Knowl Inf Syst
DOI 10.1007/s10115-012-0486-9

Skyline distance: a measure of multidimensional competence

Jin Huang · Bin Jiang · Jian Pei · Jian Chen · Yong Tang

Received: 30 May 2011 / Accepted: 5 March 2012

Abstract Skyline has been widely recognized as being useful for multi-criteria decision-making applications. While most of the existing work computes skylines in various contexts, in this paper, we consider a novel problem: how far away a point is from the skyline? We propose a novel notion of skyline distance that measures the minimum cost of upgrading a point to the skyline given a cost function. Skyline distance can be regarded as a measure of multidimensional competence and can be used to rank possible choices in recommendation systems. Computing skyline distances efficiently is far from trivial and cannot be handled by any straightforward extension of the existing skyline computation methods. To tackle this problem, we systematically explore several directions. We first present a dynamic programming method. Then, we investigate the boundary of skylines and develop a sort-projection method that utilizes the skyline boundary in calculating skyline distances. Last, we develop a space partitioning method to further improve the performance. We report extensive experiment results which show that our methods are efficient and scalable.

Keywords Skyline · Skyline distance · Skyline boundary · Query optimization

J. Huang · Y. Tang
South China Normal University, Guangdong, China

B. Jiang
Facebook Inc., Palo Alto, CA, USA

J. Pei
Simon Fraser University, Burnaby, BC, Canada

J. Chen (✉)
School of Software Engineering, South China University of Technology,
Guangzhou 510006, China
e-mail: ellachen@scut.edu.cn

Published online: 20 March 2012

1 Introduction

Skyline analysis, which finds superior trade-offs among multiple factors in multi-criteria decision-making applications, has been attracting a great amount of attention in the database community.

Example 1.1 (Skyline) In the expo business, the number of conference halls and the number of guest rooms are two major factors determining the competitiveness of a hotel in the market. Among the 4 synthesized hotels shown in Fig. 1, B has more conference halls and more guest rooms than D, and thus is a better choice than D. Using the terminology of skyline analysis, B dominates D.

Since B is not dominated by any other hotels in the set, B is in the skyline. The skyline hotels, A, B, and C, are the superior trade-offs between the number of conference halls and the number of guest rooms, the two factors in question.

As will be reviewed in Sect. 3, most of the previous studies focus on computing skylines in various contexts, and thus ignore non-skyline points. However, in many applications, questions about non-skyline points are often interesting. Particularly, it is interesting to explore how a non-skyline point can be enhanced to join the skyline.

Example 1.2 (Motivation) Suppose one wants to invest to expand hotel D in Fig. 1, so that it can compete with the skyline hotels A, B, and C. There are two possible choices with various cost. Suppose constructing a new conference hall costs 1 million dollars and constructing a new guest room costs 20 thousand dollars.

There are many different ways to upgrade D, so that it is in the skyline. For example, D can build 200 new guest rooms with 4 million dollars cost, build 4 new conference halls with 4 million dollars cost, or build 100 new guest rooms and 1 new conference hall with 3 million dollars cost. Among the above three ways, the last one costs the least.

Naturally, an optimization problem is to minimize the cost of upgrading D, so that D is not dominated by A, B, and C. The minimum cost of D can be regarded as the measure of the competence of D in the set of hotels. This measure is informative. For example, when multiple hotels are available for investment, an investor may want to consider the ones with small minimum cost. A recommendation system can rank all the candidate hotels in their minimum cost ascending order. This paper will propose the notion of skyline distance to solve such a cost minimization problem.

Computing the minimum cost of upgrading a point to the skyline is far from trivial. In a d-dimensional space, upgrades can be conducted on any of the $(2^d - 1)$ non-empty subsets of the d different dimensions. Moreover, the improvement on each dimension can be an arbitrary positive number. There are a huge number of possible combinations of improvements to boost a non-skyline point to the skyline. Existing skyline algorithms do not compute any information about how far a point is to the skyline.

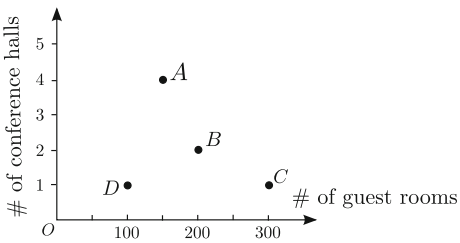


Fig. 1 Four synthesized hotels

Moreover, different cost functions may apply for different points. In the context of Example 1.2, different hotels may have different cost in building new conference halls and guest rooms. The minimum cost changes as the cost function changes. For example, in Example 1.2, if the unit price of a conference hall decreases to 5 hundred thousand dollars, then the cost of the three ways of upgrade is 4 million, 2 million, and 2.5 million dollars, respectively. The second choice of building 4 new conference halls is the best.

In this paper, we tackle the problem of computing the minimum cost of upgrading to the skyline with an arbitrary linear cost function. Some of our methods can be straightforwardly extended to monotonic cost functions. We make the following major contributions in this paper. First, we formulate the skyline distance of a query point. Second, we investigate the search space, which is called the *skyline boundary*, of the possible upgrades that may lead to the minimum cost and prove a complexity bound. Third, we develop several methods for efficient computation of skyline distances. Specifically, we start with a dynamic programming method. Then, we propose a sort-projection method to compute the skyline boundary and use it to compute skyline distances. Furthermore, we devise a space partitioning method based on the divide and conquer and the bounding techniques to boost the performance. As the last contribution, we conduct extensive experiments on both synthetic and real data sets to evaluate the efficiency and scalability of our methods.

The rest of the paper is organized as follows. In Sect. 2, we formulate the notion of skyline distances. We briefly review the related work in Sect. 3. We present a dynamic programming method in Sect. 4. Section 5 studies the boundary of the skyline and shows that the optimal solution always lays on the boundary. Section 6 presents a basic sort-projection method to compute the skyline boundary. In Sect. 7, we develop a space partitioning method to improve the performance. We report extensive experimental results in Sect. 8 and conclude the paper in Sect. 9.

2 Problem definition

We consider a set S of points in a d-dimensional space $\mathbb{D} = D_1 \times \cdots \times D_d$. A point p is written as $p = (p.D_1, \ldots, p.D_d)$, where $p.D_i$ is the value of p on dimension D_i $(1 \leq i \leq d)$. We assume that the domain of each dimension is continuous and numeric. Our techniques can be easily extended to discrete domain. Limited by space, we omit the details.

In this paper, we assume that, on each dimension, larger values are preferred. A point p *dominates* another point q, denoted by $p \prec q$, if p is not smaller than q on every dimension and larger than q on at least one dimension, i.e., $p.D_i \geq q.D_i$ for $i \in [1, d]$ and $p.D_{i_0} > q.D_{i_0}$ for at least one $i_0 \in [1, d]$. Furthermore, we say p *strictly dominates* q, denoted by $p \prec\prec q$, if p is larger than q on every dimension, i.e., $p.D_i > q.D_i$ for every $i \in [1, d]$. To facilitate the presentation, we also denote a position in $\mathbb{D}$ in the same way as a point and treat it as a virtual point, so that we can say a point dominates (or strictly dominates) a position.

The skyline of S consists of all points that are not dominated by any other point, i.e., $Sky(S) = \{p \in S | \nexists q \in S, q \prec p\}$. A point is called a *skyline point* if it is in the skyline, otherwise it is a *non-skyline point*. Example 1.1 demonstrates the concepts of dominance and skyline.

For a non-skyline point $q \in S$, we ask the following question: how can we change the values of q on the dimensions, so that q is not dominated by any other point in S? Moving a point q to a new position q' is associated with some cost $cost(q, q')$. The cost function is also called the *distance function* in our paper.

In this paper, we focus on linear cost functions, which are popularly used in practice. That is, the cost is in the form of $cost(q, q') = \sum_{i=1}^{d} w_i(q'.D_i - q.D_i)$, where $w_i > 0$ for $i \in [1, d]$. Our goal is to find q', which minimizes the cost. As will be discussed in Sect. 9, some of our methods can be straightforwardly extended to monotonic functions in general.

In this paper, we only consider positive upgrades, that is $q'.D_i \geq q.D_i$ for $i \in [1, d]$. A problem of negative upgrades can be converted to a problem only allowing positive upgrades. To be concrete, if $q'.D_i < q.D_i$ on some dimension D_i ($i \in [1, d]$), q' can take a value in the range $[x_i, y_i]$ on dimension $D_i (i \in [1, d])$ and we let $q_0 = (x_1, \ldots, x_d)$, then

$$\begin{aligned} cost(q, q') &= \sum_{i=1}^{d} w_i(q'.D_i - q.D_i) \\ &= \sum_{i=1}^{d} w_i(q'.D_i - q_0.D_i) + \sum_{i=1}^{d} w_i(q_0.D_i - q.D_i) \\ &= cost(q_0, q') + cost(q, q_0) \end{aligned}$$

Since $cost(q, q_0)$ is irrelevant to the position q', the optimization task is to minimize $cost(q_0, q')$. Clearly, $q'.D_i \geq q_0.D_i$ for $i \in [1, d]$. Thus, the problem is equivalent to a problem with only positive upgrades.

If q' can take an arbitrary value smaller than q on a dimension D_{i_0} ($i_0 \in [1, d]$), then a trivial solution is that we set $q'.D_{i_0} = -\infty$, choose an arbitrary dimension $D_{i_1} (i_1 \neq i_0)$, and set $q'.D_{i_1} = \max_{p \in S}\{p.D_{i_1}\} + 1$. Obviously, $cost(q, q') = -\infty$. In practice, for a point q, the possible upgrade point q' can only sit in a finite region. In other words, q' can take a value in a limited range on each dimension.

To avoid the triviality of the solution and also to keep our discussion simple, in the rest of the paper, we assume that $q'.D_i \geq q.D_i$ for $i \in [1, d]$. Under the assumption, the cost/distance is always positive for any pair of q and q'.

Given a point $q \in S$, we are interested in upgrading q to the skyline, i.e., moving q to a new position q' ($q'.D_i \geq q.D_i$ for $i \in [1, d]$), such that q' is not dominated by any other point in S and the cost $cost(q, q')$ is minimized. One critical observation here is that the new position does not need to be in the skyline, due to the following result.

Theorem 2.1 (Non-strict dominance) *Given a set of points S and a linear cost function $cost(\cdot, \cdot)$, for a point $q \in S$, if q is not strictly dominated by any other point in S, then with an arbitrarily small cost $\epsilon > 0$, q can be upgraded, so that it is not dominated by any other point in S.*

Proof With the total cost of ϵ, we can upgrade q to q' such that $q'.D_i = q.D_i + \frac{\epsilon}{d \cdot w_i} > q.D_i$. For any other point $p \in S$, $p \neq q$, if p does not dominate, q, p cannot dominate q'. If p dominates q, the domination is not strict. Thus, there exists at least one-dimension D_{i_0} such that $p.D_{i_0} = q.D_{i_0} < q'.D_{i_0}$. Thus, p cannot dominate q'. □

In Theorem 2.1, the cost ϵ is arbitrarily small and thus is negligible. Based on the above discussion, we are ready to formulate the notion of skyline distance.

Definition 2.1 *(Skyline distance)* Given a set S of points in a d-dimensional space $\mathbb{D}$, a query point q, and a cost function $cost(\cdot, \cdot)$, the *skyline distance* of q is the minimum cost $cost(q, q')$, where q' is a position in $\mathbb{D}$ such that $q'.D_i \geq q.D_i$ for $i \in [1, d]$ and q' is not strictly dominated by any point in S. That is, $SkyDist(q) = \min_{q' \in \mathbb{D}, q'.D_i \geq q.D_i \text{ for } i \in [1,d], \nexists p \in S, p \prec\prec q'}\{cost(q, q')\}$.

As a special case, if q is not strictly dominated by any point in S, there is no need to upgrade it, thus the cost is 0, and $SkyDist(q) = 0$.

One may be lured by linear cost functions to associating our problem with linear programming [7]. Linear programming is a technique for optimizing a linear objective function subject to linear constraints (i.e., linear equalities and linear inequalities). It is proved that the search space defined by linear constraints is always convex. However, as we will show later in the paper, the search space of our problem, called the skyline boundary, is not convex. It is difficult to convert our problem to a linear programming problem and techniques for linear programming cannot be applied to our problem. In this paper, we develop several dedicated methods to solve the problem.

3 Related work

Some representative algorithms for skyline computation include the divide-and-conquer algorithm and the Block Nested Loops algorithm [3], the Sort Filter Skyline algorithm [6], the Linear Elimination Sort for Skyline algorithm [9], the Bitmap algorithm and the Index algorithm [25], the Nearest Neighbor algorithm [13], and the Branch and Bound Skyline algorithm [19,20].

There are also numerous studies on skyline variations for different applications. For example, subspace skylines [21,27,31], k-dominant skylines [5], reverse skyline queries [8], probabilistic skyline computation on uncertain data [14], weighted attributes skylines [17], skyline queries over data streams [15,18,24,26], skyline analysis on time series data [10], spatial skyline queries [23], skyline computation in partially ordered domains [4,22], skyline maintenance for frequent updates [32], skyline cardinality estimation [16,33], and using skylines to mine user preferences, make recommendations [11,29] and microeconomic analysis [34].

In this paper, we study skylines from a different angle by investigating the distance from a query point to the skyline. The existing algorithms for skyline computation can only compute the skyline of a given data set, but cannot give any information about how far a non-skyline point is to the skyline.

Kim et al. [12] studied the same problem. They developed a Grid-Search algorithm. The algorithm partitions the data space into a uniform grid where each cell has the same size. Then, the algorithm traverses the grid in a best first fashion to compute the skyline distance. Clearly, the Grid-Search algorithm cannot scale to high-dimensional data as the number of cells to search is exponential to the dimensionality. Kim et al. [12] did not provide an empirical study on the performance of the Grid-Search algorithm. In Sect. 8, we show that our algorithms outperform the Grid-Search algorithm on almost all data sets used in our experiments, especially on high cardinality and/or high dimensionality data sets.

Lately, Wan et al. [28] also studied a problem of creating competitive products from multiple sources tables, for example, creating vacation packages from tables of hotels and flights. A competitive product is one that is not dominated by any existing product nor any possible product, which can be generated. The problem is different to our skyline distance problem. For the problem of skyline distance, we focus on improving existing products to join the skyline rather than creating new product. More importantly, we look for the best way for such improvement with respect to certain cost functions. Furthermore, [28] is designed for discrete domain obtained from source tables, e.g., table of hotels and flights, while we focus on continuous domains and our techniques can be extended to discrete domains.

Wu et al. [30] considered the exclusive dominance region of a skyline point, which is the region solely dominated by this skyline point. However, finding the exclusive dominance

region of a skyline point cannot give us the answer of the skyline distance. In fact, in our problem, a query point can be dominated by several skyline points at the same time. Hence, a query point may not be in the exclusive dominance region of any skyline point. Our problem of skyline distance and the problem of exclusive dominance region are two different problems. A solution to one cannot be used to solve the other easily.

4 Dynamic programming

Consider a set S of points in space $\mathbb{D}$. Let $Sky(S)$ be the set of skyline points in S. We assume that $Sky(S)$ is computed using an existing skyline computation algorithm.

Given a query point q, suppose q is strictly dominated by m skyline points in $Sky(S)$. For any position q' not strictly dominated by any point in S and $q'.D_i \geq q.D_i$ for all $i \in [1, d]$, the upgrade from q to q' can be viewed as a path from q to q', which always goes up along axes. Since we use linear cost functions, $cost(q, q')$ is the sum of the weighted length of the segments on the path.

We denote a path from q to q' by $P(q, q')$. Due to Definition 2.1, our objective is to find a path with the minimum cost, so that the end point q' of the path is not strictly dominated by any skyline point and $q'.D_i \geq q.D_i$ for $i \in [1, d]$.

Given a path described above, we define m turning positions. The kth ($1 \leq k \leq m$) turning position of a path, denoted by q_k, is the first position in the path such that q_k is strictly dominated by at most $m - k$ skyline points. Apparently, q_m is not strictly dominated by any skyline point. We always set q' to be q_m to minimize the length of the path. Specifically, we also let $q_0 = q$. Therefore, we can realize a path in m steps. We claim the following.

Lemma 4.1 (Cost of a path) *The cost of the path from q to q', i.e., $cost(q, q')$, is simply the sum of the cost in all steps.*

Proof We prove by mathematical induction. For $q' = q_0$, it is obviously $cost(q, q') = 0$.

For $q' = q_k$, we assume $cost(q, q') = cost(q_0, q_k) = \sum_{i=1}^{k} cost(q_{i-1}.q_i)$ holds. Then, for $q' = q_{k+1}$,

$$cost(q, q') = cost(q_0, q_{k+1}) = cost(q_0, q_k) + cost(q_k, q_{k+1}) = \sum_{i=1}^{k+1} cost(q_{i-1}.q_i).$$

In each step of the dynamic programming, due to the d dimensions in space $\mathbb{D}$, we specifically select d options of q_k, denoted by $q_{k,i}$ ($1 \leq i \leq d$). The ith choice $q_{k,i}$ is

$$q_{k,i}.D_j = \begin{cases} \min_{s \in Sky(S), s \prec q_{k-1}} \{s.D_i\} & i = j; \\ q_{k-1}.D_j & 1 \leq j \leq d,\ j \neq i. \end{cases} \tag{1}$$

We compute $SkyDist(.)$ in a particular order. Firstly, we compute $SkyDist(q_m)$ where q_m is the destination of the path (which corresponds to the base case). Secondly, we compute $SkyDist(q_{m-1})$ based on $SkyDist(q_m)$. In general, we compute $SkyDist(q_{k-1})$ based on $SkyDist(q_k)$. Finally, we compute $SkyDist(q_1)$ based on $SkyDist(q_2)$.The dynamic programming method recurs as follows,

$$SkyDist(q_{k-1}) = \min_{i=1}^{d} \{SkyDist(q_{k,i}) + cost(q_{k-1}, q_{k,i})\} \tag{2}$$

where $q_{k,i}$ is given by Eq. (1). The pseudocode is given in Algorithm 1.

Algorithm 1 The dynamic programming method $DP(q, SKY, d)$.

Input: the set $SKY = \{s_1, \ldots, s_m\}$ of skyline points which dominate the query point q; the dimensionality d;
Output: the skyline distance of q;
Description:
1: **if** there is no point in SKY that strictly dominates q **then**
2: **return** 0;
3: **else if** $SkyDist(q)$ is already computed **then**
4: **return** $SkyDist(q)$;
5: **else**
6: **for all** q_i given by Eq. (1) $(1 \leq i \leq d)$ **do**
7: $SkyDist(q_i) = DP(q_i, \{s \in SKY | s \prec q_i\}, d)$;
8: **end for**
9: **return** $\min_{i=1}^{d}\{SkyDist(q_i) + cost(q, q_i)\}$;
10: **end if**

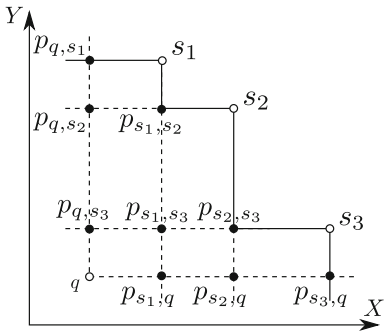


Fig. 2 A 2D example of the dynamic programming method

Algorithm 1 shows the pseudocode of the dynamic programming method developed in Sect. 4.

4.1 An example and visualization of the dynamic programming method

To visualize the dynamic programming method, we can imagine a d-dimensional grid in space $\mathbb{D}$ formed by lines parallel to the d axes crossing the query point q and the skyline points $s_1, \ldots, s_m$, which strictly dominate q. We call a point *an intersection point* if its value on every dimension is the same as some skyline points or the query point. We denote an intersection point by $p_{t_1,\ldots,t_d}$ if it has the same value as point t_i $(1 \leq i \leq d)$ on dimension i, where $t_1, \ldots, t_d$ belong to $\{q, s_1, \ldots, s_m\}$.

Example 4.1 (Intersection points) Figure 2 illustrates a 2-dimensional example, the query point q is strictly dominated by 3 skyline points s_1, s_2, and s_3. The grid is depicted by the dashed lines. There are 9 intersection points plotted as solid dots.

Basically, the dynamic programming method recursively computes the skyline distances of the intersection points and derives the skyline distance of the query point q.

Example 4.2 (The dynamic programming method) In Fig. 2, initially, we set $q_0 = q$. At the first step, q_1 has two choices, p_{q,s_3} and $p_{s_1,q}$, for upgrading. So $SkyDist(q)$ is the smaller one between $SkyDist(p_{q,s_3}) + cost(q, p_{q,s_3})$, and $SkyDist(p_{s_1,q}) + cost(q, p_{s_1,q})$. We recursively compute $SkyDist(p_{q,s_3})$ and $SkyDist(p_{s_1,q})$, respectively. Eventually, we compute the skyline distances of the intersection points listed in Table 1 from bottom up. By definition, p_{q,s_1}, p_{s_1,s_2}, p_{s_2,s_3}, and $p_{s_3,q}$ have skyline distances 0.

Table 1 A dynamic programming method example

q			
p_{q,s_3}	$p_{s_1,q}$		
p_{q,s_2}	p_{s_1,s_3}	$p_{s_2,q}$	
p_{q,s_1}	p_{s_1,s_2}	p_{s_2,s_3}	$p_{s_3,q}$

4.2 The correctness of the dynamic programming method

Given a query point q strictly dominated by m skyline points in $Sky(S)$, we denote the updating path from q to q' by $P(q, q')$, with m turning positions. q_0 and q_m are set to be q and q' as discussed before.

To prove the correctness of the dynamic programming method in Sect. 4, we first show a path defines an optimal substructure.

Clearly, if the cost of a path $P(q_0, q_m)$ from q_0 to q_m is the minimum cost of upgrading q_0 to the skyline, then, for any q_k $(0 \leq k \leq m)$ of $P(q_0, q_m)$, the cost of the path $P(q_k, q_m)$ from q_k to q_m is the minimum cost of upgrading q_k to the skyline. That is, the cost of $P(q_k, q_m)$ is the skyline distance of q_k. This provides an optimal substructure for the skyline distance problem. Consequently, we realize the dynamic programming method in m steps in a recursive way. We have

$$SkyDist(q_{k-1}) = \min_{q_k}\{SkyDist(q_k) + cost(q_{k-1}, q_k)\}. \tag{3}$$

Then, kth step, we only need to enumerate k choices (shown in Eq. 1) of the turning position q_k $(1 \leq k \leq)$. The lemma below helps us to limit the possible choices of q_k.

Lemma 4.2 *Given two points v_1 and v_2, let SKY_1 and SKY_2 be the sets of skyline points strictly dominating v_1 and v_2, respectively. If $SKY_1 = SKY_2$, then*

$$SkyDist(v_1) - SkyDist(v_2) = \sum_{i=1}^{d} w_i(v_2.D_i - v_1.D_i). \tag{4}$$

Proof Consider position v_0 such that $v_0.D_i = \max\{v_1.D_i, v_2.D_i\}$. We first show that SKY_1 is exactly the set of skyline points strictly dominating v_0 if $SKY_1 = SKY_2$. Since $SKY_1 = SKY_2$, for any skyline point $s \in SKY_1$ and any dimension $D_i (1 \leq i \leq d)$, $s.D_i > v_1.D_i$ and $s.D_i > v_2.D_i$. Therefore, $s.D_i > \max\{v_1.D_i, v_2.D_i\} = v_0.D_i$. That is, s strictly dominates v_0. Moreover, for any skyline point s strictly dominating v_0, s also strictly dominates both v_1 and v_2.

Consider a path starting from v_1 that upgrades v_1 to the skyline such that its cost is minimized (i.e., its cost is $SkyDist(v_1)$). Because a path goes along axes, we can always adjust it to pass through v_0 and maintain its cost unchanged, since v_0 and v_1 are strictly dominated by the same set of skyline points. Then,

$$SkyDist(v_1) - SkyDist(v_0) = \sum_{i=1}^{d} w_i(v_0.D_i - v_1.D_i). \tag{5}$$

Similarly, we have

$$SkyDist(v_2) - SkyDist(v_0) = \sum_{i=1}^{d} w_i(v_0.D_i - v_2.D_i). \quad (6)$$

Subtracting Eq. (6) from (5), we have Eq. (4). □

At the kth step, we select d candidates $q_{k,i}$ $(1 \leq i \leq d)$ for q_k given by Eq. (1). Clearly, there exists a $q_{k,i}$ $(1 \leq i \leq d)$ such that the set of skyline points strictly dominating q_k is the same as the set of skyline points strictly dominating $q_{k,i}$. By Lemma 4.2,

$$\begin{aligned} SkyDist(q_k) + cost(q_{k-1}, q_k) &= SkyDist(q_{k,i}) \\ &\quad + \sum_{j=1}^{d} w_i(q_{k,i}.D_j - q_k.D_j) + cost(q_{k-1}, q_k) \\ &= SkyDist(q_{k,i}) + cost(q_{k-1}, q_{k,i}). \end{aligned} \quad (7)$$

Combining Eqs. (3) and (7), we have the final recursive formula for the dynamic programming method as shown in Eq. (2).

In a d-dimensional space, a query point q and the m skyline points that strictly dominate q form a grid of $O(m^d)$ intersection points. There are d choices at every step. So, the complexity of the dynamic programming method is $O(dm^d)$ plus the cost of retrieving the m skyline points.

Taking a closer look at the dynamic programming method and Example 4.2, we notice that the optimal upgrading position of q has to be one of the intersection points p_{q,s_1}, p_{s_1,s_2}, p_{s_2,s_3}, and $p_{s_3,q}$. These intersection points have skyline distances 0. They lay on the "boundary" of the skyline. We call them boundary intersection points. Section 5 formally defines boundary intersection points and show how to use them to compute skyline distances efficiently.

5 Skyline boundary

For a set of points S in a d-dimensional space $\mathbb{D}$, the *skyline boundary* is a $(d-1)$-dimensional surface that consists of all positions that are dominated by some skyline points but not strictly dominated by any skyline point.

Definition 5.1 *(Skyline boundary)* Given a set SKY of skyline points in S, we say a point $p \in \mathbb{D}$ (p does not necessarily belong to S) is on the *skyline boundary* if there exists a point $s \in SKY$ such that $s \prec p$ and there does not exists a point $s' \in SKY$ such that $s' \prec\prec p$.

According to Theorem 2.1 and Definition 2.1, we have the following property immediately.

Property 5.1 *(Skyline boundary)* Every point on the skyline boundary has a skyline distance 0.

By Property 5.1, for a query point q, there must be a point p on the skyline boundary such that $SkyDist(q) = cost(q, p)$.

Section 4 discusses intersection points. We call an intersection point a *boundary intersection point* if it is on the skyline boundary.

Property 5.2 *(Boundary intersection points)* Let p be a point on the skyline boundary w.r.t. a set S of points and p is not a boundary intersection point. Then, there exists a boundary intersection point p' such that $p \prec p'$.

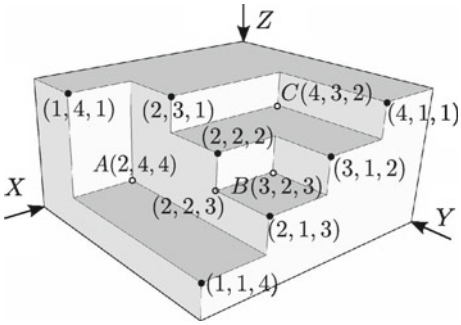


Fig. 3 A 3D example (*axes* drawn in reverse direction)

Proof The $(d-1)$-dimensional surface of the skyline boundary can be decomposed into a set of $(d-1)$-dimensional axis-aligned polyhedrons. The set of vertices of each polyhedron consists of one skyline point and a number r of boundary intersection points dominated by the skyline point. It is easy to see that such a polyhedron can be further decomposed into $r(d-1)$-dimensional axis-aligned hyper-rectangles, each of which has a boundary intersection point as the minimum corner. Because p is on the skyline boundary, it must fall into one polyhedron, then in a hyper-rectangle. Thus, it dominates the minimum corner of that hyper-rectangle, which is the boundary intersection point p' we look for. □

Example 5.1 (Skyline Boundary) Figure 2 shows a set of points and the skyline in a 2-dimensional space. The solid line in the figure illustrates the skyline boundary. Among the 9 intersection points, p_{q,s_1}, p_{s_1,s_2}, p_{s_2,s_3}, and $p_{s_3,q}$ are on the skyline boundary.

Figure 3 shows a 3-dimensional case, where A, B, and C are the skyline points. To make the figure easy to see, axes are drawn in reverse direction. The skyline boundary, which is a 2-dimensional surface, is shown in the figure.

Clearly, the skyline boundaries shown in the above example are not convex. Hence, linear programming cannot be applied to our problem.

How is the skyline distance related to the dominance relationship between points?

Lemma 5.1 (Dominance and cost) *Consider a query point q, two points p_1 and p_2 such that $p_1 \prec p_2$, $p_1.D_i \geq q.D_i$ and $p_2.D_i \geq q.D_i$ for $i \in [1, d]$. Then, $cost(q, p_1) > cost(q, p_2)$.*

Proof

$$\begin{aligned} cost(q, p_1) - cost(q, p_2) &= \sum_{i=1}^{d} w_i(p_1.D_i - q.D_i) \\ &\quad - \sum_{i=1}^{d} w_i(p_2.D_i - q.D_i) \\ &= \sum_{i=1}^{d} w_i(p_1.D_i - p_2.D_i) \end{aligned}$$

Since $p_1 \prec p_2$, $p_1.D_i \geq p_2.D_i$ and there exists at least one-dimension D_{i_0} such that $p_1.D_{i_0} > p_2.D_{i_0}$. Thus, $cost(q, p_1) - cost(q, p_2) > 0$. □

Boundary intersection points play an important role in skyline distance computation, since the skyline distance of a query point is determined by the minimum cost of upgrading the query point to a boundary intersection point, as formally shown in the following theorem.

Theorem 5.1 (Boundary intersection points) *Consider a query point q which is dominated by $m > 0$ skyline points $s_1, \ldots, s_m$. Let $p_1, \ldots, p_r$ be the r boundary intersection points determined by q and $s_1, \ldots, s_m$. Then, $SkyDist(q) = \min_{i=1}^{r}\{cost(q, p_i)\}$.*

Proof We prove the theorem by contradiction. Suppose q' is the optimal position on the skyline boundary such that $SkyDist(q) = cost(q, q')$ but q' is not a boundary intersection point. By Definition 2.1, $q'.D_i \geq q.D_i$ for $i \in [1, d]$. By Property 5.2, q' must dominate a boundary intersection point p_j ($j \in [1, r]$). By Lemma 5.1, $cost(q, p_j) < cost(q, q') = SkyDist(q)$, which leads to a contradiction. □

It is possible that one boundary intersection point dominates another one. For example, in Fig. 3, point (2, 2, 3) dominates point (2, 2, 2). By Lemma 5.1, those boundary intersection points which dominate other boundary intersection points cannot be the solution for any query point. So, we call a boundary intersection point a *local optimal point* if it does not dominate any other boundary intersection point. Combining Theorem 5.1 and Lemma 5.1, we have the following.

Corollary 5.1 (Local optimal points) *Consider a query point q dominated by m skyline points $s_1, \ldots, s_m$. Let $p_1, \ldots, p_r$ be the r local optimal points determined by q and $s_1, \ldots, s_m$. Then, $SkyDist(q) = \min_{i=1}^{r}\{cost(q, p_i)\}$.*

In summary, local optimal points have the following properties which are critical to skyline distance computation.

1. A local optimal point is dominated by some skyline points but not strictly dominated by any skyline point;
2. A local optimal point has a skyline distance 0; and
3. Local optimal points do no dominate each other.

Example 5.2 (Local Optimal points) In Fig. 2, the four boundary intersection points p_{q,s_1}, p_{s_1,s_2}, p_{s_2,s_3}, and $p_{s_3,q}$ are also local optimal points.

Figure 3 shows a 3-dimensional example, where the 7 local optimal points are plotted in solid dots. Point (2, 2, 3) is a boundary intersection point but not a local optimal point, since it dominates local optimal point (2, 2, 2).

6 The sort-projection method

In this section, we develop a sort-projection algorithm to compute local optimal points. Then, the skyline distance can be calculated straightforwardly by finding the minimum cost of the query point to the set of local optimal points.

To find local optimal points in a d-dimensional space, a d-dimensional problem is decomposed into multiple $(d-1)$-dimensional problems, which are solved recursively. We first illustrate the algorithm in the 2-dimensional case then show the recursion in high-dimensional cases.

6.1 The 2-dimensional case

Given a query point q which is dominated by m skyline points $s_1, \ldots, s_m$ (assuming no two points are the same) in a 2-dimensional space $D_1 \times D_2$, we simply sort the skyline points in the ascending order on dimension D_1. Without loss of generality, we assume

$s_1.D_1 < s_2.D_1 < \cdots < s_m.D_1$ (note that, no two points have the same value). Because skyline points do not dominate each other, $s_1, \ldots, s_m$ are also in the descending order on dimension D_2. That is, $s_1.D_2 > s_2.D_2 > \cdots > s_m.D_2$.

Clearly, there are $m + 1$ local optimal points and the ith one p_i is given by the following formula,

$$p_i = \begin{cases} (q.D_1, s_1.D_2) & i = 1; \\ (s_{i-1}.D_1, s_i.D_2) & 2 \leq i \leq m; \\ (s_m.D_1, q.D_2) & i = m + 1. \end{cases} \tag{8}$$

Figure 2 gives an illustration. Apparently, sorting is the major cost for the sort-projection algorithm in the 2-dimensional space. Thus, the time complexity is $O(m \log m)$.

Algorithm 2 gives the pseudocode of the sort-projection algorithm. Lines 1–3 summarize the processing for the 2-dimensional case. Lines 4–13 apply to the high-dimensional cases.

Algorithm 2 The sort-projection algorithm $SP(q, SKY, d)$.

Input: the set $SKY = \{s_1, \ldots, s_m\}$ of skyline points which dominate the query point q; the dimensionality d;
Output: the skyline distance of q;
Description:
1: **if** $d = 2$ **then**
2: sort points in SKY in the ascending order on dimension D_1;
3: $P = \{p_i | 1 \leq i \leq m + 1\}$ where p_i is given by Eq. (8);
4: **else**
5: select the dimension D_k of the least distinct values for sorting;
6: divide points in SKY into l partitions such that partition S_i $(1 \leq i \leq l)$ consists of the projections of points with the ith largest value on dimension D_k;
7: Let $P_1 = \{p | p.D_j = q.D_j \text{ for } j \neq k\}$; $P = \{p | p.D_k = S_1.D_k, p.D_j = p'.D_j \text{ for } j \neq k, p' \in P_1\}$; Let $SS = \emptyset$;
8: **for** i from 2 to $l + 1$ **do**
9: $SS = Sky(SS \cup S_{i-1})$;
10: Let $P_i = SP(q, SS, d - 1)$;
11: $P = P \cup \{p | p.D_k = S_i.D_k, p.D_j = p'.D_j \text{ for } j \neq k, p' \in (P_{i-1} \setminus P_i)\}$;
12: **end for**
13: **end if**
14: **return** $\min\{cost(q, p) | p \in P\}$;

6.2 The higher dimensional cases

Generally, in a d-dimensional space, again, we denote by q the query point, and by $s_1, \ldots, s_m$ the m skyline points which dominate q. The sort-projection algorithm follows a divide-and-conquer framework in three steps: dividing, conquering, and merging.

Dividing: The algorithm first sorts the skyline points in the descending order on the dimension with the least number of distinct values. The choice of the dimension will become clear in Sect. 6.3. Let D_k be the dimension used to sort skyline points. Suppose, there are l distinct values on D_k.

We project the skyline points to a $(d - 1)$-dimensional space $D_1 \times \cdots \times D_{k-1} \times D_{k+1} \times \cdots \times D_d$. The skyline points are divided into l partitions such that the points with the same value on dimension D_k are put in the same partition. Let S_i be the ith partition $(1 \leq i \leq l)$

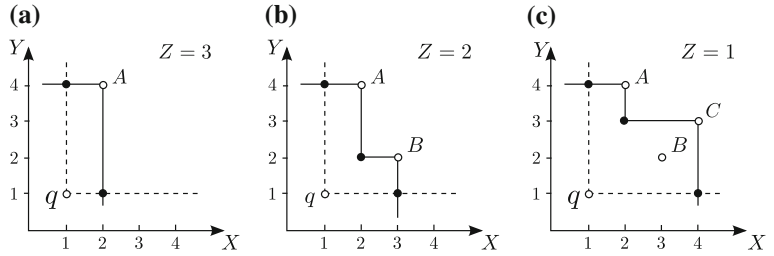


Fig. 4 The 2D projections of Fig. 3. **a** $Z = 3$. **b** $Z = 2$. **c** $Z = 1$

consisting of the projections of points with the ith largest value on dimension D_k. We also denote the ith largest value on D_k by $S_i.D_k$. As a special case, we denote $S_{l+1}.D_k = q.D_k$. Consequently, $S_i.D_k > S_j.D_k$ for any $1 \leq i < j \leq l, i \neq k, j \neq k$.

Conquering: We process the $(l+1)$ partitions one by one, starting from S_1. In the d-dimensional space $D_1 \times \cdots \times D_d$, if we fix the value on dimension D_k, we get a $(d-1)$-dimensional hyperplane $D_1 \times \cdots \times D_{k-1} \times D_{k+1} \cdots \times D_d$. The ith hyperplane is the one when we fix the value on dimension D_k to $S_i.D_k$. Let P_i $(1 \leq i \leq l+1)$ denotes the set of local optimal points on the ith hyperplane.

The only local optimal point on the first hyperplane is simply the projection of q, i.e., $P_1 = \{p | p.D_j = q.D_j \text{ for } j \neq k\}$ (line 7 in Algorithm 2). The local optimal points on the ith hyperplane $(2 \leq i \leq l+1)$ are determined by the points in the set of skyline $\bigcup_{j=1}^{i-1} S_j$ (line 9). P_i is computed by recursively applying the sort-projection algorithm.

Merging: To obtain the local optimal points in the original d-dimensional space, we first recover points in P_i $(1 \leq i \leq l+1)$ from the $(d-1)$-dimensional space to the d-dimensional space. Each point p in P_i corresponds to a point that has value $S_i.D_k$ on dimension D_k and value $p.D_j$ on dimensions D_j other than D_k. We denote the d-dimensional correspondence of p by p^d. Because $S_i.D_k > S_j.D_k$ for any $i < j$, for any point $p \in P_i$, and any point $p' \in P_j$, p'^d cannot dominate p^d. Furthermore, p^d dominates p'^d only if $p = p'$ with respect to their values on the projected $d-1$ dimensions. Therefore, in the merging step, we simply remove a point p' in P_{i-1} if there is a point p in P_i such that $p.D_i = p'.D_i$ for $1 \leq i \leq d$, $i \neq k$ (line 11).

Example 6.1 (The sort-projection algorithm) Figure 3 illustrates a 3-dimensional example in space $X \times Y \times Z$. The query point $q(1, 1, 1)$ is dominated by 3 skyline points $A(2, 4, 4)$, $B(3, 2, 3)$, and $C(4, 3, 2)$. In the dividing step, the algorithm sorts A, B, and C on dimension Z and divides them into 3 partitions $S_1 = \{A'(2, 4)\}$, $S_2 = \{B'(3, 2)\}$, and $S_3 = \{C'(4, 3)\}$.

In the conquering step, we consider the $X \times Y$ planes one by one. In the $X \times Y$ plane with $Z = 4$, the set of local optimal points $P_1 = \{(1, 1)\}$. The skylines of the other $3X \times Y$ planes with Z values 3, 2, and 1 are shown in Fig. 4. The corresponding sets of local optimal points (plotted in solid dots) are $P_2 = \{(1, 4), (2, 1)\}$, $P_3 = \{(1, 4), (2, 2), (3, 1)\}$, and $P_4 = \{(1, 4), (2, 3), (4, 1)\}$, respectively.

In the merging step, $(1, 4)$ appears in P_2, P_3, and P_4. So, we only retain the one in P_4 and remove those in P_2 and P_3. Finally, we obtain the set of local optimal points $\{(1, 1, 4), (2, 1, 3), (2, 2, 2), (3, 1, 2), (1, 4, 1), (2, 3, 1), (4, 1, 1)\}$.

6.3 Complexity analysis

Because the time complexity of the sort-project method depends on the number of local optimal points, we firstly prove the number of local optimal points in lemma 6.1 then we give Theorem 6.1 and its proof.

Lemma 6.1 (Number of local optimal points) *Let $N(m, d)$ be the number of local optimal points with m input skyline points in a d-dimensional space. Then, $N(m, 2) = O(m)$ and $N(m, d) \leq N(m, d+1) \leq mN(m, d)$ for $d \geq 2$.*

Proof The 2-dimensional case is obvious. For a $(d+1)$-dimensional space ($d \geq 2$), suppose the sort-projection algorithm divides the m skyline points into l partitions, where the ith ($1 \leq i \leq l$ partition has m_i points. We have $\sum_{i=1}^{l} m_i = m$. The ith ($2 \leq i \leq l+1$) d-dimensional hyperplane has at most $\sum_{j=1}^{i-1} m_j$ skyline points, thus at most $N(\sum_{j=1}^{i-1} m_j, d)$ local optimal points. Then, the total number of local optimal points in the $(d+1)$-dimensional space is $N(m, d+1) = 1 + \sum_{i=2}^{l+1} N(\sum_{j=1}^{i-1} m_j, d)$.

When $d = 2$, we obtain $N(m, 3) = 1 + \sum_{i=1}^{l}(l - i + 1)O(m_i)$. We establish $O(m) \leq N(m, 3) \leq (m^2)$. It is equivalent to $N(m, 2) \leq N(m, 3) \leq mN(m, 2)$. By mathematical induction, it is easy to derive $N(m, d) \leq N(m, d+1) \leq mN(m, d)$ for $d \geq 2$. □

Following with Lemma 6.1, we have the following.

Corollary 6.1 *Denote by $N(m, d)$ the number of local optimal points in a d-dimensional space. $N(m, d) = \Omega(m)$ and $N(m, d) = O(m^{d-1})$.*

Now, we can prove our complexity result stated in Theorem 6.1.

Theorem 6.1 (Complexity—sort-projection) *Let $f(m, d)$ denotes the complexity of the sort-projection algorithm with m input skyline points in a d-dimensional space. $f(m, 2) = O(m \log m)$. Moreover, $f(m, d) = \Omega(m \log m)$ and $f(m, d) = o(m^{d-1})$ for $d \geq 3$.*

Proof We prove by mathematical induction. As stated in Sect. 6.1, $f(m, 2) = O(m \log m)$.

For the $(d+1)$-dimensional ($d \geq 2$) space, the cost of the sorting step is $O(m \log m)$. The algorithm divides the m skyline points into l partitions with cardinalities $m_1, \ldots, m_l$. We have $\sum_{i=1}^{l} m_i = m$. Then, in the conquering step, the number of skyline points considered in the ith ($2 \leq l+1$) d-dimensional hyperplane is at most $\sum_{j=1}^{i-1} m_j$. Thus, the cost of computing the local optimal points in the ith hyperplane is $f(\sum_{j=1}^{i-1} m_j, d)$. In the merging step, the algorithm compares P_{i-1} and P_i. The comparison has complexity $N(m_{i-1}, d) + N(m_i, d)$ since points in P_{i-1} and P_i are sorted. Therefore, the total complexity of the sort-projection algorithm is

$$f(m, d+1) = O(m \log m) + \sum_{i=2}^{l+1} \Big(f(\sum_{j=1}^{i-1} m_j, d) + N(m_{i-1}, d) + N(m_i, d) \Big).$$

By Corollary 6.1, we have $f(m, d+1) = \Omega(m \log m)$ and $f(m, d+1) = o(m^d)$. □

According to the complexity analysis, to minimize the cost, in the dividing step, we should choose the dimension with the least distinct values such that the number of partitions is minimized.

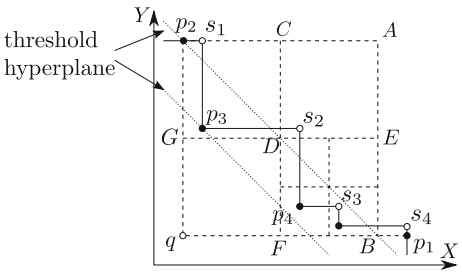


Fig. 5 An example of space partitioning

7 A space partitioning method

In this section, we develop a space partitioning method. This method partitions the space into several d-dimensional hyper-rectangles and searches for local optimal points in each hyper-rectangle individually. To reduce the search space, we propose two pruning rules to eliminate hyper-rectangles, which do not contain any local optimal point that contributes to the skyline distance with respect to a query point. For a hyper-rectangle that cannot be pruned, either it is further partitioned into sub-hyper-rectangles or the sort-projection method is applied to compute the local optimal points within it.

Given a query point q dominated by a set SKY of skyline points, initially, the space partitioning method finds d local optimal points, $p_1, \ldots, p_d$, where p_i $(1 \leq i \leq d)$ is given by the following formula,

$$p_i.D_j = \begin{cases} \max\{s.D_i | s \in SKY\} & \text{for } j = i \\ q.D_j & \text{for } j \neq i \end{cases} \tag{9}$$

Then, the space partitioning method finds the minimum cost of q to $p_1, \ldots, p_d$. Let p_{min} denotes the local optimal point such that $cost(q, p_{min}) = \min_{i=1}^{d}\{cost(q, p_i)\}$. We set $threshold = cost(q, p_{min})$.

The cost function and the threshold define a *threshold hyperplane* $\sum_{i=1}^{d} w_i \times x.D_i = threshold$. Any local optimal point p above the threshold hyperplane has cost $cost(q, p) > threshold$ and thus cannot be the optimal solution.

Example 7.1 (The space partitioning method) Let us explain the details of the space partitioning method using Fig. 5 as a 2-dimensional example, where a query point q is dominated by four skyline points s_1, s_2, s_3, and s_4. Two local optimal points p_1 and p_2 are found, and p_2 is the one with the smaller cost. Thus, we set $threshold = cost(q, p_2)$.

In Fig. 5, the dotted line through p_2 is the initial threshold hyperplane, and the local optimal point p_1 which is above the threshold hyperplane has larger cost from q and thus can be pruned. If we find more local optimal points, we may reduce the value of $threshold$ progressively and prune more local optimal points.

It is easy to see that q, p_{min}, and the cost function jointly define a d-dimensional hyper-rectangle. We partition a hyper-rectangle into 2^d equal size sub-hyper-rectangles surrounding its center.

Let R denotes a d-dimensional hyper-rectangle. Let $\max(R)$ and $\min(R)$ be the maximum and minimum corners of R, respectively. R can be pruned if any of the following two situations happen.

Algorithm 3 The space partitioning method.

Input: a query point q and the set SKY of skyline points dominating q; dimensionality d;
Output: the skyline distance of q;
Description:
find the d initial-local optimal points $p_1, \ldots, p_d$ as shown in Equation (9);
compute p_{min} among $p_1, \ldots, p_d$; $threshold = cost(q, p_{min})$;
initial a heap $\mathbb{H}$ and insert the hyper-rectangle defined by q, p_{min}, and the cost function;
while $\mathbb{H}$ is not empty **do**
  $R = \mathbb{H}.pop()$;
  if $\max(R)$ is dominated by a skyline point **then**
    continue; /* PR1 */
  end if
  if $cost(q, \min(R)) > threshold$ **then**
    break; /* PR2 */
  end if
  let S be the set of skyline points that dominate $\min(R)$;
  if $|S| \leq limit$ **then**
    $P = SP(q, S, d)$; /* invoke Algorithm 2 */
    update $threshold$ according to local optimal points in P;
  else
    partition R into 2^d equal-sized sub-hyper-rectangles and insert them into $\mathbb{H}$;
  end if
end while
return $threshold$;

PR1 if $\max(R)$ is strictly dominated by a skyline point, then R can be pruned, since any point inside R is strictly dominated by a skyline point so it is not a local optimal point. To efficiently check whether $\max(R)$ is strictly dominated by a skyline point, we can pre-build an R*-tree [1] on all skyline points.

PR2 if $cost(q, \min(R)) > threshold$, then R can be pruned, since for any local optimal point p inside R, $cost(q, p) > threshold$ (Lemma 5.1).

If a hyper-rectangle R contains a local optimal point p, then p dominates $\min(R)$. Moreover, any skyline points dominating p also dominate $\min(R)$. Therefore, if a skyline point s does not dominate $\min(R)$, s cannot lead to any local optimal points in R. In other words, to compute the local optimal points inside a hyper-rectangle R, we only need to consider the skyline points which dominate $\min(R)$.

For a hyper-rectangle R which is not pruned by rules PR1 and PR2, we check the number of skyline points dominating $\min(R)$. If the number is no more than a pre-defined threshold $limit$, we apply the sort-projection method (Algorithm 2) to compute the local optimal points. Otherwise, R is further partitioned into 2^d sub-hyper-rectangles.

Example 7.2 (Space partitioning method (continued)) In our running example (Fig. 5), the big rectangle p_2qBA plotted by the dashed lines is partitioned into 4 sub-rectangles. Hyper-rectangle $GqFD$ is pruned by PR1, and hyper-rectangle $CDEA$ is pruned by PR2.

Suppose $limit = 2$, then hyper-rectangle $DFBE$ is further partitioned, and we compute the local optimal points in hyper-rectangle p_2GDC and get local optimal point p_3, resulting in the update of the value of $threshold$.

To schedule the processing of hyper-rectangles, we build a min-heap $\mathbb{H}$ with $cost(q, \min(R))$ as the key for each hyper-rectangle R. We always process the top hyper-rectangle in $\mathbb{H}$. New rectangles are inserted into $\mathbb{H}$. Algorithm 3 gives the pseudocode of the space partitioning method.

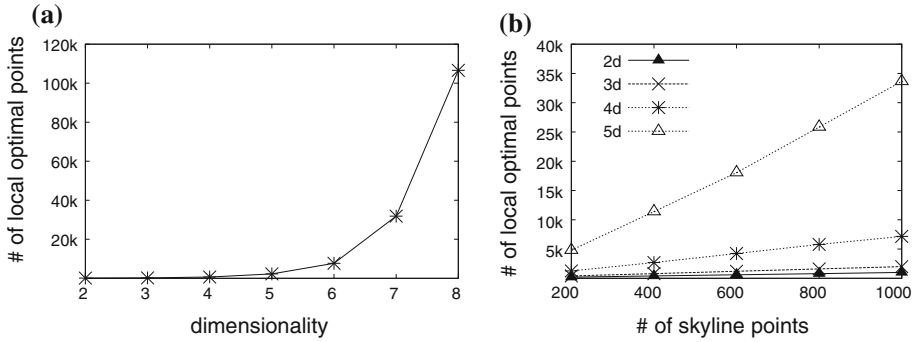


Fig. 6 Number of local optimal points. **a** Effect of d. **b** Effect of n

The space partitioning method has the same worst case complexity as the sort-projection method when the two heuristic pruning rules fail. However, as shown in our experiments, the space partitioning method works much better than the sort-projection method on various data sets.

8 Empirical studies

We conducted extensive experiments to evaluate the efficiency and scalability of the three methods developed in this paper, namely the dynamic programming method (DP), the sort-projection method (Sort-Proj), and the space partitioning method (Space-Part). All methods were implemented in C++ and compiled by Microsoft Visual Studio 2008. All experiments were conducted on a laptop computer with an Intel Core Duo 1.67GHz CPU and 2GB main memory running Windows Vista Ultimate. We used the NBA data set from http://www.nba.com and synthetic data sets. In our experiments, the skyline of a data set is pre-computed and indexed by an R-tree. The cost reports do not include the cost of skyline computation. All methods run in-memory.

We consider two major factors in our experiments, the dimensionality d of the data space and the number m of the skyline points that strictly dominate the query point, which we call *cardinality*. By default, the dimensionality is set to $d = 3$ and varies from 2 to 8. The cardinality m varies from 50 to 100,000 and is set to 500 by default. All weights in the cost function are equal to 1.

To generate a synthetic data set, we first used an anti-correlated data generator [3] to produce a large data set, from which we computed the skyline. Next, we randomly selected m points from the skyline and computed the minimum bounding box of these m points. Those points which are not selected are then ignored. Next, the minimum corner of the minimum bounding box was set to be the query point. Every experiment was repeated 10 times, and the average result is reported.

8.1 Results on the synthetic data sets

8.1.1 Number of local optimal points

We first show the number of local optimal points in different data sets, which is bounded between $O(m)$ and $O(m^{d-1})$ (Corollary 6.1 in Sect. 6.3). In Fig. 6a, the number of local optimal points increases exponentially as the dimensionality increases, while Fig. 6b shows a linear increase in the number of local optimal points with respect to the cardinality.

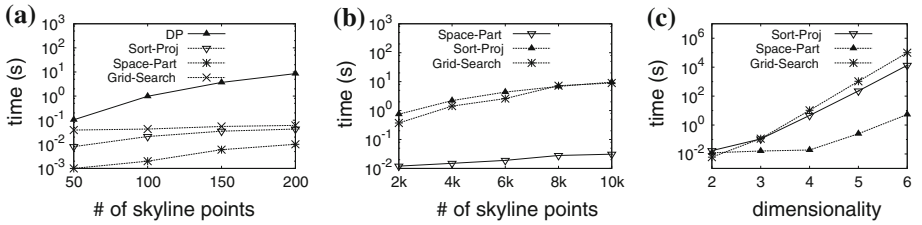


Fig. 7 Comparing the four methods. **a** Effect of n (low). **b** Effect of n (high). **c** Effect of d

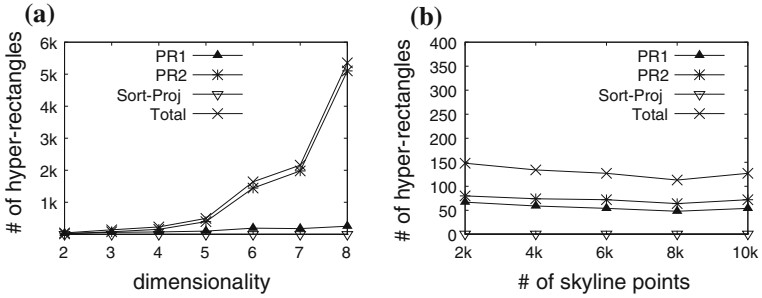


Fig. 8 Pruning power of the Space-Part method. **a** Effect of d. **b** Effect of n

8.1.2 Comparing the three methods

Figure 7 compares the running time of the three methods, as well as the Grid-Search algorithm developed in [12], on different data sets. Because the dynamic programming method needs to maintain a table of the skyline distances of all intersection points, and the table is of size $O(m^d)$, it cannot run on large data sets. Figure 7a shows the performance of the four methods on data sets with low cardinalities (from 50 to 200). We can see that the running time of DP increases quickly as the cardinality increases. The other three algorithms are much faster than DP. Due to the poor scalability of DP, we exclude DP from comparison in the rest of our experiments.

Figure 7b shows the running time of grid search and Sort-Proj increases much faster than Space-Part with respect to cardinality. Space-Part is faster than Sort-Proj by two orders of magnitude.

Figure 7c shows that the running time of grid search, Sort-Proj, and Space-Part goes up exponentially as the dimensionality goes up from 2 to 8. Still, Space-Part is orders of magnitude faster than Sort-Proj and grid search. Sort-Proj also outperforms grid search as the dimensionality increases.

Clearly, our Space-Part algorithm is faster than grid search and Sort-Proj in almost all cases, especially on data sets with high cardinality and/or high dimensionality.

8.1.3 Analysis of the space partitioning method

As the space partitioning method is exceptionally better than the other methods, we investigate it in detail. The performance of Space-Part depends on the number of hyper-rectangles generated and the number of hyper-rectangles pruned. Figure 8 shows the effectiveness of the two pruning rules used in Space-Part. Four curves are shown in both Fig. 8a, b, the number of hyper-rectangles pruned by PR1 and PR2, computed by Sort-Proj, and the total number

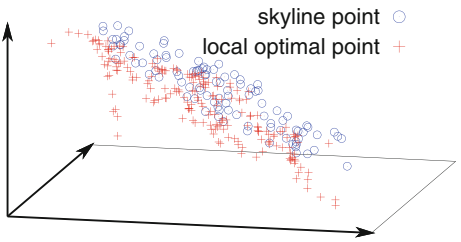


Fig. 9 A 3D illustration of local optimal points

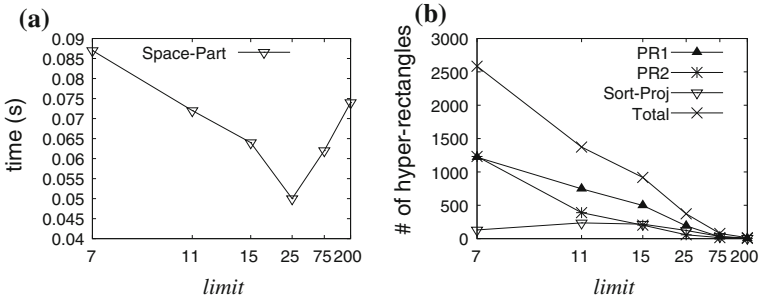


Fig. 10 Effect of $limit$ in Space-Part **a** Running time. **b** Pruning power

of hyper-rectangles generated. In fact, the total number is equal to the sum of the numbers of the hyper-rectangles pruned by PR1 and PR2 and computed by Sort-Proj. It is clear that most of the hyper-rectangles are pruned by PR1 and PR2, only a small amount of hyper-rectangles are actually processed by Sort-Proj. This explains the efficiency of Space-Part. Moreover, PR2 is more powerful in pruning than PR1.

Figure 8a shows that the total number of hyper-rectangles generated increases exponentially against the dimensionality. This is the reason that the running time of Space-Part is exponential to the dimensionality, even most of the hyper-rectangles are pruned. In Fig. 8b, the number of hyper-rectangles is insensitive to the change of the cardinality, so is the running time.

Figure 9 shows a set of skyline points, and the local optimal points generated by the skyline points in a 3-dimensional space. We can see that the local optimal points form a bowl shape. Therefore, many parts of the search space can be pruned after Space-Part finds a few local optimal points.

Figure 10 shows the effect of the parameter $limit$ used in the space partitioning method. The running time is the smallest when $limit$ is set to 25. However, the pruning power decreases all the way when $limit$ increases. The result shows that when $limit$ is small, the major cost of Space-Part is on partitioning the search space. When $limit$ is large, the number of hyper-rectangles generated is small, but the cost of computing the local optimal points in one hyper-rectangle increases. The best setting of $limit$ achieves the balance between the two aspects of cost.

Figure 11 shows the scalability of Space-Part by varying the cardinality from $20k$ to $100k$. The running time is linear with respect to the cardinality. Moreover, Space-Part can answer a skyline distance query within 1.2 s even the query point is dominated by $100k$ skyline points. To this extent, Space-Part is scalable on large data sets. In theory, the number of skyline points is $O((\log n)^{d-1})$ in a set of n uniformly distributed points [2]. So $100k$ skyline points are expected to be generated from a data set of billions of data points if the points are uniformly distributed (Fig. 10).

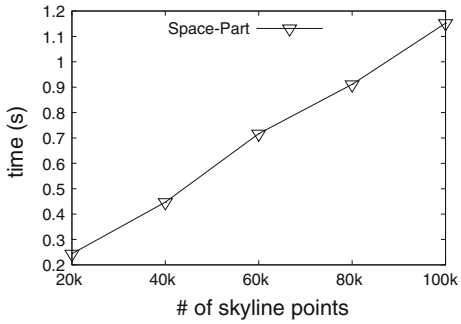


Fig. 11 The Space-Part method on high cardinality data set

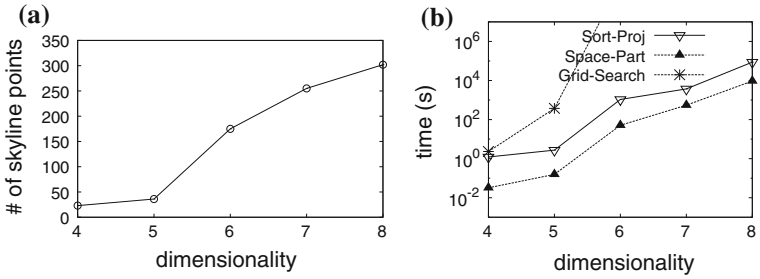


Fig. 12 The NBA data set. **a** # of skyline points. **b** Running time

8.2 Results on the NBA data set

We also conducted experiments on the well-known NBA data set. The data set consists of the seasonal average statistics of 3,986 players throughout their careers. In total, there are 15,748 points. The data set has 17 attributes. We selected the first d attributes where d varies from 4 to 8 to form a data set and compute the corresponding skyline. The list of attributes that we used are number of games played, total minutes played, total points, offense rebounds, defense rebounds, total assists, total steals, and total blocks. Figure 12a shows that the number of skyline points in the NBA data set increases sub-linearly with respect to the dimensionality. Figure 12b shows that the running time of grid search, Sort-Proj, and Space-Part increases exponentially. Still, Space-Part performs at least an order of magnitude better than grid search and Sort-Proj.

9 Discussion and conclusions

In this paper, we proposed a novel skyline distance which measures the minimum cost of upgrading a query point to the skyline. Skyline distance is a measure of competence of non-skyline points in multi-criteria decision problems and can be used for ranking non-skyline points in recommendation systems. Computing skyline distance efficiently is far from trivial. We developed several methods. First, we presented a dynamic programming method. Second, we investigated the boundary of the skyline and proposed a sort-projection method to compute skyline boundary and use it to compute skyline distances. Furthermore, we devised a space partitioning method based on the divide and conquer and the bounding techniques

to boost the performance. We conducted extensive experiments to exhibit the efficiency and scalability of our methods.

We discussed only linear cost functions in this paper. Interestingly, both the sort-projection method and the space partitioning method can be extended to handle monotonic cost functions in general. We can prove that Lemma 5.1 holds for any monotonic cost functions.

Lemma 9.1 *Given p_1, p_2, and q the same as Lemma 5.1, for a monotonic cost function*

$$cost(q, p) = f(p.D_1 - q.D_1, \ldots, p.D_d - q.D_d),$$

we have $cost(q, p_1) > cost(q, p_2)$.

Proof Since $p_1 \prec p_2$, we have $p_1.D_i - q.D_i \geq p_2.D_i - q.D_i$ for $i \in [1, d]$, and $p_1.D_{i_0} - q.D_{i_0} > p_2.D_{i_0} - q.D_{i_0}$ for at least one $i_0 \in [1, d]$. If $f(.)$ is monotonic, then $cost(q, p_1) > cost(q, p_2)$. □

Therefore, Corollary 5.1 is still valid and guarantees that the optimal upgrading position is among the local optimal points. In the space partitioning method, the two pruning rules PR1 and PR2 are still applicable, though the threshold may not define a hyperplane.

Lemma 9.2 *PR1 and PR2 are correct under any monotonic function in the space partitioning method.*

Proof Given R, $max(R)$, and $min(R)$ denoted as Sect. 7,

PR1 if $\max(R)$ is strictly dominated by a skyline point, then R can be pruned, since any point inside R is strictly dominated by a skyline point so it is not a local optimal point.

PR2 if $cost(q, \min(R)) > threshold$, then for any local optimal point p inside R, since $p \prec \min(R)$ we have $cost(q, p) > cost(q, \min(R)) > threshold$ (Lemma 9.2).

□

For future work, it is interesting to consider non-monotonic functions, a challenging problem. Another direction is to develop an index over the set of local optimal points to boost the query answering performance.

Acknowledgments We are grateful to the anonymous reviewers for their very useful comments and suggestions. Part of this work was done when Jian Chen and Jin Huang visited Simon Fraser University. The work was supported in part by the Fundamental Research Funds for the Central Universities, SCUT (Grant No. 2012ZZ0088), the Science and Technology Planning Project of Guangdong Province, China (Grant No. 2011A091000036), the National Natural Science of China (Grant No. 60970044), an NSERC Discovery grant, an NSERC Discovery Accelerator Supplement grant, and a BCFRST Foundation NRAS Endowment Research Team Program grant. All opinions, findings, conclusions, and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

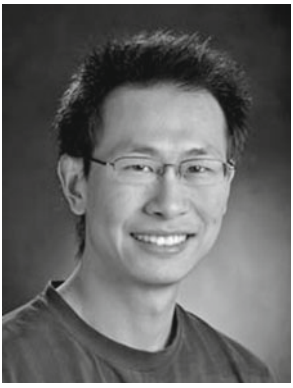
1. Beckmann N, Kriegel HP, Schneider R, Seeger B (1990) The r*-tree: an efficient and robust access method for points and rectangles. In: Proceedings of the 1990 ACM SIGMOD international conference on management of data, SIGMOD'90, New York, NY, USA, ACM, pp 322–331
2. Bentley JL, Kung HT, Schkolnick M, Thompson CD (1978) On the average number of maxima in a set of vectors and applications. J ACM 25:536–543
3. Börzsönyi S, Kossmann D, Stocker K (2001) The skyline operator. In: Proceedings of the 17th international conference on data engineering, Washington, DC, USA, IEEE Computer Society, pp 421–430

4. Chan C-Y, Eng P-K, Tan K-L (2005) Stratified computation of skylines with partially-ordered domains. In: Proceedings of the 2005 ACM SIGMOD international conference on management of data, SIGMOD'05, New York, NY, USA, ACM, pp 203–214
5. Chan C-Y, Jagadish HV, Tan K-L, Tung Anthony KH, Zhang Z (2006) Finding k-dominant skylines in high dimensional space. In: Proceedings of the 2006 ACM SIGMOD international conference on management of data, SIGMOD'06, New York, NY, USA, ACM, pp 503–514
6. Chomicki J, Godfrey P, Gryz J, Liang D (2005) Skyline with presorting: theory and optimizations. In: Intelligent Information Systems'05, pp 595–604
7. Cormen T, Leiserson C, Rivest R (1990) Introduction to algorithms. The MIT Press, Cambridge
8. Dellis E, Seeger B (2007) Efficient computation of reverse skyline queries. In: Proceedings of VLDB, pp 291–302
9. Godfrey P, Shipley R, Gryz J (2005) Maximal vector computation in large data sets. In: Proceedings of the 31st international conference on very large data bases, VLDB'05, VLDB Endowment, pp 229–240
10. Jiang B, Pei J (2009) Online interval skyline queries on time series. In: Proceedings of the 2009 IEEE international conference on data engineering, Washington, DC, USA. IEEE Computer Society, pp 1036–1047
11. Jiang B, Pei J, Lin X, Cheung DW, Han J (2008) Mining preferences from superior and inferior examples. In: Proceeding of the 14th ACM SIGKDD international conference on knowledge discovery and data mining, KDD'08, New York, NY, USA, ACM, pp 390–398
12. Kim Y, You G-W, Hwang S-W (2008) Escaping a dominance region at minimum cost. In: Proceedings of the 19th international conference on database and expert systems applications, DEXA'08, Springer, Berlin, Heidelberg, pp 800–807
13. Kossmann D, Ramsak F, Rost S (2002) Shooting stars in the sky: an online algorithm for skyline queries. In: Proceedings of the 28th international conference on very large data bases, VLDB'02, VLDB Endowment, pp 275–286
14. Lian X, Chen L (2008) Monochromatic and bichromatic reverse skyline search over uncertain databases. In: Proceedings of the 2008 ACM SIGMOD international conference on management of data, SIGMOD'08, New York, NY, USA, ACM, pp 213–226
15. Lin X, Yuan Y, Wang W, Lu H (2005) Stabbing the sky: efficient skyline computation over sliding windows. In: Proceedings of the 21st international conference on data engineering, ICDE'05, Washington, DC, USA, IEEE Computer Society, pp 502–513
16. Luo C, Jiang Z, Hou W-C, He S, Zhu Q (2011) A sampling approach for skyline query cardinality estimation. Knowl Inf Syst, 1–21. doi:10.1007/s10115-011-0441-1
17. Mindolin D, Chomick J (2009) Discovering relative importance of skyline attributes. In: Proceedings of the 35th international conference on very large data bases, VLDB Endowment, August, vol 2, pp 610–621
18. Morse M, Patel JM, Grosky WI (2007) Efficient continuous skyline computation. Inf Sci 177:3411–3437
19. Papadias D, Tao Y, Fu G, Seeger B (2003) An optimal and progressive algorithm for skyline queries. In: Proceedings of the 2003 ACM SIGMOD international conference on management of data, SIGMOD'03, New York, NY, USA, ACM, pp 467–478
20. Papadias D, Tao Y, Fu G, Seeger B (2005) Progressive skyline computation in database systems. ACM Trans Database Syst 30:41–82
21. Pei J, Jiang B, Lin X, Yuan Y (2007) Probabilistic skylines on uncertain data. In: Proceedings of the 33rd international conference on very large data bases, VLDB'07, VLDB Endowment, pp 15–26
22. Sacharidis D, Papadopoulos S, Papadias D (2009) Topologically sorted skylines for partially ordered domains. In: Proceedings of the 2009 IEEE international conference on data engineering, Washington, DC, USA, IEEE Computer Society, pp 1072–1083
23. Sharifzadeh M, Shahabi C (2006) The spatial skyline queries. In: Proceedings of the 32nd international conference on very large data bases, VLDB'06, VLDB Endowment, pp 751–762
24. Sun S, Huang Z, Zhong H, Dai D, Liu H, Li J (2010) Efficient monitoring of skyline queries over distributed data streams. Knowl Inf Syst 25: 575–606. doi:10.1007/s10115-009-0269-0
25. Tan K-L, Eng P-K, Ooi BC (2001) Efficient progressive skyline computation. In: Proceedings of the 27th international conference on very large data bases, VLDB'01, Morgan Kaufmann Publishers Inc, San Francisco, CA, USA, pp 301–310
26. Tao Y, Papadias D (2006) Maintaining sliding window skylines on data streams. IEEE Trans Knowl Data Eng 18:377–391
27. Tao Y, Xiao X, Pei J (2006) Subsky: efficient computation of skylines in subspaces. In: Proceedings of the 22nd international conference on data engineering, ICDE '06, Washington, DC, USA, IEEE Computer Society, p 65
28. Wan Q, Wong RCW, Ilyas Ihab F, Tamer Özsu M, Peng Y (2009) Creating competitive products. Proc VLDB Endow 2:898–909

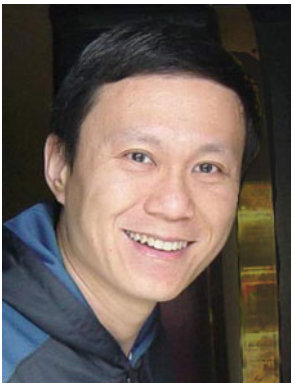
29. Wong RCW, Pei J, Fu AWC, Wang K (2007) Mining favorable facets. In: Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining, KDD'07, New York, NY, USA, ACM, pp 804–813
30. Wu P, Agrawal D, Egecioglu O, Abbadi AE (2007) Deltasky: optimal maintenance of skyline deletions without exclusive dominance region generation. In: Proceedings of the 23rd international conference on data engineering, ICDE 2007, April 15–20, 2007, The Marmara Hotel, Istanbul, Turkey, IEEE, pp 486–495
31. Yuan Y, Lin X, Liu Q, Wang W, Yu JX, Zhang Q (2005) Efficient computation of the skyline cube. In: Proceedings of the 31st international conference on very large data bases, VLDB'05, VLDB Endowment, pp 241–252
32. Zhang Z, Cheng R, Papadias D, Tung AKH (2009) Minimizing the communication cost for continuous skyline maintenance. In: Proceedings of the 35th SIGMOD international conference on management of data, SIGMOD'09, New York, NY, USA, ACM, pp 495–508
33. Zhenjie Z, Yin Y, Ruichu C, Papadias D, Tung A (2009) Kernel-based skyline cardinality estimation. In: Proceedings of the 35th SIGMOD international conference on management of data, SIGMOD'09, New York, NY, USA, ACM, pp 509–522
34. Zhu L, Li C, Tung A, Wang S (2011) Microeconomic analysis using dominant relationship analysis. Knowl Inf Syst, 1–33. doi:10.1007/s10115-010-0337-5

Author Biographies

Jin Huang received his ME and Ph.D. degrees, both in Computer Science, from Sun Yat-Sen University, China, in 2004 and 2010, respectively. Currently, he is postdoctoral researcher of South China Normal University, China. His current research interests cover database, data mining, and information retrieval.



Bin Jiang is a Research Scientist at Facebook. Bin has published in premier academic journals and conferences. He served as a reviewer for TKDE and in the program committees of several international conferences such as SIGKDD and ICDM. Bin holds a Ph.D. degree in Computer Science from Simon Fraser University, Canada, and received his B.Sc., and M.Sc., degrees from Peking University, China, and University of New South Wales, Australia, respectively.



Jian Pei is a Professor at the School of Computing Science, Simon Fraser University, Canada. He is interested in researching, developing, and deploying effective and efficient data analysis techniques for novel data intensive applications, including data mining, Web search, data warehousing and online analytic processing, database systems, and their applications in social networks and media, health-informatics, business, and bioinformatics. His research has been extensively supported in part by governmental funding agencies and industry partners. He is also active in developing industry relations and collaboration, transferring technologies developed in his group to industry applications, and developing proof-of-concept prototypes. Since 2000, he has published 1 textbook, 2 monographs, and over 170 research papers in refereed journals and conferences, which have been cited thousands of times. He has served in the organization committees and the program committees of over 160 international conferences and workshops. He is the associate editor-in-chief of IEEE Transactions of Knowledge and Data Engineering (TKDE), and an associate editor or editorial board member of the premier academic journals in his fields. He is an ACM Distinguished Speaker and a senior member of Association for Computing Machinery ACM and IEEE. He is the recipient of several prestigious awards.

Jian Chen received her BS and Ph.D. degrees, both in Computer Science, from Sun Yat-Sen University, China, in 2000 and 2005, respectively. She is currently an associate professor and director of the Data Mining Group in SSE (SCUT). Her research interests include database, data mining, social networks and their related applications. She has served as a PC member for international conferences such as PAKDD and CIKM.

Yong Tang is currently Professor in the School of Computer Science and Director of Research Center of Software Technology for Information Service at South China Normal University. His main research interests in database, service computing, and social network service; he has authored over 150 technical papers.