Skyline Computation permitting Dynamic Determination of Query Regions

Myung Kim
Dept. of Computer Science & Engineering
Ewha Womans University
Seoul, Korea

***Abstract*—The skyline of a multidimensional data set is the maximal subset whose elements are not dominated by any other elements of the set. Its computation is one of the useful operations for recommendation systems. However, when the user (or the query object) is moving on a 2-D plane, dynamically changing distance between the user and the data elements should also be considered in the skyline computation. In order to do so, previously reported algorithms focus on locating the data elements that get closer to the mobile user. However, in our work, we let the user instantly limit the query region at query time, so that the skyline is computed only from the data elements that belong to the chosen region. The skyline computed this way is more suitable for the mobile user's requirements in various applications. In order to speed up the query processing, we use a quadtree for representing various query regions, and precompute intermediate results for each quadtree node. The performance of the algorithm is evaluated by experiments.**

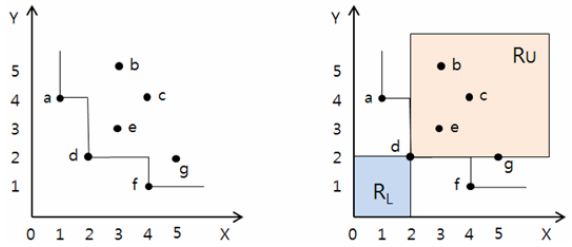
Keywords-skyline computation; decision making system; dynamic determination of query region

I. INTRODUCTION

The skyline of a multidimensional data set is the maximal subset that is considered to be optimal to the user's requirements. Skyline computation is useful especially when a huge amount of data set is analyzed, since it selects only a small amount of useful data from such a big data set. Skyline computation is first introduced in [1], and a lot of research has been conducted for computing the skyline in various environments. Recently, skyline computation for mobile users gets special attention.

The skyline of a multidimensional data set is defined as follows. Let A be a d dimensional data set, and let $p = (p_1, p_2, \ldots, p_d)$ and $q = (q_1, q_2, \ldots, q_d)$ be data elements in A, where p_i and q_i, $1 \le i \le d$, are the values of dimension i of p and q, respectively. If $p_i \le q_i$, for all i, and $p_j < q_j$ for at least one j, $1 \le i, j \le d$, then p is said to dominate q [5]. The skyline of A is defined to be the maximal subset whose elements are not dominated by any other elements of A.

For example, Fig. 1(a) shows a 2 dimensional data set consisting of 7 elements whose (X, Y) coordinates are (1, 4), (3, 5), (4, 4), (2, 2), (3, 3), (4, 1), and (5, 2). Here, we can see that data element b = (3, 5) is dominated by data element a = (1, 4), since $1 \le 3$, $4 \le 5$, and $1 < 3$. Data elements a, d, and f form the skyline of the set, since they are not dominated by any other elements of the set. Fig. 1(b) shows two regions R_L and R_U that are related to data element d. Data element d belongs to the skyline because there is no element in region R_L. Data elements in region R_U are dominated by d. Here they are b, c, e, and g. Here, let us assume that the data elements represent restaurants, the X and Y axes represent the average food price and reputation rank of the corresponding restaurants, respectively. Then, restaurants a, d, and g are considered to be 'good restaurants' that can be recommended to the user.



(a) The skyline of the set (b) Dominance relationship among data

Figure 1. A 2-dimensional data set and its skyline.

Let us now consider the case where the user is moving on the 2-dimensional space. Suppose also that the user is driving around, and wants to find quality restaurants nearby. Fig. 2 shows the restaurants of Fig. 1 on a 2-dimensional map. Assume that the user is driving inside region R. Without limiting the query region R, the user is recommended restaurants a, d, and f. However, if the user sets the region of interest to be R, restaurant e becomes alive, since no other elements in R dominate e. This is why we want to set the region of interest at query time.

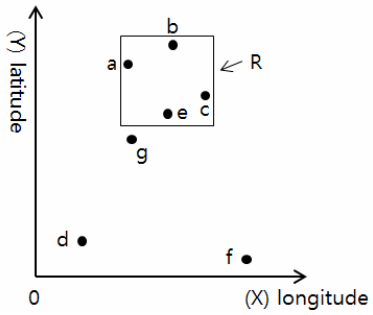


Figure 2. Positions of the user and the data elements.

In this paper, we propose an efficient algorithm for computing the skyline for the data elements that belong to the instantly determined query region at the query time. The

algorithm is divided into two phases. The first phase scans the entire data set, and filters out the elements that are dominated by some other elements. The remaining elements are partitioned and indexed by a quadtree. For each quadtree node, skyline for the corresponding region is precomputed. In the second phase, a lot of user queries are processed using the intermediate results computed at the first phase.

The paper is organized as follows. In Section 2, related works are given. In Section 3, the proposed skyline computation algorithm is presented. In Section 4, the performance of the algorithm is evaluated by experiments. In Section 5, we conclude our work.

II. RELATED WORKS

Skyline computation algorithm for a multidimension data set was first proposed by [1]. [3] solves the problem in the database context, and gives efficient algorithms called BNL (block nested loop) and D&C (divide and conquer). [4] uses sorting and filtering to speed up the computation. [5] handles a very large data set stored in hard disk. [6] assumes a client server environment, and [7] solves the problem in parallel and distributed environment. [5] uses filters and sorting efficiently. [6] uses a stop line in order to stop transmitting data from the server after reaching certain point. [7] uses an angle based space partitioning scheme for load balancing among computing nodes. [12] modifies the scheme in [7] for efficient filtering. Progressive skyline computation is given in [8, 9]. [8] uses a bitmap and indexing algorithm and produces skyline elements one by one. [9] interacts with the user during the skyline computation and adjusts the results accordingly. Skyline computations for mobile environments are proposed in [2, 11, 13, 15]. [11, 15] precomputes the skyline using static attributes, and adjusts it using the dynamically changing position of the moving user. [2] is similar to [11] except that it assumes the network of roads. [13] partitions the map into many blocks, and precomputes the skylines for each block. [14] uses P-Cube and signatures to answer boolean queries as well as preference queries together.

III. SKYLINE COMPUTATION

We now present the proposed skyline computation algorithm. The algorithm is divided into two phases. The first phase scans the input data set, and eliminates as many data elements as possible using efficient filters. Only static attributes are used at this time. Assume that the remaining data elements are (virtually) placed on a 2-dimensional map. They are partitioned into blocks, and each block is indexed by a quadtree leaf node. For each quadtree node including the internal nodes, the skyline is computed from the data elements that belong to the corresponding block. The second phase of the algorithm is to process the skyline computation queries issued by many moving users who define their own query regions.

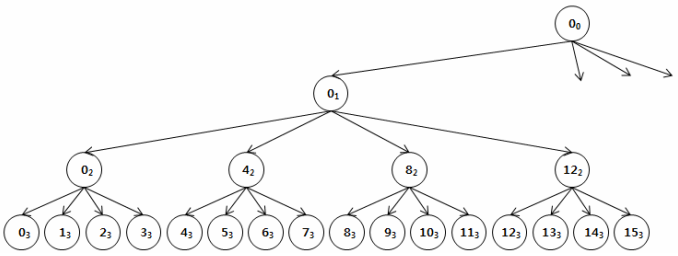
Let us now explain the quadtree index used in the first phase of the algorithm. Fig. 3(*a*) shows the partitions established on a 2-dimensional map. For example, we assume that there are 64 blocks on the 2-dimensional map. Fig. 3(*a*) shows the row-major ordering of the blocks, and Fig. 3(*b*) shows the corresponding shuffled-row-major ordering of the blocks. Fig. 3(*c*) is a part of the quadtree built on Fig 3 (*b*). Fig. 3(*c*) shows only the first 16 blocks. Each node of the quadtree is represened by i_j. Here i represents the shuffled-row-major number of the top leftmost cell of the block, and j represents the level of the node. From the total number of the blocks and the node index, we can compute the size of the block represented by a quadtree node. Note also that the index of the leftmost child node of node i_j are i_{j+1}. For example, the leftmost child of 8_2 is 8_3, and 8_3, 9_3, 10_3, and 11_3 are the four children of 8_2.

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

(*a*) Row-major order

0	1	4	5	16	17	20	21
2	3	6	7	18	19	22	23
8	9	12	13	24	25	28	29
10	11	14	15	26	27	30	31
32	33	36	37	48	49	52	53
34	35	38	39	50	51	54	55
40	41	44	45	56	57	60	61
42	43	46	47	58	59	62	63

(*b*) Shuffled row-major order



(*c*) Quadtree representing the first 16 blocks

Figure 3. Partitioning and quadtree indexing the 2-dimensional map.

The last step of the first phase of the algorithm is to compute the skyline for each quadtree node. Let us assume that $S(i_j)$ is the skyline computed for the node i_j. Assume first that the skyline for each node 0_3, 1_3, 2_3, … , 15_3 is already computed. Now consider the computation of the skyline for the block represented by 8_2, which is the parent of four blocks, 8_3, 9_3, 10_3, and 11_3. We can see that $S(8_2)$ is a subset of the union of $S(8_3)$, $S(9_3)$, $S(10_3)$, and $S(11_3)$. Using this fact, we can easily and promptly compute the skyline of each parent node of quadtree nodes. The first phase of the algorithm can be formally described as follows. Assume that s is the size of the sample, f is the number of the filters, and n is the number of blocks for each dimension of the map.

Phase I : *Precomputation Phase*

[Step 1] [*Filter determination*]
Choose s random samples from the given multidimensional data set A, and select f best possible filters.

[Step 2] [*Data Scanning and filter upgrade*]
Scan the data set A. For each data p, if p is dominated by the filters, discard it. Otherwise, place p to the predetermined block of the map. And if p dominates a filter, replace the filter with p.

[Step 3] [*Skyline Computation for quadtree leaves*]
Sort the data elements placed in each block. Read the sorted elements in order, and select skyline elements.

[Step 4] [*Skyline Computation for the internal nodes of the quadtree*]
Merge the skyline of each child node, and compute the skyline of the corresponding parent node.

Figure 4. Various cases of query region establishment.

We explain now the second phase of the skyline computation algorithm. It is in fact the query processing of many moving users. The user sets up his/her query region at the query time. Query region can be established in 4 ways as in Fig. 4. Fig. 4(*a*) shows the case that the query region consists of one quadtree node. In this case, the precomputed skyline is returned. In Fig. 4(*a*), precomputed $S(48_1)$ is returned. In Fig. 4(*b*), the query region is the union of $S(12_2)$, $S(24_2)$, $S(36_2)$, and $S(48_2)$. Thus, the skyline is computed by first merging the four skylines, and then extracting the final skyline using a step similar to [step 4] of the first phase of the algorithm. In Fig. 4(*c*), the skyline is computed from $S(48_2)$, $S(15_3)$, $S(26_3)$, $S(27_3)$, $S(37_3)$, and $S(39_3)$. For these three cases, the skyline of the query region is a combination of the blocks represented by a quadtree node. Thus, the skyline of the query region can be obtained easily from the precomputed intermediate results.

However, there can be a case where the query region does not cover the entire blocks as in Fig. 4(*d*). In this case, $S(48_2)$ is precomputed. $S(37_3)$, $S(39_3)$, $S(45_3)$, $S(56_3)$, and $S(57_3)$ need to be adjusted. For example, let us take a look at $S(37_3)$. From $S(37_3)$, we select skyline elements for the query block, and let the set be $SI(37_3)$. Let $SO(37_3) = S(37_3) - SI(37_3)$. Now, we need to select the data elements of $A(37)$ that are dominated by $SO(37_3)$, compare them with all the $SI(37_3)$ to determine the skyline of the region represented by the intersection of $S(37_3)$ and the query region. Let us say the skyline $SF(37_3)$. The skyline of the query region is computed from $S(48_2)$, $SF(37_3)$, $SF(39_3)$, $SF(45_3)$, $SF(56_3)$, and $SF(57_3)$. The summary of the second phase of the algorithm is shown below.

Phase II : *Query Processing Phase*

[Step 1] [*Quadtree node extraction*]
Select all the quadtree nodes that belong to the query region. Here, parent nodes are perferred to child nodes.

[Step 2] [*Query region is a combination of quadtree nodes*]
If the query region can be obtained from a combination of quadtree nodes, merge the skylines of the quadtree nodes, and compute the final skyline of the quadtree region.

[Step 3] [*Query region partially overlaps with a qudtree node*]
Adjust the skyline for the blocks with which query region partially overlaps. Merge the skylines of the quadtree blocks that belong to the query region.

IV. PERFORMANCE EVALUATION

We conducted some experiments on a PC equipped with a 2.40GHz Intel *i*5 CPU and 4.0GB of main memory. The programs are written in C, and are run in .NET environment. The data set used for the experiment is artificially generated. It is a 4 dimensional data set with Gaussian distribution. The number of data elements is 10millions.

Let us first show the experimental results for the first phase of the algorithm. In the first step, we select 1,000 samples, and use 10 best of them as the initial filters. In the second step, 10 millions of data elements are read, and leave the data items that are not dominated by the filters. In case a data element , p,

dominates a filter, *f*, *f* is removed from the filter set, and *p* gets included into the filter set. The number of blocks on the 2-dimensional map is 16 x 16 = 256. After the filtering step, only 19% of the entire input data set is left. They are placed to 256 blocks of the map, and there are 7,479 elements in each block, in average. After executing [step 3] of the algorithm, each block has 117 skyline elements in average.

Experimental results shown in Table I is obtained from [step 4] of the first phase of the algorithm. We can see that the size of the skyline computed for the leaf nodes on level 4 is 117 in average. The size of the skyline of their parent blocks is 224 in average. The area covered by a parent node is 4 times that of its child block. However, the size of the skyline only increases less than double. It shows that precomputation of the skyline for each internal node of the quadtree helps the query processing in the second phase.

TABLE I. THE SIZE OF THE SKYLINE FOR EACH QUADTREE NODE.

Node level	Number of blocks represented by the node	Size of skyline	Density Decrement
4	1	117	-
3	4	224	0.479
2	16	414	0.462
1	64	733	0.442
0	256	1,261	0.431

Table II. shows the execution time taken by each step of the first phase of the algorithm. Data scanning through partitioning takes 2.546 seconds, and the skyline computation for the leaf nodes takes 1.154 seconds, and the skyline computation for the blocks represented by the internal nodes of the tree takes not much time. For example, the skyline computation for each node on level 2 from the skylines for nodes on level 3 takes 0.017seconds only. Note also that it takes rather longer to compute the skyline of the root block from the skylines of the leaf nodes directly.

TABLE II. EXECUTION TIME OF THE FIRST PHASE OF THE ALGORITHM

Steps of the algorithm	Execution time (sec)
Data scanning, filering, and partitioning	2.546
Skyline computation for level 4 nodes	1.154
Skyline computation for level 3 nodes	0.023
Skyline computation for level 2 nodes	0.017
Skyline computation for level 1 nodes	0.012
Skyline computation for level 0 node	0.008
Skyline computation for level 0 node from level 4 nodes	0.84

Experimental results for the second phase of the algorithm, that is the query processing phase, are shown below. Here we assume that the experimental query regions cover 16 blocks in average. We place the query region randomly, and check to see if which levels of the quadtree nodes are used for the computation. It shows that the frequency of using level 4 nodes is 77%, but it only covers 44% of the total area. On the other hand, the frequency of level 3 node usage is only 22%. However they cover half of the entire area of the query block.

Finally, we use query blocks of size 4.5x4. In this case, there are some blocks that are partially covered by query blocks. In detail, 20% of the region needs adjustment. Execution time for such a query is 6.84. Note that it is better to reduce the size of the blocks, and try not to have partial blocks inside the query blocks.

TABLE III. QUERY EXECUTION (QUERY REGION COVERS 4 BLOCKS)

Node level	Block size	Usage frequency	Area covered
4	1	77%	44%
3	4	22%	50%
2	16	1%	6%

V. CONCLUSIONS

We presented a skyline computation algorithm that allows the user to determine his/her query regions at query time. Such a setting is useful especially when the user is moving around on a 2-dimensional space, and wants to be recommended quality data elements nearby. In order to improve the performance of the skyline computation algorithm, the 2-dimensional space on which the user is moving around is partitioned, and then indexed using a quadtree. The skyline for each node of the tree is precomputed. Experimental results show that quadtree indexing and precomputation of the skyline for each quadtree node make the entire process efficient. The grain size of the blocks affects the performance of the algorithm. In case the partitioning of the 2-dimensional space is fine grained, and the query region exactly overlaps with the blocks of the 2-dimensional space, the algorithm gives reasonably good performance.

REFERENCES

[1] H. T. Kung, F. Luccio, and F. P. Preparata. “On finding the maxima of a set of vectors,” *Journal of the ACM*, vol. 22, no. 4, pp. 469–476, 1975.

[2] S. Jang, J. You, “An Efficient Pre-computing Method for Processing Continuous Skyline Queries in Road Networks,” Journal of KIISE:Database, Vol. 36, No. 4, pp. 314-320, Aug. 2009.

[3] S. Borzsonyi, D. Kossmann, and K. Stocker, “The skyline operator,” In *ICDE* 2001, pp. 421–430, Heidelberg, Germany, Apr. 2001.

[4] J. Chomicki, P. Godfrey, J. Gryz, D. Liang, “Skyline with presorting,” In *ICDE* 2003, pp. 717–719, India, Mar. 2003.

[5] P. Godfrey, R. Shipley, and J. Gryz, “Maximal vector computation in large data sets,” In *VLDB* 2005, pp. 229–240, Trondheim, Norway, Aug. 2005.

[6] I. Bartolini, P. Ciaccia, M. Patella, “SaLSa: Computing the skyline without scanning the whole sky,” In *CIKM* 2006, pp. 405-414, Arlington, Virginia, USA, Nov. 2006.

[7] A. Vlachou, C. Doulkeridis , Y. Kotidis, “Angle-based space partitioning for efficient parallel skyline computation,” In *ACM SIGMOD* 2008, pp. 227–238, Vancouver, Canada, Jun. 2008.

[8] K.-L. Tan, P.-K. Eng, and B. C. Ooi, “Efficient progressive skyline computation,” In *VLDB* 2001, pp. 301–310, Roma, Italy, Sep. 2001.

[9] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: An online algorithm for skyline queries," VLDB 2002, pp. 275~286, Hong Kong, China, Aug. 2002.

[10] Y. Tao and D. Papadias, "Maintaining Sliding Window Skylines on Data Streams," IEEE TKDE, Vol. 18, pp. 377~391, 2006.

[11] Z. Hung, H. Lu, B. Ooi and A.Tung, "Continuous skyline queries for moving objects," IEEE TKDE, Vol. 18. pp.1645-1658, 2006.

[12] J. Kim, M Kim, "Skyline Computation Using Adaptive Filters," International Journal of Compter Science and Information Technology & Security (IJCSITS), Vol. 2, No. 2, pp. 431-434, Apr. 2012.

[13] J. Kim, M. Kim, "An Extended Skyline Computation Scheme for Recommendation Services in Mobile Environments," Journal of KIISE:Computing Practice and Letters, Vol. 18, No. 7, pp. 558-562, Jul, 2012 (in Korean).

[14] D. Xin, J. Han, "P-Cube: Answering Preference Queries in Multi-Dimensional Space," In ICDE 2008, pp. 1092-1100, Cancun, Mexico, April, 2008.

[15] M. Sharifzadeh, C. Shahabi, "The Spatial Skyline Queries," In VLDB 2006, pp. 751-762, Seoul, Korea, Sep. 2006.

AUTHORS PROFILE

Myung Kim received her BS degree in Mathematics, Ewha Womans University in 1981. She received her MS degree in Computer Science at University of Minnesota, USA, in 1990. She received her PhD in Computer Science, University of California, Santa Barbara, 1993. She is currently a faculty member of the Department of Computer Science and Engineering, Ewha Womans University. Her research interests include Data analysis, Real time recommendation Systems, Information Retrieval, and High performance computing.