Upgrading Products based on Existing Dominant Competitors

Heri Wijayanto
Department of Computer Science and Information Engineering, Asia University, No. 500, Lioufeng Rd., Wufeng, 41354, Taichung, Taiwan, ROC
Department of Informatics Engineering, Faculty of Engineering, University of Mataram, No. 62, Majapahit Rd., 83115, Mataram, Indonesia
heri@unram.ac.id

Syauki A. Thamrin
Department of Computer Science and Information Engineering, Asia University, No. 500, Lioufeng Rd., Wufeng, 41354, Taichung, Taiwan, ROC
syauki.aulia@outlook.com

Arbee L.P. Chen*
Department of Computer Science and Information Engineering, Asia University, No. 500, Lioufeng Rd., Wufeng, 41354, Taichung, Taiwan, ROC
arbee@asia.edu.tw

Abstract

In the Industry 4.0 era, manufacturers compete to produce better products that are expected to satisfy a larger number of customers. We propose a recommendation system for upgrading products considering user preferences. This approach is based on the dominating regions of dominant competitors. The dominating region represents the estimation of the number of potential customers. However, examining overlapped dominating regions for a high dimensional space is NP-hard. We propose a novel method named TDRDFS which constructs a Dominant Graph of Intersection skyline points (DGI) for modeling the dominating regions. Our experiments show that TDRDFS significantly reduces computation. Based on our approach, product vendors are able to determine the strategy of upgrading products easily.

1. Introduction

In a case, a factory needs to upgrade their product to increase its profit because the other factories produce better products. Then, the company may consider maximizing profit within the least budget. In industry 4.0, manufacturers tend to create new innovations to generate new customer preferences then the products become new trends in the market. Innovation can be made by polishing a feature that the other manufacturers do not take into consideration. For example, looking back to the year 2000, one mobile phone manufacturer produced a mobile phone with a music quality sound[1,2]. At this time, the other mobile phone companies do not take attention to the sound feature and they only provide standard sound for ringing and calling. Then, the mobile phone with music quality sound attracts customers and it became a new trend.

The products that are well accepted on the market can be said as dominating products. A product dominates the other because it is equal or better in all its attributes and it has at least one better attribute than the other. The concept of skyline operator for computing dominating data points has been proposed by Börzsönyi et al. [1], which was later extended to many variants [2], [3], [4], and [5]. A data set of smartphone products with two attributes (price and weight) is depicted in Fig. 1. The values of the two attributes have been normalized first. The skyline data points in Fig. 1 are p_1, p_2, and p_3 that are not dominated by others.

The work of upgrading products recommendation system attracts many researchers such as [6], [7], [8], [9], [10], and [11]. In general, those works assume that the customer preferences are fixed and the upgrading or choosing the profitable products are based on fixed customer preferences. However, in the real world, customer preferences are influenced by a new product that can open a new region in the competition. In this study, we propose a new approach based on the concept of dominating regions. The properties of dominating regions are investigated to provide a decision support system for upgrading products. It helps a company to discover a region that is the most profitable to upgrade a product. This region could be a new region for a competition that does not have dominated competitors or a region that has the least dominated competitors existed. The dominating region can be used to

[1] https://en.wikipedia.org/wiki/Samsung_SPH-M100

[2] https://en.wikipedia.org/wiki/List_of_best-selling_mobile_phones

URI: https://hdl.handle.net/10125/70823
978-0-9981331-4-0

represent the number of potential customers whose preferences are satisfied with the product represented by the associated data point. It is important to determine the dominating regions of a set of dominant competitor products to decide on upgrading products. Besides that, the least cost of upgrading is more preferable.

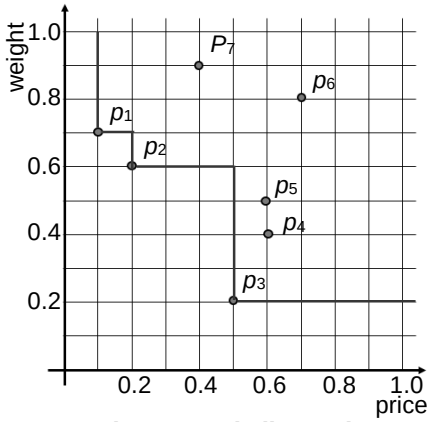

Figure 1. Skyline points

Fig. 2 shows the dominating regions of the four skyline points as the representation of dominant competitor products p_1, p_2, p_3, and p_4. In detail, the dominating region of p_1 is denoted by $A_1 \cup A_2 \cup A_3 \cup A_4$. We divide the dominating region of p_1 because those have different dominated points, for example, A_2 is the dominating region of both p_1 and p_2. It is called a shared dominating region. The probability of the customers purchasing a certain product associated with this dominating region can be computed. The computation of the shared dominating regions is NP-hard for high dimensional data [12].

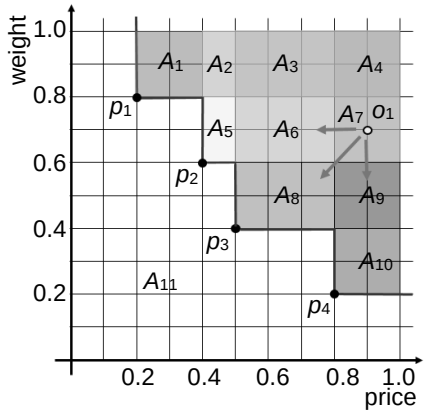

Figure 2. The dominating regions of the products

The computation of determining the dominating regions is very expensive. We show this expensive complexity by proof and experiments in this study by a naïve approach. Next, we propose an efficient method called *Top-Down Recursive Depth First Search* or *TDRDFS*. It can reduce the computation complexity that is shown in our experiments. Based on the *TDRDFS* approach, we can provide a recommendation system of upgrading a product. Our contributions in this research are summarized as follows:

- We introduce a novel upgrading product recommendation system that is based on dominating regions of a set of dominant products. It will help a company to make a better product in the current competition tendency that each company tries to find a new field for attracting the customers.
- We utilize a new type of domination graph to represent dominating regions. And, we proposed a novel method to construct the Domination Graph that is much more efficient than the naïve approach.
- Extensive experiments are provided to show the performance of our approaches.

The remaining parts of this article are organized as follows. Previous studies of the recommendation system of upgrading products are presented in Section 2. Our proposed methods are examined and explained in Section 3. Section 4 shows our experiments to evaluate the performance of the proposed method. The future works of this study and the conclusion are given in Section 5 and 6.

2. Related works

Skyline queries, which find a set of data points not dominated by others, are useful in many applications on multiple criteria decision-making. The skyline operator was first proposed in [1]; it then triggers lots of studies. Many methods were developed for reducing the computation cost of finding the skyline points, for example, the *Bitmap* method [13], the *Branch and Bound* (*BBS*) method [14], the *Nearest Neighbor* (*NN*) method [15], and the *Sort First Skyline* (*SFS*) method [16]. Furthermore, the skyline query is also extended into its variants such as dynamic skyline [2], [14], [17], [18], reverse skyline [3], [4], [19], and continuous skyline [17], [20], [5].

A product is demanding if it satisfies more customer preferences than the others. In other words, the better products dominate the larger number of user preferences. Many studies have been done on this purpose that focuses on finding potential customers and demanding products. This concept is introduced firstly by *k*-dominating queries [17]. It selects *k* skyline points that have the most number of dominating other points. Lin et al. [21], in 2007 show that selecting *k*-dominating skyline is NP-hard in three dimensions or more and they proposed a heuristic algorithm to solve it by probabilistic counting technique. Yin and Mamoulis published two articles in 2007 and 2009 to select *k*-points that has the largest number of dominating points [22], [23]. Those works are based on aggregate *R*-tree or *aR*-tree data structure. Studies in [17], [21], [22], and [23] find skyline points that dominate *k* largest dominating points. It is different from ours that our approach determines the expected number of points that could be dominated by the set of skyline points. In their approach, the points that are dominated are given. In contrast, our approach returns the dominating potential of all skyline points.

Customer preference can also be dominated by competitors. The dominance relationship analysis (*DRA*)

was introduced in [9] that utilize the skyline concept in [1], [17]. A product will be better if it dominates more customer preferences than the number of dominating customer preferences. In this approach, a data cube called *DADA* is constructed by performing lattice [24] to represent the dominant relationship between objects.

Zou et al., [10] proposed the *Pareto-based Dominant Graph* (*DG*) that is the extension of *DADA*. It is to improve the efficiency of the top-*k* query. The basic idea of this study is to implement the dominant relationship of data points. And next, the authors develop pruning strategies to be implemented on the query. The *DG* is constructed offline and it as the input of four traveler proposed algorithms. Besides that, it also provides insertion and deletion algorithms to maintain *DG*. This study is close to our methods to construct *DG* and it can be said that the *DG* in our study is a special case of Zou's *DG*. The other study that utilizes the dominant graph is done by [25] in 2014. It is to answer continuous top-k queries and the graph is called by *Close Dominant Graph* (*CDG*). Therefore, [9] and [10] are also different from our approach because it is the further study of [17] and uses a given customer preference dataset.

The study of upgrading products based on the skyline was done in 2012 by Lu et al [8]. Skyline points represent the competitive products in the market. We can refer to the skyline points to upgrade our products. The algorithms in this study utilize *R-tree* data structure and the upgrading cost is calculated by the distance function. The objective of this study is to upgrade products becoming skyline points [8]. However, in our approach, we upgrade products to maximize the dominating region in which an upgraded product need not be a skyline point. Therefore, our work and [8] are incomparable. Our focus in this study is to maximize the dominating region of skyline points for upgrading products.

In 2013, the other study of *DRA* is [11] that takes consideration of budget constraint because every vendor do not have an unlimited budget to upgrade their products. The budget constraint is represented by a plane that is constructed by a constraint function. A product is upgraded to the constraint plane and select the based position on that plane.

Our study is based on previous research, Lin et al., [6] in 2013 on finding the *k* most demanding products (called the *k-MDP* problem). It is an NP-hard problem for a dataset whose dimensions are three or larger. The proof of the NP-hardness is based on the topological relations of spatial objects [12]. The probability for each customer to purchase a given product has to be evaluated. This probability is calculated by customer preference data. However, in our study, we determine the probabilities based on dominating regions of evaluated data points that are very complex in high dimensions. Similar to previous works explained earlier, [11] and [6] also use given customer preferences to determine the upgrading product and it is also different from our proposed approach.

The other problem solved in the previous researches that are related to our study is on the finding of prospective customers [26] in 2018 and [27] in 2016. Yin et al., [26] perform reverse skyline query and similarly, Islam et al., in [27] utilize reverse skyline and dynamic skyline to find potential customers. Besides that, similar to *k-MDP* [6] on selecting potential products, Zou et al., [28] in 2019 propose algorithms to select the product combinations under the price promotion. It is based on heuristic algorithms that commonly implemented in knapsack problems because selecting potential products is also NP-hard.

3. Proposed methods

3.1. Preliminaries

Given a set of n products $P = \{p_1, ..., p_n\}$, assume each product p_j has m attributes, denoted $A = (a_1, \ldots, a_m)$, and the i^{th} attribute of p_j denoted $p_j.a_i$. if each attribute has a numeric value, the dataset can be depicted in a Euclidean space *ECS* as shown in Fig. 1 in which a point represents a product and a dimension symbolizes an attribute. Besides, all attribute values have been normalized first. If we further assume the attribute values are the lower the better, then the concept of *domination* can be defined as follows.

Definition 1. (Domination) A point p_i dominates p_j if $\forall a_k \in A$, $p_i.a_k \leqslant p_j.a_k$, and $\exists a_r$ in A, $p_i.a_r < p_j.a_r$. It is denoted $p_i \prec p_j$ and $p_i \not\prec p_j$ represents that p_i does not dominate p_j.

Definition 2. (Skyline) A skyline point is a point that is not dominated by any other points and the set of skyline points is denoted as $S=\{p_i | \forall j, j \neq i, p_j \not\prec p_i\}$.

Definition 3. (Dominating Region) A dominating region of a point (product) p_i is a subspace R of the Euclidean space *ECS* that any points (customer preferences) located on R are dominated by point p_i. It represents the *expected number of customers*. The minimum point in this subspace is p_i and the maximum point is denoted as p_{max}, where all attribute values are maximum in *ECS*. The dominating region of p_i is denoted by $R(p_i) = \langle p_i, p_{max} \rangle$. The cardinality of this subspace $|R(p_i)|$ can be calculated by Equation 1.

$$|R(p_i)| = |\langle p_i, p_{max} \rangle| = \prod_{j=1}^{m} (p_{max}.a_j - p_i.a_j) \quad (1)$$

where m is the number of attributes.

For example, in Fig. 2, the expected number of customers for product p_1 is 0.16 if there is only one product p_1 in *S*.

Definition 4. (Intersect) A set of point *Q* intersect if for any p_i and $p_j \in Q$, $p_i \not\prec p_j$ and $p_j \not\prec p_i$.

Definition 5. (Intersection Point) In a skyline data set *S* with *d* dimensions, an Intersection Point *IP* with coordinate $(ip_1, ip_2, \ldots, ip_d)$ of a subset $SS \subseteq S$ represents that $ip_i = max(p_{n,i} : p_n \in SS)$ where *i* is i^{th} dimension of *SS*. It is denoted as *IP*(*SS*).

For example, in Fig. 3.a and 3.b, *IP*(2,3,4) is an intersection point between point *p*(1,3,4) and *p*(2,2,4).

Definition 6. (Shared Dominating Region) Given a subset $P' \subseteq P$, $|P'| = j$, $j \geqslant 2$, *SR* is a subspace of *ECS* constructed by $SR(p_1,...,p_j) = R(p_1) \cap R(p_2) \cap \ldots \cap R(p_j)$. All points located in *SR* are dominated by each element in *P'*. The cardinality of the dominating region of $SR(P')$ =

$SR(p_1,...,p_j)$ is denoted by $|SR(P')|$ and can be calculated by Equation 2. The *expected number of customers* in $SR(P')$ is $|R(IP(P'))|$, which can be divided by $|P'|$ to get the expected number of customers for each product in P'. We assume that in a region, the market is shared equally to all products dominating this region.

$$|SR(P')| = |R(IP(P'))| \quad (2)$$

Definition 7. (Common Shared Dominating Region) A common shared dominating region of P is a shared dominating region for all points in P, denoted CSR. That is, $CSR = SR(P)$.

According to Definition 7, all points in P have the same value of the expected number of customers in CSR as A_4 shown in Fig. 2.

Definition 8. (Private Dominating Region) A private dominating region for p_i is denoted $PR(p_i)$, which is a subspace of $R(p_i)$ in which all data points are only dominated by p_i. The cardinality of dominating region of $PR(p_i)$ denoted by $|PR(p_i)|$ is $|R(p_i)|$ minus the cardinality of the union of all $SR(P')$ that p_i is an element of P' and $|P'| \geqslant 2$.

$$|PR(p_i)| = |R(p_i)| - |\cup_{P' \subseteq P, p_i \in P'} R(IP(P'))| \quad (3)$$

A customer will buy a product if it satisfies all the customer preferences. In other words, this product dominates the customer preferences. According to Definition 3 (Dominating Region) and assuming the customer preferences are distributed uniformly on the dominating region of a product, a larger dominating region implies a larger number of customers probably will buy this product. We call these customers potential customers. Because the number of potential customers is proportional to the area of the dominating region we call it Total Expected Number of Customers (*TEC*). Furthermore, because the dominating regions are calculated from product features all features are normalized by their respective maximum values to make them comparable.

The total expected number of customers of a product p_i is denoted by $TEC(p_i)$, which can be calculated by equation 4.

$$TEC(p_i) = \sum_{p_i \in P', P' \subseteq P} \frac{|PR(P')|}{|P'|} \quad (4)$$

From Fig. 2 and Equation 4, $TEC(p_1) = 0.04/1 + 0.02/2 + 0.06/3 + 0.04/4 = 0.08$.

If a company wants to upgrade a product to increase the total expected number of customers, it needs to consider the cost and profit for the upgrading. The upgrading cost is assumed to be proportional to the distance of moving a point p to p'. In this study, the Manhattan distance function is used. In the real world, the cost of upgrading each attribute of a product may not be the same. It can be represented as an upgrading cost vector $uc(a_1,\ldots,a_m)$, where m is the number of attributes. The total cost UC for upgrading a product p_i to p'_i can be formulated as Equation 5.

$$UC(p_i, p_i', uc) = \sum_{j=1}^{m} (|p_i.a_j - p_i'.a_j| * uc.a_j) \quad (5)$$

The *benefit* of upgrading is the difference of *TEC* before and after upgrading, $benefit(p_i', p_i) = TEC(p_i') - TEC(p_i)$. The profit of upgrading a product is *benefit* subtracted by upgrading cost *UC* as in Equation 6. However, in the real application, this cost function is adjustable according to company needs. If the company uses linear function, then the cost function can be adjusted by the weight of cost and benefit to make them compatible.

$$profit(p_i', p_i) = benefit(p_i', p_i) - UC(p_i, p_i', uc) \quad (6)$$

Problem Description. As shown in [6], [15], and [12], computation of upgrading products based on dominating relationships of competitors is NP-hard for high dimension datasets. Therefore, we develop an algorithm to address this problem as follows: Given a set of dominant competitors that is a set of skyline points S of a dataset D and a set of products for upgrading P. (1) How to calculate all *TEC* of S for upgrading products based on dominating regions. (2) By *TEC* and a given cost function *UC*, how to upgrade a product to maximize the profit

3.2. Properties of *DGI*

Calculating *TEC* that is explained in Subsection 3.4.1 is NP-hard in a dataset that the dimension is more than three. We utilize a *Dominant Graph of Intersections* (*DGI*) and propose a new method to construct it that reduces the computation complexity. We adopt this concept from [10] which is called by *Pareto-based Dominant Graph* (*DG*) and *DGI* is a special case of *DG*.

Fig. 3.a shows 3-dimensional skyline data points and those dominating regions that intersect each other. On the right hand, a dominating graph of skyline points depicted in Fig.3.a is presented. The leaf nodes of *DGI* that are located on the bottom are the skyline data points. In other words, it is the first layer of *DGI*. The next layers above are the intermediate layer that those are intersection points. In more than three dimensions, *DGI* is complex. For instance, Fig. 3.a and 3.b show the *DGI* in 3 dimensions that exist overlapping layer.

Lemma 1: Each skyline point intersects with other skyline points in a subset $SS \subseteq S$ where $|SS| \geq 2$.

Proof: By contradiction, if a point $p \in S$ does not intersect with q, $q \neq p$, it means $p \prec q$ or $q \prec p$ s.t. $p \notin S$ or $q \notin S$. because if $q \in S$ *and* $p \in S$, $\exists\ m,n \in d$, $p_m < q_m$ and $q_n < p_n$.□

Based on Lemma 1, a naïve approach given in Fig. 4 to construct *DGI* can be constructed. It starts from determining all subsets of a set of skyline points S and finds the intersection point of each subset. Next, we insert each point to *DGI* that the insert function puts a point in a correct layer and sets the parent-child relations to other points found previously following the dominating rule mentioned in Definition 1.

The naïve approach is not computable because it is $O(2^n)$ where n is the number of skyline points. It is for both time and space complexity because we need to construct and evaluate all subsets of the dataset in the naïve approach.

To improve the performance of naïve *DGI* construction, we observe some properties of this graph. Actually, [10] provides the insertion and the deletion procedures to construct *DG* that are equal to our methods in the naïve

approach. However, the calculation of intersection points is novel in this study. The number of intersection points in a set of skyline points with three or more dimensions cannot be predicted. Therefore, in the naïve approach, we evaluate all possible subsets to find intersection points. The total number of subset combinations of n skyline points is 2^n - (n+1).

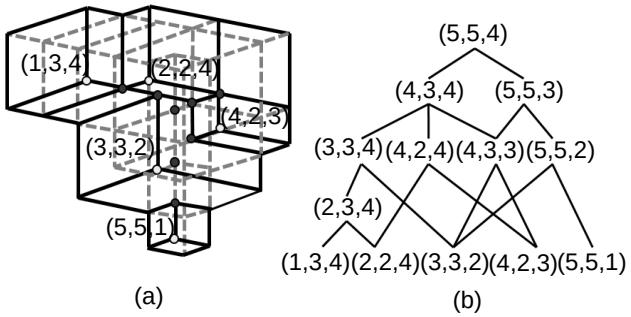

Figure 3. (a) 3D skyline points and those dominating regions. (b) Dominant graph of intersection points in 3.a

Definition 9. (Maximum Intersection Point) Maximum Intersection Point *maxIP* is an intersection point of *SS* where *SS* = *S* or *IP*(*S*).

Definition 10. (Minimum Intersection Point) Minimum Intersection Point *minIP* is an intersection point of *SS* where |*SS*| = 2.

The number of the minimum intersection point is uncertain because there is probably existed overlapped level in 3 or more dimensions of *DGI* shown in Fig. 3.b. The maximum number of minimum intersection points is n-1 with n is the number of skyline points.

Definition 11. (Extreme Skyline Points) A set of extreme skyline points ES_i, $ES_i \subseteq SS_i$, for a point $p_n \in ES_i$, p_n, $\exists$ a dimension k, s.t. $p_{n,k}$ is the maximum value in the k^{th} dimension in SS_i. For a point $p(a_1,a_2,...,a_d)$ if a_i is maximum in i^{th} dimension then p is an extreme point. $p \in ES$ where ES is a set of extreme points and $ES \subseteq SS$.

Definition 12. (Shared Dimension Set) A set of shared dimensions of an ES_i, named $SDES_i$, $SDES_i \subseteq ES_i$, $\exists$ p,q $\in$ $SDES_i$, if $\exists$ a dimension k s.t. $p[k] = q[k]$.

Lemma 2: Let $SS \subseteq S$ *and* $SS\text{-}p_i = SS'$, $p_i \in SS$, then $IP(SS)$ cannot dominate $IP(SS')$. $IP(SS')$ has only 2 conditions: $IP(SS') \prec IP(SS)$ or $IP(SS') = IP(SS)$.

Proof: It has only 3 cases: (1) If $p_i \notin ES \subseteq SS$ then $IP(SS')=IP(SS)$, because of $\exists$ $p_q \in ES$, $p_q \neq p_i$, and $\exists$ k^{th} dimension of p_q s.t. $pq_k > pi_k$. (2) If $p_i \in SDES \subseteq ES \subseteq SS$ then $IP(SS')=IP(SS)$, because of $\exists$ $p_q \in SDES$, $p_q \neq p_i$, and $\exists$ k^{th} dimension of p_q s.t. $p_{q,k} = p_{i,k}$.(3) If $p_i \in ES \subseteq SS$ and $p_i \notin SDES$ then $IP(SS') \prec IP(SS)$, because $\nexists$ $p_q \in$ SS, $p_q \neq p_i$, and $\exists$ k^{th} dimension of p_q s.t. $p_{q,k} \leq p_{i,k}$. And. $p_{i,k}$ is the maximum values in SS in k^{th} dimension and $SS' = SS\text{-}p_i$ then $IP(SS') \prec IP(SS)$. □

Based on Lemma 2, the intersection point of *SS* cannot dominate the intersection point that is formed from *SS'* and *SS'* ⊆ *SS*.

Function *naïveDGIconstruction*	
Input:	A set of skyline points *SS*
Output:	*DGI*
1.	*SS'* = Construct all subset of *SS*
2.	*IPset* = Defines the intersection point of each subset in *SS'*
3.	*member* = *IPset* ∪ *SS*
4.	*DGI* = Determine dominating relation of *member*
5.	**return** *DGI*

Figure 4. Algorithm of naïve *DGI* construction

3.3. Top-Down Recursive Depth First Search Algorithm (*TDRDFS*)

Our approach determines the intersection points layer by layer from the top of *DGI* to the bottom. As in Fig. 3b, the top layer only contains one intersection point that is called by the *root* or the maximum intersection point. The bottom layer contains all skyline data points. In each layer, the intersection points located in the same layer do not dominate each other. Furthermore, an intersection point in a level indicates the closeness of its children.

In general, the proposed approach to Construct *DGI* efficiently in this study can be presented as follows:

1. Start from a subset *SS* that contains all skyline points. Find intersection point *IP* in the top layer of the *DGI* and determine the extreme points in this subset. It is an $O(n)$ process that n is the number of skyline points in *SS*.
2. Constructs m number of subsets SS'_i based on the extreme points where m is the number of extreme skyline points and i is 1 to m.
3. For each subset SS'_i,
   a. If the element of the subset is 2 skyline points, return the intersection point of these 2 skyline points.
   b. Else push *s'* to stack, $SS = SS'_i$, and go to step 1.

The structure of a node in *DGI* represents a skyline point or an intersection point. The data of a node is a tuple of attributes in an intersection point such as (3,3,4) in *DGI* in Fig. 3.b. As the necessary of *DGI* computation, we construct an index for a node as (i, {(*del_indices*, *extreme_indices*)$_1$, …, (*del_indices*, *extreme_indices*)$_j$}). This index contains two-part, the first index is i that is an integer value to identify a point. The second index is a list of tuple (*del_indices*, *extreme_indices*). Each tuple contains *del_indices* and *extreme_indices*. The *del_indices* records the list of deleted extreme skyline points p_i in a subset of *SS* according to Lemma 2. The *extreme_indices* records the list of extreme points *ES* of *SS* in Definition 11. We need to use the list of tuples of (*del_indices*, *extreme_indices*) to anticipate the overlapped layer in *DGI* shown in Fig. 3.b. Besides, a node also has a list node of parents and a list node of children.

Because of the limited space, not all algorithms can be presented in this article. We explain the main algorithm proposed in this study as follows.

As inputs in the *DGI* construction algorithm in Fig. 5, *currentNode* is null and the *delSet* is ∅. This algorithm starts with finding the intersection point of the input *SS* by the

findMax function with $O(n)$ complexity. It is shown in Line 1, Fig. 5 that the output is a node called by *maxPoint*. We need *del_indices* in this step as one of the inputs because the output of the *findMax* function is a node then the function will create a new index of the output, especially the second index. This index is constructed by determining a set of indices of new *extreme_indices* found in this subset and by modifying the previous *del_indices*. In Line 2, by *getRoot* function, we take the root of the *currentNode*. The algorithm to find the root node is *getRoot* with an $O(\log\ n)$ computation. Then, we check that the *maxPoint* is already in the *DGI* or not by *getNode* function that is $O(n\log\ n)$ operation. If it is not on the *DGI* we set the child of this *currentNode* is *MaxPoint*. It means that if the *DGI* is empty then we determine that the *maxPoint* is as the root of our *DGI* with the parent of the root is *null*. It is done in Line 4 and 5 of the *DGI* construction algorithm in Fig. 5. If the *maxPoint* has already found in the *DGI*, it has 2 possibilities. The first is that the node has already been evaluated and the second is that the node has shared dimensions explained in Definition 12. This shared dimension causes that our *DGI* has an overlapped layer. The first possibility is handled by Line 7, 8, 9. In Line 7, if a node has been evaluated then the *del_indices* is equal to *del_indices* of *maxPoint* or *extreme_indices* of the node is equal to the *extreme_indices* of *maxPoint*. Next, in Line 8 and 9, we set the node as a child of *currentNode* and return. On the other hand, if there is a shared dimensions found in currentNode, then it executes Line 11, and 12. In Line 11, we add the *second_index* of *maxPoint* to the *second_index* of node *onRoot*. Then, we add *onRoot* as a child of *currentNode*. If the number of skyline points in *SS* is 2 then, those skyline points are added as children to the *currentNode* directly and return. It is done in Line 13, 14, and 15. However, if the number of skyline pints in SS is more than 2 then we execute Line 17, 18, 19, and 20. It generates each *SS'* from *SS* by each member of *extreme_indices* of *maxPoint*. It is done in Line 18 by $SS' = SS - p_i$, where $p_i \in maxPoint.extreme\_indices$. Next, in Line 19, the index of p_i is added to *del_indices* of *maxPoint* as *new_del_indices*. And in Line 20, recursively, it call *constructDGI* with the input are *maxPoint*, *SS'*, and *new_dell_indices*.

The next important algorithm of our approach is *setChildren* in Fig. 6 that is performed to insert new intersection points in the *DGI* in the correct position. The input of this function is the *currentNode* which is a node in *DGI* that is currently evaluating node, and the set of children of the *currentNode*. The new children cannot be inserted directly as the children of *currentNode* because in several cases, there is the same child of *currentNode* has already existed. The other case is that a new child is dominating or dominated by the existing child found earlier.

In Line 1, it processes one by one each new child from *setChildren* that is inserted to the *DGI* as a child of *currentNode*. In Line 2, the function anticipates that the *currentNode* is equal to the new child although this possibility has been handled in the *constructDGI* algorithm in Fig. 5.

```
Algorithm constructDGI
Input:   Node currentNode, a set of skyline points SS, a set of indices
         of deleted skyline points del_indices
Output:  Node outputNode (the root node)
1.   maxPoint = findMax(SS, del_indices)
2.   theRoot = getRoot(currentNode)
3.   onRoot = getNode(maxPoint, theRoot, null)
4.   if (onRoot = null)
5.       currentNode.setChildren(maxPoint)
6.   else
7.       if (onRoot.del_indices = maxPoint.del_indices) or
         (onRoot.extreme_indices = maxPoint. extreme_indices)
8.           currentNode.setChildren(onRoot)
9.           return currentNode
10.      else
11.          onRoot.addToSecondIndex(maxPoint.secondIndex)
12.          currentNode.setChildren(onRoot)
13.  if (|SS| = 2)
14.      maxPoint.setChildren(SS)
15.      return currentNode
16.  else
17.      for each pi ∈ the last of maxPoint.extreme_point_indices
18.          SS' = SS – pi
19.          new_dell_indices = dell_indices + pi.index
20.          maxPoint = constructDGI(maxPoint, SS', new_dell_indices)
21.      return currentNode
```

Figure 5. *DGI* construction algorithm

Line 3 and 4 add the new child directly as a child of *currentNode* if the *currentNode* does not have children in this time. In Line 5, the function only adds the new child that has not already been as a child of *currentNode*. Line 8 to 10 will put the new child *nc* if it dominates an existing child *c* then it recursively calls *setChildren*(c,{nc}) to insert *nc* as the child of *c*. In other hand, in Line 11 to 15, if *c* dominates *nc* then the function updates the relation of *currentNode* is the parent of *c* that becomes *currentNode* is the parent of *nc* and *nc* is the parent of *c*. Finally, in Line 16 and 17, if *nc* is incomparable to all *currentNode* existing children then *nc* is inserted as a new child of *currentNode* directly. In addition, in adding and removing children of a *currentNode* in the algorithm in Fig.6, we need to consider that this relationship is bidirectional.

If a vertex constructed by a subset is the same as other vertices constructed by other subsets then the complexity of the naïve approach can be reduced significantly as shown in our experiments. However, the number of distinct vertices in a *DGI* is uncertain, the exact complexity cannot be determined. If we estimate the complexity of *TDRDFS* by $O(2^{n/c})$ where n is the number of skyline points and $c > 1$, then c becomes smaller when the number of dimensions increases.

3.4. Upgrading products based on *DGI*

For instance, one manufacture wants to upgrade its product. First, the manufacture collects all competitors' products and selects the dominant competitors that are the skyline competitors. Next, from the skyline competitors, the manufacture constructs the *DGI* with the proposed approach. With the skyline competitors' *DGI*, the *TEC* can be calculated easily. The scenario of upgrading products is explained in this subsection.

```
Function setChildren
Input:  Node currentNode, set of Node as new children newChildren
Output: null
1.  for each nc_i ∈ newChildren
2.      if (currentNode ≠ nc_i)
3.          if (currentNode.children = ∅)
4.              add(currentNode.children, nc_i)
5.          else if (nc_i ∉ currentNode.children)
                //put c in the correct position level under currentNode
6.              boolean updated = false
7.              for each c_i ∈ currentNode.children
8.                  if (nc_i ≺ c_i)
9.                      updated = true
10.                     setChildren(c_i, {nc_i})
11.                 if (c_i ≺ nc_i)
12.                     updated = true
13.                     add(nc_i.children, c_i)
14.                     remove (currentNode.children, c_i)
15.                     add (currentNode.children, nc_i)
16.             if not updated
17.                 add(currentNode.children, nc_i)
```

Figure 6. Algorithm to insert a new node to *DGI*

In this subsection, we introduce our approach to upgrade a product based on the *DGI*. Actually, the whole of our approach complexity is in the *DGI* construction step and the utilization of *DGI* for upgrading a product is straightforward. Basically, it starts from calculating the dominating region in detail as explained in Section 1. The principle of this calculation is on the determining of private region *PR* introduced in Definition 8. Then, the other region can be computed recursively. Next, we can determine the *TEC* in each region. Those are presented in Subsection 3.4.1. Next, our scenario for upgrading a product is presented in Subsection 3.4.2.

3.4.1. Calculating Total Expected Number of Customers (*TEC*). Before we do all calculations, first we need to determine the maximum value of each dimension is determined. The algorithm to find *TEC* of a point p_i is straightforward and it is based on Equation 4 that we calculate all private regions of all points dominated by p_i. To find all points dominated by p_i, the *DGI* provides a convenience method that the dominated points of p_i are all the parents of p_i. Calculating the private region of p_i is provided in Equation 3. In general, it is also straightforward to calculate the private region of a p_i by subtracting the dominating region of p_i by all other private regions of points that are dominated by p_i.

The simple example of calculating *TEC* for 3 dimensions has been presented in Subsection 3.2 that is below Equation 4. We use Fig. 3 to show an example to make it more clear. It is a 3-dimensional dataset and this principle is applicable for higher dimensions. Let p_2 is (2,2,4) in Fig 3.b, for calculating the $PR(p_2)$ by *DGI* in Fig 3.b is as follows. First, we calculate the entire dominating region of p_2 that is noted by $R(p_2)$ that has been explained in Definition 3 and Equation 1. Next, we subtract $R(p_2)$ with all private regions of all its parents dominating regions. The parent dominating regions can be calculated recursively by the same method. We take all intersection points that are dominated by p_2 that all are the parents of p_2 in the *DGI*. Those are $IP(2,3,4)$, $IP(3,3,4)$, $IP(4,2,4)$, $IP(4,3,4)$, and $IP(5,5,4)$. Then, we calculate the private regions of all intersection points by Equation 3 recursively. Concurrently, we also perform Equation 4 to calculate the *TEC* of p_2.

3.4.2. Upgrading products based on *DGI*. Our scenario of upgrading products based on *TEC* is depicted in Fig. 7 that we provide a cost range of upgrading products in a region. The cost of upgrading has been explained in Equation 5. This computation is also straightforward because this recommendation is based on our computation to determine *DGI*.

Our product that needs to be upgraded is o_1 in Fig. 7. The upgrading product suggestions of o_1 are based on the *TEC* of regions that are dominating o_1. In Fig. 7, the possible regions of the upgrading are a whole part of A_8 and some parts of A_7, A_6, A_9, A_5, A_{10}, and A_{11}. We categorize the range upgrading recommendation into 4 types. The first is upgrading the same region with the product such as in A_7. The second is upgrading on the region that all part of the region dominates the product such as A_8. The third is upgrading in the region that is only dominated by the origin that is A_{11}. And the fourth, the target regions are only some parts that dominate the product and the product is not located on those regions such as A_5, A_6, A_7, and A_9.

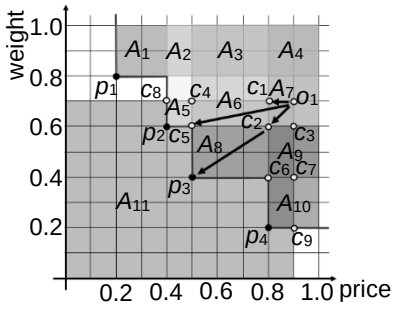

Figure 7. The scenario of upgrading a product

The first category of upgrading is to upgrade o_1 in its region that is A_7 in Fig. 7. We do not determine the minimum cost in this category because the zero cost upgrading means that we do not upgrade the product. We will find the maximum cost for this category if the product is upgraded to the intersection point c_2 that is the closest node in our *DGI* to our product o_1. In high dimensions, it is possible that we find several intersection points near to our product. Then, the maximum upgrading cost is determined by the farthest intersection point in the region.

In the second category, the minimum and maximum upgrading costs are determined by the farthest and the shortest intersection points in the region. For example, in Fig. 7, the target of upgrading region in this category is A_8 then, we got c_2 for minimum cost and p_3 for the maximum cost.

In the third category, the maximum upgrading cost is not determined because it means we upgrade our product to the origin that needs an unlimited budget. We only determine the minimum cost to upgrade a product to be skyline. In Fig. 7, we select the shortest distance from o_1 position to c_8, c_5,

c_6, and c_9. The c_5 and c_6 are the intersection points of skyline points. To determine c_8 and c_9 will be explained in the fourth category of upgrading.

The fourth category of this upgrading scenario is to upgrade a product to a region that not all part of the region dominates the product and the product is not located in those regions. Those are A_5, A_6, A_9, and A_{10} in Fig. 7 example. The determining of maximum cost is the same as the second category in this scenario. However, we need a straightforward technique to determine the minimum points in those regions. This technique is also used for determining critical points in the third category such as c_8 and c_9. This technique is to determine the projection points of our product to other regions. For example, we have our product o_1 and a point p_1 that dominates o_1 in 3-dimensional space. This example is similar to the A_7 situation in Fig. 7 for 2-dimensional space. Let the o_1 coordinate is (3, 4, 3) and p_1 is (2, 2, 2). When we substitute one by one the coordinate of o_1 by the attribute of p_1 then, we get (2, 4, 3), (3, 2, 3), and (3, 4, 2). Those are the projection points of o_1 to other regions.

4. Experiment results

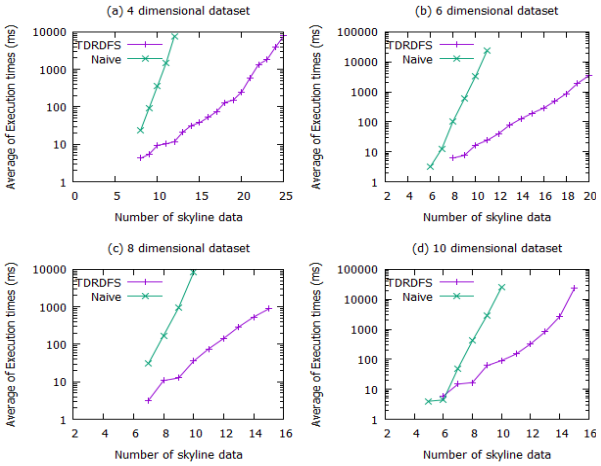

Figure 8. Construction time comparison of naïve and *TDRDFS DGI* in varying number of skyline points.

The performance of our approach is mainly influenced by the *DGI* construction algorithm that is called by *TDRDFS* algorithm. Then, we set the experiments to evaluate the algorithms as follows. First, we compare the construction time between *TDRDFS* and naïve approach in a varied number of skyline points input for 2 to 10 dimensions of synthetic datasets. The second experiment is to observe the execution time of DGI construction in a fixed number of skyline numbers for a varied number of dataset dimensions. Third, we observe the increasing number of nodes when the number of skyline points is increased. And fourth, we evaluate the number of nodes of *DGI* when the number of dimensions increases. All of the experiments are done in a personal computer 3GHz dual-core processors with 4GB RAM. Besides, we use 5 different synthetic skyline datasets and run multiple tests to present the average values in each experiment. Our proposed algorithm in this study is the further processing of skyline points in skyline query studies. Based on our previous study [29], the variation on data distributions is only effective on the processing of skyline queries. Naturally, the distribution of a set of skyline points is anti-correlated. Therefore, in this study we do not consider the data distributions of the dataset because the input of our approach is a set of skyline points.

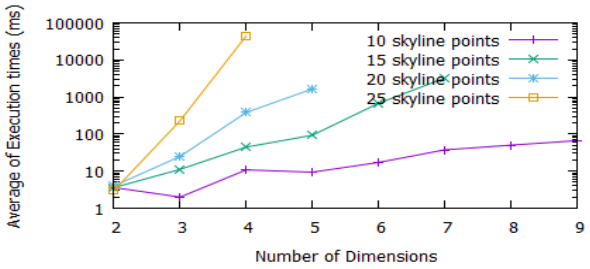

Figure 9. *DGI* construction time compared to the number of dimensions

Fig. 8.a, 8.b, 8.c, and 8.d show the execution time of *DGI* constructions for naïve algorithm and *TDRDFS* algorithm with varying number of skyline points. We use 2 to 10 dimensions of datasets and we observe starting from 2 skyline points. Then, we increase the number of skyline points until a reasonable time based on the computer machine or the capacity of memory used. Then, we have the trends of the execution time of the varied number of skyline points. In general, the naïve algorithm also has much sharper increase execution time than the *TDRDFS* shown in Fig 8.a to 8.d. And next, the naïve approach has got the "out of memory" problems when it executes the 4 dimensional 12 skyline dataset, 6 dimensional 11 skyline dataset, and 8 and 10-dimensional 10 skyline datasets as shown in Fig. 8.a to 8.d. However, we did not find the "out of memory" problem for *TDRDFS* in our experiments. We stop the execution of *TDRDFS* if the running time is more than around an hour. The *TDRDFS* outperforms the naïve approach because it can avoid $O(2^n)$ complexity of the naïve method. The naïve method needs to construct all possible subset of the dataset input to find the intersection points. On the other hand, the *TDRDFS* does not need to construct those subsets because it utilizes properties of *DGI* explained in Subsection 3.3. As a result, it can reduce the complexity of the naïve approach such that it can be implemented for upgrading products or other purposes.

The second experiment is to show the time response for the increasing number of dimensions. For this purpose, we use four fixed number of skyline data points and we increase the dimension gradually from 2 to 9 dimensions to observe the execution time. The number of skyline data points are 10, 15, 20, and 25. These results are shown in Fig. 9. In general, when the number of dimensions increases then the execution time also increases. Furthermore, a larger number of data has a significant increase. However, the increasing trend is shown in Fig. 9, which is not smooth because as mentioned earlier the number of intersection points cannot

be predicted. For example, for 10 skyline dataset, there is a small decrease when the number of dimensions is 3, because the number of intersection points is smaller than the previous results. Then, for the different datasets in the same size and the same number of dimensions may have a different number of intersection points. It is because there probably exist many shared dimension data points and the number of it is also depended on the input data. Besides that, the number of extreme points in a subset also takes effect on the execution time. It is because the number of extreme points of a subset will determine the number of children of the intersection point. Then in a high dimensional dataset, if the number of extreme points is small then the number of intersection points is also small and vice versa.

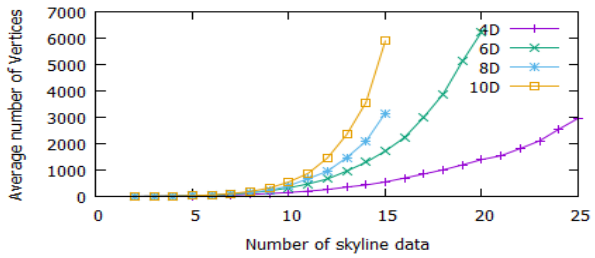

Figure 10. The number of intersection points compared to the number of skyline points

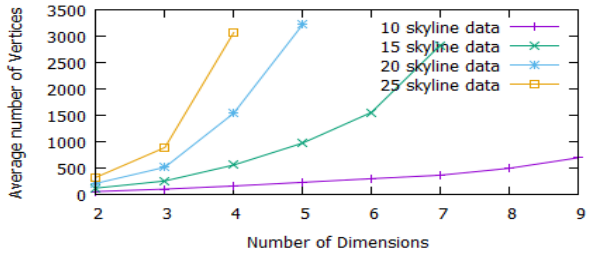

Figure 11. The number of intersection points compared to the number of dimensions

In Fig. 10, the third experiment is to know the relation between the number of input skyline data points with the number of intersection points. Based on the result of experiments, the number of intersection points increases exponentially when the number of input increases. It can be seen in Fig. 10 that those results also have slight fluctuation. Those are also caused by the extreme points and shared points in the datasets.

Fig. 11 shows the result of the experiment that observes the increasing number of intersection nodes in a varying number of dimensions. Similar to the experiment presented in Fig. 9, we use 10, 15, 20, and 25 number of skyline points. The experimental results show that the number of intersection points increases exponentially when the number of dimension increase. Furthermore, the larger number of skyline points has more significant increasing intersection points.

5. Future work and conclusions

5.1. Future work

The method to construct *DGI* for upgrading products in this study can be extended to many fields. First, our experiments of the *DGI* construction performance still have exponential increase when the number of skyline data and the number of dimensions are high. This problem is solved by performing parallel processing. For instance, we utilize *MapReduce* framework that is the well-known parallel computing framework today. It consists of two phases that the first phase is *Map* phase and the second is *Reduce* phase. To implement *TDRDFS* in *MapReduce* phases, the property written in Lemma 2 can be explored more. We can map the children of a node (intersection point) to several machines to recursively construct a subgraph below a child. Next, we combine all subgraphs in the *Reduce* phase.

Second, our algorithms to calculate the *TEC* is based on the *DGI*. We can make it more efficient by designing a heuristic function based on the *DGI*. As shown in our experiments that the number of intersection points increases significantly for high dimensional and large size of datasets.

Third, we can extend the algorithms mentioned in the related works for selecting *k* best product based on the calculation of *TEC*. As mentioned in our Related Works section, the previous approaches for product recommendation systems assume that the customer preferences are static. In the extension, we utilize *TEC* for anticipating the changing of customer preferences.

5.2. Conclusions

This article provides methods to analyze the advantages of dominant product competitors that are skyline points. Based on our observations of dominating regions of skyline points, we construct efficient algorithms to utilize the benefit of skyline points. The dominating region of a product represents the number of expected customers. The dominating regions of skyline points are overlapped each other. It is incomputable for high dimensions and a high number of input data. To make it computable, we utilized a *Dominant Graph* that in this article is called by *Dominant Graph of Intersection* points or *DGI*. The naïve method computation of constructing *DGI* is very expensive. Next, we developed an efficient algorithm to construct *DGI* that is called by *TDRDFS*. Therefore, the expected number of customers can be determined by using *DGI* for a recommendation of upgrading products. It is applicable for upgrading a product in Industry 4.0 that the customer preferences are changing when a new product is introduced to the market. This phenomenon motivates manufacturers to compete to create innovations to take over the market. Besides that, the proposed algorithms in this article are challenging for researchers to extend the performance and applications.

6. References

[1] S. Borzsony, D. Kossmann, and K. Stocker, “The Skyline operator,” *Proc. 17th Int. Conf. Data Eng.*, pp. 421–430, 2001.

[2] D. Sacharidis, P. Bouros, and T. Sellis, “Caching dynamic skyline queries,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5069 LNCS, pp. 455–472, 2008.

[3] E. Dellis and B. Seeger, “Efficient Computation of Reverse Skyline Queries,” *VLDB 07 Proc. 33rd Int. Conf. Very large data bases*, pp. 291–302, 2007.

[4] L. Xiang and L. Chen, “Monochromatic and bichromatic reverse skyline search over uncertain databases,” in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 2008, pp. 213–226.

[5] H. Z. Su, E. T. Wang, and A. L. P. Chen, “Continuous Probabilistic Skyline Queries over Uncertain Data Streams,” pp. 105–121, 2010.

[6] C. Y. Lin, J. L. Koh, and A. L. P. Chen, “Determining k-most demanding products with maximum expected number of total customers,” *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 8, pp. 1732–1747, 2013.

[7] Y. C. Chung, I. F. Su, C. Lee, and P. C. Huang, “Finding All Competitive Products Using the Dominant Relationship Analysis,” *Proc. - 2015 Int. Conf. Intell. Inf. Hiding Multimed. Signal Process. IIH-MSP 2015*, pp. 133–137, 2016.

[8] H. Lu and C. S. Jensen, “Upgrading uncompetitive products economically,” *Proc. - Int. Conf. Data Eng.*, pp. 977–988, 2012.

[9] C. Li, B. C. Ooi, A. K. H. Tung, and S. Wang, “DADA : A Data Cube for Dominant Relationship Analysis,” in *SIGMOD’06, June 26–29*, 2006.

[10] L. Zou and L. Chen, “Pareto-Based Dominant Graph : An Efficient Indexing Structure to Answer Top-K Queries,” *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 5, pp. 727–741, 2011.

[11] S. Ge, L. Hou U, N. Mamoulis, and D. W. L. Cheung, “Dominance Relationship Analysis with Budget Constraints,” *Knowl. Inf. Syst.*, 2013.

[12] X. Lin, Q. Liu, Y. Yuan, X. Zhou, and H. Lu, “Summarizing level-two topological relations in large spatial datasets,” *ACM Trans. Database Syst.*, vol. 31, no. 2, pp. 584–630, 2006.

[13] B. C. Tan, K.-L.; Eng, P.-K. Eng; Ooi, “Efficient progressive skyline computation,” in *VLDB*, 2001.

[14] D. Papadias, Y. Tao, G. Fu, and S. Bernhard, “An Optimal and Progressive Algorithm for Skyline Queries,” in *ACM SIGMOD*, 2003.

[15] D. Kossmann, F. Ramsak, and S. Rost, “Shooting Stars in the Sky: An Online Algorithm for Skyline Queries,” *{VLDB} 2002, Proc. 28th Int. Conf. Very Large Data Bases, August 20-23, 2002, Hong Kong, China*, pp. 275–286, 2002.

[16] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, “Skyline with Presorting,” in *ICDE*, 2003.

[17] D. Papadias, Y. Tao, G. Fu, and B. Seeger, “Progressive skyline computation in database systems,” *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 41–82, 2005.

[18] W. C. Wang, E. T. Wang, and A. L. P. Chen, “Dynamic skylines considering range queries,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6588 LNCS, no. PART 2, pp. 235–250, 2011.

[19] P. M. Deshpande and P. Deepak, “Efficient reverse skyline retrieval with arbitrary non-metric similarity measures,” *Proc. 14th Int. Conf. Extending Database Technol. - EDBT/ICDT ’11*, p. 319, 2011.

[20] Y. Tao and D. Papadias, “Maintaining Sliding Window Skylines on Data Streams,” *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 3, pp. 377–391, 2006.

[21] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang, “Selecting stars: The k most representative skyline operator,” *Proc. - Int. Conf. Data Eng.*, pp. 86–95, 2007.

[22] M. L. Yiu and N. Mamoulis, “Efficient processing of top-k dominating queries on multi-dimensional data,” *VLDB ’07 Proc. 33rd Int. Conf. Very large data bases*, pp. 483–494, 2007.

[23] M. L. Yiu and N. Mamoulis, “Multi-dimensional top- k dominating queries,” *VLDB J.*, vol. 18, pp. 695–718, 2009.

[24] G. Birkhoff, *Lattice Theory*. American Mathematical Society, 1948.

[25] B. J. Santoso, G. Chiu, and I. C. Society, “Close Dominance Graph : An Efficient Framework for Answering Continuous Top- k Dominating Queries,” *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 8, pp. 1853–1865, 2014.

[26] B. Yin, K. Gu, X. Wei, S. Zhou, and Y. Liu, “A cost-efficient framework for finding prospective customers based on reverse skyline queries,” *Knowledge-Based Syst.*, vol. 152, pp. 117–135, 2018.

[27] M. S. Islam and C. Liu, “Know your customer: computing k-most promising products for targeted marketing,” *VLDB J.*, vol. 25, no. 4, pp. 545–570, 2016.

[28] X. Zhou, K. Li, Z. Yang, and K. Li, “Finding Optimal Skyline Product Combinations under Price Promotion,” *IEEE Trans. Knowl. Data Eng.*, vol. 4347, no. c, pp. 1–14, 2019.

[29] H. Wijayanto, W. Wang, W.-S. Ku, and A. Chen, “LShape Partitioning: Parallel Skyline Query Processing using MapReduce,” *IEEE Trans. Knowl. Data Eng.*, 2020.