An almost Linear I/O Algorithm for Skyline Query

Xiangquan Gui, Yuanping Zhang
College of Computer and Communication, Lanzhou University of Technology,
Lanzhou, P.R.China,
Email: {xqgui, ypzhang}@lut.cn

Xiaohong Hao
College of Electrical and Information Engineering, Lanzhou University of Technology,
Lanzhou, P.R.China,
Email: haoxh@lut.cn

Abstract—Skyline query processing has recently received a lot of attention in database community. Even though there existed several algorithms in the field of skyline query, none of them has linear I/O complexity. In this paper, the existed algorithms have been summarized, and a new kind of external memory skyline query algorithm has been presented. Moreover, the reliability of algorithm has been validated from experiments and theory, the I/O complexity and the inner memory complexity of the algorithm is both almost linear.

Index Terms—skyline query, skyline point, external memory algorithm

I. INTRODUCTION

The data size of databases has grown rapidly with the unceasing development of database technology and the widespread use of database management system nowadays. How to find out the most interested information in massive data, in order to help people make effective decisions, has become an important research subject. So Skyline query processing has been received a lot of attention recently, because of its potential applications in the fields of multi-objective decision, data mining, database visualization and so on.

The Skyline query problem is also called as the maximum vector problem. Given an $n$-dimensional space database, a data point dominates another data point if it is as good or better in all dimensions and better in at least one dimension. The Skyline Point (SP) is defined as a point that is not dominated by any other point in the database. The Skyline query problem is to find out all such SP as quickly and exactly as possible in the database.

A classic illustrative example of Skyline query is to search for hotels in Nassau (Bahamas) which are cheap and close to the beach [4]. Suppose you are going on holiday to Nassau and you are looking for a hotel that is cheap and close to the beach from a large numbers of hotels. In the Figure 1, every point denotes a hotel, the abscissa denotes the price and the ordinate denotes the distance to the beach. It can be seen that you just need consider the point (hotel) in the dot line. Because that you can always find a point (hotel) in the dot line, either the price is cheaper or the distance is closer to the beach. Then, we say those points in the dot line dominate other points and can not be dominated by each other. Those points make up the result set of Skyline query. Basing on the Skyline query, the travelers can make decisions easily according to his preference from the Skyline set whose size is much smaller than that of original database.

Barndorff-Nielsen and Sobel initiated the study of the number of maximum vector in 1966 [1]. A number of algorithms have been proposed for the Skyline query problem. Kung gave an algorithm whose query complexity is $O(n \log n)$ in lecture [11], when specific to 2-dimensional or 3-dimensional data set. For higher dimensionality, he also gave an algorithm whose query complexity is $O(n(\log n)^{d-2})$, where $d$ denoted the number of dimensions. Bentley gave a linear query complexity algorithm, which we hoped, when attribute value of data set is independent and without repetition in every dimension [2]. All those algorithms are specific to the condition of small data quantity, which means the data set can be loaded into main memory completely.
To the condition of large data quantity, which means the data set can not be loaded into main memory completely, Skyline query algorithms can be classified into two types: with index and without index. The without index algorithms mainly include the BNL(block-nested-loops) algorithm [4], the SFS(sort-filter-skyline) algorithm [6], the D&C(divide-and-conquer) algorithm [4], the Bitmap algorithm [14], and the LESS(linear elimination sort for skyline) algorithm [7]. Where the BNL algorithm has a widespread application in every dimensional data set. It dissatisfies incrementality, and will misdiagnose in running time (consider a non-SP point as a SP during the middle of computation). The SFS algorithm, based on the BNL algorithm, has a presort to the data set. It can accelerate computing, and it has the merits of nonblocking and pipeline output. The D&C algorithm divides the data set into several parts, and make sure that the every parts can be loaded into main memory. Then computes the SP using inmemory algorithm separately, gets the final SP set by merging all SP in every parts and removing dominated points. But when it deals with the large data set, the I/O overhead of divide procedure is expensive. The D&C algorithm also dissatisfies incrementality, and will misdiagnose in running time. The Bitmap algorithm uses bitmap structure to decide whether a point is a SP. It needs preprocess (create bitmap). It satisfies incrementality, and has short responsive time. But it is inapplicable to dynamic databases and different query conditions. The LESS algorithm combines some characteristics of the SFS, BNL, and FLET [3] algorithm. It is the improved algorithm of the SFS. The best case complexity of LESS algorithm is O(kn) and the worst case complexity is O(kn^2). The with index algorithms mainly include the Using B-trees algorithm [4], the Index algorithm (a B'-tree-based algorithm) [14], the NN (nearest neighbor) algorithm [10], the BBS (branch-and-bound skyline) algorithm [13], the SDC (stratification by dominance classification) algorithm [5], and some improved algorithms of those. The Using B-trees algorithm makes an ordered index for all 2-dimensional data set. It is applicable to 2-dimensional space only, and has high efficiency only under the condition that the data set is small and the first matching SP can be found early. The Index algorithm transforms the high dimensional data into one dimensional, and then establishes a B’-tree index to the transformed data set. It has short responsive time, satisfies incrementality, but it is inapplicable to different query conditions and different dimensional subspace queries. The NN algorithm divides the data set recursively using the nearest searched neighbor. It is applicable to different query conditions and dynamic databases, but its performance is low and the space spending is expensive when the dimensionality of data set is higher than 3. The BBS algorithm is also based on the nearest neighbor search strategy. But it is applicable only to the data set with the complete ordered attribute field. The SDC algorithm aims at the data set with the partly ordered attribute field, it is a classification query algorithm based on dominance relations. It has short responsive time and good incrementality.

For the Skyline query problem of massive data, the every existed algorithm at present has its own advantages and disadvantages. However, in order to increase the efficiency of those algorithms, pertinent preprocessors (make the index or presort for query) are needed. But only the preprocessor will cost a lot of I/O communications. Such as the I/O times of external sorting will cost $\Theta(N \log_{M/B} N)$, where $N$ denotes the data size, $M$ the size of main memory and $B$ the size of loading data every time [15]. Moreover, none of the existed algorithms has linear I/O complexity yet. Due to the application requirement of Skyline query problem in massive data set, a new kind of external memory Skyline query algorithm (EMSQ for short) is presented in this paper. The algorithm is almost linear in number of I/O, and the corresponding reliability has been validated from both of experiments and theory. In addition, to make the inner memory complexity is linear too, a heuristic stratage is used to organize the SP in inner memory.

The remainder of this paper is structured as following. Section II describes the algorithm in detail, Section III describes the experiment data, and Section IV discusses the theory proof of I/O complexity. Section V discusses the inner memory complexity of the algorithm. Conclusions and future works are then offered in Section VI.

II. The External Memory Skyline Query Algorithm

The algorithm is designed according to the principle that the number of SP is much less then the number of data points in the whole database. Let $N$ be the number of $R$-dimensional data points, with the Completely Independent (CI) distribution. The expectation of SP number in the set approximates to $\log^{R-I} N$ [8, 9].Therefore, we consider setting a resident area in the main memory called as the general SP room, used to store all SP that have been found during running of the algorithm. Initially the room is empty. The data points is loaded from the external memory to main memory by paging and each data point in the page, say $P$, needs to be compared with all points in the general SP room. There will be two cases happened as follow: $P$ may be dominated by some point in the general SP room then $P$ will be neglected and the next point in the page will be selected to do the same thing; otherwise as a SP, $P$ will be inserted to the general SP room. In the second case, $P$ may dominate a series of points in the general SP room, then these dominated points will be deleted. The pseudo code of the algorithm is described as following:

the External Memory Skyline Query Algorithm
1. Setting the general SP room of size $r$ in main memory.
2. Setting the size of pages (G points).
3. While (have not computed all points in the external memory) do
   4. Loading a page(with G points except the last one)
into the main memory.
5. For $I=1$ to size of the page
6. If (the $I$-th point, say $P$, is dominated by some point in the room) then
7. Continue.
8. Else
9. Delete those points $P$ dominates from the room.
10. Insert the point $P$ into the room.
11. If (the number of SP in the room > $r$) return overflow.
12. Output all points in the general SP room.

The size of pages can be calculated by the size of main memory. The size of general SP room is set to store $\left[ \frac{\log N}{(R-1)!} \right]$ points where $k > 1$. A suitable $k$ is important in the algorithm. On one hand if $k$ is large, there is enough space in the room to store all SP during program running and without overflow, but the main memory is limited and may not provide this enough space. On the other hand if $k$ is small, then it is easy to provide the room space in main memory, but the probability of overflow will be increased. In the following two sections we will show from the experiments and theory proof that, when $k \in (1, 5)$, the probability of overflow almost equals to 0 and in this case the room space can be sufficed easily.

In the 10th sentence of the algorithm, which position is the point $P$ located, when inserting $P$ into the general SP room. Sequence of points in the general SP room will effect the running time of the algorithm in the inner memory. This problem we will discuss in section V.

### III. EXPERIMENT DATA

In experiments of the EMSQ algorithm, Let the every coordinate value of data points is double float number in $(0,1)$ created randomly. Let the better in the definition of dominate be larger instead. i.e. A point $p$ dominates a point $q$ if the value of every coordinates of $p$ is larger than corresponding value of $q$. It is easy to modify the real data to this distributed area and this relation of dominate. Set the page size $G=200$ (200 data points perpage). The general SP room size is set to store

![Figure 2. Experiment results of 2-dimensional data.](image_url)

![Figure 3. Experiment results of 3-dimensional data.](image_url)

![Figure 4. Experiment results of 5-dimensional data.](image_url)

![Figure 5. Experiment results of 8-dimensional data.](image_url)
IV. THEORY PROOF

The general SP room may overflow during the program running. If this case happens the algorithm will fail. In this section we will prove that when data points chosen from any R-dimensional CI distribution, the failure probability is almost 0 for a suitable \( k \) value.

**Theorem 1.** [12] \[
\Pr[X \geq (1 + \delta)\mu] \leq \frac{e^\delta}{(1 + \delta)^{1+\delta}}
\]

This is a standard Chernoff-type bound from [12] and where \( X \) is a binomially random variable with mean \( \mu \) and \( \delta > 0 \).

Let \( M_N \) be the number of SP in a data set \( S \) of \( N \) data points in \( R \)-dimensional space. We know that \( \mu = E(M_N) = \left[ \log^{k-1} N \right]/(R-1)! \) mean of \( M_N \) when data set follows a CI distribution. But whether \( M_N \) is a binomially random variable? At first sight, it looks like whether a data point is the SP or not is not a independent property of a single data point, it depends on the whole content of the data points set. But when the whole data points set is fixed, the SP of this data points set is also fixed. We can construct an area \( \Delta \), although we do not know what the exact shape \( \Delta \) is and what the exact position \( \Delta \) located. Then if a data point \( q \) is located in \( \Delta \), it is a SP, if \( q \) is not located in \( \Delta \), it is not a SP. The probability that \( q \) is a SP is \( p \), it is not hard to see that

\[
\Pr(M_N = i) = \binom{N}{i} p^i (1 - p)^{N-i}.
\]

i.e., \( M_N \) is a binomially distributed random variable with parameters \( N \) and \( p \).

Setting \( X=M_N \), \( \mu=E(M_N) \) and \( \delta=k-1(k>1) \) back to Theorem 1, then we get that,

**Lemma 1.**

\[
\Pr[M_N \geq kE(M_N)] \leq \left( \frac{e^{k-1}}{k^k} \right)^{E(M_N)}
\]

**Lemma 2.**

\[
\Pr[M_{pG} \geq kE(M_N)] \leq \frac{e^{kE(M_N)}/E(M_{pG})}{\left[ kE(M_N)/E(M_{pG}) \right]^{kE(M_N)/E(M_{pG})}}^{E(M_{pG})}
\]

Where \( G \) denotes the number of points in every loading page, \( p \) the number of pages that have been loaded in main memory currently, valuing between \([1, \left[ N/G \right]\])

\( M_{pG} \) denotes the number of SP points after running the algorithm with \( p \) pages loaded in. Apparently \( \Pr[M_{pG} \geq kE(M_N)] \) denotes the probability that the general SP room is overflowed at present.

**Proof.** From Lemma 1 we have,

\[
\Pr[M_{pG} \geq kE(M_N)] \leq \left( \frac{e^{k-1}}{k^k} \right)^{E(M_{pG})}
\]

Let \( k=pE(M_N)/E(M_{pG}) \). So

\[
\Pr[M_{pG} \geq kE(M_N)] = \Pr[M_{pG} \geq kE(M_{pG})] \leq \left( \frac{e^{kE(M_N)/E(M_{pG})}}{[kE(M_N)/E(M_{pG})]^{kE(M_N)/E(M_{pG})}} \right)^{E(M_{pG})}
\]

From above mentioned we have:

**Theorem 2.** \( \Pr(\text{Overflow}) \leq \)

\[
\left[ \frac{1}{N_G} \prod_{p=1}^{G} [1 - \Pr(M_{pG} \geq kE(M_N))] \right]^{N/G} \quad \text{if } N_G = 0
\]

\[
1 - \left[ \frac{1}{N_G} \prod_{p=1}^{G} \left[ 1 - \Pr(M_{pG} \geq kE(M_N)) \right] \right] \quad \text{if } N_G \neq 0
\]

Where \( \Pr(\text{Overflow}) \) denotes the probability of the algorithm fails.

![Figure 6. The function figure of Theorem 2](image-url)
When $k$ is not large enough, the probability of overflow will increase with the increase of $N$ from the Figure 6 shows, and will minish alone with the augmentation of $G$ from the Figure 7. But when $k$ is large enough the size of $N$ and $G$ have a few effects to the overflow probability of the algorithm. In practice, just need set $G$ by the size of actual main memory. A reference value of $k$ in every dimension can be seen from the Figure 8. At the same time the exact value of $k$ can be calculated by Theorem 2 using some mathematics software. Such as there have a data set of $N=20000$ points in 3-dimensional space ($R=3$), set $G=200$. In order to reduce the probability of algorithm fails below a given number, say $10^{-10}$, the value of $k$ should large then 2.16 from solve the inequality of $Pr(Overflow)<10^{-10}$. So from Theorem 2, we have that when we set a suitably value of $k$, the probability of algorithm failed will almost equals to 0.

V. INNER MEMORY COMPLEXITY OF THE ALGORITHM

We have mentioned before that the sequence of points in the general SP room will effect the running time of the algorithm in the inner memory. In order to reduce the inner memory running time, let the sequence of points in the general SP room is descending order of the product of its coordinates in every dimension, i.e. Let point $p$ in $R$-dimensional space with coordinates $x_1, x_2, ..., x_R$, the product of its coordinates in every dimension is $\prod_{i=R}^{i} x_i$.

Intuitively the greater the product of a point is, the more dominant power it owns. Therefore in the 10th sentence of the algorithm, $P$ is inserted into the general SP room according the descending order of the product. In this way SP points that near the front of the sequence of the general SP room are powerful dominators. They can quickly dominate most of the input points which are not in the final output. Using this strategy, we find that the inner memory running time is almost linear too by many experiment and we can prove the 2-dimensional case by the following Theorem 3.

**Theorem 3.** Choosing $N$ point, $p_1, p_2, ..., p_N$, from a 2-dimensional CI distribution. Then, with probability $1 - N^{-\Omega(log N)}$, the inner memory running time of EMSQ algorithm finds the SP set using only $O(N^{2/3} \log^4 N)$ point comparisons.

The $N^{-\Omega(log N)}$ term can be thought of as being a super polynomially small probability, since it is smaller then $N^{-k}$ for any constant $k$. To prove the Theorem 3 we need some random regions and variables first. Let $p_1, p_2, ..., p_N$, be the input points listed in the order in which they are examined in 2-dimensional space. We partition distributed area of input points into three regions, $A$, $B$, $C$, dependent upon parameter as shown in Figure 9. Distributed area of input points is $0 \times 0$. It is not difficult to modify the real data by a mapping from CI distribution to this uniformly distribution, so that the proof remains valid when the points are chosen from any CI distribution.

![Figure 8. Input points area partitions in 2-dimensional space.](image)

Formally

$$A = (0,1 - N^{-\alpha}) \times (0,1 - N^{-\alpha})$$

$$C = \{ (x, y) : xy \geq 1 - N^{-\alpha}, 1 - N^{-\alpha} \leq x, y \leq 1 \}$$

$$B = (0,1) \times (0,1) - A - C$$
Set
\[ F_c = \min \{ i : p_i \in C \text{ or } i = N \}, \]
\[ N_{BC} = | \{ p : p \in B \cup C \} |, \]
\[ M_i = | \{ j : j \leq i \leq N, p_j \text{ is SP in } p_1, ..., p_i \} |, \]
\[ M = \max_{i \in N} M_i. \]

\( F_c \) is the index of the first point in \( C \) (if there is no such point, then \( F = N \), \( N_{BC} \) is the number of points found in the region \( B \cup C \). \( M \) is the number of SP in the point set \( \{ p_1, ..., p_{F_c} \} \). \( M_i \) is the largest of the \( M_i \) it is an upper bound on the number of point comparisons that can be performed while examining any point to see if it is SP.

**Lemma 3.** The number of point comparisons performed by EMSQ algorithm when run on a sequence \( N \) 2-dimensional points, \( p_1, p_2, ..., p_N \) is at most \( N+MF_C+MN_{BC} \).

Let \( \alpha > 0, p_1, p_2, ..., p_N \) be a sequence of \( N \) 2-dimensional points chosen from a uniform distribution over the unit square. Then
\[ \Pr( F_C > 2N^{2\alpha} \log^2 N ) < N^{-\Omega(\log N)}, \]
\[ \Pr( N_{BC} > 14 N^{1-\alpha} ) < N^{-\Omega(\log N)}, \]
\[ \Pr( M > 2 \log^2 N ) < N^{-\Omega(\log N)} . \]

The constants implicit in the \( \Omega() \) notation are dependent only upon \( \alpha \). These two lemmas separate the deterministic part of the analysis from the probabilistic part. Inserting the probabilistic bounds of Lemma 4 into the deterministic one of Lemma 3 yields that the EMSQ algorithm in inner memory performs
\[ N + O(\max(N^{2\alpha} \log^4 N, N^{1-\alpha} \log^2 N)) \]
point comparisons with probability \( 1 - N^{-\Omega(\log N)} \). To get the best possible bound we set
\[ N^{2\alpha} \log^4 N = N^{1-\alpha} \log^2 N . \]

Solving this equation yields \( \alpha < 1/3 \). Substituting \( \alpha = 1/3 \) back into (**) shows that, with probability \( 1 - N^{-\Omega(\log N)} \), the EMSQ algorithm finds the SP set of \( N \) 2-dimensional points using only \( O(N^{2/3} \log^4 N) \) point comparisons and proves Theorem 3.

Now we begin prove the Lemmas 3 and 4. We partition the input sequence \( p_1, p_2, ..., p_N \) into two subsequences, \( \{ p_1, p_2, ..., p_{F_c} \} \) and \( \{ p_{F_c+1}, p_{F_c+2}, ..., p_N \} \). We will show that the number of point comparisons performed while examining each subsequence can be bounded by functions of random variables.

**Claim 1.** The total number of point comparisons needed to examine \( p_1, p_2, ..., p_{F_c} \) is at most \( MF_C \).

**Claim 2.** The total number of point comparisons needed to examine \( p_{F_c+1}, p_{F_c+2}, ..., p_N \) is at most \( N+MN_{BC} \). \( M \) is an upper bound on the number of point comparisons performed while examining any point. The number of points in subsequence \( p_1, p_2, ..., p_{F_c} \) is \( F_c \). So the total number of point comparisons needed to examine this subsequence is at most \( MF_C \). After examining this subsequence the point \( p_{F_c} \) will be inserted into the general SP room and located in the most front position, because the product of coordinates of point \( p_{F_c} \) is the largest one currently. After that, the most front position of the general SP room can only be replaced by other points in \( C \), since those points is listed according the descending order of the product of coordinates and only points in \( C \) can have a larger product. Thus the most front point of the general SP room will always in \( C \). When examining \( p_i \), \( p_i \in \{ p_{F_c+1}, p_{F_c+2}, ..., p_N \} \), there can be divided into two cases: First, if \( p_i \in A \), the number of point comparisons needed to examine \( p_i \) is 1, because the most front point of the general SP room is in \( C \) and it can dominate \( p_i \). The number of such \( p_i \) is at most \( N \). Second, if \( p_i \in B \cup C \), the number of point comparisons needed to examine \( p_i \) is at most \( M \). The number of such \( p_i \) is \( N_{BC} \). So the total number of point comparisons needed to examine \( p_{F_c+1}, p_{F_c+2}, ..., p_N \) is at most \( N+MN_{BC} \). That explain the Claims 1 and 2 are correct. Combining Claims 1 and 2 proves Lemma 3.

**Proof of Lemma 4:** First we prove
\[ \Pr( F_C > 2N^{2\alpha} \log^2 N ) < N^{-\Omega(\log N)} . \]

Suppose that \( q \) is a point chosen from a uniform distribution over the unit square. To bound \( F_C = \min \{ i : p_i \in C \text{ or } i = N \} \) we set
\[ p = \Pr(q \in C) = \text{Area}(C) = \frac{1}{2} N^{-2\alpha} . \]

Then
\[ \Pr(F_C > 2N^{2\alpha} \log^2 N) \]
\[ < \left( \frac{1}{2} N^{-2\alpha} \right)^{2N^{2\alpha} \log^2 N} . \]
\[ = N^{-\Omega(\log N)} . \]

We now prove \( \Pr(N_{BC} > 14 N^{1-\alpha} ) < N^{-\Omega(\log N)} \).

Because \( N_{BC} \) is a binomially distributed random variable with mean \( Np = 2 N^{1-\alpha} - N^{1-2\alpha} \). Using Theorem 1 and set \( \delta = 6 \) then
\[ \Pr(N_{BC} > 14 N^{1-\alpha} ) \]
\[ = \Pr(N_{BC} > 7(2 N^{1-\alpha} - N^{1-2\alpha} ) + 7 N^{1-2\alpha} ) \]
\[ < \Pr(N_{BC} > (1 + 6)(2 N^{1-\alpha} - N^{1-2\alpha} )) \]
\[ < \left( \frac{e^6}{7} \right)^{2 N^{1-\alpha} - N^{1-2\alpha}} \]
\[ < (e^2)^{2 N^{1-\alpha} - N^{1-2\alpha}} \]
\[ = e^{-7(2 N^{1-\alpha} - N^{1-2\alpha})} \]
\[ = N^{-\Omega(\log N)} . \]

The final inequality given by Lemma 4, \( \Pr(M > 2 \log^2 N ) < N^{-\Omega(\log N)} \), has proved in the
literature [12] by Golin. In order to complete our proof. We prove it here again using the same way. It is know by [12] that in 2-dimensional space with points in CI distribution the SP number $M_i$ satisfy

$$\Pr(M_i - H_i \geq \varepsilon) \leq e^{-c\varepsilon^2/(2H_i+c\varepsilon)} \quad [12].$$

Where $H_i = \sum_{j=1}^i \frac{1}{j}$ is the $i$th harmonic number and $c>0$.

When $i$ gose to infinity, $Hi - \log i \rightarrow \gamma = 0.5772...$ (Euler’s constant). Setting $\varepsilon = \log^2 N$, if $N$ is large enough, then, for all $i<N$,

$$\Pr(M > 2\log^2 N) \leq e^{-\log^4 N/(2H_i+c\log^2 N)} \leq N^{-\Omega(\log N)}.$$

We have proved Theorem 3 completely. According many experiment data, we find that the inner memory running time is almost linear too, but the theoretical proof of higher dimensional case is not clearly yet.

VI. CONCLUSION

A new kind of skyline query algorithm has been presented in this paper. The main result, the EMSQ algorithm, can also be used as a maximum vector finding algorithm. So it not only have applications in the fields of database but also may have a number of applications in geographic information systems (CIS) and computer graphics, as data in those areas become more and more large today. More important, the I/O complexity and inner memory complexity of EMSQ algorithm is both almost linear. In addition, it is easy programming in practice.

There are several aspects about the EMSQ algorithm need more investigation. First, the proof of inner memory running time complexity when point dimensionality larger than 2. Second, the performance of the algorithm when the data set is chosen from a non-CI distribution. We will discuss those problems in future works.

REFERENCES

in 2002. He is currently a professor in the College of Computer and Communication, Lanzhou University of Technology, P.R.China. His research interest includes algorithm, graph theory, combinatorics and information theory.

Xiaohong Hao was born in Gansu, P.R.China, in February, 1960. He received M.E. degree in control theory and control engineering from Lanzhou University of Technology in 1996. He is currently a professor in the College of Electrical and Information Engineering, Lanzhou University of Technology. His research interest includes complex automatic control system, computer control technology and field bus control system.