An Efficient Approach to Finding Potential Products Continuously

Yu-Ling Hsueh[a], He Ma[b,*], Chia-Chun Lin[a], Roger Zimmermann[c]

[a]*Dept. of Computer Science & Information Engineering, National Chung Cheng University, Taiwan*
[b]*Sino-Dutch Biomedical Information and Engineering School, Northeastern University, Shenyang, China, 110169*
[c]*School of Computing, National University of Singapore, Singapore*

Abstract

Skyline points and queries are important in the context of processing datasets with multiple dimensions. As skyline points can be viewed as representing marketable products that are useful for clients and business owners, one may also consider non-skyline points that are highly competitive with the current skyline points. We address the problem of continuously finding such potential products from a dynamic d-dimensional dataset, and formally define a potential product and its upgrade promotion cost. In this paper, we propose the *CP-Sky* algorithm, an efficient approach for continuously evaluating potential products by utilizing a *second-order skyline* set, which consists of candidate points that are closest to regular skyline points (also termed the *first-order* skyline set), to facilitate efficient computations and updates for potential products. With the knowledge of the second-order skyline set, *CP-Sky* enables the system to (1) efficiently find substitute skyline points from the second-order skyline set only if a first-order skyline point is removed, and (2) continuously retrieve the top-k potential products. Within this context, the *Approximate Exclusive Dominance Region* algorithm (*AEDR*) is proposed to reduce the computational complexity of determining a candidate set for second-order skyline updates over a dynamic data set without affecting the result accuracy. Additionally, we extend the *CP-Sky* algorithm to support the computations of top-k potential products. Finally, we present experimental results on data sets with various distributions to demonstrate the performance and utility of our approach.

Keywords: Skyline Queries, Query Processing, Multi-dimensional Databases, Data Management.

1. Introduction

Skyline queries have made a huge contribution to data filtering over large data sets with multiple dimensions for decision makers. The formal definition of skyline queries is given as follows: given a data set P in d-dimensional space, a distinct object set S is returned, where S contains all objects p_i which are not dominated by another object p_j in P. We say p_1 dominates p_2 ($p_1 \prec p_2$ for short), if and only if p_1 is better than or equal to p_2 on all dimensions, and p_1 is strictly better than p_2 on at least one dimension. The early work on skyline queries assumed that data objects are static [26, 28]. Subsequently, the existing approaches [12, 16, 27, 38] have addressed the efficient update support for skyline queries over moving objects with d dynamic dimensions, each of which represents a spatial or non-spatial value. As skyline points can be viewed as marketable products, one may also consider the non-skyline points that are highly competitive with the current skyline points, and we term such non-skyline points as *potential products*. We use the following examples to describe the motivation of our key idea.

Example 1. *(Finding Potential Products) For hotel customers, the final decision regarding choosing hotels is made based on multiple hotel attributes (*e.g.*, lower price, closer to the city center, higher rating). With the assistance of skyline queries, the candidate hotels are provided, while most of the undesirable hotels are filtered out, even those with very competitive attributes. In Table 1, the skyline points are $\{p_1, p_2\}$. However, one may also observe that p_5, eliminated by p_2 during the skyline query processing, is very competitive with p_2. When the two skyline points p_1 with Price = \$209, Distance (to the city center) = 0.1 miles, and Rating = 3, and p_2 with Price = \$89, Distance = 0.8 miles, and Rating = 4 are provided, p_5 might also be a good alternative for the customer, because the price of p_5 is only \$1 more than that of p_2 and it is a bit farther to the city center. From another perspective, a hotel owner might be seeking an opportunity to increase business sales. By searching for the potential products, the owner may adjust the price or other adjustable attributes of the hotel to increase its competitiveness. We refer to p_5 as a potential product, because the cost of upgrading p_5 to a marketable product is minimal among other non-skyline products.*

Example 2. *(Finding Potential Products Continuously) For stock traders, when they choose to either buy or sell a certain stock, multiple attributes (*e.g.*, current stock price, number of shares they own, binding price and number of shares offered by other traders, company news,* etc.*) affect their final decisions. Similar to Example* 1*, stock traders also need to monitor these potential products and buy/sell the shares once they change to become skyline products, in order to earn maximum*

*Corresponding author: mahe@bmie.neu.edu.cn
Email addresses: hsueh@cs.ccu.edu.tw (Yu-Ling Hsueh), mahe@bmie.neu.edu.cn (He Ma), lcc103p@cs.ccu.edu.tw (Chia-Chun Lin), rogerz@comp.nus.edu.sg (Roger Zimmermann)

Table 1: An example of finding potential products using a hotel data set. The skyline points are $\{p_1, p_2\}$

Hotel ID	Price	Distance	Rating
p_1	\$209	0.1 miles	3
p_2	\$89	0.8 miles	4
p_3	\$95	5 miles	2
p_4	\$250	0.5 miles	3
p_5	\$90	1.1 miles	4

profits. Since these attributes are updated frequently, continuously finding potential products for an extended period of time is significantly important for stock traders.

A *promotion* [11, 20, 29–31] is a further action to upgrade a potential product (*i.e.*, to become a skyline point), when the business owners are willing to adjust one or some of its attributes. One must consider the promotion cost, which is the least cost to perform a promotion operation.

Example 3. *(Promotion) Consider that a promotion operation is conducted on p_5. The promotion cost for p_5 is taken by decreasing the price (an adjustable attribute) by at least \$1 + ϵ to turn p_5 into a marketable product, where ϵ is a small positive value. For example, the price of p_5 is set to \$88.99, the largest price strictly better (smaller) than \$89. The final marketable products (i.e., skyline points) after the promotion operation are $\{p_1, p_2, p_5\}$.*

The objective of the proposed algorithm is to continuously retrieve the potential products (*i.e.*, non-skyline data points) with the minimal promotion cost from a dynamic data set, as some of the data attributes change over time. These changes are caused by real time customer feedback (*e.g.*, ratings) or the adjustments made by the business owners for balancing the cost and profit. Finding potential products is useful for further decision making, especially when investigating the potential products for promotion. To achieve an efficient computation of potential products continuously, the following challenges must be addressed: (1) an effective query result update mechanism is needed to provide a short response time when reporting the current query results, and (2) an efficient strategy to reduce the search space dimensionality is also required.

Peng *et al.* [29–31] have addressed a similar idea for finding potential products. Their main emphasis is to find potential "stars" in social networks whose graphical data are transformed into a coordinate system before the search is conducted. Several pruning techniques, such as Skyboundary and Infra skyline, have been designed to reduce the search space; however, heavy computation is still incurred in order to retrieve the alternate skyline set, and the performance degrades when processing a large data set. Furthermore, none of these approaches consider dynamic data, regardless of the fact that data attributes in social networks change frequently. The use of a Skyband [27] query helps to find a set of non-skyline points which are dominated by at most k points. The potential products, however, have no direct association with the number of dominated points. In other words, the points in a 1-Skyband are not guaranteed to contain the potential product, which might be dominated by several skyline points. As a result, one must perform k-Skyband, where k is set to be as large as the number of skyline points to ensure finding the potential products correctly.

In this paper, to efficiently answer the queries of potential product computations from a dynamic data set, we utilize the *second-order skyline* as a candidate set such that the query processor can avoid accessing the entire dynamic data set with high dimensionality. An example of finding potential products is shown in Figure 1, where the black solid points are the skyline set (*i.e.*, the first-order skyline, $S1 = \{s_1^1, s_1^2, s_1^3, s_1^4\}$), and the solid grey points represent the second-order skyline set ($S2$ for short). The potential product with the minimal promotion cost is represented by $s_2^5 \in S2$, as an arrow from s_2^5 indicates the minimal cost to promote s_2^5. The use of potential product computations enables a user to search for such potential products that are highly competitive with the existing marketable products. A promotion operation (*i.e.*, reducing the dimensional values) can be conducted afterwards to upgrade the potential products. The second-order skyline is utilized as a fundamental technique to answer such a query over dynamic data sets and enable to continuously maintain the efficient updates of $S1$ and $S2$ sets. We propose the **C**ontinuous **P**otential **Sky**line algorithm (*CP-Sky* for short) in this paper. We show that the second-order skyline technique, which avoids expensive calculations on large data sets, facilitates the potential product computations as well as the first-order skyline updates, which need to be completed first before finding potential products, because the promotion cost is computed based on the first-order skyline.

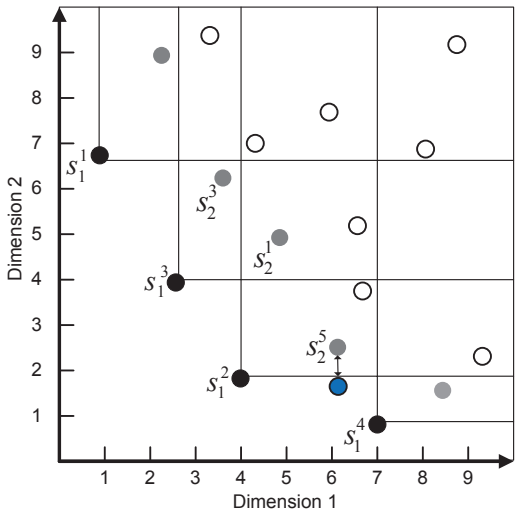


Figure 1: An illustration of the potential product: s_2^5.

The remainder of this paper is organized as follows. Section 2 describes the related work. We formally define a potential product and address the problem statement of our work in Section 3. Sections 4.1 and 4.2 present the details of the baseline algorithm and our proposed approach to compute potential

products, respectively. In particular, we detail the updated technique for $S1$ and $S2$ in Section 4.2.1, and the *CP-Sky* algorithm in Section 4.2.2. The *CP-Sky* algorithm is further extended to support the computations of top-k potential products, and the details of which are described in Section 4.2.3. We extensively verify the performance of our technique in Section 5, and finally conclude with Section 6.

2. Related Work

Börzsönyi *et al.* [5] proposed the straightforward non-progressive *Block-Nested-Loop* (BNL) and *Divide-and-Conquer* (DC) algorithms for static skyline processing. The BNL approach recursively compares each data point with the current set of candidate skyline points, which might be dominated later. BNL does not require data indexing or sorting. The DC approach divides the search space and evaluates the skyline points from its sub-regions, respectively, followed by merge operations to evaluate the final skyline points. Both algorithms may incur many iterations and are inadequate for on-line processing. Tan *et al.* [34] presented two progressive processing algorithms: the *bitmap* approach and the *index* method. The bitmap approach encodes dimensional values of data points into bit strings to speed up the dominance comparisons. The index method classifies a set of d-dimensional points into d lists, which are sorted in increasing order of the minimum coordinate. The index scans the lists synchronously from the first to the last entry. With the pruning strategies, the search space is reduced.

The *nearest neighbor* (NN) method [15] indexes the data set with an R-tree. NN utilizes nearest neighbor queries to find the skyline results. The approach repeats the query-and-divide procedure and inserts the new partitions that are not dominated by some skyline point into a to-do list. The algorithm terminates when the to-do list is empty. Similar to the NN method, Papadias *et al.* proposed the *skyband* query [27], where a k-skyband query reports a set of points dominated by at most k points. The skyband query is usually used to process top-k queries over different datasets. For example, Gong *et al.* [8] used the k-skyband partition to complete the top-k query. The original dataset is partitioned into two sub-datasets depending on whether they fall into k-skyband or not. Another study [23] utilized the skyband to process the top-k query over incomplete datasets, *i.e.*, data missing several dimensions. They designed multiple structures such as expired skyline, shadow skyline, and thickness warehouse to improve the searching performance. The *branch and bound skyline* (BBS) algorithm [26] traverses an R-tree to find skyline points. Although BBS outperforms the NN approach, the performance can deteriorate due to many unnecessary dominance checks. In order to improve the efficiency of searching for the skyline and pruning unnecessary search paths, Bartolini *et al.* [2] proposed the $SaLSa$ algorithm to pre-sort the input data and to minimize the data accesses using a monotone limiting function. Another study [4] that combines collaborative filtering and the skyline operator was presented to generate a skyline according to personalized features and requirements. Bartolini *et al.* [1] proposed a distributed access model that processes m sub-queries to find an overall best matching result by integrating the results of each sub query that deals with only a subset of the query features. The definition of the second-order skyline was firstly introduced and the *Best* operator was designed to search for a certain-order skyline (*e.g.*, first-order skyline, second-order skyline). Furthermore, the MPO and $iMPO$ algorithms were presented to efficiently process data sets with partial orders and to return the result objects ordered according to user-specified preferences of the skyline attributes. Trimponias *et al.* [36] introduced a distributed approach to vertically reduce the search complexity by proposing a two-phase framework. To solve *size constrained skyline queries, Lu* et al. *[21] proposed the skyline ordering algorithm to form a skyline-based partition which is used to facilitate the computation of size constraints.*

Many of the recent techniques aim to provide continuous skyline support for moving objects and data streams. Although the existing static methods can be applied to obtain continuous results by repeatedly performing them over an interval of time, the trade-off lies in compromising either the accuracy or efficiency during query processing. If the query interval is too short, the system incurs high computational cost. On the contrary, if the interval is too long, the query results are not accurate since some data updates might not be detected by the system. To achieve both accuracy and efficiency, it is essential to apply continuous algorithms to deal with dynamic data sets. Lin *et al.* [18] presented n-of-N skyline queries against the most recent n of N elements to support on-line computation against sliding windows over a rapid data stream. Morse *et al.* [24] proposed a scalable *LookOut* algorithm for efficiently updating the continuous time-interval skyline. Sharifzadeh *et al.* [33] introduced the concept of *Spatial Skyline Queries* (SSQ). Given a set of data points P and a set of query points Q, SSQ retrieves those points of P which are not dominated by any other point in P considering their derived spatial attributes with respect to query points in Q. A continuous skyline query processing strategy was presented in [12] with a kinetic-based data structure. However, providing prompt query responses was not considered in the design. A suite of novel skyline algorithms based on a *Z-order* curve [7] was proposed in [16]. Among the solutions, *ZUpdate* facilitates incremental skyline result maintenance by utilizing the properties of a Z-order curve. In our early work [9], we proposed the ESC algorithm to efficiently manage the query results by delegating the time-consuming skyline update computations to another independent procedure, which is processed after the query processor reports the latest skyline query results. The key idea is to maintain a second-order skyline set which is a skyline candidate set pre-computed when a traditional skyline point requests an update. The approach had also been implemented as a system framework presented in [10]. Other related techniques can be found in the literature [3, 6, 19, 25, 35, 38].

Chan *et al.* [6] presented three algorithms for evaluating skyline queries with partially ordered attributes. Their solution transforms each partially ordered attribute into a two-integer domain value, which allows the users to utilize index-based algorithms to compute skyline queries in the transformed space.

However, all of the techniques proposed in [6] have limited progressiveness and pruning abilities. In real applications, dynamic preferences for categorical attributes are more common than a fixed ordering for skyline query evaluation. One straightforward solution is to enumerate all of the possible preferences and to materialize all of the results of the preferences; however, the costs of a full materialization are usually prohibitive. Therefore, Wong *et al.* [37] proposed a semi-materialization method named the *IPO-tree* search, which stores only partial useful results. With these partial results, the result of each possible preference can be efficiently returned. However, an IPO-Tree only considers very simple totally-order-like user preferences. Sacharidis *et al.* designed a topological sort-based mechanism called *topologically-sorted skylines* (TSS) [32], which involves a novel dominance check function to eliminate false hits and misses. In addition, TSS can handle dynamic skyline queries. Zhang *et al.* [39] extended the lattice theorem and an off-the-shelf skyline algorithm and then designed a mechanism that employs an appropriate mapping of a partial order to a total order.

In [13], Jang *et al.* used the Manhattan distance to evaluate the cost between a skyline point and a query point. The skyline minimum vector was utilized to promote a non-skyline point to a new skyline point. Kim *et al.* [14] designed a grid-based cell searching algorithm to prune out unnecessary searches. Based on this structure, they sorted the cells in ascending order according to the cost and found the cheapest updating region in the early stage. Later, Peng *et al.* [29–31] proposed a series of works on the topic of skyline promotion. In [31], the authors designed a pruning mechanism for the *potential rising stars* retrieved from a social network. First, the social network graph is transformed into a coordinate system by which they tried to improve the inefficiency of the brute force algorithm which was adopted in [29] for retrieving potential stars. Based on the characteristics of a skyline set, the designed approach adds a promotion boundary presented in [31] to achieve a boost to the search for the potential stars. In [31], although optimization by adding a boundary achieves better performance, the approach still needs to search a large data space for the non-skyline points set, which is not a promising solution. Therefore, reducing the search space has become a key step. In order to reduce the designated search space, an *Infra* skyline computation for the skyline promotion was designed. The main emphasis of the Infra-skyline design is to retrieve another skyline by excluding the original skyline. However, this approach still incurs heavy computation to retrieve the alternate skyline set, and the performance degrades when processing a large data set. Lu and Jensen [20] proposed a probing algorithm and a spatial-join approach to solve the top-k products updating problem. The R-tree is used to index both the competitive objects and the upgrading candidates so as to estimate the lower bound upgrading cost. As a result, the spatial-join approach then returns the incremental results on-the-fly before testing all the objects in the dataset. Another related approach on a similar topic, named skyline upgrade, was presented in [11], which tried to solve the *skyline distance* by using a space partitioning mechanism to enhance the original dynamic programming technique. However, all the aforementioned studies differ from the main goal of this research which is to retrieve potential products continuously from a dynamic data set while providing quick responses to the users. Therefore, we utilize the concept of the ESC algorithm [9] in this paper for efficient updates over moving objects so as to further deal with the computations of top-k potential products continuously.

3. Notations and Problem Definitions

We formally define potential products in this section. We first introduce the notations of symbols used throughout the paper as shown in Table 2. Note that some symbols may be used as prefixes and their meanings are to be explained when used. The definitions of potential product, promotion cost (*pcost*), and skyline promotion operation are defined first. Subsequently, the problem statement is addressed to clarify the purpose of our work. Throughout this paper, the terms "skyline points", "first-order skyline points", and "marketable products" are used interchangeably. A potential product is a non-skyline point with the minimal promotion cost among all other non-skyline points.

Table 2: Symbols and functions.

Symbols	**Descriptions**
P	Number of data objects, where each point in P, *e.g.*, $p_1 = (x^1, x^2, \ldots, x^d)$ or $p_2 = (y^1, y^2, \ldots, y^d)$, contains the d dimensional values.
d	Number of dimensions.
$p_1 \prec p_2$	p_1 dominates p_2.
$p_1 \not\prec p_2$	p_1 does not dominate p_2.
$S1$ and m	$S1$ is the first-order skyline point set (traditional skyline query result set). $S1 = \{s_1^1, s_1^2, \ldots, s_1^m\}$, where m is the size of $S1$.
$S2$ and m'	$S2$ is the second-order skyline point set. $S2 = \{s_2^1, s_2^2, \ldots, s_2^{m'}\}$, where m' is the size of $S2$.
$EDR(s_1^i)$	Exclusive dominance region of s_1^i.
N and n_i	$N = P - S1$ is a non-skyline set, consisting of $\{n_1, n_2, \ldots, n_\ell\}$, where each $n_i \in N$ is a non-skyline point and ℓ is the size of N.
$C_{n_i}[1 \ldots d]$	Dimensional promotion cost of dimension 1 to d of n_i.
$n_i.pcost$	Promotion cost of a non-skyline point $n_i \in N$.
$cor_w(n_i)$	A function to return the w-th dimensional value of n_i.
ϵ	A user-specified small positive value that makes the to-be-promoted product completely escape from the dominance of skyline points.
$D(s_1^i)$	A dominance set containing a group of $S2$ points which are dominated by s_1^i to substitute a removed or moving s_1^i point.

Definition 1. *(*A Potential Product*)*
A potential product n_i exists among $N = \{n_1, ..., n_\ell\}$, where N

is a non-skyline set (i.e., $N = P - S1$), and ℓ is the size of N. The following condition holds: $n_i.pcost \leq n_j.pcost$, where $n_i, n_j \in N$.

Definition 2. *(*Promotion Cost: pcost*)*
A promotion cost pcost is the minimal cost for upgrading a non-skyline point n_i to a skyline point. Let $S1' \subseteq S1$ be a set of skyline points that dominate $n_i \in N$.

The list of dimensional promotion cost for all dimensions is denoted by $C_{n_i} = \{c_1, c_2, ..., c_d\}$, which is preserved for the promotion operations defined in Definition 3. $C_{n_i}[w]$, which is the w^{th} element in the C_{n_i} list, represents the dimensional promotion cost for dimension w. Therefore, the promotion cost (as a squared distance) of n_i can be measured by Equation 1.

$$n_i.pcost = \sum_{w=1}^{d} C_{n_i}[w]^2 \tag{1}$$

Specifically, the promotion cost of n_i is given by the following equations:

$$n_i.pcost = \begin{cases} \min(minCost, minCost') : & \text{if } |S1'| > 1 \\ minCost^*_w : & otherwise. \end{cases} \tag{2}$$

where

$$minCost = \min_{w=1}^{d}(\max(cor_w(n_i) - cor_w(s_1^j) + \epsilon), \forall s_1^j \in S1')^2 \tag{3}$$

$$minCost' = \sum_{w=1}^{d} \min(cor_w(n_i) - cor_w(s_1^j) + \epsilon, \forall s_1^j \in S1')^2 \tag{4}$$

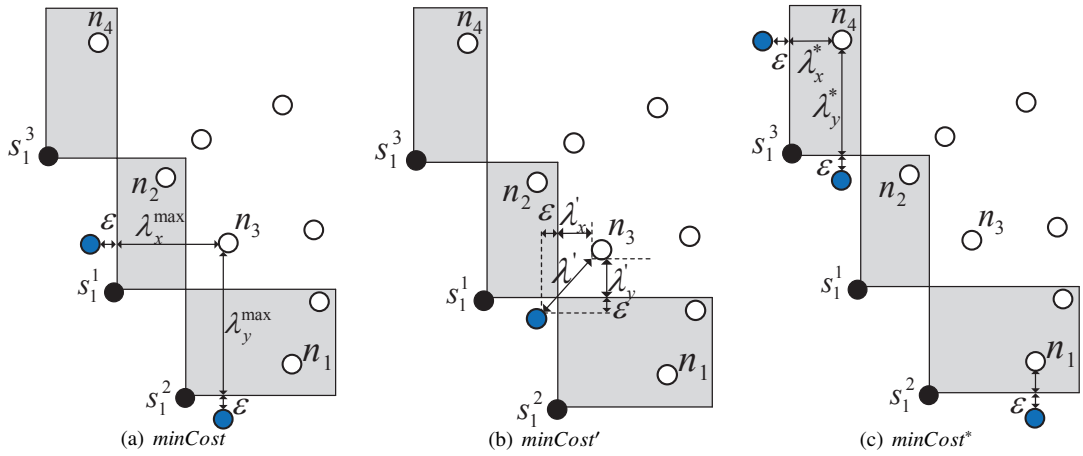
$$minCost^* = \min_{w=1}^{d}(cor_w(n_i) - cor_w(s_1^j) + \epsilon)^2 \tag{5}$$

Equation 2, equivalent to Equation 1 and used for measuring $C_{n_i}[1 \ldots d]$, shows the promotion cost for a non-skyline point n_i. There are two cases in Equation 2 to determine the promotion cost. Assume that $S1' \subseteq S1$ is a set of skyline points that dominate n_i. When n_i is dominated by multiple skyline points (*i.e.*, $|S1'| > 1$), $n_i.pcost$ is set to the minimal cost chosen between *minCost* (Equation 3) and *minCost′* (Equation 4). *minCost* is set to the minimal cost of the maximum difference between n_i and $s_1^j \in S1'$ for each dimension among all skyline points in $S1'$ as shown in Equation 3, where ϵ is a user-specified small positive value that makes the to-be-promoted product completely escape from the dominance of skyline points in S'. Similarly, *minCost′* is the minimal cost by calculating the minimal difference between n_i and $s_1^j \in S1'$ for each dimension among all skyline points in $S1'$ as shown in Equation 4. When n_i is only dominated by one skyline point s_1^j (*i.e.*, $|S1'| = 1$), n_i is located in an *exclusive dominance region* [27] (*EDR*). In such a case, the promotion cost is set to $cor_h(n_i) - cor_h(s_1^j) + \epsilon$, when dimension h satisfies the minimal difference among all other dimensions. As a result, we set $minCost^*$ to $cor_h(n_i) - cor_h(s_1^j) + \epsilon$, and set this value to $C_{n_i}[h]$. Otherwise, the rest of the other promotion cost $C_{n_i}[w]$ is set to 0, $\forall w$ and $w \neq h$.

The example in Figure 2 illustrates the two cases of promotion cost, where the black solid points are the first-order skyline points. For the first case, in Figure 2(a), n_3 is dominated by more than one first-order skyline point. Based on Equation 3, we first obtain the maximum difference $\lambda_x^{max} = n_3.x - s_1^1.x$ for the x dimension, because it is larger than $n_3.x - s_1^2.x$. Next, we obtain $\lambda_y^{max} = n_3.y - s_1^2.y$ as the maximum difference for the y dimension. Therefore, *minCost* is set to $\lambda_x^{max} + \epsilon$, because it is smaller than $\lambda_y^{max} + \epsilon$. The procedure then calculates *minCost′* based on Equation 4. In Figure 2(b), for each dimension, we obtain the minimal difference among s_1^1 and s_1^2, which are the two points dominating n_3. The minimal difference for the x dimension is obtained by $n_3.x$ - $s_1^2.x + \epsilon$, and the minimal difference for the y dimension is obtained by $n_3.y$ - $s_1^1.y + \epsilon$. We sum up the squared difference of each dimension to obtain the minimal cost λ', and set $minCost' = \lambda'$. Because *minCost′* is smaller than *minCost*, the promotion cost is set to *minCost′*. Furthermore, $C_{n_3}[x] = \lambda'_x + \epsilon$ and $C_{n_3}[y] = \lambda'_y + \epsilon$. For the second case, in Figure 2(c), n_4 residing in an *EDR* (the gray-shaded area) is only dominated by one first-order skyline point; therefore, among all dimensions, the minimal difference indicated by an arrow (*i.e.*, $minCost^* = \lambda_x^* + \epsilon$) is obtained based on the x dimension. We then set $C_{n_4}[x] = \lambda_x^* + \epsilon$ and $C_{n_4}[y] = 0$. Similarly, the minimal difference for n_1 is obtained based on the y dimension.

Algorithm 1 illustrates the calculation of the promotion cost and C_{n_i} for a non-skyline point following Equations 2 to 5. When the non-skyline point n_i is dominated by more than one first-order skyline point, there are two cases for the promotion cost computation. One is: for each individual dimension, the maximal value (denoted as $cost_w^+$) among the differences between n_i and each $S1$ point that dominates n_i (Lines 7–8). The other is: for each individual dimension, the minimal value (denoted as $cost_w^-$) among the differences between n_i and each $S1$ point that dominates n_i (Lines 9–10). The promotion cost *minCost′* is then the sum of $C_{n_i}^-[w]^2$ for all the dimensions (Line 11). The promotion cost *minCost* is then the square of the minimal value of the set $C_{n_i}^+$ among all the dimensions (Line 12). Hence, the promotion cost is the minimal value between *minCost* and *minCost′*. In Lines 13–15, *minCost* is the promotion cost and the promotion cost for each dimension of C_{n_i} is set to 0 and $C_{n_i}[h]$ is set to *minCost*, where h satisfies the minimal difference among all other dimensions. Lines 16–18 show the other case. *minCost′* is the promotion cost; therefore, for all $w = 1$ to d, $C_{n_i}[w]$ is set to $C_{n_i}^-[w]$. Otherwise, we calculate the difference between n_i and the only $S1$ point dominating n_i for each dimension, and treat the square of the minimal value among all dimensions as the promotion cost $minCost^*$ in Lines 19–24. The promotion cost for each dimension of C_{n_i} is set to 0 and $C_{n_i}[h]$ is set to $minCost^*$, where h satisfies the minimal difference among all other dimensions.

The promotion costs defined in our paper (Definition 2) are different from the upgrading cost [20] and the skyline distance [11]. In [20], the upgrading cost is computed based on a product cost function, which was not specifically defined because the authors stated that there may be different ways to define a product cost function that captures the overall cost of



(a) *minCost* (b) *minCost′* (c) *minCost**

Figure 2: Examples of promotion cost of Equation 2.

Algorithm 1: *pcost(n_i)*

```
1  Initialization:
   let n_i be a non-skyline point
   S1′ be a set of skyline points that dominate n_i, s_1^j ∈ S1′
2  pcost = 0, minCost = ∞, minCost′ = 0, minCost* = 0
3  C+_{n_i}[1:d] = 0, C−_{n_i}[1:d] = ∞, C_{n_i}[1:d] = 0
4  if (sizeof(S1′) > 1) then
5      for (w = 1 to d) do
6          for (∀ s_1^j ∈ S1′) do
7              if (cor_w(n_i) − cor_w(s_1^j) > C+_{n_i}[w]) then
8                  C+_{n_i}[w] = cor_w(n_i) − cor_w(s_1^j) + ε
9              if (cor_w(n_i) − cor_w(s_1^j) < C−_{n_i}[w]) then
10                 C−_{n_i}[w] = cor_w(n_i) − cor_w(s_1^j) + ε
11         minCost′ += C−_{n_i}[w]^2
12     minCost = min_{w=1···d} C+_{n_i}[w]^2, where k finds the min value
13     if (minCost < minCost′) then
14         C_{n_i}[w] = 0, ∀w = 1 to d, and C_{n_i}[k] = minCost
15         pcost = minCost
16     else
17         C_{n_i}[w] = C−_{n_i}[w]^2, ∀w = 1 to d
18         pcost = minCost′
19 else
20     for (w = 1 to d) do
21         C−_{n_i}[w]^2 = cor_w(n_i) − cor_w(s_1^j) + ε
22     minCost* = min_{w=1···d} C−_{n_i}[w]^2, where k finds the min value
23     C_{n_i}[w] = 0, ∀w = 1 to d, and C_{n_i}[k] = minCost*
24     pcost = minCost*
25 n_i.pcost = pcost
```

manufacturing the products. The upgrading cost is generally obtained by $f_p(t') - f_p(t)$, where f_p is a product cost function, t is a potential product and t' is a skyline point after t is upgraded. In other words, the upgrading cost is the increased cost to upgrade t. In our work, on the other hand, we basically generalize and quantify the promotion cost. We have defined the promotion cost as the least cost to upgrade a potential product. We have specifically defined the promotion cost in Equation 2, which is fundamental to obtaining the minimum difference among all the skyline points to promote the potential product plus a small positive value ϵ.

As for the skyline distance in [11], a linear cost function was adopted to evaluate the cost of moving a non-skyline point q to a new position q' so that q is not dominated by any other point in the skyline set. The skyline distance is defined as the cost in the form of $cost(q, q') = \sum_{i=1}^{d} w_i(q'.D_i - q.D_i)$, where w_i is a weight for dimension i, representing the level of importance of the dimension, d is the total number of dimensions, and $(q'.D_i - q.D_i)$ represents the difference between the dimensional values of q and q'. Therefore, the skyline distance is basically the summation of each weighed difference between the dimensional value of q and q'. Similarly, our promotion cost is generally a squared distance between q and q'. Although the definition of the promotion cost used to serve the purpose of computing potential products in the end is similar, we have clearly defined and provided an algorithm to compute the promotion cost in our paper.

Definition 3. *(Promotion: $f(n_i)$)*
Given a non-skyline point n_i, a promotion can be performed by $cor_w(n_i) - C_{n_i}[w]$, where $w = 1$ to d for each dimension and $cor_w(n_i)$ returns the w^{th} dimensional value of n_i.

A promotion function $f(n_i)$ implemented based on the above definition returns a newly promoted skyline point for a potential product. The following lemma and proof of correctness support this process.

Lemma 1. *Given the current skyline set $S1=\{s_1^1, s_1^2, \ldots, s_1^m\}$, and a newly promoted skyline point s_1', resulting from the promotion operation (Definition 3), $S1$ must contain s_1'. That is, s_1' is the final skyline point after promotion.*

Proof: (By definition) Since s_1' is a newly promoted skyline point, s_1' cannot be dominated by another point in P. Therefore, the $S1$ set must contain s_1'. ∎

Problem Statement

The problem statement for our work is described as follows. Given a set of dynamic data set P and a time period $\triangle t$, a pair of (n_i, t_j) is returned for each time period between t_{j-1} and t_j, where $n_i \in P - S1$ is a top potential product with the minimal promotion cost (based on Equation 2), and t_{j-1} and t_j are two consecutive time stamps within $\triangle t$.

$$\arg \min_{n_i \in P-S1} f_{t_j}(n_i)$$

where $f(n_i)$ indicates the promotion function at a given time stamp $t_j \in \triangle t$.

4. Finding Potential Products Continuously

The general setup of the problem consists of a set of dynamic data objects with d dimensions. Moving objects can freely maneuver in an unrestricted and unpredictable fashion, meaning that their parameters may arbitrarily change their values. As a result, existing approaches have addressed designing a query processor with the capability of processing skyline queries continuously when a set of dynamic data is given. In reality, interesting features can be further discovered beyond the skyline set. In particular, we term the *continuous potential products* retrieved from the non-skyline set as the products that would require the minimal cost to be promoted as marketable products (*i.e.*, become skyline points). For example, investors might seek profitable stocks to invest in, and a potential stock could be a good candidate. For health care, physicians would want to discover the patients who have test screening results that are close to approaching high risk levels. In the following sections, we first introduce the baseline algorithm. Since the potential products are close to the existing skyline points in nature, we adopt a range-based algorithm as the baseline approach, which uses range queries to look for surrounding non-skyline points. Next, we design a search algorithm termed *CP-Sky*, which efficiently updates the query results as the data attributes change over time, and retrieves potential products by utilizing the second-order skyline set, which is essentially similar to skyline ordering in [21]. However, we consider a dynamic data set which implies that the second-order skyline might be different from time to time. Therefore, we propose an update strategy to efficiently update the $S1$ and $S2$ sets, which need to be maintained always up-to-date so that the potential products can be accurately retrieved continuously by the algorithms we developed based on the $S1$ and $S2$ sets. The *CP-Sky* algorithm greatly reduces the search space to look for potential products. Furthermore, we extend our approach to compute the top-k potential products ordered by the promotion cost in ascending order.

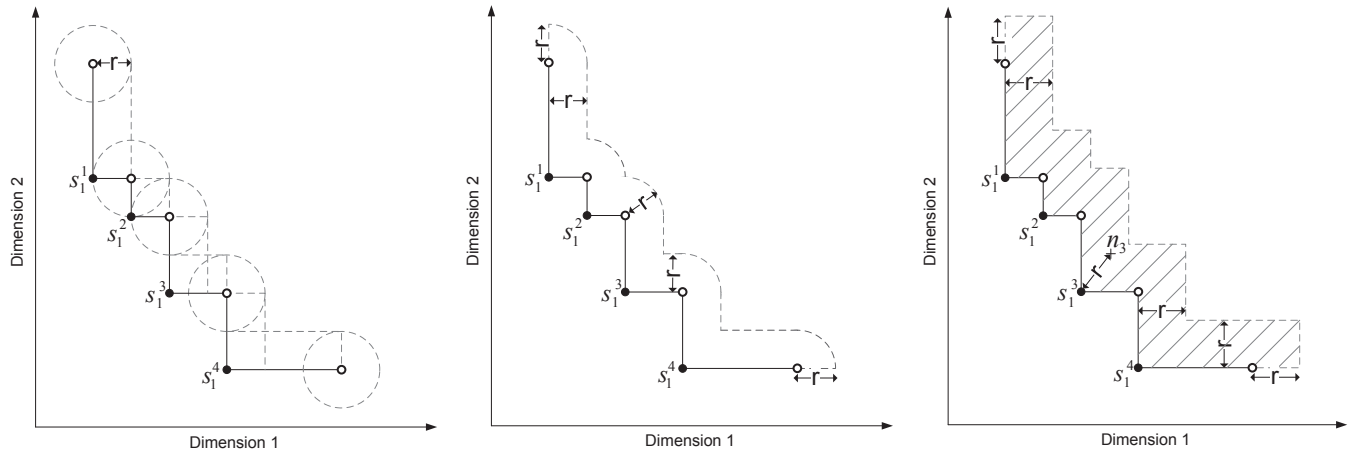
4.1. Range-based Query Processing

In this section, we introduce a baseline algorithm to find the potential product shown in Algorithm 2. The basic idea is to search around the skyline points and find the non-skyline point with the minimal promotion cost as the potential product. Initially, the search radius of a range query along the skyline is set to r, and it is set to be doubled in each search round if no non-skyline point is found. The search procedure is not terminated until at least one non-skyline point is found or all points have been searched. The initial value of r has an impact on the number of promotion cost calculations and the search rounds. For example, when r is set to a small value, the range query can possibly be processed several times in order to find the potential product. Conversely, the range query might involve a large number of promotion cost calculations in the first round, when r is set to a large number.

The procedure is continuously performed and the potential product is re-evaluated only when a point in P changes its one or more dimensional values (Lines 2−3). The skyline set is retrieved from the typical *BBS* algorithm (Line 4) [26]. Ideally, a range query needs to be processed around the skyline points (Figure 3(a)). In the skyline query, the exact query region is shown in Figure 3(b). However, in practice, considering the drawback between the value of r and the search cost, we randomly choose one skyline point (*e.g.,* s_1^3 in Figure 3(c)), and find its nearest neighbor (*i.e.,* n_3). The Euclidian distance between s_1^3 and n_3 is then set as the initial value of r (Lines 5−7 in Algorithm 2). Hence, the search region SR can be illustrated with the shaded region (shown in Figure 3(c)), where all the non-skyline points, whose distance to the skyline points is not larger than r, are covered. These non-skyline points within SR are considered as the *candidate set* and are inserted into the potential product set named *cpSkyLine*. This method guarantees that the potential product can be retrieved from the first search round and no further search is necessary. The points in *cpSkyLine* are then sorted by promotion cost in ascending order and the potential product with the minimal promotion cost is the first element (Lines 14−15). This algorithm can be easily modified to compute the top-k potential products. In Line 6, a k-NN query is performed instead to obtain the k_{th} nearest neighbor for computing the search radius (*i.e.*, the distance between s_1^r and the k_{th} nearest neighbor). Note that the calculation of the promotion cost follows the rules stated in Equations 2 to 5 and is detailed in Algorithm 1. The drawback of the range-based algorithm is the random selection procedure of one skyline point and finding its nearest neighbor. The best case is that this nearest neighbor is exactly the potential product, and hence no other point will be found within the distance r. Conversely, the worst case is that this nearest neighbor makes all non-skyline points be within r of the skyline points. Therefore, the performance of the range-based algorithm depends on a random procedure and the data distribution, which is not stable in terms of efficiency.

4.2. CP-Sky Query Processing

In this section, we introduce the *CP-Sky* algorithm by formally defining the second-order skyline ($S2$) utilized for the efficient computations and updates for potential products. The major challenging issue of continuous computations of potential products is to avoid unnecessary dominance checking on irrelevant data points for query result updates. After observing the *BBS* algorithm [26], we deduce that when evaluating the skyline query results, a set of the second-order skyline points



(a) The search region of the range query around the skyline points

(b) The exact search region

(c) The search region in practice

Figure 3: Illustration of determining the search region SR for the range-based algorithm.

Algorithm 2: *Range-based Algorithm(P)*

```
1  Initialization:
   let r be the initial search radius for a range query
   let cpSkyLine = ∅ be the potential product set
   let top_cpSkyLine = ∅ be the final potential product
2  while (TRUE) do
3      if (any point in P moves) then
4          S1 = traditional-BBS(P)
5          randomly choose s_1^r ∈ S1
6          find NN as the nearest neighbor of s_1^r , NN ∈ (P − S1)
7          r = dist(s_1^r, NN)
8          calculate the search region SR based on r
9          while (cpSkyLine.isEmpty()) do
10             for (∀n_i ∈ SR) do
11                 if (n_i ∉ cpSkyLine) then
12                     n_i.pcost = proCost(n_j)
13                     cpSkyLine.insert(n_i, n_i.pcost)
14         cpSkyLine.sort() /* sorting by n_i.pcost
           ascendingly                                  */
15         top_cpSkyLine ← the top element in cpSkyLine
```

can always be obtained with a little extra work, while retrieving the *first-order skyline* ($S1$) points. We refer to the traditional skyline query result as the *first-order skyline*, consisting of $S1 = \{s_1^1, ..., s_1^m\}$, required to be available for finding potential products. The *second-order skyline* $S2 = \{s_2^1, ..., s_2^{m'}\}$ is defined as follows:

Definition 4. *(Second-order Skyline)*
A data point p is a second-order skyline point iff *there exists no p' that dominates p, where $p, p' \in (P - S1)$ and $p \neq p'$. Informally, all $S2$ points are dominated by $S1$ and the rest of the data points $(P - S1 - S2)$ are dominated by both $S1$ and $S2$.*

Notably, as dynamic data sets are considered, we describe how to efficiently maintain both $S1$ and $S2$ to support the evaluation of continuous query processing for potential products in Section 4.2.1. Unlike the range-based algorithm (described in Section 4.1) which processes the query results periodically without considering previous results for updates, *CP-Sky* is an efficient algorithm that utilizes the second-ordered skyline to handle continuous computations of potential products. Finally, we detail the *CP-Sky* algorithm in Section 4.2.2.

4.2.1. Efficient Update Techniques for $S1$ and $S2$

To handle continuous computations in a dynamic environment, we must first maintain the $S1$ set before finding potential products, because the promotion cost is computed based on the first-order skyline. With the knowledge of the second-order skyline set, the system is able to efficiently find the substitute skyline point from the $S2$ set when it is removed or at least one value of its dimensions changes. The $S2$ set is further utilized to retrieve the top-k potential products without accessing the entire dynamic data set with high dimensionality. Therefore, in this section, we describe how to efficiently maintain both $S1$ and $S2$ sets to support continuous computations for potential products.

The features of an $S2$ set are as follows: (1) it is a data set that covers all the new $S1$ candidate points, and (2) $S2$ might be a relatively small data subset of the entire data set. Therefore, the query processor can efficiently update the $S1$ set. The size of the $S2$ sets retrieved from different distributed data sets are shown in our experiments (Section 5.1). An example is shown in Figure 4. If an $S1$ point s_1^2 moves to Region I, the search space for *CP-Sky* to update the query result only involves the $S1$ and the $S2$ sets. In this case, s_1^2 remains an $S1$ point, but it dominates s_1^1. *CP-Sky* needs to remove s_1^1 from the $S1$ set and s_1^1 becomes a new $S2$ point, since no existing $S2$ point can dominate it. Due to the movement of s_1^2, *CP-Sky* searches for new $S1$ points from the $S2$ set only. Since s_2^2 (an *EDR data point* which is only dominated by one skyline point) is left un-

dominated, s_2^2 becomes a new $S1$ point and is removed from the $S2$ set.

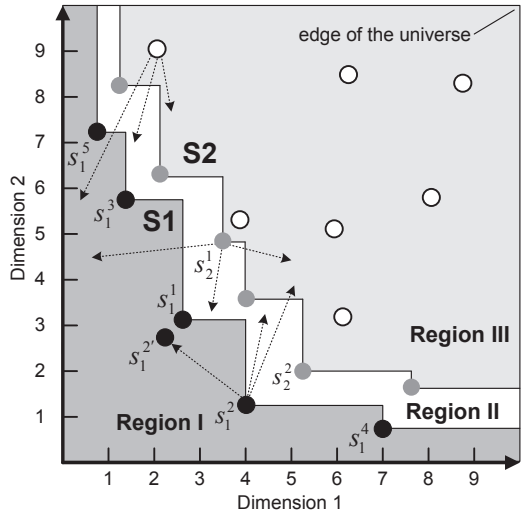


Figure 4: Examples of first-order ($S1$) and second-order ($S2$) skyline sets.

To further reduce the search space of visiting $S2$ points to update the skyline query result, we define a *dominance set* in Definition 5 for each $S1$ point s_1^i. A *dominance set* contains a group of $S2$ points which are dominated by s_1^i (denoted by $D(s_1^i)$) to substitute a removed or moving s_1^i point, when the dominance relationship has changed. For example in Figure 2, the dominance set of s_1^3 includes s_2^1 and s_2^3. If s_1^3 is removed, the system only has to check the $S2$ points in $D(s_1^3)$, instead of all of the $S2$ points. In this example, s_2^3 becomes a new $S1$ point, so it is removed from $S2$. We formally define a *dominance set* and establish Lemma 2 which states that a dominance set must contain all the necessary $S1$ candidate points as follows:

Definition 5. *(*Dominance Set: $D(s_1^i)$*)*
A dominance set *(denoted by* $D(s_1^i) = \{s_2^r, ...s_2^v\}$*) of a skyline point* s_1^i *is an* $S2$ *subset where* $\forall s_2^w \in D(s_1^i), s_1^i \prec s_2^w$.

Lemma 2. *Let A be the skyline points extracted from* EDR(s_1^i). $D(s_1^i)$ *must contain A, which is a subset of* $D(s_1^i)$.

Proof: (By contradiction) Let $p \in A$ be a point not included in $D(s_1^i)$. This is a contradiction, since p is only dominated by s_1^i. Therefore, it must be in $D(s_1^i)$. It follows that $D(s_1^i)$ must contain all points in A. ∎

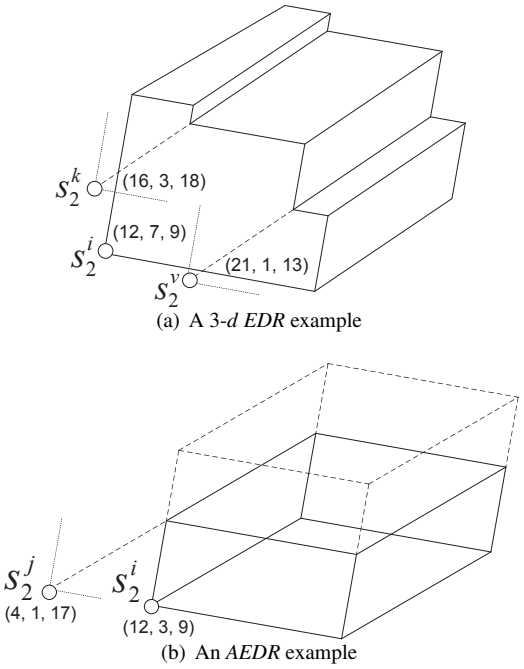
The *CP-Sky* algorithm delegates the necessary $S2$ maintenance (an independent procedure from $S1$ updates) to the query processor after the $S1$ updates are completed. For example, new $S2$ points must be retrieved to substitute s_2^2, which is now an $S1$ point. To avoid scanning through all of the data points in Region *III* for new $S2$ points, we propose an *approximate exclusive dominance region* (*AEDR*) computation in contrast to a traditional *exclusive dominance region* (*EDR*) computation [27]. The existing work [27, 38] performs time-consuming *EDR* data point computations for the skyline query result updates. An *EDR* is not usually pre-computed because of the complexity of the calculation, especially in higher dimensional spaces. In our algorithm, we utilize the *BBS* approach to initially compute the $S1$ set, and the $S2$ points are computed during the execution of the modified dominance-checking procedure which runs a window query to determine a set of candidate skyline points.

The main procedures for skyline updates include *S1Evaluation* for the $S1$ updates and *S2Evaluation* for the $S2$ set maintenance. To improve the performance of *S2Evaluation*, we introduce the concept of an *approximate exclusive dominance region* (*AEDR*), which helps to reduce the amortized cost of the $S2$ updates without affecting the result accuracy. When $d = 2$, the traditional *EDR* is a regular rectangle. However, an *EDR* has an irregular shape in higher dimensions. For example, in Figure 5(a), s_2^i is a skyline point to be deleted. The *EDR* of s_2^i is an irregular polygon after deleting the overlapping area with the dominance area of s_2^k and s_2^v. Based on this observation, we can obtain a regular shaped *EDR* only when we consider the skyline points which have a value x^h larger than that of s_2^i in only one dimension. Because these points are completely "outside" of the *EDR*, they can trim all of the areas that represent the upper dimensional value x^h. Our *AEDR* may cover non-exclusive regions such that the data points in *AEDR* are a super set of those in the traditional *EDR*. Therefore, the result accuracy is not affected. Using the data set in the *AEDR* to find the substitute skyline point for a to-be-removed skyline point s_2^i must incur higher computational time than using the data points in the *EDR*, because of the correspondingly larger number of data points in an *AEDR*. However, in our experimental results, we show that the amortized cost of the S2 update, including the cost of *AEDR* computation and finding a new substitute skyline point from the *AEDR*, is mostly lower compared to using a traditional *EDR*. Therefore, we can confirm the usability of the *AEDR*. The definition of an *AEDR* is given in Definition 6.

Definition 6. *(*Approximate EDR: AEDR(s_2^i))
Let $s_2^i = (x^1, x^2, ..., x^d)$*, and* $s_2^j = (y^1, y^2, ..., y^d)$ *be two* $S2$ *points.* AEDR(s_2^i) = s_2^i.DomArea − (s_2^i.DomArea ∩ s_2^j.DomArea*), if there exists exactly one* $x^h < y^h$, $1 \le h \le d$, $\forall s_2^j \in (S2 - s_2^i)$.

For example, in Figure 5(b), s_2^i is the skyline to be deleted and the solid rectangular box is an *AEDR*, which is a regular shape resulting from trimming the overlapping dominance areas of s_2^i and s_2^j. We utilize the *AEDR* to search for the new $S2$ points by traversing the data R-tree. Each *MBR e* extracted from the heap is checked to find whether it intersects with the *AEDR*. If true, we check whether *e* is dominated by the existing $S2$ points.

The skyline update algorithm is implemented in an event-driven fashion to handle the skyline query updates. Note that each time stamp in this paper represents the triggering time when a point (a $S1$ point, a $S2$ point, or a regular data point) issues an update to the system. All the incoming update requests with their time stamps are inserted into a queue and the query processor then sequentially handles each request in the queue. The $S1$ and $S2$ sets are up-to-date, assuming that



(a) A 3-*d EDR* example

(b) An *AEDR* example

Figure 5: Traditional *EDR* vs. *AEDR*.

Algorithm 3: *skylineUpdate*(P)

```
1  (S1, S2) = modified-BBS(P)
2  while (TRUE) do
3      listen to an update request from p
4      if (p) then
5          let S̃2 = ∅ be a new S2 point set
6          let S̄2 = ∅ be the S2 points to be removed
7          S1Evaluation(p, S1, S2, S̃2, S̄2)
8          output the updated S1 set to the query user and continue
9          S2Evaluation(p, S1, S2, S̃2, S̄2)
10         update p to p′
```

Algorithm 4: *S1Evaluation*(p, $S1$, $S2$, $\widetilde{S2}$,$\overline{S2}$)

```
1  let p′ be the updated point of p
2  checkFreeS2 = false
3  if (S1.contains(p)) then
4      remove p from S1
5      checkFreeS2 = true
6  if (S1 ⊀ p′) then
7      find I ⊂ S1 to be the new S2 points dominated by p′
8      S1 = S1 ∪ p′ − I
9      S̃2.insert(I)
10 if (checkFreeS2) then
11     find D(p) ⊂ S2 to be the S2 points dominated by p
12     for (each o ∈ D(p)) do
13         if (S1 ⊀ o) then
14             S1 = S1 ∪ o
15             S̄2.insert(o)
```

the query processor is able to responsively complete the computations and there is no communication delay. The update procedure shown in Algorithm 3 includes *S1Evaluation* (Algorithm 4) and *S2Evaluation* (Algorithm 5). The initial $S1$ and $S2$ sets are computed through the BBS algorithm in Line 1. When the query processor receives an update request from p in Line 3, which can be a first-order, second-order or a regular data point, it first performs *S1Evaluation* to examine whether the request affects the $S1$ set (the query result). In Line 7, the existing $S1$ and $S2$ sets are passed into the procedure for updates. Additionally, because of the update of p, $\widetilde{S2}$ stores the new $S2$ set and $\overline{S2}$ stores the $S2$ points to be removed from $S2$. The processes involved with these two sets, which are subsequently passed into *S2Evaluation*, do not impact the $S1$ set. Therefore, the procedure outputs the updated $S1$ points as soon as *S1Evaluation* is completed. In Line 9, *S2Evaluation* then processes the rest of the non-$S1$-related computations to update the $S2$ set. Line 10 updates p to p', which is the updated point of p.

In the *S1Evaluation* procedure (see Algorithm 4), Line 1 obtains the updated point p' with the new position of p. Lines 3−5 remove p, if p is currently in the $S1$ set and $checkFreeS2$ is set to true, because we need to ensure the update of p results in any new $S1$ points released from the $S2$ set. Lines 6−9 update the $S1$ set if p' is a new $S1$ point (*i.e.*, when $S1 \not\prec$ (does not dominate) p'), and delete the I set, which is the new $S2$ set dominated by p'. Since all the points in I become new $S2$ points, which are inserted into $\widetilde{S2}$ in Line 9, the $S2$ set is updated later in the *S2Evaluation* procedure. Lines 10−15 check whether all the $S2$ points in $D(p)$ are still dominated by p'. In Line 11, we first obtain $D(p)$, which contains the $S2$ points originally dominated by p. In Line 14, since o (a new $S1$ point after p moves) can never dominate any $S1$ point, o is directly added to the $S1$ set. This is because o is an *EDR* data point, and therefore it must not dominate any existing $S1$ points. o is inserted into the $\overline{S2}$ set, which is to be removed later in the *S2Evaluation* procedure.

S2Evaluation (see Algorithm 5) is a more expensive procedure than *S1Evaluation*, because it involves *AEDR* computations to find a set of new $S2$ points to substitute a moving or removed $S2$ point. First, we insert p into the $\overline{S2}$ set before the update procedure is performed. *DataRtree-AEDR* is performed to find the I set, where each point is only dominated by a corresponding $S2$ point in $\overline{S2}$. We find the I set to substitute the to-be-removed $S2$ points in $\overline{S2}$. Line 5 removes the entire $\overline{S2}$ set from $S2$, and I, each of which is a new $S2$ point, is added to $S2$. The $\widetilde{S2}$ set contains new $S2$ points to be inserted into $S2$. Line 6 finds the existing $S2$ points that are dominated by at least one point in $\widetilde{S2}$. Therefore, each point in $D(\widetilde{S2})$ becomes a regular data point. Subsequently, in Line 6, the $S2$ set is updated by adding the $\widetilde{S2}$ set and removing the $D(\widetilde{S2})$ set. Similarly, since each point in $\widetilde{S2}$ was originally an $S1$ point, the $D(\widetilde{S2})$ set is directly removed from the $S2$ set without further checking whether any points in $S2$ are dominated. Lines 7−10 are processed if p' is a new $S2$ point. The insertion of p' may

Algorithm 5: *S2Evaluation*(p, $S1$, $S2$, $\widetilde{S2}$, $\overline{S2}$)

```
1  let p' be the updated point of p
2  if (S2.contains(p)) then
3      S2̄.insert(p)
4  I = DataRtree-AEDR(S2̄)
   /* I is a regular data set, each of which is only
   dominated by a S2 points in S2̄                    */
5  S2 = S2 ∪ I − S2̄
6  find D(S2̃) ⊂ S2 to be a regular data set due to the insertion of S2̃
   S2 = S2 ∪ S2̃ − D(S2̃)
7  if (S1 ≺ p') then
8      if (S2 ⊀ p') then
9          find I' ⊂ S2 to be a regular data set due to the insertion
           of p'
10         S2 = S2 ∪ p' − I'
```

dominate some existing $S2$ points; therefore, Line 9 finds the dominated $S2$ points (*i.e.*, I') and removes them from the $S2$ set in Line 10.

4.2.2. The CP-Sky Algorithm

We propose the *CP-Sky* algorithm to search for the potential products. The key idea is to continuously maintain the first-order and the second-order skylines during the query procedure such that we can efficiently retrieve the potential products. The maintenance of the $S1$ and $S2$ skyline sets over dynamic data have been detailed in Section 4.2.1. The lemma and the proof of correctness related to the *CP-Sky* algorithm are provided as follows.

Lemma 3. *Given the $S1$ and $S2$ sets, the top potential product n^* can be found from $S2$. There exists no n_i, which has a promotion cost smaller than that of any point in* $(P - S1)$.

Proof: (By definition) A non-skyline point $n_i \in (P - S1 - S2)$ is dominated by at least one point s_2^j in $S2$. Therefore, $cor_w(n_i) \geq cor_w(s_2^j), \forall w = 1$ to d. By the definition of a promotion cost defined in Definition 2, the $s_j.pcost$ must be smaller than that of n_i. Let n^* in $S2$ be a point with the minimal promotion cost among all $S2$ points. n^* must be the potential product with the minimal promotion cost among all non-skyline points ($P - S1$). Therefore, using the $S2$ set is sufficient to find a potential product. ∎

Algorithm 6 illustrates the algorithm to continuously search for the potential products and efficient updates over moving data, given $S1$ and $S2$ updated by *skylineUpdate* (Algorithm 3). The procedure then updates potential products if either $S1$ or $S2$ is updated (Line 3). Because all the second-order skyline points in $S2$ are treated as candidates of potential products, they are sorted by promotion cost in ascending order (Lines 4–7). The top potential product is then the first element (Line 8), which is guaranteed to be the potential product with minimal promotion cost.

Algorithm 6: *CP-Sky(S1, S2)*

```
1  Initialization:
   let cPSkyLine = ∅ be the potential product set
   let top_cPSkyLine = ∅ be the final potential product
2  while (TRUE) do
3      if (S1 or S2 is updated) then
4          for (∀ s_2^i ∈ S2) do
5              s_2^i.pcost = proCost(s_2^i)
6              cPSkyLine.insert(s_2^i, s_2^i.pcost)
7          cPSkyLine.sort()
8          top_cPSkyLine ← the top element in cPSkyLine
```

4.2.3. Top-k CP-Sky Query Processing

In this section, we discuss how to extend the *CP-Sky* algorithm to find the top-k potential products from all non-skyline points, and the modified approach is called *kCP-Sky* algorithm. Based on Lemma 3, the system can only guarantee the top potential product to be retrieved from $S2$. Therefore, we need to modify the *CP-Sky* algorithm to return the top-k potential products. We utilize the $S2$ set to look for the initial top-k potential products (*kPP* for short) from $S2$ first. Next, to guarantee finding the top-k potential products, we must perform k range queries (as the details described in Algorithm 2) over the non-skyline points for each product in the *kPP* set. The lemma and the proof of correctness followed by the pseudo codes for computing top-k potential products are described as follows.

Lemma 4. *Let kPP contain k products in ascending order of their promotion cost retrieved from $S2$ and let kPP' contain the kPP set and the union set of the products found in the range queries performed on each product in kPP. The final top-k potential products must be found from kPP'.*

Proof: (By definition) Let s_2^i be an $S2$ point, $s_2^i \notin kPP$. The promotion cost of s_2^i must be larger than that of any product in *kPP*. According to the rule of transitivity, the promotion cost of any product in the range query of s_2^i must be larger than that of any product in *kPP*. Therefore, the final top-k potential products must be found from kPP'. ∎

The time complexity of the *kCP-Sky* algorithm is bounded to the number of potential products (*i.e.*, k), whereas the range-based algorithm checks the entire $S1$ points to look for surrounding points as the candidates of potential products. The time complexity of the *kCP-Sky* algorithm is $O(k \times log|P|)$, where $|P|$ is the number of data points and the complexity of each range query is $O(log|P|)$, because a range query evaluation is based on the R-tree index structure. Therefore, at most k range queries are performed to obtain the top-k potential products. The time complexity of the range-based algorithm is $O(|S1| \times log|P|)$, since we need to check every skyline point to find the top-k potential products. Generally, k is a relatively smaller number than the size of the $S1$ points, particularly when the data cardinality is large and the dimensionality is high. Therefore, *kCP-Sky* significantly outperforms the range-based approach.

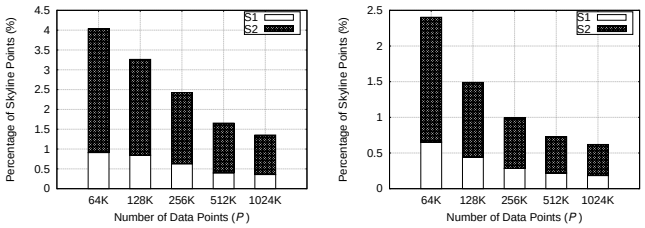
Algorithm 7: *kCP-Sky(k, S1, S2)*

```
1  Initialization:
   let cpSkyLine = ∅ be the potential product set
   let topK_cpSkyLine = ∅ be the final top-k potential products
2  while (TRUE) do
3      if (S1 or S2 is updated) then
4          for (∀s_2^i ∈ S2) do
5              s_2^i.pcost = proCost(s_2^i)
6              cpSkyLine.insert(s_2^i, s_2^i.pcost)
7          cpSkyLine.sort()
8          let kPP be the top-k products retrieved from cpSkyLine
9          kPP' ← kPP
10         for (∀s_2^j ∈ kPP) do
11             find kNN as the k_th nearest neighbor of s_2^j
12             r = dist(s_2^j, kNN)
13             calculate the search region SR based on r
14             for (∀n_w ∈ SR) do
15                 n_w.pcost = proCost(n_w)
16                 kPP' ← n_w
17         cpSkyLine.sort()
18         topK_cpSkyLine ← the top-k products from cpSkyLine
```

Finally, Algorithm 7 illustrates the approach of continuously finding the top-k potential products, where Lines 1−7 are identical to Algorithm 6. The procedure is performed in an event-driven fashion. That is, if any $S1$ or $S2$ point is updated (Line 3), the potential products are also re-evaluated. Line 8 obtains the initial top-k products from $S2$ and Line 9 first inserts the initial top-k products into kPP'. In Lines 10−16, the promotion cost is calculated for each product in the range query of each product in kPP. As a result, all candidates for top-k potential products are found and stored in Line 18.

5. Experimental Evaluation

The experiments were conducted to evaluate the performance of potential product computations. We compared the proposed *CP-Sky* approach with the range-based algorithm stated in Section 4.1 and the *Infra* skyline algorithm [30]. We use $k = 1$ as the default value in all experiments. All of these algorithms utilize R-trees as the underlying structure for indexing the data and skyline points. We use the Spatial Index Library [17] for the R-tree index. A page size of 4Kbytes is deployed, resulting in node capacities between 94 ($d = 5$) and 204 ($d = 2$). The $S1$ and $S2$ sets are indexed by a main-memory R-tree to improve the performance of the dominance checks. Our data sets were generated on a terrain service space of $[0, 1000]^2$. We initially generated from 64,000 to 1,024,000 (1) uniformly distributed and (2) anti-correlated data points with dimensions in the range of 2 to 5, respectively. The dimensional values of the generated data points were then updated based on the random-walk mobility model [22] to simulate the movement patterns of the moving data objects. Each object moves with a constant velocity until an expiration time. The velocity is then replaced by a new velocity with a new expiration time. The object update ratio, defined as the percentage of all data points that issue updates, is set in a range from 1% to 5%. Additionally, we obtained two real-world data sets of the air quality and households in our experimental evaluation.



(a) Uniform distribution (b) Anti-correlated distribution

Figure 6: The ratios of the size of $S1$ and $S2$ to the entire data set ($d = 5$, $f_{update} = 3\%$, $k = 1$).

The query results are evaluated in an event-driven fashion. Therefore, the query processor calls different procedures based on each specific event type. We consider the overall CPU and I/O cost because the *CP-Sky*, range-based and *Infra* algorithms that we evaluated return potential products after the second-order skyline or a candidate set is found. Our experiments use several metrics to compare these algorithms and the details of the experimental results are presented in the following sections. In summary, Sections 5.1− 5.5 contain the evaluation results using the synthetic data sets. Subsequently, Section 5.6 presents the performance evaluation using the real-world data sets. Table 3 summarizes the default parameter settings in the following simulations.

Table 3: Simulation parameters.

Parameter	Symbol	Default	Range
Number of data points	P	128,000	64,000, 128,000, 256,000, 512,000, 1,024,000
Number of dimensions	d	5	2, 3, 4, 5
Update ratio	f_{update}	3%	1%, 2%, 3%, 4%, 5%
Number of requested potential products	k	1	1, 5, 10, 15, 20

5.1. Sizes of $S1$ and $S2$ on Different Distrusted Datasets

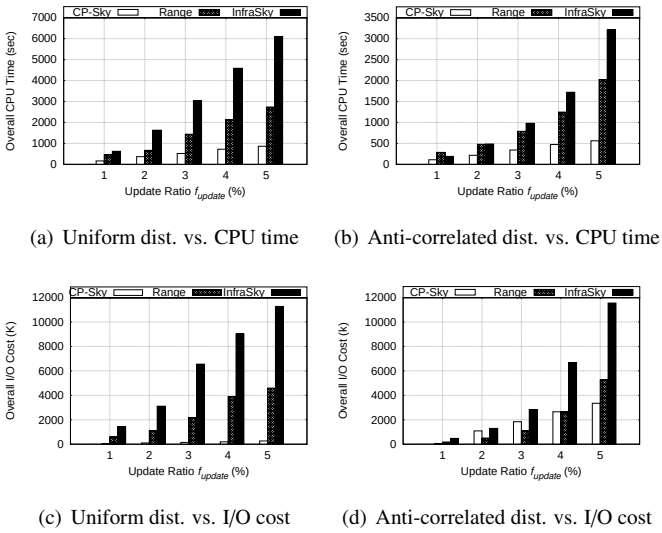
The second-order skyline set as defined in Definition 4 is a relatively small data set based on our observation. Both $S1$ and $S2$ are accessed by the *CP-Sky* algorithm; therefore, the size of both sets affects algorithm performance. In this section, we show the ratios of the sizes of $S1$ and $S2$ to the entire data set in percentages for three different distributed datasets used for our experiments. As shown in Figure 6(a) for the uniformly distributed data set, we can see that the percentages for $S1$ and $S2$ are only up to 1%, and 3%, respectively. The ratio decreases

as the data cardinality increases, because the sizes of the $S1$ and $S2$ sets do not significantly affected by the data cardinality. Figure 6(b) show the percentages for $S1$ and $S2$ on the anti-correlated data set. The percentages for both sets are small. As a result, the use of $S1$ and $S2$ greatly reduces the the update cost of continuous skyline queries and continuous computations of potential products.

5.2. Effect of Update Ratios

Figures 7(a)−(b) show the average CPU time by varying the update ratio for evaluating the performance of the *CP-Sky*, range-based, and *Infra* algorithms during the potential product computations for the uniformly and anti-correlated distributed data sets, respectively, while Figures 7(c)−(d) show the I/O cost for each distributed data set. The data cardinality is fixed at 128,000, the dimensionality at 5, and k at 1. For the uniformly distributed data set, as we can see in Figures 7(a) and (c), compared with the range-based algorithm and *InfraSky* algorithm, the *CP-Sky* approach outperforms them both in terms of both CPU time and I/O cost. With an increase in the update ratio, the overall CPU time and I/O cost of all these three approaches also increases. Dealing with the point updates, the *InfraSky* algorithm monitors whether the updated points are promoted by comparing them with the promotion boundary, which applies the BBS algorithm two times.

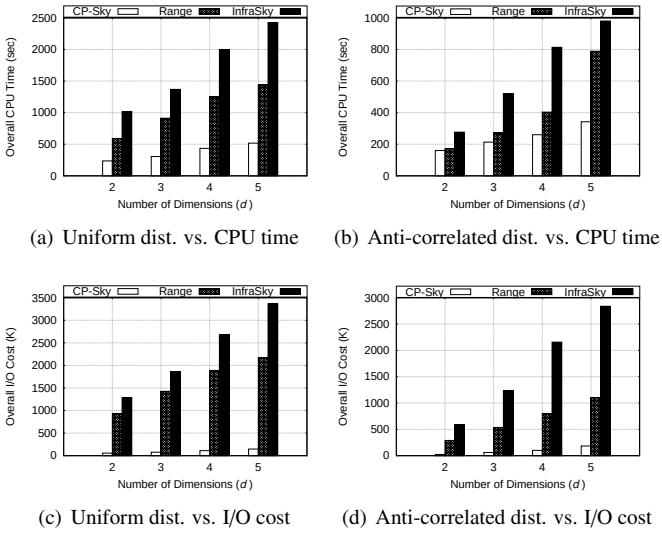
This results in the *InfraSky* algorithm performing the worst. Conversely, the range-based and the *CP-Sky* approaches only apply the BBS algorithm once. The proposed *CP-Sky* approach calculates the update only when the $S1$ or $S2$ sets are updated, which results in less calculation and fewer page accesses. We can observe that for the uniformly distributed data set in Figures 7(a) and (c), all of the three approaches take less computational cost both in CPU time and I/O cost, because most of data points are dominated, resulting in fewer steps to terminate the computations. Furthermore, because the number of the $S2$ set is small, the computation time is less, while looking for potential products. Therefore, since these three approaches perform extremely well, the performance gap among these algorithms can be ignored. For the anti-correlated distributed data set in Figures 7(b) and (d), similar results are produced.



(a) Uniform dist. vs. CPU time (b) Anti-correlated dist. vs. CPU time

(c) Uniform dist. vs. I/O cost (d) Anti-correlated dist. vs. I/O cost

Figure 7: Effect of update ratios: performance comparison among three approaches with various update ratios and data distributions ($P = 128k$, $d = 5$, $k = 1$).



(a) Uniform dist. vs. CPU time (b) Anti-correlated dist. vs. CPU time

(c) Uniform dist. vs. I/O cost (d) Anti-correlated dist. vs. I/O cost

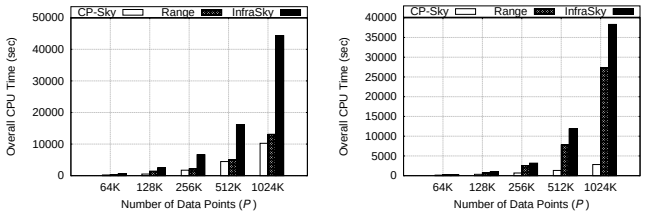
Figure 8: Effect of dimensionality: performance comparison among three approaches with various dimensionality and data distributions ($P = 128$k, $f_{update} = 3\%$, $k = 1$).

5.3. Effect of Dimensionality

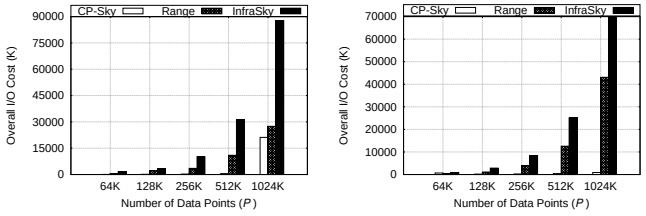
Next, we studied the impact of dimensionality on the performance of computing potential products. The data cardinality is fixed at 128,000, the update ratio at 3%, and k at 1. As shown in Figures 8(a)−(d), the *CP-Sky* approach achieves better performance in terms of CPU time and I/O cost. As the dimensionality increases, the performance of the range-based algorithm and *InfraSky* algorithm is degraded significantly, while the *CP-Sky* approach is not affected much. The gap between the *CP-Sky* approach and the other two approaches grows sharply. The reason is that fewer points can be placed on the same page and the *CP-Sky* approach accesses fewer pages. It is inferred that the *CP-Sky* approach can continuously handle the computation of potential products in a higher dimension.

5.4. Effect of Cardinality

The correlation of the performance and cardinality are reported with dimensionality fixed at 5, the update ratio at 3%, and k at 1. Figures 9(a)−(d) show the average CPU time and I/O cost with respect to different cardinalities. As the dataset size increases, the performance of all three algorithms deteriorates. The difference lies in the speed of performance degradation, with the *CP-Sky* approach achieving almost constant I/O cost and smaller CPU time growth. The reason is that the *CP-Sky* approach is affected only when the $S1$ or $S2$ sets are updated. It can be inferred that the *CP-Sky* approach is scalable among large-scale datasets.

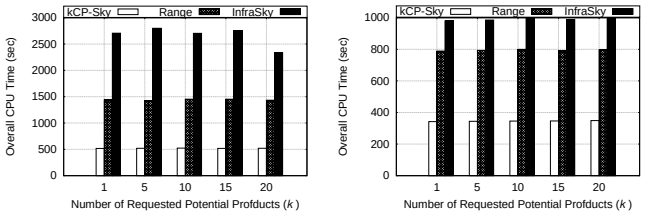


(a) Uniform dist. vs. CPU time (b) Anti-correlated dist. vs. CPU time

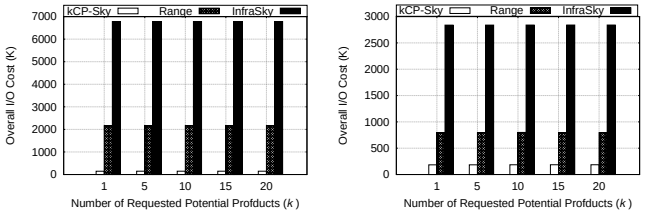


(c) Uniform dist. vs. I/O cost (d) Anti-correlated dist. vs. I/O cost

Figure 9: Effect of cardinality: performance comparison among three approaches with various cardinality and data distributions ($d = 5$, $f_{update} = 3\%$, $k = 1$).



(a) Uniform dist. vs. CPU time (b) Anti-correlated dist. vs. CPU time



(c) Uniform dist. vs. I/O cost (d) Anti-correlated dist. vs. I/O cost

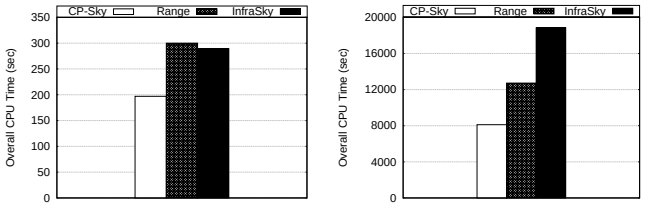
Figure 10: Effect of k value: performance comparison among three approaches with various k and data distributions ($P = 128k$, $d = 5$, $f_{update} = 3\%$).

5.5. Effect of k Value

Finally, we evaluate the effect of k. The data cardinality is fixed at 128,000, the dimensionality at 5, and the update ratio at 3%. We extend the three approaches to compute top-k potential products. In Figures 10(a)−(d), as k increases, we can see that the *kCP-Sky* algorithm remains stable in terms of average CPU time and I/O cost, because *kCP-Sky* performs at most k range queries at the $S2$ set regardless the size of $S2$. While the range-based, and *Infra* algorithms perform range queries at every point in $S1$ to guarantee to get top-k potential products. Therefore, the CPU and I/O performance are affected by the size of the $S1$ set. Hence, we can see that both range-based and *Infra* algorithms also have stable performances. However, because both algorithms perform more range queries, their CPU time and I/O cost are significantly higher than those of the *kCP-Sky* algorithm for uniform and anti-correlated data sets only.

5.6. Real-world Data Set Evaluation

In this section, we evaluate the *CP-Sky*, range-based, and *Infra* algorithms with two real-world data sets of US air quality and households. The air quality data was downloaded from the United States Environmental Protection Agency website (https://www3.epa.gov), which publishes air quality data of 288 counties in the United States in 2015. There are five dimensions (*i.e.*, air quality indicators) for each county, including $PM_{2.5}$, PM_{10}, SO_2, NO, and O_3, which change their values every day. As we can see in Figure 11(a), our *CP-Sky* algorithm outperforms the other two approaches in terms of CPU time. The range-based and *Infra* algorithms have worse, but not terrible, performance. The reason is that the cardinality of the real-world data set is quite small, and hence the methods incur no significant performance gap. The second real-world data set is a static data set containing 127,931 US households downloaded from http://www.ipums.org. In Figure 11(b), we can see that our *CP-Sky* algorithm outperforms the other two approaches, and the Infra algorithm has the worst performance. Therefore, we can conclude that *CP-Sky* is an efficient approach in practice.



(a) Air quality dataset (b) US household dataset

Figure 11: Experimental evaluation using real-world data sets.

6. Conclusions

In this paper, we formally define potential products and propose the *CP-Sky* algorithm by leveraging the *second-order skyline* set that facilitates efficient updates for continuous computations of potential products. Our *CP-Sky* algorithm achieves a faster response time and better overall CPU performance. With the adoption of the $S2$ sets, *CP-Sky* can efficiently update the potential products. An approximate exclusive dominance region (*AEDR*) is proposed, and our experiments confirm the feasibility of *AEDR* which has a low amortized cost of the exclusive data evaluation in high dimensional and dynamic data environments. The *S1Evaluation* procedure first examines all the incoming data requests and updates the $S1$ result if necessary, while the *S2Evaluation* procedure integrates our lemmas and heuristics to achieve a low CPU overhead and reduced I/O cost. Additionally, we further extend the *CP-Sky* algorithm to support the computations of top-k potential products. An extensive experiment on various data sets with different distributions indicates that our system outperforms the range-based and the *Infra*

skyline approaches and is especially well-suited for interactive applications that require a shorter response time.

7. References

[1] Ilaria Bartolini, Paolo Ciaccia, Vincent Oria, and M. Tamer Özsu. Flexible Integration of Multimedia Sub-queries with Qualitative Preferences. *Multimedia Tools and Applications*, 33(3):275–300, 2007.

[2] Ilaria Bartolini, Paolo Ciaccia, and Marco Patella. Efficient Sort-based Skyline Evaluation. *ACM Transactions on Database Systems (TODS)*, 33(4):133–135, 2008.

[3] Ilaria Bartolini, Paolo Ciaccia, and Marco Patella. The Skyline of a Probabilistic Relation. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 25(7):1656–1669, 2013.

[4] Ilaria Bartolini, Zhenjie Zhang, and Papadias. Collaborative Filtering with Personalized Skylines. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 23(2):190–203, 2011.

[5] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The Skyline Operator. In *Proceedings of the 17th IEEE International Conference on Data Engineering (ICDE)*, pages 421–430, 2001.

[6] Chee Yong Chan, Pin-Kwang Eng, and Kian-Lee Tan. Stratified Computation of Skylines with Partially-Ordered Domains. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 203–214, 2005.

[7] Volker Gaede and Oliver Günther. Multidimensional Access Methods. *ACM Computing Surveys (CSUR)*, 30(2):170–231, 1998.

[8] Zhenqiang Gong, Guangzhong Sun, Jing Yuan, and Yanjing Zhong. Efficient top-k query algorithms using K-skyband partition. In *Proceedings of the 4th International Conference on Scalable Information Systems (ICST)*, pages 288–305, 2009.

[9] Yu-Ling Hsueh, Roger Zimmermann, and Wei-Shinn Ku. Efficient updates for continuous skyline computations. In *Proceedings of the 19th International Conference on Database and Expert Systems Applications (DEXA)*, pages 419–433.

[10] Yu-Ling Hsueh, Roger Zimmermann, Wei-Shinn Ku, and Yifan Jin. Skyengine: Efficient skyline search engine for continuous skyline computations. In *Proceedings of the 27th IEEE International Conference on Data Engineering (ICDE)*, pages 1316–1319, 2011.

[11] Jin Huang, Bin Jiang, Jian Pei, Jian Chen, and Yong Tang. Skyline distance: a measure of multidimensional competence. *Knowledge and Information Systems*, 34(2):373–396, 2013.

[12] Zhiyong Huang, Hua Lu, Beng Chin Ooi, and Anthony K. H. Tung. Continuous Skyline Queries for Moving Objects. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 18(12):1645–1658, 2006.

[13] Su Min Jang, Choon Seo Park, and Jae Soo Yoo. Skyline Minimum Vector. In *Proceedings of the 12th International Asia-Pacific Web Conference (APWEB)*, pages 358–360, 2010.

[14] Youngdae Kim and You Seung Won Hwang. Escaping a Dominance Region at Minimum Cost. In *Proceedings of the 19th International Conference on Database and Expert Systems Applications (DEXA)*, pages 800–807. Springer, 2008.

[15] Donald Kossmann, Frank Ramsak, and Steffen Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB)*, pages 275–286, 2002.

[16] Ken C. K. Lee, Baihua Zheng, Huajing Li, and Wang-Chien Lee. Approaching the Skyline in Z Order. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB)*, pages 279–290, 2007.

[17] Spatial Index Library. http://www.research.att.com/ marioh/spatialindex/index.html.

[18] Xuemin Lin, Yidong Yuan, Wei Wang, and Hongjun Lu. Stabbing the Sky: Efficient Skyline Computation over Sliding Windows. In *Proceedings of the 21st IEEE International Conference on Data Engineering (ICDE)*, pages 502–513, 2005.

[19] Xuemin Lin, Yidong Yuan, Qing Zhang, and Ying Zhang. Selecting Stars: The k Most Representative Skyline Operator. In *Proceedings of the 23rd IEEE International Conference on Data Engineering (ICDE)*, pages 86–95, 2007.

[20] Hua Lu and Christian S. Jensen. Upgrading Uncompetitive Products Economically. In *Proceedings of the 28th IEEE International Conference on Data Engineering*, pages 977–988, 2012.

[21] Hua Lu, Christian S Jensen, and Zhenjie Zhang. Flexible and efficient resolution of skyline query size constraints. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 23(7):991–1005, 2011.

[22] Andrew Bruce McDonald. A Mobility-Based Framework for Adaptive Dynamic Cluster-Based Hybrid Routing in Wireless Ad-Hoc Networks. *Ph.D. Dissertation proposal, University of Pittsburgh.*, 1999.

[23] Xiaoye Miao, Yunjun Gao, Lu Chen, Gang Chen, Qing Li, and Tao Jiang. On Efficient k-Skyband Query Processing over Incomplete Data. In *Proceedings of the 18th International Conference on Database Systems for Advanced Applications (DASFAA)*, pages 424–439, 2013.

[24] Michael D. Morse, Jignesh M. Patel, and William I. Grosky. Efficient Continuous Skyline Computation. *Information Sciences*, 177(17):3411–3437, 2007.

[25] Michael D. Morse, Jignesh M. Patel, and H. V. Jagadish. Efficient Skyline Computation over Low-Cardinality Domains. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB)*, pages 267–278, 2007.

[26] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. An Optimal and Progressive Algorithm for Skyline Queries. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 467–478, 2003.

[27] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. Progressive Skyline Computation in Database Systems. *ACM Transactions on Database Systems (TODS)*, 30(1):41–82, 2005.

[28] Jian Pei, Wen Jin, Martin Ester, and Yufei Tao. Catching the Best Views of Skyline: A Semantic Approach Based on Decisive Subspaces. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB)*, pages 253–264, 2005.

[29] Zhuo Peng, Chaokun Wang, Lu Han, Jingchao Hao, and Yiyuan Bai. Discovering the Most Potential Stars in Social Networks. In *Proceeding of the third International Conference on Emerging Databases (EDB)*, 2011.

[30] Zhuo Peng, Chaokun Wang, Lu Han, Jingchao Hao, and Xiaoping Ou. Discovering the Most Potential Stars in Social Networks with Infra-skyline Queries. In *Web Technologies and Applications*, pages 134–145. Springer, 2012.

[31] Zhuo Peng, Chaokun Wang, Fangbo Tao, and Lu Han. SkyBoundary: An Improved Approach to Member Promotion in Social Networks. In *Proceedings of the 9th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC)*, pages 838–845, 2011.

[32] Dimitris Sacharidis, Stavros Papadopoulos, and Dimitris Papadias. Topologically Sorted Skylines for Partially Ordered Domains. In *Proceedings of the 25th IEEE International Conference on Data Engineering (ICDE)*, pages 1072–1083, 2009.

[33] Mehdi Sharifzadeh and Cyrus Shahabi. The Spatial Skyline Queries. In *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB)*, pages 751–762, 2006.

[34] Kian-Lee Tan, Pin-Kwang Eng, and Beng Chin Ooi. Efficient Progressive Skyline Computation. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB)*, pages 301–310, 2001.

[35] Li Tian, Le Wang, Peng Zou, Yan Jia, and Aiping Li. Continuous Monitoring of Skyline Query over Highly Dynamic Moving Objects. In *Proceedings of the 6th ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE)*, pages 59–66, 2007.

[36] George Trimponias, Ilaria Bartolini, Dimitris Papadias, and Yin Yang. Skyline Processing on Distributed Vertical Decompositions. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 25(4):850–862, 2013.

[37] Raymond Chi-Wing Wong, Ada Wai-Chee Fu, Jian Pei, Yip Sing Ho, Tai Wong, and Yubao Liu. Efficient Skyline Querying with Variable User Preferences on Nominal Attributes. *Proceedings of the Very Large Database Endowment (PVLDB)*, 1(1):1032–1043, 2008.

[38] Ping Wu, Divyakant Agrawal, Ömer Egecioglu, and Amr El Abbadi. Deltasky: Optimal Maintenance of Skyline Deletions without Exclusive Dominance Region Generation. In *Proceedings of the 23rd IEEE International Conference on Data Engineering (ICDE)*, pages 486–495, 2007.

[39] Shiming Zhang, Nikos Mamoulis, Ben Kao, and David Wai-Lok Cheung. Efficient Skyline Evaluation over Partially Ordered Domains. *Proceedings of the Very Large Database Endowment (PVLDB)*, 3(1):1255–1266, 2010.