Dynamic Skyline Computation in a Mobile Environment

Myung Kim[1], Jihyun Kim[2]

[1,2]Department of Computer Science and Engineering, EwhaWomans University,
Seoul, Korea

Abstract

Skyline computation is an operation that extracts from a multidimensional data set the maximal subset whose elements optimally satisfy the user's requirements. This operation is considered to be useful to a mobile user, if such subset can be promptly computed from the data set that surrounds the user within a certain distance limit at any given time. In this work, we propose an efficient algorithm for continuously providing a mobile user with such skylines of the data sets located near him/her.The algorithm computes the skylines in sequence according to the movements of the mobile user.The *i*-th skyline in the sequence is computed by updating the (*i*-1)-th skyline. The performance of the proposed algorithm is shown by experiments.

Keywords:**Skyline Computation,Recommendation Service, Mobile User, Dynamic environment.**

1. Introduction

The skyline of a multidimensional data set consists of all the elements that satisfy the users' requirements better than any other elements of the given set. Skyline computation is a very useful operation in decision making especially when dealing with a large data set. Recently, a lot of skyline computation algorithms have been reported for various environments [1, 2, 4, 6, 8, 12]. Here, we focus on a mobile environment meaning that the user is moving around, and is interested in obtaining instantly the skyline of the data set (or objects) that are located within a certain distance from him/her. We are interested in providing a mobile user with such skyline at any given time.

Formally, the skyline of a multidimensional data set is defined as follows [2]. 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, $1 \leq i \leq d$, are the values of dimension i of p and q, respectively. If $p_i \leq q_i$, for all i, and $p_j < q_j$ for at least one j, $1 \leq i, j \leq d$, then p is said to *dominateq*. 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 8 elements whose (X, Y) coordinates are (1, 5), (3, 6), (2, 4), (3, 2), (4, 1), (4, 3), (5, 4) and (6, 2). Here, we can see that data element b = (3, 6) is dominated by data element a = (1, 5), since $1 \leq 3$, $5 \leq 6$, and $1 < 3$. Data elements a, c, d, and e form the skyline of the set, since they are not dominated by any other elements of the set. Note that all the data elements that are dominated by at least one of the skyline points are located above the line named "skyline" in the figure. We now apply the skyline computation to a restaurant recommendation problem. Assume that the data elements in Fig. 1 represent restaurants, the X and Y axes represent the average food price and the reputation of the corresponding restaurants, respectively. Assume also that the smaller rank is better. Then, restaurants a, c, d, and e are considered to be 'good restaurants' that can be recommended to the user.

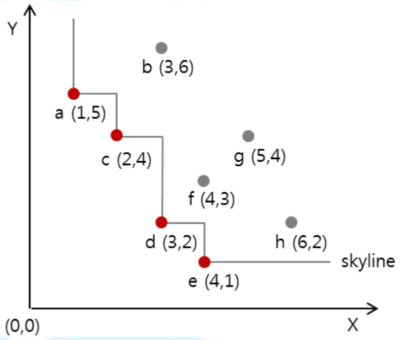


Fig. 1 A 2-dimensional data set and its skyline.

Let us now turn our attention to a situation where users are moving around, and each of them wants to find quality restaurants nearby. In this case, we can see that the multidimensional data sets of users'interests (i.e., neighbor regions)vary depending on the location of the users at query time. Here, we propose an algorithm for efficiently computing the skylines for such mobile users. This algorithm can be used for a recommendation system that serves many mobile users or clients concurrently.

Our algorithm works as follows: The entire plane (or a map), on which the users are moving around, is partitioned into fine grained grid cells. Let's say, the plane is divided into 1km × 1km square grid cells. A mobile user, p, is initially placed at a grid cell on the map, as shown in Fig. 2(a). Assume that user p is interested in finding a quality restaurant in the surrounding area of 3 cells × 3 cells. Here R_1 is such region.We first compute the skyline,S_1, of the data set in R_1.

Mobile users can move in 8 directions: up, down, left, right, upper left, upper right, lower left, and lower right. Fig. 2(a) shows that user p is at the center of region $R_1$and p's next move out of region $R_1$is in the upper right direction. Sometime later, user p will arrive at the center of region R_2,as in Fig. 2(b). The algorithm then computes the skyline of the data set in region R_2, which is S_2.Skyline computation will be carried out similarly, whenever user p enters a neighboring grid cell.Thus, for the example in Fig. 2, skylines, S_1, S_2, S_3, and S_4, are computed for R_1, R_2, R_3, and R_4 in sequence.

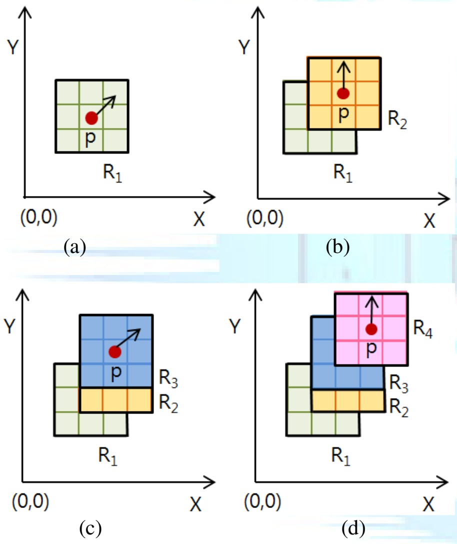


Fig. 2 User$p$'s movement and the corresponding regions of interests.User p moves from the center of R_1 to the centers of R_2, R_3, and R_4 in sequence.

The algorithm computes the skylines for regions R_1, R_2, ... ,R_{i-1}, R_i,R_{i+1}, ... in sequence. The skyline,S_i, for region $R_i$is obtained by updating S_{i-1} for region R_{i-1}. In order to efficiently do so, we developed two operations called *DeleteBlock*( ) and *AddBlock*( ). *DeleteBlock*( ) is used to eliminatethe influence of area R_{i-1}- R_i. *AddBlock*( ) is used to include the influence of area R_i - R_{i-1}.Performance evaluation has been conducted,and shows that the algorithm is efficient and useful.

The paper is organized as follows: In Section 2 we briefly overview the background research results. In Section 3 we propose our skyline computation algorithm. In Section 4, performance evaluation is given. In Section 5, we conclude our work.

2. Related Works

Skyline operation was first introduced in [2]. Since then a lot of research has been conducted to develop efficient algorithms for skyline computation [1, 2, 3, 6, 9, 11,14].Skyline computation algorithms assume various conditions, and a 'mobile environment' is considered to be one such condition that recently receives a lot of attention [4, 5, 7, 8, 10, 12, 13].

Early attempt [1, 2, 11, 14] is to use the location information of the user to extract from the original data set the subset of interest to the user. Skyline is computed on the selected subset. R-tree index, Grid index, or angle-based partitioning schemes are used to speed up the execution of the skyline computation.

Algorithm in [3,4, 10] continuously computes the skylines from the data sets surrounding the mobile user. It uses a grid structure to manage the entire data set and creates a skyline influence region to filter out the unnecessary skyline processing over the data sets. However, this method assumes that the distribution of the data should be uniform and the performance of the algorithm is affected by the size of grid cells.

Algorithms proposed in [6, 7, 8] also assume a mobile environment. [6] and [8] uses a quadtree structure in order to represent 2-dimensional space efficiently, and [7] computes extended skyline that contains quality data elements that is near the mobile user. However, these are not real time algorithms.

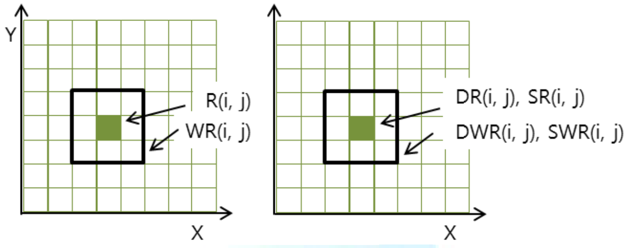
A distributed mobile environment is assumed in [5, 9,12, 13].Skylines are computed in sequence from the data stream, one skyline per sliding window. It maintains separate timer for each mobile user, thus maintenance overhead is relatively high. The algorithm proposed in [5] consists of three processing phases, and each phase is to reduce the network bandwidth consumption, network delay and query response time. However, in case the user moves around quickly, the overhead for index maintenance and skyline updates becomes very high.

The reported algorithms are not very much adequate to serve many mobile users in real time. In this paper, we propose a skyline computation algorithm for mobile users in such a way that the skyline for each user is updated as soon as he/she moves to a predetermined grid type region. Thus, skylines are computed in sequence. In order to speed up the execution of the algorithm, we propose two functions, *AddBlock*( ) and *DeleteBlock*( ), to efficiently update the pre-computed skyline in sequence. In our algorithm, we assume only one user. However, it can serve a lot of users simultaneously.

3. Skyline Computation for a Mobile User

Consider a two dimensional space R as in Fig.3 (a). Assume that R is partitioned into an $n \times n$ grids, and $R(i, j)$ represents the cell (or region) located at i-th row and j-th column of the grids. We also define‘wide region’,$WR(i, j)$, as the block (or region) that consists of $R(i, j)$ and the surrounding eight cells, that is placed at left, right, up, down, upper left, upper right, lower left, and lower right corner of $R(i, j)$. Formally, $WR(i, j)$ consists of $R(k, l)$, where $i - 1 \leq k, l \leq i + 1$.

Suppose that DR is a multidimensional data set that we are concerned, and their elements are placed on the 2 dimensional space$R$, as defined above. It is like restaurants (DR) are all over the place on the map (R). The set of data elements of R, i.e., DR, that is located inside $R(i, j)$ is called $DR(i, j)$. The skyline of $DR(i, j)$ is called $SR(i, j)$. $DWR(i, j)$ represents the data set that consists of the data elements of DR that belongs to $WR(i, j)$. $SWR(i, j)$ is the skyline of the set $DWR(i, j)$. See figure 3.



(a) Regions on the map (b) Data sets and their skylines for regions

Fig. 3 A 2-dimensional space and the definitions of regions and data sets located on the regions.

Suppose now that a user moves around on the map (R), and wants to find a set of quality restaurants nearby. The purpose of our algorithm is to provide him/heras soon as he/she enters $R(i, j)$good quality restaurants in$WR(i, j)$, that is, $SWR(i, j)$. Formally speaking, assume that the user is initially in $R(i_0, j_0)$, and moves in sequence to $R(i_1, j_1)$, $R(i_2, j_2)$, $R(i_3, j_3)$, … , $R(i_{k-1}, j_{k-1})$, $R(i_k, j_k)$, …, where $|i_{k-1} - i_k| \leq 1$ and $|j_{k-1} - j_k| \leq 1$. That is, the user moves to one of the eight adjacent regions. Our algorithm computes $SWR(i_0, j_0)$, $SWR(i_1, j_1)$, $SWR(i_2, j_2)$, $SWR(i_3, j_3)$, … , $SWR(i_{k-1}, j_{k-1})$, $SWR(i_k, j_k)$, … in sequence. Furthermore, $SWR(i_k, j_k)$ is computed as soon as the user enters region $R(i_k, j_k)$.Our skyline computation algorithm can be described as follows.

We explain the algorithm in detail. The first step of the algorithm is the initialization step, and is thus executed only once right before the recommendation service begins. However, in case the original data set DR gets updated later on, each $SR(i, j)$ can be updated independently and efficiently. For example, if a data element p is inserted to $DR(i, j)$,p only needs to be compared with the elements in $SR(i, j)$. If data element q is deleted from $DR(i, j)$-$SR(i, j)$, no change should be done on $SR(i, j)$. If q is from $SR(i, j)$, then the elements in $DR(i, j)$-$SR(i, j)$ that are dominated by q need to be compared with all the element of $SR(i, j)$.

Algorithm 1:
Dynamic Computation of Skylines for a mobile user

[Step 1] [*Compute the skyline for each* $R(i, j)$]
Scan the multidimensional data set DR, and partition the elements into $DR(i, j)$, $0 \leq i \leq n - 1$, $0 \leq j \leq n - 1$. Compute the skyline$SR(i, j)$ for each data set $DR(i, j)$.

[Step 2] [*DynamicSkyline computation for user* Q]
Supposer that the user Q is initially located at $R(i_0, j_0)$. Compute $SWR(i_0, j_0)$. Keep track of Q’s movement. As soon as Q moves from $R(i_{k-1}, j_{k-1})$ to one of its eight neighbor region, $R(i_k, j_k)$, compute $SWR(i_k, j_k)$ by updating $SWR(i_{k-1}, j_{k-1})$. Do this until the user stops moving.

The second step of the algorithm is in fact a continuous query processing part. In this algorithm, we describe the step only for one mobile user. However, query processing for many users can also be concurrently done in a similar manner. The mobile user’s initial position is$R(i_0, j_0)$. Thus, we first compute $SWR(i_0, j_0)$, which is the skyline of $DWR(i_0, j_0)$. In other words, $SWR(i_0, j_0)$ is the skyline of the data set that is located in wide region $WR(i_0, j_0)$ whose center is $R(i_0, j_0)$.

During the query processing, let us say that the user is in $R(i_{k-1}, j_{k-1})$ andis about to move to $R(i_k, j_k)$. We then update the current skyline $SWR(i_{k-1}, j_{k-1})$ to obtain the next skyline $SWR(i_k, j_k)$.At this moment, let us take a moment to think about how to compute $SWR(i_k, j_k)$, for some $k \geq 0$. One might suggest to compute $SWR(i_k, j_k)$, for all $k \geq 0$ in advance in step 1.However, we assume that data set $DR(i, j)$ located in each $R(i, j)$ gets updated frequently. One such example might be the data on the restaurants in the street. They get updated frequently. Thus we decide to compute $SWR(i_k, j_k)$ on the fly.

Let us now turn our attention to the computation of $SWR(i,j)$, which is the skyline for the wide region whose center is $R(i, j)$. One simple way of doing it is to obtain$DWR(i, j)$ from $\bigcup_{k=i-1}^{i+1} \bigcup_{l=j-1}^{j+1} DR(k, l)$ and compute the skyline from $DWR(i, j)$ directly. However, it would be very time consuming to compute the skyline this way, in case therecommendation system serves many mobile users.Thus, we propose an algorithm for updating the current skyline $SWR(i_{k-1}, j_{k-1})$ to get the next skyline $SWR(i_k, j_k)$.

We describe the scheme in detail. We are interested in computing the skyline for wide region *SWR*(*i*, *j*), assuming that the skyline for region *R*(*k*, *l*)'s, *SR*(*k*, *l*), $i-1 \le k \le i+1$ and $j-1 \le l \le j+1$, were already computed. It is obvious that the skyline for wide region*SWR*(*i*, *j*) is a subset of $\bigcup_{k=i-1}^{i+1}\bigcup_{l=j-1}^{j+1} SR(k,l)$. In fact, *SWR*(*i*, *j*) can be represented by set$A = \bigcup_{k=i-1}^{i+1}\bigcup_{l=j-1}^{j+1} TR(k,l)$, and each *TR*(*i*, *j*) of set *A*is a subset of *SR*(*i*, *j*), as in Fig. 4.In the figure,white circles represent *SR*(*i*, *j*)'s and black circles inside white circles represent *TR*(*i*, *j*)'s. Note that the elements in set *SR*(*i*, *j*)-*TR*(*i*, *j*) are the newly found elements during the computation of *SWR*(*i*, *j*), that are dominated by an element in set $\bigcup_{k=i-1}^{i+1}\bigcup_{l=j-1}^{j+1} SR(k,l)$, where $k \neq i$ and $l \neq j$. In addition, for each *SR*(*i*, *j*), we define *UR*(*i*,*j*,*k*,*l*), $i-1 \le k \le i+1$, $j-1 \le l \le j+1$, $k \neq i$, and $l \neq j$, such that the elements of *UR*(*i*,*j*,*k*,*l*) are the elements of *SR*(*i*, *j*) that are dominated by an element of *SR*(*k*, *l*). Thus there are 8 *UR*(*i*,*j*,*k*,*l*)'s for *SR*(*i*, *j*) since $k \neq i$, and $l \neq j$.

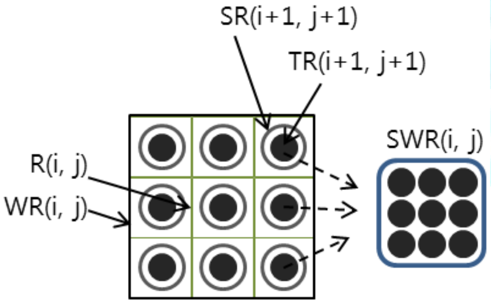
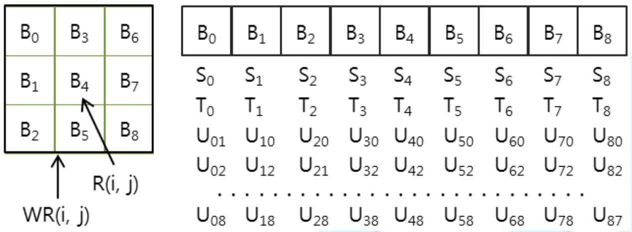


Fig. 4 Skyline for wide region *SWR*(*i*, *j*).



B_0	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8
S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
U_{01}	U_{10}	U_{20}	U_{30}	U_{40}	U_{50}	U_{60}	U_{70}	U_{80}
U_{02}	U_{12}	U_{21}	U_{32}	U_{42}	U_{52}	U_{62}	U_{72}	U_{82}
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
U_{08}	U_{18}	U_{28}	U_{38}	U_{48}	U_{58}	U_{68}	U_{78}	U_{87}

Fig. 5 Renamed regions and data sets.

We now describe the computation of *SWR*(*i*, *j*). In order to simplify the description of the algorithm, we rename the terms regarding regions and data sets as in Fig. 5. First, 9 regions (or grid cells) of *WR*(*i*, *j*) are renamed B_0, B_1, ... , B_8. Skyline for B_i, $0 \le i \le 8$, is renamed S_i.The subset of S_i that consists of *SWR*(*i*, *j*) is called T_i. The set S_i- T_i is divided into 8 sets, U_{io}, U_{i1}, ... , U_{ik}, ... ,U_{i8}, where $i \neq k$ and the union of T_i and$\bigcup_{k=0}^{8} U_{ik}$ becomes S_i. Note that the elements of U_{ik} are the elements of S_i but they are dominated by an element of S_k, thus they do not belong to *SWR*(*i*, *j*).

Suppose that the mobile user is initially at *R*(*i*, *j*).We first need to compute *SWR*(*i*, *j*). Computation of *SWR*(*i*, *j*) is given in step 1 of Algorithm 2. In the algorithm, *SWR* represents the skyline for the wide region whose center is the region where the mobile user is currently located. Initially, *SWR* is the same is *SWR*(*i*, *j*). Note that *AddBlock*( ) is called 9 times. When *AddBlock* (*k*) is called, T_k and U_{ki} are computed, and all the previously computed T_i and U_{ij}, $0 \le i,\ j \le k-1$, are updated. *AddBlock*( ) is described in Algorithm 3.

As mentioned earlier, the purpose of our algorithm is to provide the mobile user with the skyline of the wide region which he/she just moves in. Thus, skylines are updated in sequence by the while loop of step 2 of algorithm 2.Every update of the skyline is done in two phases as follows. In order to simplify the description, we rename the blocks so that blocks B_0, B_1, ... ,B_a are no longer inside the wide region of the mobile user. For example, if the user moves right, B_0, B_1, B_2 are such blocks. If the user moves diagonally, B_0, B_1, B_2, B_3, B_4 are such blocks. In the first phase, the skyline *SWR* is updated by eliminating the influence of those blocks. Blocks are eliminated one by one by *DeleteBlock*(*k*), which is described in Algorithm 4. What is done in *DeleteBlock*(*k*) is to find the data elements in U_{jk}, $k+1 \le j \le 8$ that were dominated by an element of T_k.We then check to see if they can be skyline elements again. Newly found skyline elements are then inserted to the set they belong to.

Let us take a look at an example. In case that the user moves to the right, we delete T_0, T_1, T_2, and update T_3~T_8 so that the influence of T_0, T_1, T_2 is removed. Suppose that *DeleteBlock*(0) and *DeleteBlock*(1) were already called, and *DeleteBlock*(2) is about to be called. That means, it is time to delete T_2, and to update T_3~$T_{8.}$ In order to do so, we use U_{32}, U_{42}, ... ,U_{82}. These are the elements that were deleted because they are dominated by an element of T_2. Now these U_{i2}, $3 \le i \le 8$, are compared with all T_j, $3 \le j \le 8$ and $j \neq i$, are compared. If the elements of U_{i2},$\le i \le 8$ are not dominated by any element in this step, they come back to the corresponding skyline set, which is T_i.

In the second phase of the step 2 of algorithm 2 is to add skyline elements from the blocks that are inserted to the wide region of the mobile user. For the simplicity, assume that B_0, B_1, ... ,B_{8-a} are remaining blocks and B_{8-a+1}, ... , B_8 are the blocks to be added. Insertion of these blocks are done by calling *AddBlocks*(*k*), $8-a+1 \le k \le 8$, where *a* is the number of blocks to be added.

Algorithm 2:
Computationof the skylines SWRin sequence

```
[Step 1]  [ Computation of the first skyline ]
          for ($k$ = 0; $k$< 9; $k$++)
             AddBlock ($k$);
[Step 2]  [ Adjust the skyline acording to the next move ]
          while ( 1 ) {
             (1) determine the next move; rename the blocks so that $\{B_0, B_1, \dots, B_a\}$ are the blocks to be out of the mobile user's next wide region. If the move is up, down, left, or right $a$ = 2, otherwise $a$ = 4;
             (2) for ($k$ = 0; $k$<= $a$; $k$++)
          DeleteBlock($k$);
             (3) rename the remaining blocks
                    $\{ B_0, B_1, \dots, B_{8-a}\}$
             (4) assume that $\{ B_{8-a+1}, \dots, B_8 \}$ are the blocks to be added to the wide region of the mobile user.
             (5) for ($k$ = 8-$a$+1; $k$<= 8; $k$++)
          AddBlock($k$);
          }
```

Algorithm 3: *AddBlock*(k)

```
  $T_k = S_k$;
  for ($m$ = 0, $m$<$k$; $m$++){
       // compare $T_m$ with $T_k$
       for each ($p$, $q$), where $p$ in $T_m$, and $q$ is in $T_k$ {
if $p$ is dominated by $q$, move$p$ to$U_{mk}$ ;
else if $q$ is dominated by $p$, move$q$ to$U_{km}$ ;
            else $p$ stays in $T_m$, and $q$ stays in $T_k$;
     }
```

Algorithm 4: *DeleteBlock*(k)

```
  Delete $T_k$from $SWR$ and discard $U_{ki}$'s;
  for ($m$ = $k$ + 1;$m$< 9; $m$++){
    for ($j$ = $m$ + 1; $j$< 9; $j$++){
      // compare $U_{jk}$ with $T_m$
      for each ($p$, $q$) where $p$ in $U_{jk}$ and $q$ is in $T_m${
            if $p$ is dominated by $q$, move$p$ to $U_{jm}$;
            else if $q$ is dominated by $p$, move$q$ to $U_{mj}$;
      }
      if  $U_{jk}$ is not empty, move all the elements to $T_j$.
    }
}
```

4. Performance Evaluation

We conducted some experiments on a PC equipped with a 2.40GHz Intel Q6600 CPU and 4.0GB of main memory.The test programs are in C, and run in the Microsoft .NET environment.

The purpose of our experiments is to show how efficiently the proposed algorithm runs so that it can be used to serve a lot of mobile users concurrently. Thus it is assumed that the area the users are moving around is already divided into many grid cells, and the data set the users are interested in is already partitioned, and their skylines are computed. In other words, for each grid cell $R(i, j)$, Data set $DR(i, j)$ is provided and its skyline$SR(i, j)$ is computed using previously published efficient algorithms [5, 6, 7]. With such assumptions, we only evaluate the performance of computing $SWR(i, j)$'s in sequence.

We use artificially generated four data sets, and their properties are given in Table 1. What is important in the data set is the ratio of $TR(i, j)$ to $SR(i, j)$. The smaller the ratio, the more likely the performance of the algorithm can be better. That is, the proposed algorithm is expected to run better for the data set A than for the date sets C or D.

Table 1: Characteristics of the test data sets

Data Sets	Average number of elements in $SR(i, j)$	Average number of elements in $TR(i, j)$	Ratio of$TR(i, j)$ to $SR(i, j)$
A	50	18	36%
B	50	28	56%
C	50	40	80%
D	50	50	100%

We ran the program with the four data sets, as given in Table 1. The test results are shown in Fig. 5. It is assumed that the user moves randomly in any of the 8 directions.The chart in Fig. 5 shows the execution time of the algorithm for the cases that the user made 2500moves, 5000moves, 7500moves, and 10000 moves, respectively.

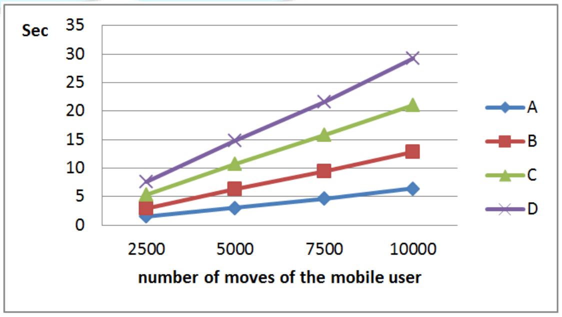


Fig. 5 Skyline for wide region $SWR(i, j)$.

For example, when the number of moves is 5000, the time taken by the algorithm with data sets, *A*, *B*, *C*, *D* are 2.98sec, 6.36 sec, 10.71 sec, and 14.80 sec, respectively. It shows that the smaller the ratio of *TR*(*i*, *j*) to *SR*(*i*, *j*), the better the algorithm works. In case of data set *A*, the size of *TR*(*i*, *j*) is only 36% of *SR*(*i*, *j*). Note that with data set *D*, no update can be done. In other words, when a skyline is computed for the user, no previously computed skylines can be used for time saving. Fig. 5 shows that the proposed algorithm works well.

The efficiency of the proposed algorithm mainly depends on how much is saved by *DeleteBlock*( ) and *AddBlock*( ). When a block B_i is added to *SWR* by *AddBlock*( ), U_{io}, U_{i1}, …. , U_{ik}, … ,U_{i8} are computed. These are later used by *DeleteBlock*( ). Here, we compare the execution time taken by *AddBlock*( ) and that taken by *DeleteBlock*( ). The test results are shown in Fig. 6. In this experiment, we assume that the user made 5000 moves. The chart shows for each data set, the time taken by *AddBlock*( ) and that taken by *DeleteBlock*( ) seperately. For example, for data set *A*, the time taken by *AddBlock*( ) is 2.36 sec, but the time taken by *DeleteBlock*( ) is 0.62 sec. We can see how efficient *DeleteBlock*( ) is. Note that for up, down, left, right moves, we only need to call *AddBlock*( ) 3 times. However, without having *DeleteBlock*( ), we need to call*AddBlock*( ) 6 more times. In other words, currently, there are 21 block-block comparisons, however, without *DeleteBlock*( ), there are 15 more block-block comparions. The reason is when *i*th block is added, *i*-1 block-block comparisons are needed.

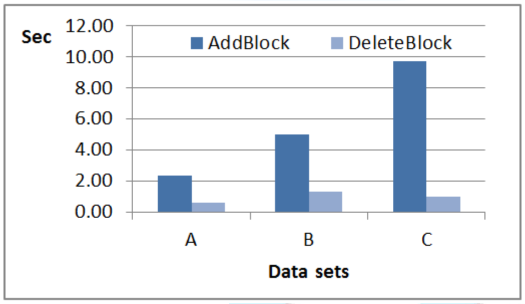


Fig. 6 Comparison of execution time between *AddBlock*( ) and *DeleteBlock*( )

Mobile users move in 8 different directions. Among them four types of moves, such as left, right, up, or down moves, use *DeleteBlock*(*i*, *j*) and *AddBlock*( ) 3 times each. However diagonal moves uses *DeleteBlock*(*i*, *j*) and *AddBlock*( ) 5 times each. We compared the execution time assuming that the user made 5,000 moves. Fig. 7 shows the execution time. Roughly, ratio of diagonal moves to other type of moves is 1.378 : 1.

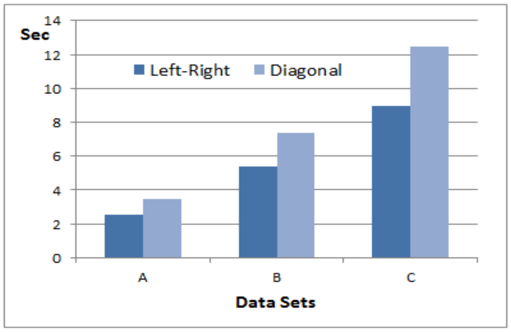


Fig. 7 Comparison of execution time taken by left-right type moves and diagonal moves.

5.Conclusions

We propose an efficient algorithm for computing skylines in sequence for mobile users. Skylines in sequence are computed by updating the skyline that was computed just before. In order to do so, we propose two algorithms, *DeleteBlock*( ) and *AddBlock*( ) and a storage for $\bigcup_{k=0}^{8} U_{ik}$. In case that the size of *SWR*(*k*, *l*) is smaller than that of the union of *SR*(*k*, *l*) , $0 \leq k$, $1 \leq 9$, our algorithm works much better. Such an environment is quite common. Here we assume the case that only one user is served. However, the algorithm can serve many users without interference. Experimental results show that the proposed algorithm works better.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education (2013R1A1A2013124).

References

[1] I. Bartolini, P. Ciaccia, and M. Patella, “SaLSa: Computing the skyline without scanning the whole sky,” In CIKM, pp.405-414, 2006.
[2] S. Borzsonyi, D. Kossmann, and K. Stocker, “The skyline operator,” In ICDE, pp. 421-430, 2001.
[3] J. Chomicki, P. Godfrey, J. Gryz, and D.Liang, “Skyline with presorting,”In ICDE, pp.717-719, 2003.
[4] Z. Hung, H. Lu, B. Ooi, and A.Tung, “Continuous Skyline queries for moving objects,” IEEE TKDE, Vol.18, pp.377-391, 2006.
[5] X. Lin, J. Xu, and H. Hu, “Range-based Skyline Queries in Mobile Environment,” IEEE TKDE, Vol. 25, No. 4, 2013.
[6] J. Kim, and M. Kim, “Skyline Computation Using Adaptive Filters,” International Journal of Computer Science and Information Technology & Security(IJCSITS), Vol. 2, No.2, pp. 431-434, Apr. 2012.

[7] J. Kim, and 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.
[8] J. Kim, and M. Kim, "Skyline Computation permitting Dynamic Determination of Query Regions," International Journal of Computer Science and Information Technology & Security(IJCSITS), Vol. 2, No.5, pp. 939-943, Oct. 2012.
[9] M.Morse, J.M. Patel, and W.I.Grosky, "Efficient continuous skyline computation," Inf. Sci. Vol. 177, No. 17, pp.3411-3437, 2007.
[10] L. Tian, L. Wang, P. Zou, Y. Jia and A. Li, "Continuous Monitoring of Skyline Query over Highly Dynamic Moving Objects." In MobiDE'07, pp.59-66, 2007.
[11]A. Vlachou, C. Doulkeridis, and Y. Kotidis, "Angle-based space partitioning for efficient parallel skyline computation,"In ACM SIGMOD, pp. 227-238, 2008.
[12] Y.Y. Xiao, Y.G. Chen,"Efficient distributed skyline queries for mobile applications," Journal of Computer Science and Technology, Vol. 25, pp.523-536, 2010.
[13] Y. Xiao, K. Lu, and H. Deng,"Location-Dependent SkylineQuery Processing in Mobile Databases," In WISA, pp.3-8, 2010.
[14] B. Zheng, K.C.K. Lee, W.C.Lee,"Location-dependent skyline query," In MDM, pp.148-155, 2008.