An External Memory Algorithm of Maxima-Finding Problems*

Xiang-Quan Gui, Yuan-Ping Zhang, Li Li
College of Computer and Communication, Lanzhou University of Technology, Lanzhou, 730050, Gansu, P.R.C.,
e-mail: xqgui@lut.cn, ypzhang@lut.cn, lili0226@mail2.lut.cn

Xue-Rong Yong
Department of Mathematics, University of Puerto Rico at Mayaguez, P.O.Box 9018, PR 00681, USA
e-mail: xryong@math.uprm.edu

***Abstract*—Maxima-finding problems are the fundamental problem in computational geometry with a great deal of application in many areas, and it has resurfaced with the advent of Skyline Queries for relational databases and data mining recently. The existed algorithms in the field of Maxima-finding problem (Skyline Queries) have been summarized in this paper. But for the massive data sets, there has no I/O linear algorithm yet. A new kind of External Memory Algorithm of Maxima-finding problem (EMMF) has been presented, the I/O complexity of algorithm is linear, the corresponding reliability has been validated from experiments, and the status of 2-dimensional space has been proved in theory too.**

Keywords-Maxima-finding problem; Skyline query; external memory algorithm; I/O algorithm

I. INTRODUCTION

The Maxima-finding problem is to find the subset of the points such that each is not dominated by any of the points from the set. Given a set S of N points in R-dimensional space, Point P is said to *dominate* point Q if each coordinate of P is greater than the corresponding coordinate of Q. A point in S that is not dominated by any other point in S is a *maximal* point. The undominated points in S are the *maxima* of the point set [13]. This problem has been considered for many years, a number of algorithms have been proposed for efficiently finding the maximal.

The Maxima-finding problem has been rediscovered recently in the database context with the introduction of skyline queries. Instead of vectors or points, this is to find the maximal over *tuples*. Certain columns (with numeric domains) of the input relation are designated as the skyline criteria, and dominance is then defined with respect to these. The non-dominated tuples then constitute the skyline set [10]. A classic illustrative example of skyline queries is to search for hotels in Nassau (Bahamas) which are cheap and close to the beach [4]. Suppose each hotel has two attributes: the price and the distance to the beach. Hotel A dominates hotel B (or, A is a better choice than B in the context of this example) if A.price $\leqslant$ B.price, A.distance $\leqslant$ B.distance and at least one inequality holds. Those hotels not dominated by others in terms of price and distance to the beach form the skyline. In other words, the skyline hotels are all possible trade-offs between price and distance to the beach that are superior to other hotels.

Skyline queries have attracted a fair amount of attention since their introduction in [4]. It is thought that skyline offers a good mechanism for incorporating preferences into relational queries, and its implementation could enable data mining more efficiency. Actually, skyline queries and maxima-finding problems are almost the same in fundamental concepts and algorithms.

From 1966 Barndorff-Nielsen and Sobel study the number of maxima in the literature [1] as beginning, the idea itself is old, the maxima problems has been a lot of research results. There are a considerable number of algorithms to find the maxima in a data set. And we classified as Sequential algorithm[2, 13], Divide-and-conquer algorithm [3, 4, 5, 7, 13], Bucket algorithm[8, 9], Selection algorithm[6], Sieve algorithm [2, 10].

But for the massive data sets, that can be seen in the areas of data mining and knowledge discovery often, the Maxima-finding algorithm is very lack. Because for dealing with the massive data, the data are usually too large to fit the inner memory, the main constraints of algorithm efficiency has become the cost of data transfer between the inner and outer of the memory. Although there are seldom parts of algorithm can deal with massive data, such as the BNL algorithm in the literature [4] and the SFS algorithm in the literature [5], the I/O complexity of these algorithms is expensive. Because these algorithms all need to sorting points beforehand, when dealing with massive data, only the I/O cost of sorting will be $\Theta(N \log_{M/B} N)$, where N denotes the points (tuples) number, M the size of main memory and B the size of loading data every time. About the details of the massive data-processing algorithms can be seen in the Vitter's review literature [14]. To deal with these cases, a new kind of external memory maxima-finding algorithm (EMMF for short) is presented in this paper, the corresponding reliability has been validated from experiments, the status of 2-dimension has been proved in theory also. The EMMF algorithm is linear in number of I/O.

Section II of this paper describes the algorithm in detail, Section III describes the experiment data, and Section IV

* This work was supported by the National Science Foundation of Gansu province, P.R.C..(Grant No. 3ZS051-A25-037).

978-0-7695-3557-9/09 $25.00 © 2009 IEEE
DOI 10.1109/ETCS.2009.439

discusses the theory proof. Conclusions are then offered in Section V.

II. THE EXTERNAL MEMORY MAXIMA-FINDING ALGORITHM（EMMF）

The algorithm is designed according the principle that the number of maximal points is very little relative to the number of whole points set. Let N be the number of R-dimensional data points, with the independently and identically distribution (I.I.D.). The expectation of maxima number in the set approximate to $\frac{\log^{R-1} N}{(R-1)!}$ [11]. Therefore, we consider opening a resident area in main memory that called as the *general maxima index room*, used to store all index of maximal points that has been found in running time. In practice, data sets may carry some other information that is useless in comparison. Let those information always in main memory will waste a lot of memory space. So the local maximal points (i.e. points that be considered as maximal point temporarily in running time and may not as a maximal point finally) will be established index and others is unnecessary for reducing I/O and CPU time. Initially, insert a virtual point to the general maxima index room to dominate more points and to insure the room not overflows at the beginning. Then of all points in the external memory, loaded by paging and compared with points in the general maxima index room. There will be three cases happened as follow, if it can be dominated by a point in the general maxima index room, choose another candidate point to do the same things. If it can dominate some points in the general maxima index room, replace those points. If it incomparable with each points in the general maxima index room, make index of it and insert to the room. The point P and point Q are incomparable means P can not dominate Q and Q can not dominate P too. The pseudo code of the algorithm as follow.

the External Memory Maxima-Finding algorithm（EMMF）

1. Opening a general maxima index room (the room for short)in main memory.
2. Insert a virtual point.
3. Setting the size of pages (G points).
4. While（have not computed all points in the external memory）do
5. Loading a page (G points) into the main memory.
6. For i=1 to G do
7. If（the i-th point can be dominated by a point in the room）then
8. CONTINUE.
9. Else if （the i-th point can dominate some points in the room）then
10. Delete those points from the room.
11. Else if （the i-th point incomparable with each points in the room）then
12. Make index of the i-th point and insert into the *room*.
13. If the virtual point has not been dominated, the algorithm is failed. Exit and use other algorithms.
14. Output all points in the general index maxima room.

The algorithm setting a space in main memory, called as the general maxima index room, used to store all maximal points at the 1st sentence. The $\left\lceil k[\frac{\log^{R-1} N}{(R-1)!}] \right\rceil$ points index can be stored in it (let N be the number of all points in R-dimensional space and k>1). Then insert a virtual point into the general maxima index room at the 2nd sentence, set the every dimension number of the virtual point is $1-(\frac{\ln N}{N})^{\frac{1}{R}}$ (let every dimension is bounded in [0, 1]). This position will have good performance in practice and can be dominated easily in running time. The probability of virtual point can not be dominated finally is at most $1/N$ (the prove can be seen latter in Theorem 1). Then loading a page (G points) into the main memory each time. The general maxima index room will overflow (this algorithm will fail) when insert points into the general maxima index room at the 12^{th} sentence. If this status happened, can use other algorithms to find the maxima set. Whereas the probability of this status happened is very tiny in I.I.D. from the examination and theory prove. Now we will give experiment data as follows to explaining the credibility of the algorithm.

III. EXPERIMENT DATA

In simulant experiment of EMMF algorithm, let main memory is 512MB. The every dimension coordinate value of points is 32 bit double float number in (0, 1) that created randomly. For every point, let the space usage is 1MB by adding some additional data. Let G=200, the size of 200MB's points load into main memory a time. And Let the size of general maxima index room is $\left\lceil k[\frac{\log^{R-1} N}{(R-1)!}] \right\rceil$ KB ($k\in$ [2, 3], every point index own 1KB, $R\in$ {1,..,8}). We create 100 teams data in randomly, and the points number N of teams from 2000 to 200200 (the space size of points from 2000MB to 200GB).

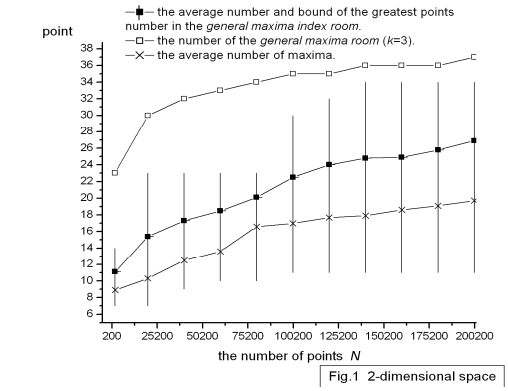


Fig.1 2-dimensional space

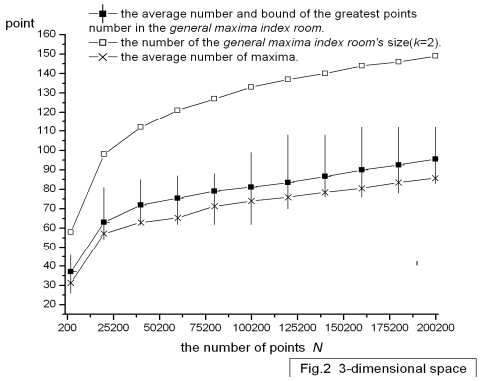


Fig.2 3-dimensional space

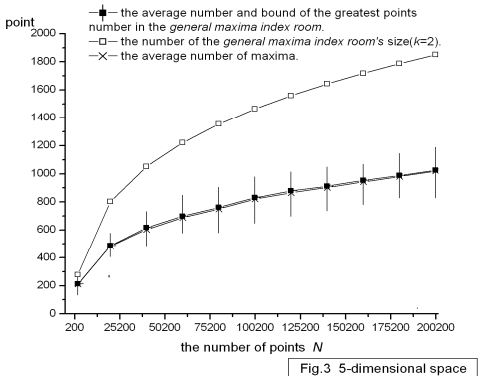


Fig.3 5-dimensional space

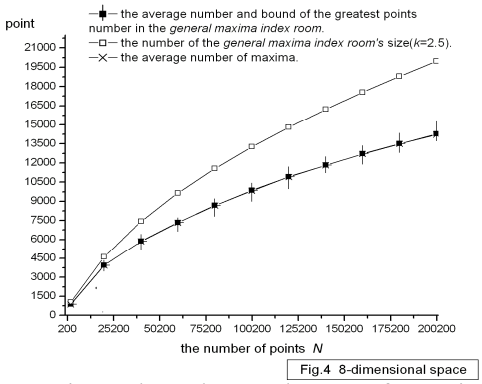


Fig.4 8-dimensional space

Experimental results can be seen from Fig.1 to Fig.4. Without lose of generality, we only show experimental results in these four figures with data of 2, 3, 5, 8-dimensions respectively. The results completely match expectation. In the figures, the capacity of general maxima index room always bound the greatest number of maximal points that means there is no overflow happening during running algorithm in every set of N points. In fact, this overflow may happen but the probability is very small which is almost 0. The proof will be given in next section.

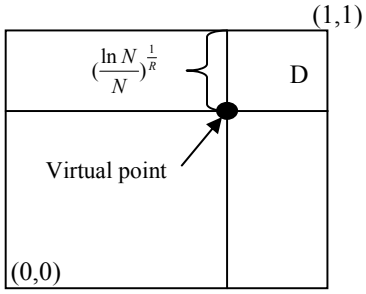
The EMMF algorithm's I/O time is linear, because of loading G points a time and only the loading in of points without loading out. The I/O times of EMMF is only N/G. Moreover, for about 200GB data, there can not be computed by common microcomputer or simplicial using the paging technology of OS. Because with the increase of I/O times, the running efficiency of programs will significant decline. The EMMF algorithm not only completes the computation, but also the I/O time is linear.

IV. THEORY PROOF

We insert a virtual point that is not necessarily a number of the set S at the beginning of the EMMF algorithm. If this virtual point can not be dominated by a point in set S, the EMMF algorithm will be failed. The probability that none of the N points is dominate the virtual point is at most $1/N$, proved by the Theorem 1. After then the corresponding reliability in 2-dimension space has been proved in Theorem 3. The cases of higher dimension space have not proved in theory yet, since the proof will be very complex and we will discuss in the later paper.

Theorem 1. The probability that none of the N points is dominate the virtual point is at most $1/N$.

Proof. The instance of 2-dimension space is illustrated in this picture:



The probability that none of the N points is dominate the virtual point equals to the region D is empty. This probability is $\{1-[(\frac{\ln N}{N})^{\frac{1}{R}}]^R\}^N$, since $\{1-[(\frac{\ln N}{N})^{\frac{1}{R}}]^R\}^N=[1+(\frac{-\ln N}{N})]^N \le e^{-\ln N}=1/N$ (for any $X, (1+\frac{X}{N})^N \le e^X$).

Theorem 2[12]**.**

$$\Pr(M_N - H_N \ge \varepsilon) \le e^{-\varepsilon^2/(2H_N+\varepsilon)}$$

Where M_N denotes the number of maximal points in a point set S that points number equal to N,

$H_N = \sum_{j \le N} 1/j$ is *N*-order Harmonic Series, and $\varepsilon > 0$.

Lemma 1. $\Pr(M_N \ge k \log N) < e^{-\frac{[(k-1)\log N - \gamma]^2}{(k+1)\log N + \gamma}}$

Proof. When the i tends to infinity, $H_i - \log i \to \gamma = 0.57721...$ (Euler constant). So let $\varepsilon = k\log N - H_N$, and because that $H_N > \log N + \gamma$, we have,

$$\Pr(M_N \ge k \log N) \le e^{-\frac{(k\log N - H_N)^2}{2H_N + k\log N - H_N}} = e^{-\frac{(k\log N - H_N)^2}{k\log N + H_N}} < e^{-\frac{[(k-1)\log N - \gamma]^2}{(k+1)\log N + \gamma}}.$$

Lemma 2. $\Pr(M_{pG} \ge k \log N) < e^{-\frac{[k\log N - \log(pG) - \gamma]^2}{k\log N + \log(pG) + \gamma}}$

Where *G* denotes the number of points in every loading pages. *p* the number of pages that has loaded in main memory currently, value between [1，$\lfloor N/G \rfloor$]. M_{pG} the number of maximal points after loading *p* pages. $\Pr(M_{pG} \ge k \log N)$ the probability of the maximal points number in *p* pages bigger than the number of points in the general maxima index room.

Proof. From lemma 2. we have,

$\Pr[M_{pG} \ge k_1 \log(pG)] < e^{-\frac{[(k_1-1)\log(pG) - \gamma]^2}{(k_1+1)\log(pG) + \gamma}}$. Let $k_1 = k\log N / \log(pR)$, there have

$$\Pr(M_{pG} \ge k_1 \log pG) =$$

$$\Pr(M_{pG} \ge k \log N) <$$

$$e^{-\frac{[(\frac{k\log N}{\log(pG)} - 1)\log(pG) - \gamma]^2}{(\frac{k\log N}{\log(pG)} + 1)\log(pG) + \gamma}} = e^{-\frac{[k\log N - \log(pG) - \gamma]^2}{k\log N + \log(pG) + \gamma}}.$$

From above-mentioned we have:

Theorem 3. Pr(Overflow)<

$$\begin{cases} 1 - \prod_{p=1}^{\lfloor N/G \rfloor} \{1 - e^{-\frac{[k\log N - \log(pG) - \gamma]^2}{k\log N + \log(pG) + \gamma}}\} & N \% G = 0 \\ 1 - \{1 - e^{-\frac{[k\log N - \log(N) - \gamma]^2}{k\log N + \log(N) + \gamma}}\} \prod_{p=1}^{\lfloor N/G \rfloor} \{1 - e^{-\frac{[k\log N - \log(pG) - \gamma]^2}{k\log N + \log(pG) + \gamma}}\} & N \% G \ne 0 \end{cases}$$

Where Pr(Overflow) denotes the probability of overflow in the general maxima index room when algorithm was running.

The Fig.5 shows that the probability of overflow decrease with the augmentation of *G*, but decrease is not significant. In other words, the size of *G* has a few infections to the running of algorithm. In practice, just need sets *G* by the size of actual memory. The Fig. 6 shows that when $k \ge 3$, i.e. the general maxima index room can have points at lest 3log*N*, the probability of overflow will be very teeny.

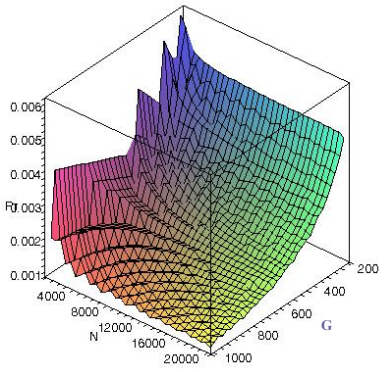


Fig. 5. the function figure of theorem 3 where *N* denotes the points number of set *S*, *G* the number of points in every loading pages, γ=0.577, *k*=3(i.e. the points number of the general maxima index room is 3log*N*).

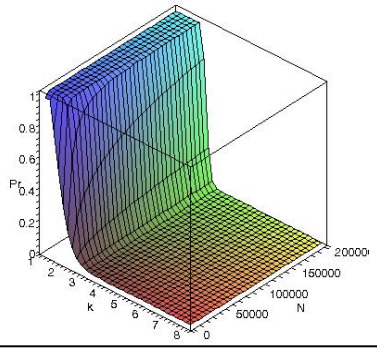


Fig. 6. the function figure of theorem 3 where *N* denotes the points number of set *S*, the points number of the general maxima index room is *k*log*N*, γ=0.577,*G*=200.

V. CONCLUSION

The existed algorithms in the field of Maxima-finding problem (Skyline Queries) have been summarized in this paper. A new kind of Maxima-finding algorithm (EMMF) that dealing with I.I.D.'s massive data has been presented. On this basis, the corresponding reliability has been proved from two aspects of theory and experiments. To our knowledge, this is the first demonstration of the pure external memory algorithm of maxima-finding problem. The main result, the EMMF algorithm, may have a number of applications in data mining, spatial databases, geographic information systems (CIS), and computer graphics, as data in those areas become more and larger today and the EMMF algorithm have good reliability and its I/O complexity is linear. In addition, it is easy programming in practice.

We will study the data in others distribution and theory prove higher dimensional space in the future work.

REFERENCES

[1] O. Barndorff-Nielsen and M. Sobel, On the distribution of the number of admissible points in a vector random sample, *Theory of Prob. and its Appl*., 1966, 11(2), pp. 249–269.

[2] J.L. Bentley, K.L. Clarkson and D.B. Levine, Fast linear expected-time algorithms for computing maxima and convex hulls, *Algorithmica*, 1993, 9, pp. 168–183.

[3] J.L. Bentley and M.I. Shamos, Divide and conquer for linear expected time, *Information Processing Letters*, 1978, 7, pp. 87–91.

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

[5] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, Skyline with presorting, In ICDE, Mar. 2003, pp. 717–719.

[6] K.L. Clarkson, More output-sensitive geometric algorithms (extended abstract), in IEEE 35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, 1994, pp. 695–702.

[7] L. Devroye, Moment inequalities for random variables in computational geometry, *Computing*, 1983, 30, pp. 111–119.

[8] L. Devroye, A note on the expected time required to construct the outer layer, *Information Processing Letters,* 1985, 20, pp. 255–257.

[9] L. Devroye, Lecture Notes on Bucket Algorithms, Progress in Computer Science, 6, Birkhäuser, Boston, MA,1986.

[10] P. Godfrey, R. Shipley, J. Gryz, Maximal vector computation in large data sets. VLDB 2005 - Proceedings of 31st International Conference on Very Large Data Bases, 2005, pp. 229-240.

[11] M.J. Golin, How many maxima can there be?, *Theory of Prob. and its Appl.,* 1993, 2, pp. 335-353.

[12] M.J. Golin, A provably fast linear-expected-time maxima-finding algorithm , *Algorithmica*, 1994, 11, pp. 501-524.

[13] H.T. Kung, F. Luccio and F.P. Preparata, On finding the maxima of a set of vectors, *Journal of the ACM*, 1975, 22, pp. 469–476.

[14] J.S. Vitter, External Memory Algorithms and Data Structures: Dealing with Massive Data, *ACM Computing Surveys*, 2001, 33, pp. 209-271