traversed peers keeps its around 50% percentage with variant network sizes for ChordSky. For different sizes of the network, a slight decrease in the traverse peers' percentage is shown in our new approaches as the size increases. It decreases slowly to reach around 7%. Our approach's prunability power, thus, increases as network size increases.

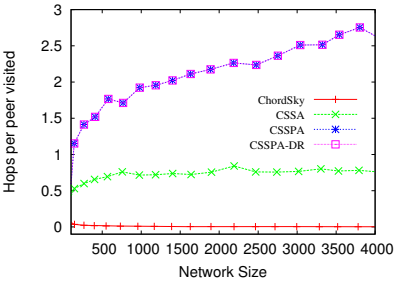


Fig. 9. Lookup hops per traversed peer

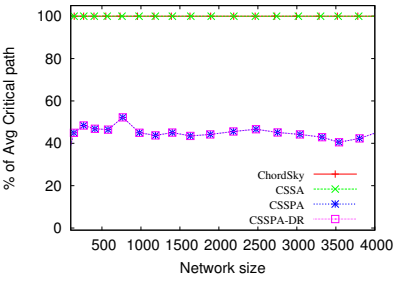


Fig. 10. Avg. critical line

4.2 Bandwidth Consumption

Fig. 7 shows transferred data for the algorithms. The pruning power and the reduction in the visited peers resulted in a reduction in transferred data points. As the size of the network increases, the traveled data also increases. Due to lack of peer prunability in ChordSky, more data are transferred. Our parallel algorithms show less traveled data than our serial approach because it does not exploit incomparable peers. Thus, it sends irrelevant data to incomparable peers. As a result of the increase number of traversed peers, which due to the increase in network size, a slight increase in the transferred data is noted.

Fig. 8 shows the huge saving for our approaches as compared to ChordSky. Up to 80% of the transferred data in ChordSky is saved. In CSSPA-DR, more than 90% is saved due the new data reduction techniques . As for CSSA and CSSPA, upto 80% are saved. Because CSSPA shows better saving than CSSA.

4.3 Expenses

ChordSky's unprunablity results in no cost. But the new algorithms' pruning ability may require a target lookup. A target in Chord may need up to $O = log_2 n$ hops, but it is not the case for our Z-order distribution. Queries are usually traversed between neighboring due to the addressing scheme. Jumps could mean pruning. Fig. 9 shows that the average number of hops is less than three hops per traversed peer. The small cost in the figure encourages the usage of those new algorithms. The cost of parallel algorithms is more than that of serial ones since a one-dimension neighbor peer could be looked by different peers.

4.4 A Bird's-eye View of the Algorithms' Results

Fig. 11 summarizes our findings. The percentage of visited peers to the overall network size is the highest when the baseline is used. Using the same new pruning techniques in all new algorithms is reflected in the same traversed peers for the new algorithms. The percentage of the transferred data to the overall data is low for our algorithms. It is minimum when the data reduction technique is used. ChordSky's visited peers seems to be strongly directly related to the

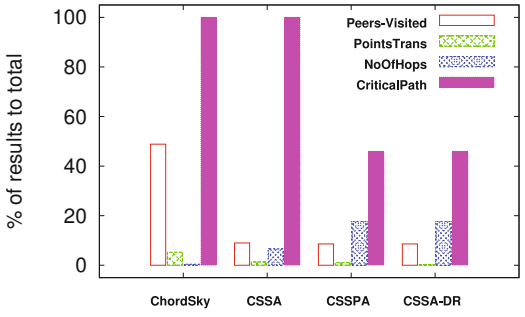


Fig. 11. A bird's eye view on adv. and disadv of each algorithm

network size. ChordSky's serial traversal is reflected in the minimum lookup cost. Parallelism increases cost because the peer is reached by different previous peers. Its bandwidth usage, however, is the minimum. By assuming that n is the number of traversed peers, both the baseline and the CSSA are completed only after n steps. When parallelisms are used, only 40% of the n steps are needed.

5 Conclusion

This paper addresses the *constrained subspace skyline* queries that have not been addressed in structured P2P systems. By converting the multi-dimensional data into a single index using a Z-order filling space, simpler P2P structures are used for a more general skyline queries (i.e. *constrained subspace skyline* queries). By a horizontal partitioning of the dataspace and assigning them Z-addresses, serial and parallel algorithms for such queries traversal are designed. They exploit incomparable partitions featured in skyline computations. Sending only necessary points to each next unpruned *one-dimension neighbor* and exploiting feature of this neighboring relation, a reduction in bandwidth usage is achieved. The efficiencies of the algorithms are shown in progressiveness and reduction in both the bandwidth consumption and traversed peers. New efficient approaches for such queries need to be found for unstructured P2P. Streams and partial order attributes should also be considered in our future work.

Acknowledgments. This research is partially supported by National Natural Science Foundation of China under grants 61173170 and 60873225, Innovation Fund of Huazhong University of Science and Technology under grants 2011TS135 and 2010MS068, and CCF Opening Project of Chinese Information Processing.

References

1. Stoica, I., Morris, R., Karger, D.R., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: SIGCOMM, pp. 149–160 (2001)

2. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: 17th International Conference on Data Engineering, ICDE 2001, pp. 421–432. IEEE, Washington (2001)
3. Dellis, E., Vlachou, A., Vladimirskiy, I., Seeger, B., Theodoridis, Y.: Constrained subspace skyline computation. In: CIKM, pp. 415–424 (2006)
4. Wu, P., Zhang, C., Feng, Y., Zhao, B.Y., Agrawal, D., El Abbadi, A.: Parallelizing Skyline Queries for Scalable Distribution. In: Ioannidis, Y., Scholl, M.H., Schmidt, J.W., Matthes, F., Hatzopoulos, M., Böhm, K., Kemper, A., Grust, T., Böhm, C. (eds.) EDBT 2006. LNCS, vol. 3896, pp. 112–130. Springer, Heidelberg (2006)
5. Rocha-Junior, J., Vlachou, A., Doulkeridis, C., Nørvåg, K.: Efficient execution plans for distributed skyline query processing. In: Proceedings of the 14th International Conference on Extending Database Technology, pp. 271–282. ACM (2011)
6. Chen, L., Cui, B., Lu, H.: Constrained skyline query processing against distributed data sites. IEEE Trans. Knowl. Data Eng. 23(2), 204–217 (2011)
7. Tao, Y., Xiao, X., Pei, J.: Subsky: Efficient computation of skylines in subspaces. In: ICDE, vol. 65 (2006)
8. Vlachou, A., Doulkeridis, C., Kotidis, Y., Vazirgiannis, M.: SKYPEER: Efficient subspace skyline computation over distributed data. In: ICDE, pp. 416–425. IEEE (2007)
9. Papadias, D., Tao, Y., Fu, G., Seeger, B.: Progressive skyline computation in database systems. ACM Transactions on Database Systems 30(1), 41–82
10. Lee, K.C., Lee, W.C., Zheng, B., Li, H., Tian, Y.: Z-SKY: an efficient skyline query processing framework based on Z-order. VLDB Journal: Very Large Data Bases 19(3), 333–362 (2010)
11. Balke, W.-T., Güntzer, U., Zheng, J.X.: Efficient Distributed Skylining for Web Information Systems. In: Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K. (eds.) EDBT 2004. LNCS, vol. 2992, pp. 256–273. Springer, Heidelberg (2004)
12. Wang, S., Vu, Q.H., Ooi, B.C., Tung, A.K.H., Xu, L.: Skyframe: a framework for skyline query processing in peer-to-peer systems. VLDB J. 18(1), 345–362 (2009)
13. Zhu, L., Tao, Y., Zhou, S.: Distributed skyline retrieval with low bandwidth consumption. IEEE Trans. Knowl. Data Eng. 21(3), 384–400 (2009)
14. Cui, B., Chen, L., Xu, L., Lu, H., Song, G., Xu, Q.: Efficient skyline computation in structured peer-to-peer systems. IEEE Trans. Knowl. Data Eng. 21(7), 1059–1072 (2009)
15. Pei, J., Jin, W., Ester, M., Tao, Y.: Catching the best views of skyline: A semantic approach based on decisive subspaces. In: Proceedings of the 31st International Conference on Very large Data Bases, pp. 253–264. VLDB Endowment (2005)
16. Yuan, Y., Lin, X., Liu, Q., Wang, W., Yu, J., Zhang, Q.: Efficient computation of the skyline cube. In: Proceedings of the 31st International Conference on Very Large Data Bases, pp. 241–252. VLDB Endowment (2005)
17. Jin, W., Tung, A., Ester, M., Han, J.: On efficient processing of subspace skyline queries on high dimensional data. In: 19th International Conference on Scientific and Statistical Database Management, SSBDM 2007, pp. 12–12. IEEE (2007)
18. Chen, L., Cui, B., Xu, L., Shen, H.T.: Distributed Cache Indexing for Efficient Subspace Skyline Computation in P2P Networks. In: Kitagawa, H., Ishikawa, Y., Li, Q., Watanabe, C. (eds.) DASFAA 2010. LNCS, vol. 5981, pp. 3–18. Springer, Heidelberg (2010)
19. Zhu, L., Zhou, S., Guan, J.: Efficient skyline retrieval on peer-to-peer networks. In: FGCN, pp. 309–314. IEEE (2007)