The VLDB Journal (2016) 25:545–570
DOI 10.1007/s00778-016-0428-3

REGULAR PAPER

Know your customer: computing k-most promising products for targeted marketing

Md. Saiful Islam[1] · **Chengfei Liu**[1]

Received: 11 August 2015 / Accepted: 31 March 2016 / Published online: 13 April 2016

Abstract The advancement of World Wide Web has revolutionized the way the manufacturers can do business. The manufacturers can collect customer preferences for products and product features from their sales and other product-related Web sites to enter and sustain in the global market. For example, the manufactures can make intelligent use of these customer preference data to decide on which products should be selected for targeted marketing. However, the selected products must attract as many customers as possible to increase the possibility of selling more than their respective competitors. This paper addresses this kind of product selection problem. That is, given a database of existing products P from the competitors, a set of company's own products Q, a dataset C of customer preferences and a positive integer k, we want to find k-most promising products (k-*MPP*) from Q with maximum expected number of total customers for targeted marketing. We model k-*MPP* query and propose an algorithmic framework for processing such query and its variants. Our framework utilizes grid-based data partitioning scheme and parallel computing techniques to realize k-*MPP* query. The effectiveness and efficiency of the framework are demonstrated by conducting extensive experiments with real and synthetic datasets.

Keywords Product selection · Dynamic skylines · Reverse skylines · Algorithms · Complexity analysis

✉ Md. Saiful Islam
mdsaifulislam@swin.edu.au

Chengfei Liu
cliu@swin.edu.au

[1] Swinburne University of Technology, Melbourne, Australia

1 Introduction

The competitive products are the alternative choices potential customers can decide to buy over any available product. The rampant use of World Wide Web for selling goods online allows the manufacturer to collect customer preferences for product features, e.g., search queries of online users and thereby, make intelligent use of these preference data to identify the competitive products as well as the potential buyers for them. The study of competitive products is crucially important for the manufacturers to sustain in the global market and has attracted considerable attention to the community ([1,6,9,13–15,20,25,27,28,30] for survey), e.g., the sale department can exploit this kind of study to find customer groups who are most likely to buy their products and also, to design specialized promotions, advertisement campaigns, coupons or similar promotions to expedite the sales of their products. In general, the promotion events are meant to increase the sales of the products and thereby, increase the overall revenue. However, several products might be interesting for the same customer and not all products contribute equally to attract customers in the market. Therefore, the manufacturers wish to identify only a subset of products that can possibly attract the highest number of customers in the market so that the advertising and other costs spread over a larger number of customers.

Consider another example from the job market area where a software firm wishes to hire some employees to fill up a few vacant positions. The firm can advertise these positions along with the required skills and employment benefits. However, the advertised positions need to compete with other job openings available in the market. Therefore, the firm wishes to attract as many candidates as possible so that the probabilities of getting more interviewees are increased and the firm can select some best employees by interviewing them. Here,

we can treat the firm as the manufacturer, job openings as products and the candidates as the customers.

The above product selection problem is defined as follows: given a set of products P from existing market, a set of own products Q called query products, a dataset of customer preferences C where $C \gg P \gg Q$, find k products from Q with maximum expected number of total customers. We term this as finding *k-most promising products* (*k-MPP*). The solution to this *k-MPP* product selection problem requires a *product adoption* model for the customers and a *product selection* strategy for the manufacturer. The *product adoption model* tells which products are competing with each other to attract a particular customer and how much a product contributes to attract a customer. On the contrary, the *product selection strategy* tells how to select products with maximum expected number of customers.

In this paper, we present a novel *product adoption model* and a *product selection strategy* based on dynamic skyline [2,18] and reverse skyline [4,12] queries. The *dynamic skyline* query is used to retrieve data (products) from customer's perspective, and *reverse skyline* query is used to retrieve data (customers) from manufacturer's perspective. Given a database of products P and a customer preference c as query, the *dynamic skyline* query for c retrieves all products $p_1 \in P$ that are preferable to c than other products $p_2 \in P$. That is, products p_1 match the preference of c better than other products p_2 in P. On the other hand, given a database of customer preferences C, products P and a query product p_1, the *reverse skyline* query retrieves all customers $c \in C$ that prefer p_1 than any other product $p_2 \in P$. In other words, the reverse skyline of p_1 consists of all customers c that include p_1 in their dynamic skylines. Both the dynamic skyline and reverse skyline queries follow the around-by semantics, i.e., absolute differences between the product attributes' values and the customer preferences in those attributes. These types of queries are studied in [29] and [1] for supporting market research queries. These works assume that a customer appearing in the product's reverse skyline contributes 100 % for the sustainment of the product in the market, i.e., the influence of a product is defined by its reverse skyline size. However, the dynamic skyline of a customer may consist of products from the competitors in the market, not only the company's own products. That is, a customer may have other products in her preference list and none of them is dominated by the manufacturer's own product (and vice versa). Therefore, the manufacturers cannot be firm about the adoption of the product by a customer; rather, they may associate a probability with it. Existing market research queries [1,29] disregard the above.

In our model, we associate a probability with the product adoption among customers based on dynamic dominance (i.e., around-by semantics) and skylines (i.e., preference-based queries). The models in [13] and [30] also associate a probability with the product adoption among customers. However, these models [13,30] assume that a product satisfies the requirements of a customer if the product attributes' values are less than or greater than the customer-specified preferences in those attributes. Therefore, these models fail to model the scenarios where a customer might not like to minimize or maximize certain quality metric of a product. For example, a customer may not like a too small or a too big screen for a laptop; rather, he/she may like a certain range for it. This kind of preferences can only be modeled appropriately in dynamic skyline and reverse skyline queries with around-by semantics, i.e., the absolute differences between product attributes' values and the customer preferences in those attributes.

We also present a novel *product selection strategy* for finding *k-most promising products* (*k-MPP*) with maximum expected number of customers for the manufacturers based on our product adoption model. The basic computational units of a *k-MPP* query are dynamic and reverse skylines. Though there are a number of established studies on dynamic [18] and reverse [1,4,5,19,29] skylines, none of them are efficient for processing *k-MPP* queries. Most of these works either rely on index structures that are query dependent or are not optimized for multiple units. In this paper, a query-independent grid-based data partitioning scheme is developed to process *k-MPP* queries by designing parallel algorithms for them. Our index scheme selectively stores one or more data objects as pivots from each partition to filter data objects belonging to a partition as early as possible while processing the basic computational units of a *k-MPP* query. Further, an approach is developed for grouping multiple queries and processing them together with expedite the efficiency.

The main contributions of this paper are as follows:

- We develop a novel probabilistic product adoption model among customers based on dynamic and reverse skylines.
- We develop a novel product selection strategy for finding the k-most promising products (*k-MPP* for short) with maximum expected number of customers.
- We present a parallel computing approach for processing *k-MPP* product selection query and its variants by designing a simple yet efficient query-independent grid-based data partitioning scheme.
- We also demonstrate the effectiveness and efficiency of our approach by conducting extensive experiments with both real and synthetic datasets.

The paper is organized as follows: Sect. 2 provides the necessary background and preliminaries; Sect. 3.1 presents our product adoption model; Sect. 3.2 describes the proposed product selection strategy, the *k-MPP* query and its vari-

ants; Sect. 4 analyzes the complexity of computing *k-MPP* queries; Sect. 5 describes our approach; Sect. 6 presents the experimental results; Sect. 7 describes the related work; and finally, Sect. 8 concludes the paper.

2 Background

Data model We consider each product $p \in P$, query product $q \in Q$ and customer (customer preference) $c \in C$ as a d-dimensional data object. Without any loss of generality, we assume that each data object stores only numeric values in its dimensions. The ith-dimensional values of a product, query product and customer are denoted by p^i, q^i and c^i, respectively. In general, we use ob to denote any data object from these three datasets, unless otherwise it is explicitly specified.

Definition 1 (*Dynamic dominance*) A data object ob_1 dynamically dominates another data object ob_2 w.r.t. a third data object ob_3, denoted by $ob_1 \prec_{ob3} ob_2$, if ob_1 is closer to ob_3's values in all dimensions and is strictly closer to ob_3's value in at least one dimension than ob_2. Mathematically, the relation $ob_1 \prec_{ob3} ob_2$ holds iff (a) $|ob_3^i - ob_1^i| \leq |ob_3^i - ob_2^i|, \forall i \in [1, ..., d]$ and (b) $|ob_3^i - ob_1^i| < |ob_3^i - ob_2^i|, \exists i \in [1, ..., d]$.

Example 1 Consider the dataset of products and customers given in Fig. 1. According to Definition 1, p_1 dynamically dominates p_3 w.r.t. c_1, i.e., $p_1 \prec_{c_1} p_3$, as $|c_1^1 - p_1^1| = |2 - 6| = 4 \leq |c_1^1 - p_3^1| = |2 - 6| = 4$ and $|c_1^2 - p_1^2| = |8 - 6| = 2 \leq |c_1^2 - p_3^2| = |8 - 20| = 12$.

From Definition 1, it is easy to verify that *dynamic dominance* relies on around-by semantics, which is also exemplified in Example 1. Dynamic dominance plays a very important role in modeling the customer preferences for a product in dynamic skyline [18] and reverse skyline [4] queries and studied extensively to establish the customer–product relationship [1,29].

ID	Dim1	Dim2
p_1	6	6
p_2	4	18
p_3	6	20
p_4	9	15
p_5	12	18
p_6	16	14
p_7	12	6
p_8	16	6
p_9	20	8
p_{10}	20	20

(a)

ID	Dim1	Dim2
c_1	2	8
c_2	4	10
c_3	6	16
c_4	8	18
c_5	10	10
c_6	16	14
c_7	12	2
c_8	18	6
c_9	18	18
c_{10}	20	13

(b)

Fig. 1 A dataset of products (P) and customers (C), **a** products in the market, **b** customer preferences

2.1 Dynamic skyline

Definition 2 (*Dynamic skyline* [18]) The dynamic skyline of a customer $c \in C$, denoted by *DSL(c)*, consists of all products $p_1 \in P$ that are not dynamically dominated by other products $p_2 \in P$ w.r.t. the customer c, i.e., $p_2 \not\prec_c p_1$.

Example 2 Consider the dataset given in Fig. 1. According to Definition 2, the dynamic skyline of customer c_5 consists of products p_4, p_7 and p_9 as these products are not dynamically dominated by other products in P w.r.t. c_5. Similarly, the dynamic skylines of c_3 and c_4 are $\{p_2, p_3, p_4\}$ and $\{p_2, p_3, p_4, p_5\}$, respectively.

The dynamic skyline $DSL(c)$ consists of all products $p_1 \in P$ that dynamically match the customer preference c better than any other product $p_2 \in P$. Therefore, the dynamic skyline query is used to retrieve products from customer's perspective or point of view. We say every product $p \in DSL(c)$ *competes* with each other for the customer c to enter into the market, e.g., products p_4, p_7 and p_9 compete with each other for c_5.

2.1.1 Dynamic skyline computation

A large number of dynamic skyline of a customer indicate more choices for the customer. The dynamic skyline of c can be computed by any traditional skyline computing algorithm [2] having all products $p \in P$ transformed to a new data space where point c is considered as the origin and the absolute distances to c are used as the mapping functions [18] as shown in Fig. 2. The *mapping function*, f^i, is defined as $f^i(p^i) = |c^i - p^i|$.

2.2 Reverse skyline

Definition 3 (*Reverse skyline* [4]) The reverse skyline of a product p_1, denoted by $RSL(p_1)$, consists of all customers $c \in C$ such that p_1 is not dynamically dominated by other products $p_2 \in P$ w.r.t. the customer c, i.e., $p_2 \not\prec_c p_1$. In other words, the reverse skyline of a product p_1 retrieves all customers $c \in C$ such that p_1 appears in the dynamic skylines of c.

Example 3 Consider the dataset of products and customers given in Fig. 1. The reverse skyline of product p_4 consists of customer c_3, c_4 and c_5 as each of them includes the product p_4 in their dynamic skylines. Similarly, the reverse skyline of p_3 and p_5 is $RSL(p_3) = \{c_3, c_4\}$ and $RSL(p_5) = \{c_4, c_9\}, respectively.$

We say that all customers appearing in $RSL(p)$ prefer the product p to others. Therefore, these customers are considered to be the *potential buyers* for the product p_1 in the

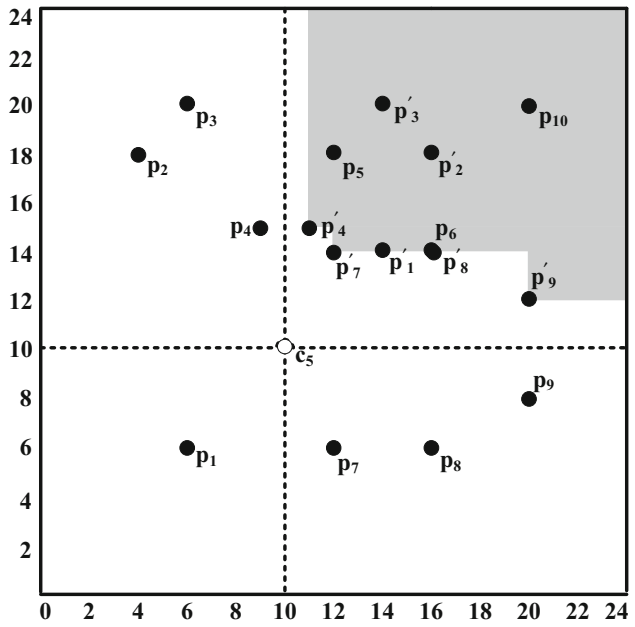


Fig. 2 Computing the dynamic skyline of the customer c_5

market [1], e.g., c_3, c_4 and c_5 are the potential buyers for the product p_5. A large number of reverse skyline indicate more customers for a product.

Definition 4 (*Influence set of a product* [1,29]) The influence set of a product $p \in P$, denoted by $IS(p)$, consists of the customers $c \in C$ that appear in the reverse skyline of p, i.e., $RSL(p)$. The *influence score* of p is measured by the size of the set, i.e., $|IS(p)|$.

Example 4 Consider the dataset of products and customers given in Fig. 1. The influence set of product p_4 consists of customer c_3, c_4 and c_5, i.e., $IS(p_4) = \{c_3, c_4, c_5\}$ and the *influence score* of p_4 is $|IS(p_4)| = 3$. Similarly, the influence score of p_3 and p_5 is $|IS(p_3)| = |\{c_3, c_4\}| = 2$ and $|IS(p_5)| = |\{c_4, c_9\}| = 2$, respectively.

2.2.1 Reverse skyline (influence set) computation

To compute the reverse skyline of a product p_1, firstly we need to compute the midpoint skyline (also known as *midskyline* [29]) for each orthant O of p_1. A product p_1 has 2^d orthants in a d-dimensional dataspace. The orthant of an object ob_2 with respect to an object ob_1 is calculated as follows: $O^i = 0$ if $ob_2^i \leq ob_1^i$, 1 otherwise . The midpoint for each product $p_2 \in P$ w.r.t. p_1 is calculated as $m_2^i = (p_1^i + p_2^i)/2$. Then, we need to compute the skyline of these midpoints for each orthant (as shown for p_4 in Fig. 3). Secondly, we need to check whether a customer c is dominated by these midpoint skylines w.r.t. p_1 or not. A customer c is dominated by a midskyline point m w.r.t. the product p_1 iff (a) $|p_1^i - m^i| \leq |p_1^i - c^i|, \forall i \in [1, ..., d]$ and (b)

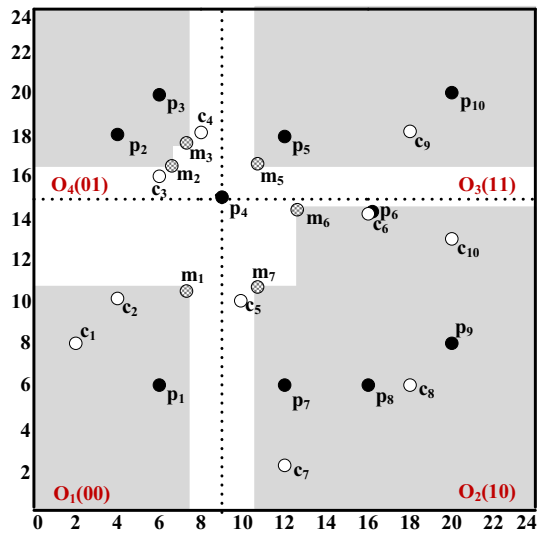


Fig. 3 Computing the reverse skyline of the product p_4

$|p_1^i - m^i| < |p_1^i - c^i|, \exists i \in [1, ..., d]$. If c is not dominated by any of the corresponding midpoint skylines w.r.t. p_1, then c appears in the reverse skyline of p_1. To know more about midpoint skyline and its role in reverse skyline computation, interested readers are referred to [29].

Table 1 shows the list of symbols used in the paper.

3 The proposed product adoption model and selection strategy

This section presents our product adoption model and the product selection strategy.

3.1 The proposed product adoption model

We propose a *uniform product adoption* (*UPA*) model based on dynamic and reverse skylines. In our model, we assume that every product $p \in P$ appearing in the dynamic skyline of the customer $c \in C$, i.e., $DSL(c)$, competes with each other to attract the customer c. Also, the customers $c \in C$ that appear in the reverse skyline of $p \in P$, i.e., $RSL(p)$, are the potential buyers for the product p. The *UPA* model is described below.

Definition 5 Given a set of existing products P in the market, the probability by which the customer will buy the product p, denoted by $Pr(c, p|P)$, is given as:

$$Pr(c, p|P) = \begin{cases} \frac{1}{|DSL(c)|} & \text{if } p \in DSL(c) \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

It is easy to verify that Eq. (1) does not show any biasness toward any particular product that appears in $DSL(c)$. That

Table 1 The list of symbols

Symbol	Meaning
P	A set of products in the market
C	A set of customers
Q	A set of company's own product
D	$P \cup C$
d	The number of dimensions in D
p	A product in P
c	A customer in C
q	A product in Q
ob	An object in D
$DSL(c)$	Dynamic skyline of c
$RSL(p)$	Reverse skyline of p
$Pr(c, p\|P)$	Probability by which c buy p
$E(C, p\|P)$	Market contribution of p given P
$E(C, P'\|P)$	Market contribution of the set P' given P
$\|IS(p)\|$	Influence score of p
$\|IS(P')\|$	Influence score of P'
k-MPP	The k-most promising products
k-MPP_{ind}	Independent k-MPP
k-MPP_{dep}	Dependent k-MPP
m	Number of threads (i.e., worker nodes)
n	Grid size
D_l	The lth partition of D
pos_l	The positional vector of D_l
δ_i	Side length of a partition in ith dimension
$loc(ob)$	Partition of D that contains ob
$\mathcal{N}_c(D)$	Search space of $DSL(c)$
$\mathcal{N}_q(D)$	Search space of $RSL(q)$
$\mathcal{N}^+{}_q(D)$	Extended search space of $RSL(q)$
$\mathcal{M}$	The midpoint skyline

is, every product in $DSL(c)$ has the equal chance of being selected by c and the products not appearing in $DSL(c)$ do not have any chance of being selected by c. Therefore, we get the following:

$$\sum_{\forall p \in P} Pr(c, p|P) = 1 \tag{2}$$

Under *UPA* model, a customer c is considered to be a *valued customer* for a product p if the customer c contains the product p in its dynamic skyline exclusively. It is assumed that the manufacturers like to increase the number of *valued customers* for their products.

Example 5 Consider the dataset given in Fig. 1. The probability of p_4 being chosen by the customer c_5 is $Pr(c_5, p_4|P) = \frac{1}{|DSL(c_5)|} = \frac{1}{3}$ as $|DSL(c_5)| = 3$ (from Example 2). Similarly, the probabilities of p_4 being chosen by customers c_3 and c_4 are $\frac{1}{3}$ and $\frac{1}{4}$, respectively.

3.1.1 Market contribution

The *market contribution* of a product $p \in P$ is measured by the expected number of total customers in C that might choose to buy the product p over other products in P. We assume that a customer would be equally interested in each product that appears in her dynamic skyline. Thus, the customer assigns an equal weight to the product that is inversely proportional to the size of the dynamic skyline as described in Sect. 3.1. Finally, the contribution of a product becomes the sum of the weights it receives from all the customers in the market.

Definition 6 Given a set of products P and a set of customers C, the market contribution of a product p, denoted by $E(C, p|P)$, is obtained by adding the probabilities $Pr(c, p|P)$ of each customer c in C as follows:

$$E(C, p|P) = \sum_{\forall c \in C} Pr(c, p|P) \tag{3}$$

As the customers c appearing in the reverse skyline of p are the only potential buyers for p (as per Definition 3) and the probabilities of p being chosen by customers that do not contain p in their dynamic skylines are zero (as per Eq. 1), Eq. 3 reduces to the following:

$$E(C, p|P) = \sum_{\forall c \in RSL(p)} Pr(c, p|P) \tag{4}$$

Example 6 Consider the dataset given in Fig. 1. The market contribution of p_4 is $E(C, p_4|P) = Pr(c_3, p_4|P) + Pr(c_4, p_4|P) + Pr(c_5, p_4|P) = \frac{1}{3} + \frac{1}{4} + \frac{1}{3} = \frac{11}{12}$. Similarly, the market contribution of p_3 and p_5 is $E(C, p_3|P) = Pr(c_3, p_3|P) + Pr(c_4, p_3|P) = \frac{1}{3} + \frac{1}{4} = \frac{7}{12}$ and $E(C, p_5|P) = Pr(c_4, p_5|P) + Pr(c_9, p_5|P) = \frac{1}{4} + \frac{1}{2} = \frac{3}{4}$, respectively.

Definition 7 Given a set of products P and a set of customers C, the market contribution of a product set P', denoted by $E(C, P'|P)$, is obtained by adding the market contribution of each product $p \in P'$ as follows:

$$E(C, P'|P) = \sum_{\forall p \in P'} E(C, p|P) \tag{5}$$

Example 7 Consider the dataset of products and customers given in Fig. 1. Assume that P' is a product set consisting of p_3, p_4 and p_5. The market contribution of the product set P' is $E(C, P'|P) = E(C, p_3|P) + E(C, p_4|P) + E(C, p_5|P) = \frac{7}{12} + \frac{11}{12} + \frac{3}{4} = 2.25$.

Theorem 1 *The market contribution of a product set $P' \subseteq P$ is bounded as follows:* $0 \leq E(C, P'|P) \leq |C|$.

Proof By putting Eq. 3 into the formula of $E(C, P'|P)$ given in Eq. 5, we get the following:

$$E(C, P'|P) = \sum_{\forall c \in C} \sum_{\forall p \in P'} Pr(c, p|P) \tag{6}$$

Now, assume that $P' = P$; then, all customers $c \in C$ are the potential buyers for the products p in P'. From Eq. 2, we get $\sum_{\forall p \in P'} Pr(c, p|P) = 1$ as $P' = P$. Therefore, Eq. 6 becomes as given as follows:

$$E(C, P'|P) \quad = \sum_{c \in C} 1 = |C|.$$

Again, assume that $P' \subset P$ and $\exists c_1 \in C$ that includes at least a product $p_1 \in P \setminus P'$ in its dynamic skyline, i.e., $p_1 \in DSL(c_1)$. Then, Eq. 2 becomes:

$$\sum_{\forall p \in P'} Pr(c_1, p|P) + \sum_{\forall p_1 \in P \setminus P'} Pr(c_1, p_1|P) \quad = 1.$$

We know that $\sum_{\forall p_1 \in P \setminus P'} Pr(c_1, p_1|P) > 0$ as $p_1 \in DSL(c_1)$, which gives us the following:

$$\sum_{\forall p \in P'} Pr(c_1, p|P) < 1.$$

By putting the above into Eq. 6, we get:

$$E(C, P'|P) < \sum_{c \in C} 1 = |C|.$$

Finally, assume that $P' \subset P$ and every product $p \in P'$ is dynamically dominated by a product $p_1 \in P \setminus P'$ w.r.t. a $c \in C$. Then, Eq. 2 becomes $\sum_{\forall p \in P'} Pr(c, p|P) = 0$. Therefore, Eq. 6 becomes as given as follows:

$$E(C, P'|P) = \sum_{c \in C} 0 = 0.$$

Hence, the theorem. □

3.1.2 Market contribution versus influence score

Definition 8 (*Influence score of a product set* [1]) The influence set of a product set P', denoted by $IS(P')$, consists of all customers $c \in C$ that appear in the reverse skyline of $p \in P'$, i.e., $IS(P') = \bigcup_{\forall p \in P'} IS(p)$. The influence score of P' is measured by the size of the set $IS(P')$ and is denoted by $|IS(P')|$.

Example 8 Consider the dataset given in Fig. 1. The influence set of the product set $P' = \{p_3, p_4, p_5\}$ consists of customers c_3, c_4, c_5 and c_9, i.e., $IS(P') = \{c_3, c_4, c_5, c_9\}$. Therefore, the influence score of P' is $|IS(P')| = 4$.

There is a clear distinction between the *market contribution* proposed in this paper and the *influence score* [1,29]. We argue that the *market contribution* metric is more realistic than the *influence score* for measuring the product sustainment in the market. For example, if we consider the *influence score* metric to judge the product sustainment in the market, then both product p_3 and p_5 are the same (see Example 4). However, if we consider the *market contribution*, then p_5 is preferable to the manufacturer than p_3 as the expected number of customers of p_5 is greater than that of p_3. The *influence score* metric [1] assumes that the number of actual customers of a product set is equivalent to its influence set size, which is an overestimate of the expected number of customers in the market. The *market contribution* metric measures the expected number of customers probabilistically by taking into account all plausible choices of a customer, i.e., the *market contribution* combines both the customers' perspective and the manufacturers' perspective into the metric. The existing *influence score*[1] considers the manufacturers' perspective only.

3.2 The proposed product selection strategy

We propose a novel *product selection strategy* for the manufacturers based on the *UPA* model developed in Sect. 3.1. The new query is called k-most promising products (k-*MPP*) selection query as given below.

Definition 9 (*Generalized k-**MPP** query*) Given a set of existing products P, a set of own products Q, a set of customers C and a positive integer k less than $|Q|$, the k-*MPP* query, denoted by k-$MPP(Q, P, C)$, selects a subset of k products Q' from Q which has the market contribution greater than any other subset of k products Q'' of the product set Q.

The above presents a generalized product selection query, which can be specialized. We present two variants of it based on the competition among the own products: (a) independent and (b) dependent k-*MPP*.

3.2.1 Independent k-MPP

We define a k-*MPP* query as independent k-*MPP*, denoted by k-MPP_{ind}, where the query products in Q do not compete with each other. That is, products in Q are considered independently while computing their contribution in the market. For example, the competitors of q_1 w.r.t. customer c_2 is shown in Fig. 4a, where q_1 does not compete with any other

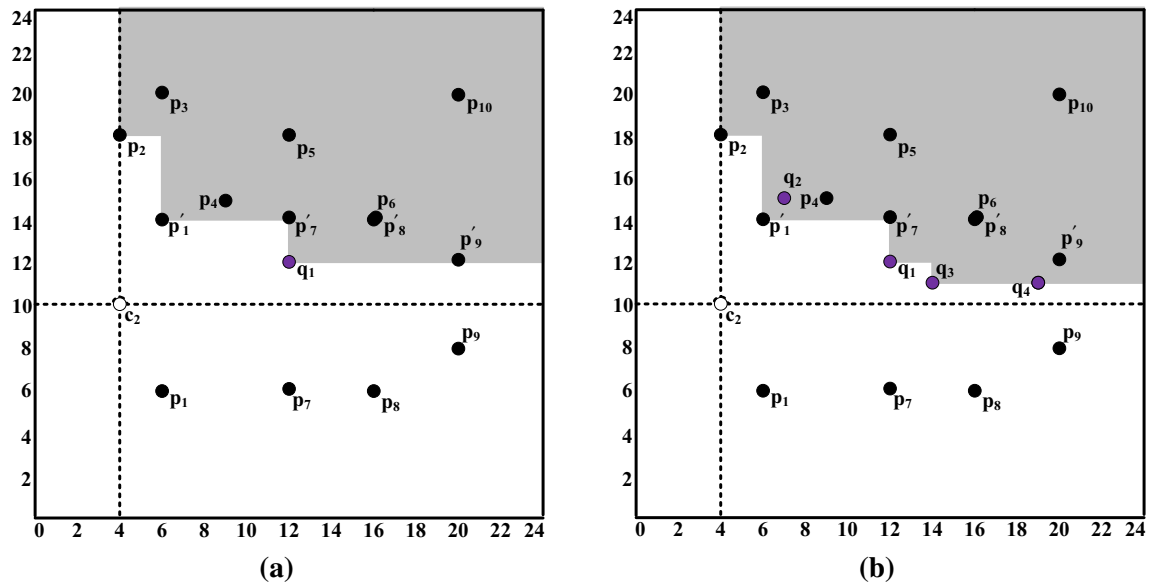


Fig. 4 Competitors of the product q_1 in **a** independent and **b** dependent k-**MPP** query settings w.r.t. customer c_2

products from Q, but products from their competitors only, i.e., P. This kind of *k-MPP* query is suitable in scenarios where a manufacturer considers to offer variants of the same type of product among the customers and likes to expedite the sales via market segmentation; or a job applicant is wished to be interviewed by each sub-team in a company if she satisfies the requirements of each of them.

Definition 10 (*k-MPP$_{ind}$ query*) Given Q, P and C, a $k-MPP_{ind}$ query is realized as:

$$k\text{-}MPP_{ind}(Q, P, C) = \underset{Q' \subseteq Q, |Q'|=k}{\text{argmax}} \; E(C, Q'|P) \tag{7}$$

Example 9 Consider the dataset of existing products and customers given in Fig. 1. Assume that we are given four query products $Q : \{q_1(12, 12), q_2(7, 15), q_3(14, 11), q_4(19, 11)\}$ and the manufacturer wishes to retrieve top 3-products from Q with the maximum number of total customers. The market contribution of q_1 is $E(C, q_1|P) = Pr(c_2, q_1|P) + Pr(c_5, q_1|P) = \frac{1}{2} + \frac{1}{3} = \frac{5}{6}$. Similarly, we get $E(C, q_2|P) = Pr(c_3, q_2|P) + Pr(c_4, q_2|P) = \frac{1}{2} + \frac{1}{5} = \frac{7}{10}$, $E(C, q_3|P) = Pr(c_2, q_3|P) + Pr(c_5, q_3|P) = \frac{1}{3} + \frac{1}{3} = \frac{2}{3}$ and $E(C, q_4|P) = Pr(c_2, q_4|P) + Pr(c_5, q_4|P) + Pr(c_8, q_4|P) + Pr(c_9, q_4|P) + Pr(c_{10}, q_4|P) = \frac{1}{3} + \frac{1}{3} + \frac{1}{2} + \frac{1}{3} + \frac{1}{3} = \frac{11}{6}$. In $k - MPP_{ind}$, we chose q_1, q_2 and q_4 as $E(C, \{q_1, q_2, q_4\}|P) = \frac{5}{6} + \frac{7}{10} + \frac{11}{6} = 3.37$ is the highest than any other 3-query products in the set Q.

3.2.2 Dependent k-MPP

We define a *k-MPP* query as dependent *k-MPP*, denoted by *k-MPP$_{dep}$*, where we allow the query products in Q to compete with each other. That is, we compute the market contribution of q considering the products not only from P, but also from Q as q's competitors. For example, the competitors of q_1 w.r.t. customer c_2 is shown in Fig. 4b where q_1 competes with the products from P as well products from the query product set Q. This kind of *k-MPP* query is useful for scenarios where a manufacturer wishes to judge the sustainment of a new product which is a variant of an existing product; or a job applicant is wished to be interviewed by only one sub-team in a company.

Definition 11 (*k-MPP$_{dep}$ query*) Given Q, P and C, a *k-MPP$_{dep}$* query is realized as follows:

$$k\text{-}MPP_{dep}(Q, P, C) = \underset{Q' \subseteq Q, |Q'|=k}{\text{argmax}} \; E(C, Q'|P \cup Q) \tag{8}$$

Example 10 Consider the dataset of existing products and customers given in Fig. 1. Assume that we are given four query products $Q : \{q_1(12, 12), q_2(7, 15), q_3(14, 11), q_4(19, 11)\}$ and the manufacturer wishes to retrieve top 3-products from Q with the maximum number of total customers. The market contribution of q_1 is $E(C, q_1|P \cup Q) = Pr(c_2, q_1|P \cup Q) + Pr(c_5, q_1|P \cup Q) = \frac{1}{4} + \frac{1}{3} = \frac{7}{12}$. Similarly, we get $E(C, q_2|P \cup Q) = Pr(c_3, q_2|P \cup Q) + Pr(c_4, q_2|P \cup Q) = \frac{1}{2} + \frac{1}{5} = \frac{7}{10}$, $E(C, q_3|P \cup Q) = Pr(c_2, q_3|P \cup Q) + Pr(c_5, q_3|P \cup Q) = \frac{1}{4} + \frac{1}{3} = \frac{7}{12}$ and $E(C, q_4|P \cup Q) = Pr(c_8, q_4|P \cup Q) + Pr(c_9, q_4|P \cup Q) + Pr(c_{10}, q_4|P \cup Q) = \frac{1}{2} + \frac{1}{3} + \frac{1}{3} = \frac{7}{6}$. In $k - MPP_{dep}$, we chose q_2, q_3 and q_4 as $E(C, \{q_2, q_3, q_4\}|P \cup Q) = \frac{7}{10} + \frac{7}{12} + \frac{7}{6} = 2.45$ is the highest than any other 3-query products in the set Q.

3.3 Related product selection models

There are a number of related product selection models exist in the literature [1,13,29] and [30]. Given a dataset of products and customers, Wu et al. [29] propose a new algorithm called *BRS* to compute the *influence set* (IS) of a given query product q, which outperform the branch and bound algorithm proposed by Dellis et al.[4]. Later, Arvanitis et al. [1] extend this to select a subset of query products Q' from a given product set Q that jointly maximize the size of the influence set, i.e., $|IS(Q')|$. The products in this set are termed as *most attractive candidates* (MAC) and if the size of Q' is k, then this is called the *k-MAC* query.

Theorem 2 *Assume that Q' is the set of k products selected by the k-MAC query and Q'' is the set of k products selected by the k-MPP query optimally from Q. We get the following: $|IS(Q')| \geq |IS(Q'')|$ but $E(C, Q'|P \cup Q) \leq E(C, Q''|P \cup Q)$ (also $E(C, Q'|P) \leq E(C, Q''|P)$).*

Proof Assume that there are no existing products in the market, i.e., $P = \emptyset$. Then, every customer $c \in C$ prefers one or more products only from the product set Q. Consider a special case where $|Q| = 1$ and also the value of k is 1. We get $Q' = Q'', |IS(Q')| = |IS(Q'')| = |C|$ and $E(C, Q'|P \cup Q) = E(C, Q''|P \cup Q) = |C|$ (also $E(C, Q'|P) = E(C, Q''|P) = |C|$). Consider another case where there are three query products q_1, q_2 and q_3 in Q with $RSL(q_1) = C_1 \cup C_2$, $RSL(q_2) = C_1$ and $RSL(q_3) = C_3$. We assume that the following relationships hold: (a) $C = C_1 \cup C_2 \cup C_3$ and (b) C_1, C_2 and C_3 are mutually disjoint. We get the followings:

(i) $E(C, q_1|P \cup Q) = \frac{|C_1|}{2} + |C_2|$;
(ii) $E(C, q_2|P \cup Q) = \frac{|C_1|}{2}$;
(iii) $E(C, q_3|P \cup Q) = |C_3|$;
(iv) $IS(\{q_1, q_3\}) = C_1 \cup C_2 \cup C_3$ and
(v) $IS(\{q_1, q_2\}) = C_1 \cup C_2$.

Now, if the value of k is 2, then the optimal *k-MAC* query selects q_1 and q_3 as Q' from Q as $|IS(\{q_1, q_3\})| = |C_1| + |C_2| + |C_3| > |IS(\{q_1, q_2\})| = |C_1| + |C_2|$. However, the optimal *k-MPP* selects q_1 and q_2 as Q'' as $E(C, Q''|P \cup Q) = |C_1| + |C_2| > E(C, Q'|P \cup Q) = \frac{|C_1|}{2} + |C_2| + |C_3|$ for $|C_1| > 2 \times |C_3|$. Similarly, we can prove $E(C, Q''|P) > E(C, Q''|P)$. Hence, the theorem. □

Given a set of products and customers, Lin et al. [13] propose a product selection model called *discovering k-most demanding products*, denoted by *k-MDP*, where a product p is assumed to satisfy a customer c iff $p^i \leq c^i$. This work also maximizes the number of expected customers for a subset of products of a given set of candidate products Q as we do in our model. Xu et al. [30] propose two types of product selections models, called *k-best selling products* (*k-BSP*) and (b) *k-best-benefit products* (*k-BBP*), that increase the expected sales in the market. This work also develops a probabilistic product adoption model by assuming that a product satisfies a customer iff $p^i \geq c^i$. However, the product adoption models of these works [13,30] are not realistic in the sense that customers do not always wish to minimize/maximize the attributes of a product, rather they may prefer certain ranges in them. These attributes can also be modeled in our approach by setting the customer preferences to their MIN/MAXes. Therefore, we argue that our product adoption model is more robust compared to [13] and [30] as we model product adoption among the customers through around-by semantics (i.e., dynamic dominance and skyline) and more sustainable compared to [1,29] as we model product selection by considering both the customer and the manufacturer perspectives (recall from Sect. 3.1.2).

3.4 Design decision models versus *k-MPP* queries

The proposed product selection strategy is orthogonal to the design decision models studied in other domain [7,16]. Like [7,16], the *k-MPP* queries consider the product (i.e., the product attributes), the consumer (i.e., the customer), the firm (i.e., the manufacturer) and its competitors (i.e., competitors' products in the market). The *k-MPP* queries tend to maximize the *profit* (i.e., modeled via the expected number of total customers) by selecting a subset of products from Q that can beat the competitors' product by fulfilling the *demand* of the *consumer*. Unlike top-k queries [3] where weights are used to rank the objects, the *demand* of a product is measured by the customers c that appears in $RSL(c)$ and the alternative choices of a customer c are determined by the products p that appears in $DSL(c)$.

4 Complexity analysis

This section analyzes the complexity of processing k-MPP query by providing a serial execution approach and then, a hypothetical parallel approach for improving the efficiency. The *k-MPP* is a special type of query which requires a different kind of data indexing scheme to expedite its execution time. We conclude that existing data indexing policies and parallel skyline computing techniques are inefficient for *k-MPP* queries.

4.1 Serial execution of *k-MPP* query

To process *k-MPP* query, one can start computing the dynamic skyline of each customer $c \in C$ and then, check whether $DSL(c)$ contains the product $q \in Q$. Once we

know $DSL(c)$ of each customer $c \in C$, we can compute $Pr(c, q|P)$ (or $Pr(c, q|P \cup Q)$) and $E(C, q|P)$ (or $E(C, q|P \cup Q)$) for each product $q \in Q$. However, this naive solution is not efficient as we need to compute $|C|$ dynamic skylines (recall $C \gg P \gg Q$). The dynamic skyline is itself computationally very expensive [18]. Not all products in Q may also be readily available for computing $DSL(c)$ offline to expedite the computation, e.g., some of the products may not be manufactured yet, rather they are prototyped on the fly based on the demand received from the customers (e.g., product survey), which requires online computation of $DSL(c)$. An alternative/better solution to the above is to compute the reverse skyline of each query product $q \in Q$, then compute the dynamic skyline only for those customers that appear in the reverse skyline of q (avoiding unnecessary dynamic skyline computations) as suggested in Eq. 4. We term this as the *baseline approach*. The efficiency of this solution depends on the number of query products in Q, the time required to compute the reverse skylines and the dynamic skylines of the customers c that appear in $RSL(q)$ of $q \in Q$.

Assume that the average run time of computing $RSL(q)$ of $q \in Q$ and $DSL(c)$ of a customer c in $RSL(q)$ are T_{RSL} and T_{DSL}, respectively. Also, assume that both T_{RSL} and T_{DSL} include the time required to build the data index. Then, the average runtime complexity, T^s_{k-MPP}, of the baseline approach is given as follows:

$$T^s_{k-MPP} = |Q| \times (T_{RSL} + |RSL(q)| \times T_{DSL}) \tag{9}$$

Equation 9 represents the runtime complexity of executing a k-*MPP* query in serial where $|RSL(q)|$ is the average reverse skylines of the query products q in Q.

4.2 Parallel processing of *k-MPP* query

Assume that there are $m + 1$ computing resources (e.g., threads, processors and machines) $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_m$ available. We term these computing resources as nodes, where $\mathcal{T}_0$ is regarded as the master node and others as the worker nodes. The efficiency of the baseline approach can be improved if we compute the reverse skylines and the corresponding dynamic skylines in parallel. Figure 5 shows a hypothetical parallel processing strategy of k-*MPP* query for $|Q| = 3$ in a 4-nodes system. Here, we compute either a reverse skyline or a dynamic skyline query in each node at a time. Assume that there are three customers for each $q \in Q$ on average. In serial execution, we would have complexity: $3 \times T_{RSL} + 9 \times T_{DSL}$. As we can execute three skyline queries in parallel in our hypothetical parallel processing strategy of k-*MPP*, we will have complexity: $T_{RSL} + 3 \times T_{DSL}$. That is, the hypothetical parallel processing strategy can achieve speed up ratio 3 as illustrated in Fig. 5. Here, we ignore the implementation overhead of synchronization, data communication, result processing, etc. as indicated by T_Q in Fig. 5. Therefore, the runtime complexity of processing k-*MPP* query in parallel, T^p_{k-MPP}, in an $(m + 1)$-nodes ideal system is:

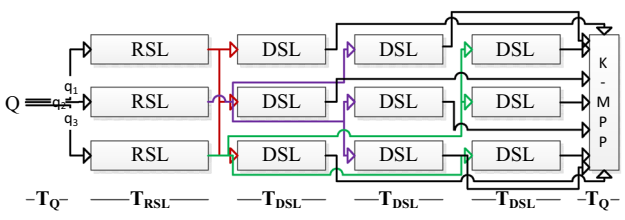


Fig. 5 A hypothetical parallel processing strategy of k-*MPP* query

$$T^p_{k-MPP} = \frac{|Q|}{m} \times (T_{RSL} + |RSL(q)| \times T_{DSL}) \tag{10}$$

The efficiency of the hypothetical parallel processing strategy of k-*MPP* can be further expedited as follows: (1) building query-independent reusable data index; (2) grouping queries based on their similarities (e.g., significant overlap of the search spaces of two or more skyline queries) and processing them together by sharing their computations; and (3) computing the $DSL(c)$ offline of each customer $c \in C$ considering the product objects p in P and later, updating it for query products $q \in Q$.

4.3 Limitations of existing approaches

As outlined in Sect. 4.1, the basic units of computation for processing k-*MPP* queries are dynamic and reverse skylines and the efficiency of realizing a k-*MPP* query in a hypothetical ideal system is much dependent on parallelizing these units as conceptualized in Sect. 4.2. Though there are a number of established works on parallelizing traditional skyline query exist in the literature ([10,17,21,24,31,32] for survey), none of them are suitable for processing k-*MPP* queries in parallel (recall that k-*MPP* relies on dynamic and reverse skylines, not the traditional skyline query). To apply these technique for our problem, we need to transform every objects into a new space considering each query ($q \in Q$ for reverse skyline) as well as the customer ($c \in C$ for dynamic skyline) as the origin, which is certainly not efficient for solving k-*MPP*.

The only work on computing dynamic and reverse skylines in parallel is proposed in [19] using quad-tree. However, the quad-tree index is query dependent and is designed to facilitate the computation of a single dynamic and reverse skyline query in parallel, not multiple skyline queries. There is also no technique for computing bichromatic reverse skyline involving both competitor's products and customer preferences using quad-tree index. The index needs to be rebuilt every time if we want to process a new dynamic or

reverse skyline query. Therefore, the approach proposed in [19] cannot omit the data index-building time from T_{RSL} and T_{DSL} in Eq. 10 for processing *k-MPP* query in parallel. To the best of our knowledge, there is also no work on grouping multiple skyline queries and processing them together by sharing their data space and computation.

5 Our approach

This section presents our approach for processing the *k-MPP* product selection queries in parallel. Firstly, we design a simple yet very efficient grid-based index structure, which is query independent, and therefore, the resultant index is reusable. Then, we show how to efficiently compute the basic units of a *k-MPP* query by reducing their search spaces. We also present an approach for grouping multiple query products $q \in Q$ together based on their (extended) search spaces and thereby, expedite the processing of the *k-MPP* further.

5.1 Data indexing

We partition the whole data space into regular grids. Then, we index our dataset P and C by scanning them once. We also carefully select some of the products $p \in P$ as *pivots* to establish the partitionwise dominances.

5.1.1 Index structure

We apply an $n \times n$ grid to partition the whole data space $D = \{P, C\}$. That is, each dimension is divided into n parts and there are n^d partitions in total for a d-dimensional data space D. These partitions are termed as $D_0, D_1, \ldots, D_{n^d-1}$ and are read in column-major order. Each partition $D_l, l \in \{0, 1, \ldots, n^d - 1\}$, is qualified by a d-dimensional *positional vector* pos_l, which locates the corresponding partition in the grid structure. The range of values covered in the ith dimension of a partition are given as follows: $[pos_l^i \times \delta^i, (pos_l^i + 1) \times \delta^i)$, where $\delta^i = \frac{max(ob^i)}{n}, \forall i \in \{1, 2, \ldots, d\}$ and $pos^i \in \{0, 1, \ldots, n-1\}$. The positional vector pos_l of the partition in which the data objects belong to are computed as follows: $pos_l^i = \frac{ob^i}{n}, \forall i \in \{1, 2, \ldots, d\}$. Therefore, the index structure can identify the *location* of a data object *ob* in the grid, denoted by $loc(ob)$, in $O(1)$ time.

Example 11 Consider the data objects given in Fig. 1. A 3×3 grid index structure of this dataset is shown in Fig. 6. In this index structure, we have 9 partitions: $D_0, D_1 \ldots, D_8$ and the corresponding positional vectors of these partitions are: $< 0, 0 >, < 0, 1 >, \ldots, < 2, 2 >$, where $\delta^1 = \frac{24}{3} = 8$ and $\delta^2 = \frac{24}{3} = 8$. Here, we assume $max(ob^i) = 24, \forall i \in \{1, 2, \ldots, d\}$. The range of values covered in the 1^{st} and 2^{nd} dimensions of partition D_1 are $[0, 8)$ and $[8, 16)$, respectively. Similarly, the range of values covered in the 1^{st} and 2^{nd} dimensions of D_8 are $[16, 24)$ and $[16, 24)$, respectively. The location of p_2 is $< \frac{6}{8}, \frac{18}{8} >=< 0, 2 >$, i.e., partition D_2.

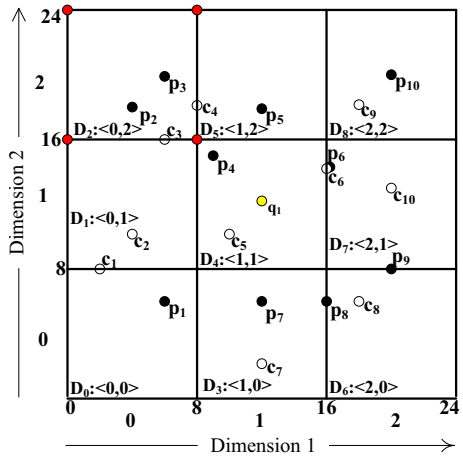


Fig. 6 A 3×3 grid index structure of the two-dimensional dataset given in Fig. 1

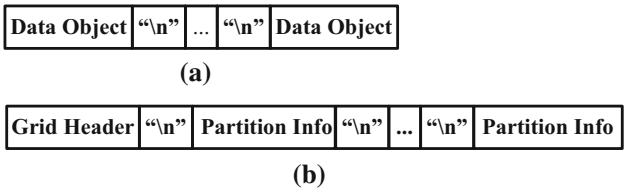


Fig. 7 The **a** data object and **b** grid index file format

5.1.2 Index creation

To create the index, we scan the data objects in P and C sequentially and then, rewrite them in text file(s) as data objects separated by newlines as shown in Fig. 7a. Each data object is modeled as containing the following information: (a) the vector pos_l of the partition D_l containing the data object *ob*; (b) the *id* of *ob* and (c) values of the data object *ob*. We maintain two different text files, one for products P, denoted by "gridproducts.txt" and another one for customers C, denoted by "gridcustomers.txt." The rationale is that we can scan the indexed products once and pass it to each skyline query if P fits entirely in memory ($C \gg P$). We can also scan objects from C page by page and process them one after another for computing the reverse skylines.

While scanning and rewriting indexed objects, we save information about the partitions of the indexed data objects (e.g., positional vector of the partition, #product objects and #customer objects in it). When we finish scanning objects from P and C, we write the information about the partitions

and the pivot objects of the partition in a text file, denoted by "gridinfo.txt," consisting of the followings: (a) a grid header and (b) partition infos as shown in Fig. 7b. The grid header consists of the following information: (i) #dimensions; (ii) grid size (n) and (iii) the δ. The partition info(s) contain information about the non-empty partitions consisting of the followings (i) pos_l of the partition D_l; (ii) #product objects; (iii) #customer objects; (iv) #pivot objects and (v) the values of the pivot objects. We do not save information about an empty partition, i.e., partitions that do not contain any type of data object. The index is built once and shared by all queries.

5.1.3 Pivot object and partitionwise dominance

Definition 12 A partition D_1 dominates another partition D_2 w.r.t. a data object ob_3, denoted by $D_1 \prec_{ob_3} D_2$, if every data object $ob_2 \in D_2$ is dominated by a data object $ob_1 \in D_1$ w.r.t. ob_3, i.e., $ob_1 \prec_{ob_3} ob_2$. If ob_1 is a product object from P, we say D_1 productwise dominates D_2 and is denoted by $D_1 \prec^P_{ob_3} D_2$. We term the product object ob_1 as the pivot object of D_1.

Example 12 Consider the datasets given in Fig. 1 and the query product $q_1(12, 12)$ as shown in Fig. 6. According to Definition 1, every object in D_2 is dominated by $p_4 \in D_4$ w.r.t. q_1. Therefore, D_2 is dominated by D_4 w.r.t q_1, i.e., $D_4 \prec_{q_1} D_2$. Here, p_4 is the pivot object for D_4 which is used to establish $D_4 \prec_{q_1} D_2$ and $D_4 \prec^P_{q_1} D_2$. Similarly, p_4 can also be used to establish the following relationships: $D_4 \prec_{c_5} D_2$ and $D_4 \prec^P_{c_5} D_2$.

To establish the partitionwise dominance between partitions D_1 and D_2 w.r.t. ob_3, we do not need to check the pairwise dominance between the data object $ob_1 \in D_1$ and every data object $ob_2 \in D_2$. We know that the data objects of a partition D_2 are bounded by its hypothetical corner objects (as marked by the red circles in Fig. 6). In a d-dimensional data space, a partition has 2^d corner objects. The values of these corner objects of D_2 in the ith dimension are: $< pos^i_2 \times \delta^i + b \times \delta^i >, \forall b \in \{0, 1\}$. We only need to check the pairwise dominance of ob_1 with these hypothetical corner objects of D_2 w.r.t. ob_3. If $ob_1 \in D_1$ dominates all of these corner objects of D_2 w.r.t. ob_3, then we can ensure that all data objects of $ob_2 \in D_2$ will be dominated by $ob_1 \in D_1$ w.r.t. ob_3. Therefore, we save ob_1 as pivot objects for D_1 to establish $D_1 \prec_{ob_3} D_2$ in "gridinfo.txt."

Example 13 Consider the datasets given in Fig. 1, the index structure and the query product $q_1(12, 12)$ as shown in Fig. 6. The hypothetical corner objects of D_2 are marked as red circles in Fig. 6. It is easy to verify that all of these corner objects of D_2 are dominated by $p_4 \in D_4$ w.r.t. q_1. We get $D_4 \prec_{q_1} D_2$ and also, $D_4 \prec^P_{q_1} D_2$. Therefore, p_4 is stored as pivot object for D_4 in "gridinfo.txt" to establish the partitionwise dominance with D_2 in D w.r.t. q_1.

Algorithm 1: Search Space of Dynamic Skyline

```
Input  : customer c
Output: N_c(D)
1  begin
2      L ← loc(c); // initialize the FIFO list L
       with loc(c)
3      while L ≠ ∅ do
4          D_1 ← pop an element from L;
5          add D_1 to N_c(D);// D_1 is a non-dominating
               partition
6          N_1(D_1) ← immediate neighboring partitions of D_1;
7          for each D_2 ∈ N_1(D_1) do
8              if ∄D_1 ∈ N_c(D) : D_1 ≺^P_c D_2 then
9                  add D_2 to N_c(D);// D_2 is a
                       non-dominating partition
10                 insert D_2 into L;
```

Lemma 1 *If a partition D_1 dominates another partition D_2 w.r.t. a data object ob and D_2 dominates D_3 w.r.t. ob too, then D_1 also dominates D_3 w.r.t. ob, i.e., $D_1 \prec_{ob} D_2$ and $D_2 \prec_{ob} D_3 \implies D_1 \prec_{ob} D_3$.*

Proof Let us assume that there are three data objects $ob_1 \in D_1, ob_2 \in D_2$ and $ob_3 \in D_3$. The proof of this lemma then immediately follows the transitivity: $ob_1 \prec_{ob} ob_2$ and $ob_2 \prec_{ob} ob_3 \implies ob_1 \prec_{ob} ob_3$, and the partitionwise dominance given in Definition 12. □

5.2 Processing the basic computational units

This section describes how we can compute the basic units (i.e., dynamic and reverse skylines) of the *k-MPP* query by reducing their search spaces and thereby, avoid checking the pairwise dominances as much as possible. We establish partitionwise dominance relationships so that that we can filter all objects belonging to a partition without checking the pairwise dominances.

5.2.1 Computing dynamic skyline of a customer

We already know that a dynamic skyline of a customer c retrieves all products $p_1 \in P$ that are not dynamically dominated by other products $p_2 \in P$ w.r.t. the customer c as explained in Sect. 2.

Lemma 2 *We can safely remove every product p_2 contained in a partition D_2 for processing $DSL(c)$ iff $\exists D_1 \in D : D_1 \prec^P_c D_2$.*

Proof From definition 12, we know that $\exists p_1 \in D_1$ such that the products $p_2 \in D_2$ are dominated by p_1 w.r.t. the customer c as $D_1 \prec^P_c D_2$. Therefore, $p_2 \in D_2$ can not appear in the dynamic skyline of the customer c and can be removed safely while computing $DSL(c)$. □

Algorithm 2: Dynamic Skyline of a Customer

```
   Input  : customer c, N_c(D), products P
   Output: DSL(c)
1  begin
2      if N_c(D) <-= null then
3          N_c(D) ← searchSpaceOfDynamicSkyline(c);
           // Algo. 1
4      for each p ∈ P do
5          if loc(p) ∈ N_c(D) then
6              insert p to the min heap H_c;
7      DSL(c) ← ∅; // initialization
8      while H_c ≠ ∅ do
9          p_1 ← retrieve the root element from H_c;
10         if ∄p_2 ∈ DSL(c) : p_2 ≺_c p_1 then
11             add p_1 to DSL(c);
```

The search space $\mathcal{N}_c(D)$ of a dynamic skyline query c consists of all non-dominating partitions in D (according to Lemma 2 and Lemma 1), which can be determined by accessing the information stored in "gridinfo.txt." As we are going to retrieve products $p_1 \in P$ that are preferable to c than other products $p_2 \in P$, we need to access the product data stored in "gridproducts.txt" only. The steps of computing $DSL(c)$ are described below:

(1) Computing $\mathcal{N}_c(D)$ for $DSL(c)$: We initialize a FIFO list $\mathcal{L}$ by the location of the customer c, i.e., $loc(c)$. Then, we do the following until $\mathcal{L}$ is empty: (a) pop an element D_1 from $\mathcal{L}$ and add it to $\mathcal{N}_c(D)$; (b) compute the immediate neighbors $\mathcal{N}_1(D_1)$ of D_1 as follows: $< pos_1^i + b >, \forall b \in \{-1, 0, +1\}$ and $i \in \{1, 2, ..., d\}$; and (c) $\forall D_2 \in \mathcal{N}_1(D_1)$, if D_2 is not dominated by any partition in $\mathcal{N}_c(D)$ (Lemma 2) w.r.t. c, then add D_2 to $\mathcal{N}_c(D)$ and also, insert D_2 into $\mathcal{L}$. We add D_2 to $\mathcal{L}$ because we can not guarantee that the immediate neighbors of D_2 will be dominated either by D_1 or D_2 , e.g., neighboring partitions of D_2 that are positioned into the vertical and horizontal directions. The above steps are pseudo-coded in Algorithm 1.

(2) Retrieving Products from P: We access the products p stored in "$gridproducts.txt$" sequentially and filter them as follows: if $loc(p) \in \mathcal{N}_c(D)$, then we insert p into a min heap $\mathcal{H}_c$, otherwise drop it. To compare two objects for the min heap $\mathcal{H}_c$, we use the Euclidean distances of products p_1 and p_2 to c. We assume that the min heap $\mathcal{H}_c$ can be stored in the main memory (recall $P \ll C$). These steps are pseudo-coded in lines 4–6 of Algorithm 2.

(3) Computing Dynamic Skyline $DSL(c)$: Firstly, we initialize $DSL(c)$ to $\emptyset$. Then, we do the following until $\mathcal{H}_c$ is empty: (a) retrieve the root product p_1 from $\mathcal{H}_c$; and (b) add p_1 to $DSL(c)$ if $\nexists p_2 \in DSL(c) : p_2 \prec_c p_1$. These steps are pseudo-coded in lines 7–11 of Algorithm 2.

Correctness of Algorithm 2. The lines 2–10 of Algorithm 1 computes the search space, i.e., the set of non-dominating partitions $\mathcal{N}_c(D)$ for $DSL(c)$ starting from the $loc(c)$. Then, it grows $\mathcal{N}_c(D)$ by repeatedly adding the non-dominating neighboring partitions into it. It stops only if the neighboring partitions are dominated by some of the partitions already added to $\mathcal{N}_c(D)$. Therefore, we can say that $\mathcal{N}_c(D)$ consists of the legitimate partitions which may contain the dynamic skyline objects for $DSL(c)$. The lines 4–6 of Algorithm 2 scan the product objects from "gridproduct.txt" and insert them in a min heap $\mathcal{H}_c$ in order of their distances to c. The lines 7–11 of Algorithm 2 compute $DSL(c)$ by repeatedly retrieving the root product from $\mathcal{H}_c$. The mean heap $\mathcal{H}_c$ ensures that the root product is either dominated by some products already added in $DSL(c)$ or part of it [18]. Therefore, we correctly compute the dynamic skyline of a customer $c \in C$, i.e., $DSL(c)$.

5.2.2 Computing the reverse skyline of a product

A reverse skyline query of a query product q, $RSL(q)$, consists of all customers $c \in C$ that prefer to buy q over other products $p \in P$. The $RSL(q)$ is computed by retrieving all customers $c \in C$ that are not dynamically dominated by the midskylines of products $p \in P$ w.r.t. q in each of its orthant, i.e., $m \not\prec_q c$, where m is the midskyline for P as explained in Sect. 2.

Observation-1: If a product p dominates a customer c w.r.t. a query product q in an orthant O, then the midpoint m of p also dominates c w.r.t. q. Therefore, the customer c cannot be in the reverse skyline of q if $p \prec_q c$ in an orthant O.

Observation-2: If a product p_1 dominates another product p_2 w.r.t. a query product q in an orthant O, then the mid point m_1 of p_1 also dominates the midpoint m_2 of p_2 w.r.t. q. Therefore, if p_2 dominates c w.r.t. q in an orthant O, then m_1 also dominates c w.r.t. q. As a result, c cannot be in the reverse skyline of the query product q if $p_1 \prec_q p_2$ and $p_2 \prec_q c$.

Example 14 Consider the datasets given in Fig. 1, the index structure, the query product $q_1(12, 12)$, customer $c_3 \in C$ and the midpoint m_4 of product $p_4 \in P$ w.r.t. q_1 as shown in Fig. 8. It is easy to verify that $c_3 \in C$ is dominated by m_4 w.r.t. q_1 in orthant O_4 as $p_4 \prec_{q_1} c_3$. Therefore, the customer c_3 cannot be in $RSL(q_1)$. Similarly, any customer in D_2 dominated by p_2 or p_3 w.r.t. q_1 would also be dominated by m_4 w.r.t. q_1 and therefore, could not be in $RSL(q_1)$.

Lemma 3 *We can safely remove each product p_2 and customer c_2 contained in D_2 for processing $RSL(q)$ iff (1) $\exists$ pivot $p \in D_1$ such that the pivot p and the partition D_2 are in the same orthant of q and (2) $D_1 \prec_q^P D_2$.*

Proof The proof of this lemma immediately follows the Observation 1 and 2, also the definition of partitionwise dominance given in the Definition 12. □

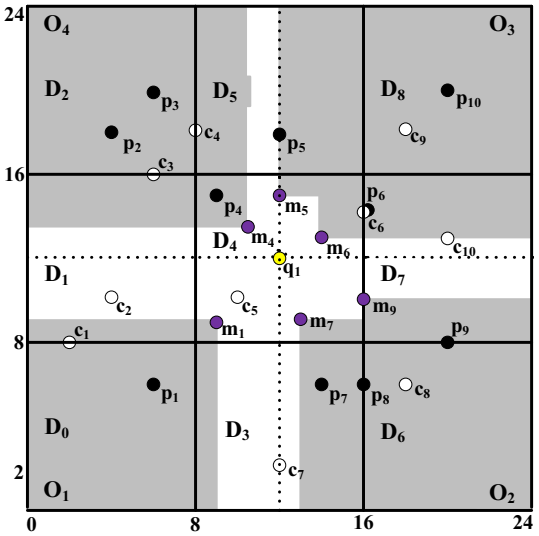


Fig. 8 Midpoint of p_4 dominates c_3 w.r.t. q_1 for dataset given in Fig. 1

Algorithm 3: Search Space of Reverse Skyline

```
Input  : query product q
Output: N_q(D)
1  begin
2    L ← loc(q); // initialization
3    while L ≠ ∅ do
4      D_1 ← pop an element from L;
5      add D_1 to N_q(D); // D_1 is a non-dominating
       partition
6      N_1(D_1) ← immediate neighboring partitions of D_1;
7      for each D_2 ∈ N_1(D_1) do
8        if ∄D_1 ∈ N_q(D) : D_1 ≺_q^P D_2 & O_q(p) = O_q(D_2)
         then
9          add D_2 to N_q(D); // p is a pivot
           object for D_1
10         insert D_2 into L; // D_2 is a
           non-dominating partition
```

The search space $\mathcal{N}_q(D)$ of a reverse skyline query q consists of all non-dominating partitions (according to Lemma 3 and Lemma 1), which can be determined by accessing the information stored in "gridinfo.txt" only. However, as we are going to retrieve customers that find q preferable to other products in P, we need to access both product data and customer data stored in "gridproducts.txt" and "gridcustomers.txt," respectively.

The steps of computing $RSL(q)$ are follows:

(1) Computing $\mathcal{N}_q(D)$ for $RSL(q)$: We initialize a FIFO list $\mathcal{L}$ by the location of the query q, i.e., $loc(q)$. Then, we do the following until $\mathcal{L}$ is empty: (a) pop an element D_1 from L and add it to $\mathcal{N}_q(D)$; (b) compute the immediate neighbors $N_1(D_1)$ of D_1 as follows: $< pos_1^i + b >, \forall b \in \{-1, 0, +1\}$ and $i \in \{1, 2, ..., d\}$; and (c) $\forall D_2 \in N_1(D_1)$, if D_2 is not dominated by any partition in $\mathcal{N}_q(D)$ (Lemma 3), then add D_2 to $\mathcal{N}_q(D)$ and also, insert D_2 into $\mathcal{L}$. These steps are pseudo-coded in lines 2–10 of Algorithm 3.

Algorithm 4: Midpoint Skyline of a Product

```
Input  : query product q, N_q(D), products P
Output: midpoint skyline M
1  begin
2    for each p ∈ P do
3      if loc(p) ∈ N_q(D) then
4        insert p to the min heap H_q;
5    M ← ∅; // initialization
6    while H_q ≠ ∅ do
7      p_1 ←retrieve the root element from H_q;
8      m_1 ← compute midpoint of p_1 w.r.t. q;
9      if ∄m_2 ∈ M : m_2 ≺_q m_1 & O_q(m_1) = O_q(m_2) then
10       add m_1 to M; // m_1 is a midpoint
         skyline
```

(2) Retrieving Products from P: We access the product objects p stored in "$gridproducts.txt$" sequentially and filter them as follows: if $loc(p) \in \mathcal{N}_q(D)$, then we insert p into a min heap $\mathcal{H}_q$, otherwise drop it. To compare two objects for the min heap $\mathcal{H}_q$, we use the Euclidean distances of products p_1 and p_2 to q. We assume that the min heap $\mathcal{H}_q$ can be stored in the main memory (recall $P \ll C$). These steps are pseudo-coded in lines 2–4 of Algorithm 4.

(3) Computing Midpoint Skyline of P: Firstly, we initialize the midpoint skyline set $\mathcal{M}$ to $\emptyset$. Then, we do the following until $\mathcal{H}_q$ is empty: (a) retrieve the root product p_1 from $\mathcal{H}_q$; and (b) add the midpoint m_1 of p_1 to $\mathcal{M}$ if $\nexists m_2 \in \mathcal{M} : m_2 \prec_q m_1$ and m_1 and m_2 are in the same orthant O of the query product q (**Observation-2**). These steps are pseudo-coded in lines 5–10 of Algorithm 4.

(4) Computing Reverse Skyline from C: We access the customers c stored in "$gridcustomers.txt$" sequentially and filter them as follows: (a) if $loc(c) \in \mathcal{N}_q(D)$, then go to step (b), otherwise drop it; and (b) we add c to the $RSL(q)$ if $\nexists m \in \mathcal{M} : m \prec_q c$ and both c and m are in the same orthant O of q. These steps are pseudo-coded in lines 6–10 of Algorithm 5.

Correctness of Algorithm 5. The lines 2–10 of Algorithm 3 computes the set of non-dominating partitions $\mathcal{N}_q(D)$ for $RSL(q)$ starting from the $loc(q)$. Then, it grows $\mathcal{N}_q(D)$ by repeatedly adding the non-dominating neighboring partitions into it. It stops only if the neighboring partitions are dominated by some of the partitions already added into $\mathcal{N}_q(D)$ in the same orthant of q. Therefore, we can say that $\mathcal{N}_q(D)$ consists of the legitimate partitions which may contain the midpoint skyline and reverse skyline objects for $RSL(q)$. The lines 2–4 of Algorithm 4 scan the products from "gridproduct.txt" and insert them into a min heap $\mathcal{H}_q$ in order of their distances to q. The lines 5–10 of Algorithm 4 compute

Algorithm 5: Reverse Skyline of a Product

```
Input  : query q, midpoint skyline M, N_q(D), products P,
         customers C
Output: RSL(q)
1  begin
2    if N_q(D) <-= null then
3      N_q(D) ← searchSpaceOfReverseSkyline(q); // Algo. 3
4    if M <-= null then
5      M ← midpointSkyline(q, N_q(D), P); // Algo. 4
6    RSL(q) ← ∅; // initialization
7    for c ∈ C do
8      if loc(c) ∈ N_q(D) then
9        if ∄m ∈ M : m ≺_q c & O_q(m) = O_q(c) then
10         add c to RSL(q);
```

midpoint skyline $\mathcal{M}$ by repeatedly retrieving the root product from $\mathcal{H}_q$. The mean heap $\mathcal{H}_q$ ensures that the current root product is either dominated by some products already added in $\mathcal{M}$ in an orthant of q or a part of $\mathcal{M}$. Finally, lines 6–10 of Algorithm 5 scan customers c from "gridcustomers.txt" and check whether they are dominated by an $m \in \mathcal{M}$. The customer c is added to $RSL(q)$, only if it is not dominated by any $m \in \mathcal{M}$ in an orthant of q. Therefore, Algorithm 5 correctly computes the reverse skyline of q.

5.2.3 Optimizing partitionwise dominance for RSL

Consider a hypothetical query point q and the grid index structure in a hypothetical two-dimensional data space as shown in Fig. 9. We see that we could establish partitionwise dominances for D_4 with D_0, D_2, D_6 and D_8 if we had a pivot product for each of the four orthants of q. Therefore, we argue to have 2^d pivots per partition in the index structure for $RSL(q)$ and the grid size n should be selected in a way so that each partition of D contains at least 2^d objects. Enforcing this may incur additional steps while indexing the data. The selected pivot objects should be closer to the corners of a partition to increase the probability of having a pivot object for each orthant of a random query product. However, the above is different for dynamic skylines. As we do not consider orthants in dynamic skyline, one pivot object per partition is sufficient for $DSL(c)$.

5.3 Processing *k-MPP* query

To process a *k-MPP* query, we need to do the following: (a) computing the reverse skyline for each query product $q \in Q$; (b) computing the dynamic skyline of each customer $c \in RSL(q)$; and (c) selecting the k query products from Q based on their contribution in the market. However, the processing of dynamic skyline of a customer $c \in RSL(q)$ varies for $k-MPP_{ind}$ and $k-MPP_{dep}$. To compute $DSL(c)$ and $RSL(q)$ for $k-MPP_{dep}$ query, we use $P \cup Q \setminus q$ as P.

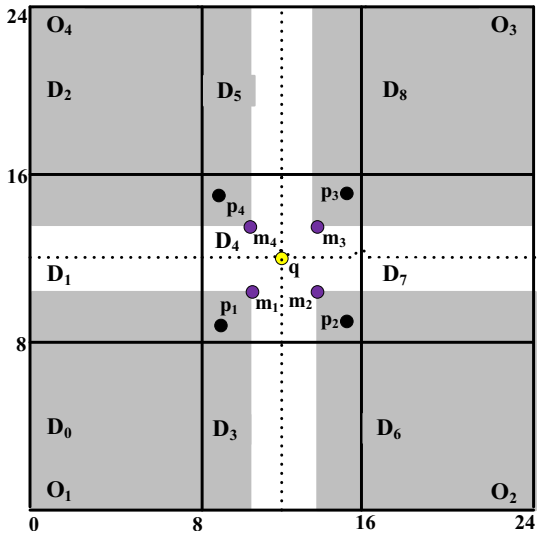


Fig. 9 Selection of pivot objects for a hypothetical query product q in partition D_2 for $RSL(q)$

5.3.1 An straightforward strategy

This strategy computes *k-MPP* in light of the hypothetical parallel strategy of processing *k-MPP* as explained in Sect. 4.2. That is, we first compute the $RSL(q)$ for each query product $q \in Q$ and then, we compute the $DSL(c)$ for each $c \in RSL(q)$ in parallel. We consider each (reverse/dynamic) skyline query as an independent job which needs to be executed by the worker nodes supervised by the master node. However, the number of skyline queries can be larger than the number of available worker nodes. Therefore, we insert all jobs into a *waiting queue* and start executing them in *first-come-first-serve* order. Once a job is finished by a worker node, the result is returned to the master node. These steps are pseudo-coded in Algorithm 6.

5.3.2 An optimized strategy

This section describes an optimized strategy for computing *k-MPP* product selection query in parallel based on: (1) grouping similar query products in Q and processing their reverse skylines together; and (2) computing $DSL(c)$ offline and updating it for the query products in $q \in Q$. Our approach is described below.

A) Grouping similar query products We propose to group queries in Q based on their similarities to achieve the followings: (1) the reverse skylines of the similar query products can be processed together by a single working node and (2) the disk accesses can be reduced by sharing the data objects

Algorithm 6: An Straightforward Strategy

Input : query products Q, products P, customers C
Output: subset of query products Q'

begin
  $\mathcal{T}_0$: **for** *each* $q \in Q$ **do**
    $E(C, q|P) \leftarrow 0$; // initialization
  $\mathcal{T}_0$: **parallel for** *each* $q \in Q$ **do**
    $\mathcal{T}_j \leftarrow$ retrieveAFreeNode ($\{\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_m\}$);
    $RSL(q) \leftarrow \mathcal{T}_j$: reverseSkyline($q, null, null, P, C$); // Algo. 5
  $\mathcal{T}_0$: **for** *each* $q \in Q$ **do**
    parallel for *each* $c \in RSL(q)$ **do**
      $\mathcal{T}_j \leftarrow$ retrieveAFreeNode($\{\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_m\}$);
      $DSL(c) \leftarrow \mathcal{T}_j$: dynamicSkyline($c, null, P \cup q$); // Algo. 2
      $E(C, q|P) \leftarrow E(C, q|P) + \frac{1}{DSL(c)}$;
  $\mathcal{T}_0 : Q' \leftarrow$ selectKProducts(Q); // based on $E(C, q|P)$

among the basic computational units while processing the k-*MPP* query.

Definition 13 (*Strong similarity*) Two query products q_1 and q_2 in Q are said to be strongly similar iff $\mathcal{N}_{q_1}(D) = \mathcal{N}_{q_2}(D)$.

However, the checking of the above strong similarity requires computing the actual search spaces $N_{q_1}(D)$ and $N_{q_2}(D)$ of q_1 and q_2 and then, inspecting whether these search spaces are the same, which is time-consuming for the master node. Therefore, grouping queries based on strong similarity is not durable for efficiently parallelizing k-*MPP* query. As an alternative, we propose to rely on the *extended search space* $\mathcal{N}_q^+(D)$ of the reverse skylines of the query products $q \in Q$ to measure their similarity. The extended search space of the reverse skyline of a query product $q \in Q$ is computed as follows: Firstly, we add the location of the query product q, i.e., $loc(q)$ to $\mathcal{N}_q^+(D)$. Then, we initialize a FIFO list $\mathcal{L}$ by the immediate neighbors of $loc(q)$ and do the following until $\mathcal{L}$ is empty: (a) pop an element D_1 from L; and (b) if $\nexists$ pivot $p \in loc(q) : p \prec_q D_1 \ \& \ O_q(p) = O_q(D_1)$, then add D_1 to $\mathcal{N}_q^+(D)$ and insert the immediate neighbors $N_1(D_1)$ of D_1 into $\mathcal{L}$. The above is pseudo-coded in Algorithm 7. It is easy to verify that $\mathcal{N}_q(D) \subseteq \mathcal{N}_q^+(D)$.

Lemma 4 *Two query products q_1 and q_2 share the same extended search space, i.e., $\mathcal{N}_{q_1}^+(D) = \mathcal{N}_{q_2}^+(D)$, if (1) $loc(q_1) = loc(q_2)$ (rename them as $loc(q_{12})$); and (2) $\forall p \in loc(q_{12})$, $O_{q_1}(p) = O_{q_2}(p)$ in $loc(q_{12})$, where $p \in P$ is a pivot for $loc(q_{12})$.*

Proof Assume that the extended search spaces $\mathcal{N}_{q_1}^+(D)$ and $\mathcal{N}_{q_2}^+(D)$ are not the same but the conditions (1) and (2) in Lemma 4 hold for q_1 and q_2. Also, assume that a partition D_1

Algorithm 7: Search Space$^+$ of Reverse Skyline

Input : query product q
Output: extended search space $\mathcal{N}_q^+(D)$

begin
  add $loc(q)$ to $\mathcal{N}_q^+(D)$;
  $\mathcal{N}_1(loc(q)) \leftarrow$ immediate neighboring partitions of $loc(q)$;
  $\mathcal{L} \leftarrow \mathcal{N}_1(loc(q))$; // initialization
  while $\mathcal{L} \neq \emptyset$ **do**
    $D_1 \leftarrow$ pop an element from $\mathcal{L}$;
    if $\nexists$ *pivot* $p \in loc(q) : p \prec_q D_1 \& O_q(p) = O_q(D_1)$ **then**
      add D_1 to $\mathcal{N}_q^+(D)$;
      insert immediate neighboring partitions of D_1 into $\mathcal{L}$;

does not appear in $\mathcal{N}_{q_1}^+(D)$ but does in $\mathcal{N}_{q_2}^+(D)$. Since $D_1 \notin \mathcal{N}_{q_1}^+(D)$, there must be a pivot product p_1 in $loc(q_1)$ such that (1) $O_q(p_1) = O_q(D_1)$ and (2) $p_1 \prec_{q_1} D_1$. However, the above cannot happen. Since, $loc(q_1) = loc(q_2)$ and p_1 appears in the same orthant for both q_1 and q_2, then p_1 must also dominate D_1 w.r.t. q_2, i.e., $p_1 \prec_{q_2} D_1$. □

Definition 14 (*Weak similarity*) Two queries q_1 and q_2 in the query product set Q are said to be weakly similar if (1) $loc(q_1) = loc(q_2)$; and (2) $\mathcal{N}_{q_1}^+(D) = \mathcal{N}_{q_2}^+(D)$.

The first condition in Definition 14 can be decided in $O(1)$ time as explained in Sect. 5.1.1 and the second can also be decided in constant time as per Lemma 4. Therefore, it is obvious that checking query similarity based on *extended search spaces* $\mathcal{N}_q^+(D)$ is much more cheaper than checking query similarity by computing their *actual search spaces*, $\mathcal{N}_q(D)$. Once, we know the similarity of two or more queries in Q, we can group them, share the (extended) search space (which is computed for one query product per group) among them and finally, compute their reverse skylines together. However, if there is only one query in a group, we compute the actual search space of the reverse skyline for it.

Example 15 Consider the datasets given in Fig. 1, the index structure and the queries as shown in Fig 10. The location of q_1 and q_3 is D_4. Assume that p_4 is a pivot object for D_4. It is easy to verify that p_4 is in the same orthant for both q_1 and q_3, i.e., (1) if $p_4^i \leq q_1^i$, then $p_4^i \leq q_3^i$ and (2) if $p_4^i > q_1^i$, then $p_4^i > q_3^i$. Therefore, the (extended) search spaces of the reverse skylines for q_1 and q_3 are the same, i.e., $\{D_0, D_1, D_3, D_4, D_5, D_6, D_7, D_8\}$. We say q_1 and q_3 are similar.

B) Computing $DSL(c)$ offline and updating for Q To compute k-*MPP*, we need to compute the dynamic skyline for each customer $c \in RSL(q)$. The dynamic skyline is computationally known to be very expensive [18]. To expedite this operation, we precompute the dynamic skyline of each customer $c \in C$ considering only the existing products P. Then,

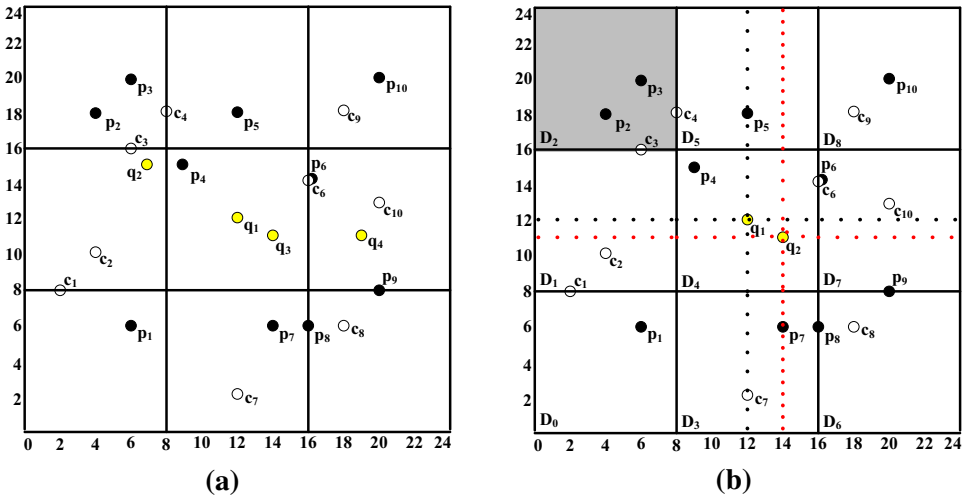


Fig. 10 **a** The locations of the query products in a 3 × 3 index structure and **b** Search space [Reverse Skyline] of the query products q_1 and q_3

Algorithm 8: An Optimized Strategy

Input : query products Q, products P, customers C
Output: subset of query products Q'

1 **begin**
2 $\mathcal{T}_0$: **for** *each* $q \in Q$ **do**
3 $E(C, q|P) \leftarrow 0$;
4 $\mathcal{T}_0 : \mathcal{Q} \leftarrow$ groupQueriesBasedOnSimilarity(Q); // Definition 14
5 $\mathcal{T}_0$: **parallel for** *each* $Q_1 \in \mathcal{Q}$ **do**
6 $\mathcal{T}_j \leftarrow$ retrieveAFreeNode ($\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_m$);
7 $\mathcal{T}_j$:**if** $Q_1.size() = 1$ **then**
8 $q \leftarrow$ retrieveTheQuery(Q_1);
9 $RSL(q) \leftarrow$ reverseSkyline($q, null, null, P, C$);
10 **else**
11 $q \leftarrow$ retrieveAnyQuery(Q_1);
12 $\mathcal{N}_q^+(D) \leftarrow$ extendedSearchSpace(q); // Algo. 7
13 $P' \leftarrow$ filteredProducts($\mathcal{N}_q^+(D)$, P);
14 **for** *each* $q \in Q_1$ **do**
15 $\mathcal{N}_q(D) \leftarrow$ refineSearchSpace($q, \mathcal{N}_q^+(D)$);
16 $\mathcal{M}(q) \leftarrow$ midpointSkyline($q, \mathcal{N}_q(D), P'$); // Algo. 4
17 $RSL(q) \leftarrow$ reverseSkyline($q, \mathcal{M}(q), \mathcal{N}_q(D), null, C$); // Algo. 5
18 $\mathcal{T}_0$: **for** *each* $q \in Q$ **do**
19 **parallel for** *each* $c \in RSL(q)$ **do**
20 $\mathcal{T}_j \leftarrow$ retrieveAFreeNode ($\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_m$);
21 $\mathcal{T}_j : DSL(c) \leftarrow$ precomputedDSL(c);
22 $DSL(c) \leftarrow \mathcal{T}_j$: updateDSL($DSL(c)$, Q);
23 $E(C, q|P) \leftarrow E(C, q|P) + \frac{1}{DSL(c)}$;
24 $\mathcal{T}_0 : Q' \leftarrow$selectKProducts($Q$); // based on $E(C, q|P)$

we update this precomputed dynamic skyline for the query product set Q. The precomputed dynamic skylines are stored as fixed-length records in a text file "dynamicskylines.txt." Each record in the file consist of the identifiers (comma separated) of the products $p \in P$ that appear in the dynamic skyline of the customer $c \in C$ (spaces are padded at the end if needed). The first record represents the dynamic skyline of the first customer c_1 in C and so on.

To retrieve the precomputed dynamic skyline for a customer c, we position the file pointer as follows: $(c_{id} - 1) \times r_c$, where c_{id} and r_c denote the *id* and the *size* of the fixed-length record of the dynamic skyline of c, respectively. The indexed products are also stored as fixed-length records in"gridproduct.txt" file. To update the dynamic skyline of a customer $c \in RSL(q)$, firstly we retrieve the product identifiers from the precomputed dynamic skylines and then, we retrieve the product values from "gridproduct.txt." The product objects are retrieved by positioning the file pointer as follows: $(p_{id} - 1) \times r_p$, where p_{id} and r_p denote the *id* and the *size* of the fixed-length record of the product object p, respectively. Finally, the dynamic skyline $DSL(c)$ of a customer c is updated for Q as follows: (1) find a subset Q' of Q such that no two queries in Q' dominate each other w.r.t. the customer c; and (2) for each query product $q \in Q'$ we do the followings: (a) if $\nexists p \in DSL(c) : p \prec_c q$, then q is added to $DSL(c)$ and if $\exists p \in DSL(c) : q \prec_c p$, then p is removed from $DSL(c)$; (b) otherwise, q is ignored.

The pseudo-code of efficiently computing *k-MPP* query in parallel considering the above is given in Algorithm 8. Firstly, the algorithm group queries based on their predicted $\mathcal{N}_q^+(D)$ (line 4) computed by the master node. Then, for each group of queries Q_1, a worker node is assigned by the master node to do the following: if $|Q_1| = 1$, process it by calling Algorithm 5, otherwise: (a) compute the extended search space $\mathcal{N}_q^+(D)$ and the filtered products $P' \subseteq P$ and (b) compute $RSL(q)$for each $q \in Q_1$ as follows (i) compute the actual search space $\mathcal{N}_q(D)$ by refining $\mathcal{N}_q^+(D)$ for q, (ii) compute midpoint skyline $\mathcal{M}(q)$ based on $\mathcal{N}_q(D)$ and P' and (iii) finally, compute $RSL(q)$ based on $\mathcal{N}_q(D)$, $\mathcal{M}(q)$ and C (lines 11–17). Lines 18–23 compute the dynamic skyline for each $c \in RSL(q)$ by retrieving the precomputed $DSL(c)$

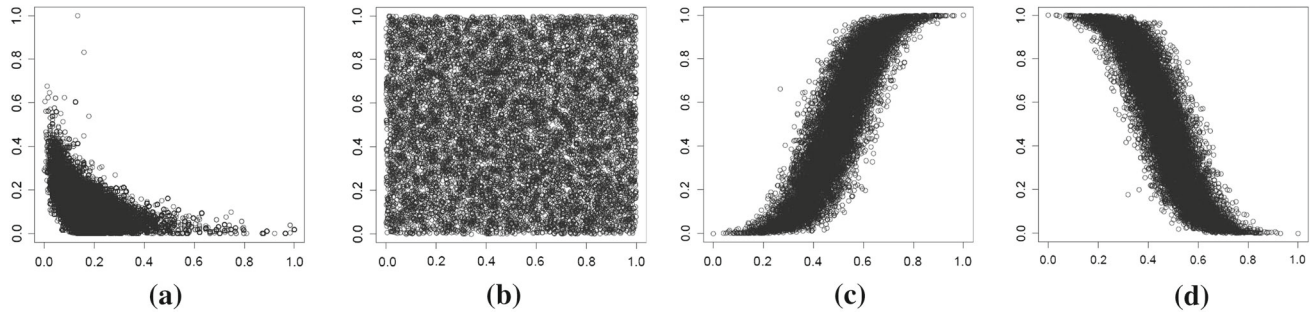


Fig. 11 Examples of tested **a** real CarDB; synthetic, **b** uniform; **c** correlated and **d** anticorrelated datasets

and then updating it for Q. Finally, line 24 selects k best query products in Q based on $E(C, q|P)$.

6 Experiments

This section presents the experimental studies of our approach. More specifically, we show the effectiveness of our product selection model as well as evaluate the performance of processing *k-MPP* query in parallel by comparing our approach with the existing counterparts.

6.1 Settings

Datasets We empirically evaluate the performance of our proposed technique for processing *k-MPP* query in parallel using real data, namely CarDB[1], consisting of 2×10^5 objects. This is a six-dimensional dataset with attributes: *make*, *model*, *year*, *price*, *mileage* and *location*. The three numerical attributes *year*, *price* and *mileage* are considered in our experiments. The dataset is normalized into the range [0, 1]. We randomly select half of these normalized car objects as products and the rest as the customers. The use of the CarDB dataset makes excellent sense in our experiment. The *k-MPP* can be exploited to estimate the market contribution of the advertised cars for a particular seller and thereby, select the k-most promising cars for designing specialized promotions. We also present experimental results based on synthetic data: uniform (UN), correlated (CO) and anti-correlated (AC), consisting of varying number of products, customers and dimensions. The cardinalities of these datasets in products and customers are 1×10^5, 2×10^5, 3×10^5, 4×10^5 and 5×10^5. The dimensionality is in the range of 2–4. Examples of these datasets consisting of 2×10^5 objects in two dimensions are shown in Fig. 11a–d.
Data indexing Each dataset is normalized in the range [0,1] and is indexed by our proposed grid-based index structure. We vary the grid size from 4 to 80.

Table 2 Values of different parameters used in experiments

Parameter	Values
Datasets	Real (CarDB), Synthetic (UN, CO, AC)
Data cardinality	$1 \times 10^5, 2 \times 10^5, 3 \times 10^5, 4 \times 10^5, 5 \times 10^5$
Dimensionality	2–4
Grid size	4–80
Threads	5–50
QTree samples [19]	1000
Split threshold [19]	40

Queries For each experiment we run a number of queries generated (synthetic) and selected (CarDB) randomly by following the distribution of the tested datasets.
Environment We perform our experiments in Swinburne High Performance Computing (HPC) system[2] with nodes 1–5, 10 processors per node (ppn) and 10GB main memory. All of our algorithms are implemented in Java utilizing its multi-threading packages. The nodes are selected by following the formula given below:

$$\underset{nodes}{\text{argmin}} \; (nodes - 1) \times ppn \leq (m + 1) \leq nodes \times ppn \quad (11)$$

where m is the number of workers (threads in Java). The datasets are indexed by the proposed grid-based data indexing scheme in a Windows PC with Intel(R) Core (TM) Duo 2.99 GHz CPU and 3.49 GB RAM. Table 2 summarizes the values of different parameters.

6.2 Effectiveness study

We know that the *market contribution* measure effectively combines both the customer and the manufacturer perspective into the same metric, which is (theoretically) more realistic than the *influence score* measure as demonstrated in Sects. 3.1 and 3.4. This section evaluates how effectively our approach trades-off between the *valued customers* and the

[1] https://autos.yahoo.com/.

[2] http://www.astronomy.swin.edu.au/supercomputing/.

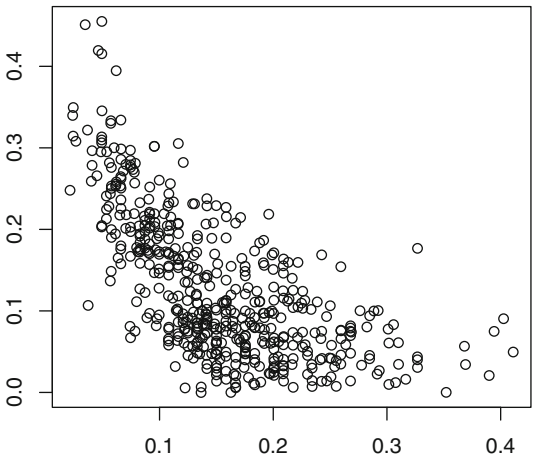


Fig. 12 CarDB products used in the effectiveness study

influence score. Recall that the valued customers are those customers in the market that are prone to prefer the manufacturer's own products as explained in Sect. 3.1.

To demonstrate the effectiveness of the proposed product selection model, we randomly select 500 products (i.e., cars) from the real CarDB dataset by following the distribution of the dataset as shown in Fig. 12[3]. Then, we issue various product selection queries, e.g., *k-MAC*, *k-MPP* and the greedy variations of them. The variation of *k-MAC* is the $(1 - \frac{1}{e})$ greedy approximation of the optimal *k-MAC* as explained by Arvanitis et al. in [1]. We consider the $k\text{-}MPP_{dep}$ (where the manufacturer's own products compete with each other to attract customers) and a greedy approximation of it, which is constructed as follows: (a) select a product q from Q with the maximum influence score (simply, the number of customers in its reverse skyline) and (b) keep adding product(s) into Q' from Q which has the maximum number of customers in common with the influence set of products already selected, if there is no such product left, then we choose the one with maximum influence score. The rationale of the above greedy approximation is that the selected products should increase the market contribution of the products already in the set (recall the proof of Theorem 2). This greedy version of *k-MPP* differs from the *k-MAC* one only in the sense that we select the next product in the set by maximizing the overlap of their influence set. Finally, we show the percentage of valued customers retained in our *k-MPP* and the existing *k-MAC* product selection models by setting value for k from 10 to 50.

From Fig. 13a, it is evident that our product selection model *k-MPP* outperforms other two models *k-MAC* and greedy *k-MPP* in terms of market contribution metric, which is expected as we model this in our approach. We also see that the greedy *k-MPP* is very close to the optimal *k-MPP* in terms of market contribution. However, our model is very close to the optimal *k-MAC* model in terms of influence score as we see from Fig. 13b. We already know that product selections are meant to increase the sales by provoking the customers in buying the manufacturer's own products, which is achieved by designing special product promotional events for the customers (recall from Sect. 1). This urges that we should make a good balance between the percentage of valued customers and the influence score while maximizing the market contribution so that the promotional costs are distributed among the actual customers. Figure 13c shows the % valued customers of different product selection models (% valued customers $= \frac{\text{market contribution}}{\text{influence score}}$), which demonstrates that our model can make a good trade-off between them. However, to achieve this we need to compute the market contribution of each query product $q \in Q$, which adds another level of complexity in our product selection model. In this paper, we design a specialized grid-based query-independent reusable data indexing scheme to expedite the performance of *k-MPP* by parallelizing the basic computational units for it. The efficiency of our approach is studied in Sects. 6.3 and 6.4.

6.3 Data indexing evaluation

We evaluate the proposed grid-based index structure in terms of *index-building time* and *index size* (i.e., size of the "grid-info.txt" file in disk). More specifically, we study the effect of data cardinality, dimensionality and the grid size on the above two metrics. Figure 14 shows the effect of different parameters on index-building time and size in disk in synthetic datasets UN, CO and AC. The index size in disk remains constant for the certain grid size and dimensions if the cardinality in customer and product datasets varies as shown in Fig. 14a–f. However, this is different for index-building time. The index-building time increases if the cardinality increases in either product or customer datasets or both. In general, the index-building time and the index size in disk increase if the dimensionality in the datasets and the grid size in the index structure increase as we see in Fig. 14g–l.

6.4 Performance study

This section studies the execution efficiency of processing *k-MPP* in parallel based on our index structure. More specifically, we study the effect of different parameter settings on the efficiency of the proposed approaches *SROND* (the Baseline), *SROFD* (a variation of the Baseline), *PROND* (the straightforward strategy) and *PROFD* (a variation of the straightforward strategy) as summarized in Table 3. We also compare the efficiency of our approach with the existing approach of computing dynamic and reverse skylines in parallel using quad-tree (QTree) indexing scheme [19].

[3] We find only 67 query products are non-dominating w.r.t. attracting customers in the market, i.e., $RSL(q) \neq \emptyset$.

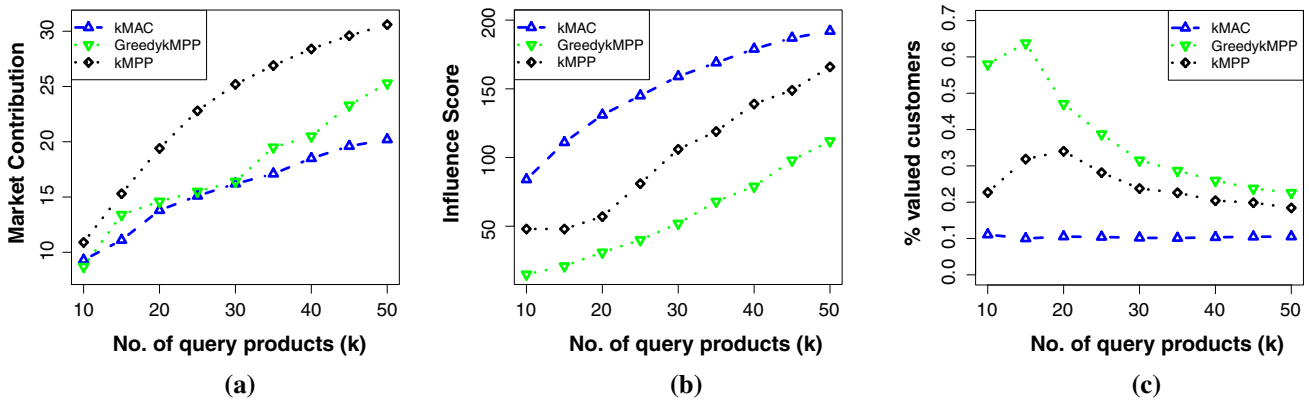


Fig. 13 The quality of results in different influence-based product selections models, **a** market contribution, **b** Influence Score, **c** %valued customers

As our *k-MPP* query involves both products and customer preferences, we extend the mono-chromatic reverse skyline computing technique given in [19] for bichromatic reverse skyline using the QTree index as follows: (a) a QTree node n_1 is marked as pruned for a $q \in Q$ if there exists a QTree node n_2 such that n_1 and n_2 are in the same orthant of q and $p_2 \in n_2$ dominates all corner points of n_1 w.r.t q; (b) all products $p \in P$ are scanned and used to compute midpoint skyline for q if not located in the pruned node and (c) finally, all $c \in C$ are scanned and marked as plausible customer for q (i.e., reverse skyline point) if not located in the pruned node of the QTree and not dominated by any midskyline point. We utilize 1000 samples (selected by applying the reservoir sampling method) and split threshold 40 to build the QTree for best performance as suggested in [19]. We compare two variants of this approach, i.e., *QPROND* and *QPROFD* as summarized in Table 3.

6.4.1 Effect of dimensionality

We study the effect of data dimensions on our proposed algorithms by setting the number of worker nodes (number of threads in the thread pool of Java) to *10*. We run experiments on 2–3 dimensional real CarDB datasets and 2–4-dimensional synthetic datasets. The cardinality is set to 1×10^5 for both products and customers. To index each tested dataset, the grid size (n) is established empirically. The value of k is set to 20 for $k\text{-}MPP_{dep}$. We run 100 queries following the distribution of the tested dataset. The results are shown in Fig. 15. We see that the execution time of every approach increases if the number of dimension increases in general. However, the proposed parallel algorithms *PROFD* and *PROND* take far less time compared to the baseline approaches, i.e., *SROFD* and *SROND* and significantly outperform the existing counterparts, i.e., *QROFD* and *QROND* for the increased number of dimensions in the datasets. The existing *QROFD* and *QROND* performs worst compared to our approach because of its query dependent data indices as per our analysis given in Sect. 4.3. The existing counterpart *QROND* is not scalable in higher dimensions as shown in Fig. 15b. These results demonstrate that the baseline approaches as well as the existing counterparts are not suitable for processing *k-MPP* queries. We conclude that the proposed reusable query-independent grid-based data indexing and our approach of processing *k-MPP* queries are efficient for processing *k-MPP* queries in parallel.

6.4.2 Effect of query products

This section studies the effect of cardinality in the query products on the execution time of our approaches. We run experiments on CarDB and CO datasets in two dimensions by varying the number of query products, $|Q|$, from 20–500, setting $|P| = 1 \times 10^5$, $|C| = 1 \times 10^5$, k to 25 for $k\text{-}MPP_{dep}$ and the number of worker nodes (threads in Java) to 20. The results are shown in Fig. 16. The grid size (n) is established empirically. From Fig. 16, we see that the execution time of every technique increases if the number of query products in Q increases. However, our approaches *PROFD* and *PROND* are much faster compared to the existing counterparts *QROFD* and *QROND*. The existing counterpart *QROND* is not scalable in terms of $|Q|$ due to its query dependent quad-tree index. On the other hand, our approaches are highly scalable in terms of $|Q|$.

6.4.3 Effect of cardinality in datasets

This section studies the effect of cardinality of product and customer datasets on the execution time of our approaches of processing *k-MPP* queries in parallel. We run experiments on UN dataset in three dimensions by varying the cardinality in both products P and customers C. We set the grid size $n = 10$, $|C| = 1 \times 10^5 \sim 5 \times 10^5]$, $|P| = 1 \times 10^5 \sim 4 \times 10^5]$, $|Q| = 100$, k to 25 for $k\text{-}MPP_{dep}$ and the workers (threads in

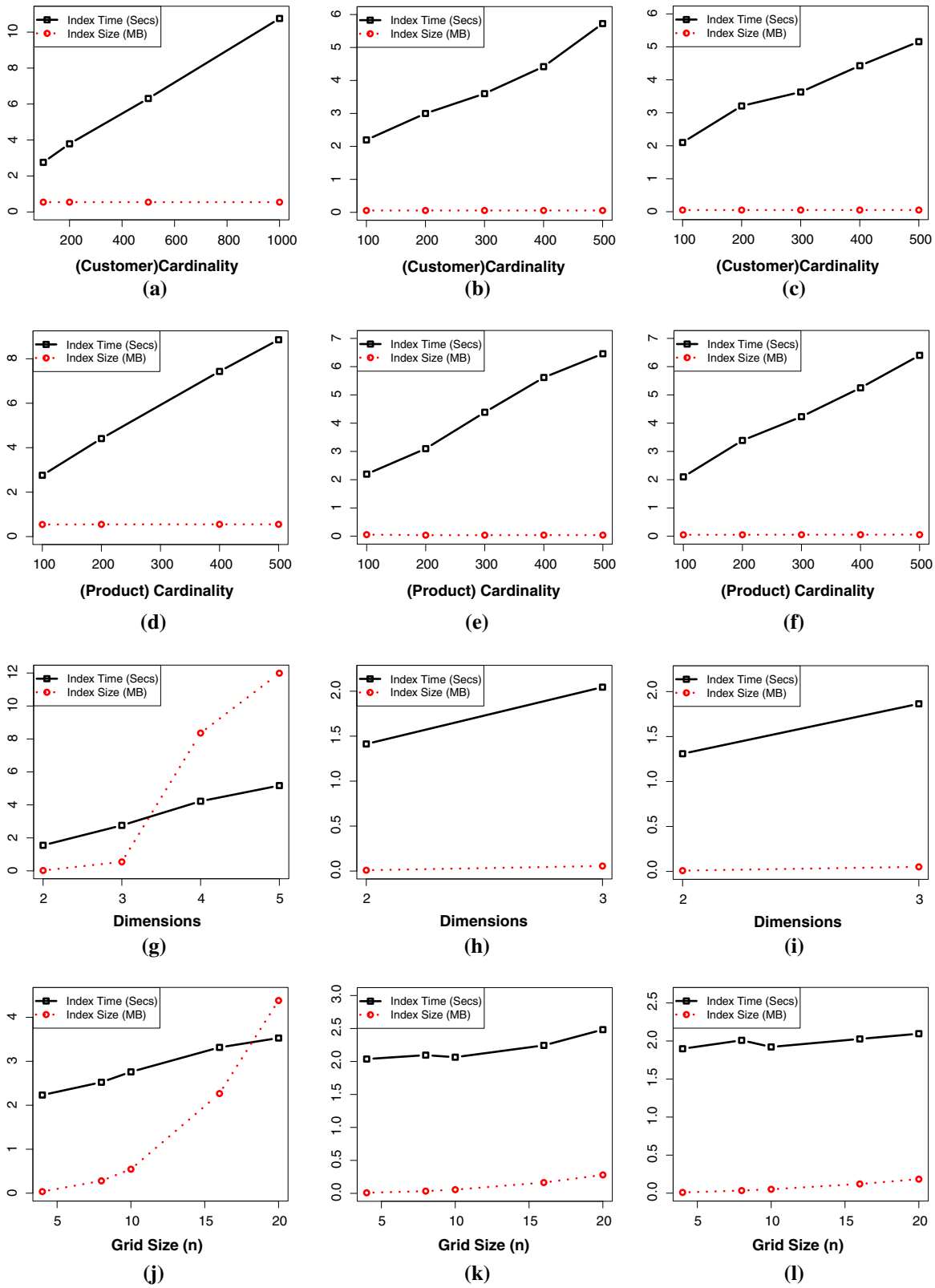


Fig. 14 Effect of **a–c** cardinality [$n = 10, d = 3, |P| = 1 \times 10^5$] , **d–f** cardinality [$n = 10, d = 3, |C| = 1 \times 10^5$] **g–i** dimensions [$n = 10, |D| = 2 \times 10^5$] and **j–l** grid [$d = 3, |D| = 2 \times 10^5$] on index-building time and size in synthetic datasets

Table 3 Different approaches of processing *k-MPP*

Acronym	Description
SROND	*S*erial *R*SL+ *ON*line DSL
SROFD	*S*erial *R*SL + *OF*fline *D*SL
PROND	*P*arallel *R*SL + *ON*line *D*SL
PROFD	*P*arallel *R*SL + *OF*fline *D*SL
QPROND	*P*arallel *R*SL + *ON*line *D*SL using *Q*Tree [19]
QPROFD	*P*arallel *R*SL + *OF*fline *D*SL using *Q*Tree [19]
PMROND	*P*arallel *M*ultiple *R*SL + *ON*line *D*SL
PMROFD	*P*arallel *M*ultiple *R*SL + *OF*fline *D*SL

Java) to 20. The results are shown in Fig. 17. We see that the execution times increase if the cardinalities in either products or customers increase. However, the execution times of our approaches are far less than the existing counterparts i.e., *QROFD* and *QROND*. These results demonstrate that our approaches are scalable in terms of cardinality increases in the datasets.

6.4.4 Effect of threads

This part of performance study investigates the effect of the number of threads on the execution time of processing *k-MPP* queries. We run experiments on two-dimensional real and synthetic datasets. The cardinality is set to 1×10^5 for both products and customers. To index each tested dataset, the grid size (n) is established empirically. We run 100 queries by following the distribution of the tested dataset and setting k to 25 for $k\text{-}MPP_{dep}$. The number of threads is varied from 5 to 50. The results are shown in Fig. 18. The execution times of both our approaches and the existing counterparts tend to decrease up to a certain number of threads, after that the execution times tend to increase again. This indicates that the overhead (delay due to the communication between the workers and the master and the synchronization between them) of processing *k-MPP* in parallel becomes more costly after a certain number of threads for a particular data and query settings.

6.4.5 Effect of data indexing

This section studies the effect of data indexing, i.e., grid size (n), on the execution times of our approaches. We run experiments on CardDB and UN datasets in two dimensions for varying grid size $n = 10 - 80$ by setting $|Q| = 100$, $|P| = 1 \times 10^5$, $|C| = 1 \times 10^5$, the workers (threads in thread pool of Java) to 20 and k to 25 for $k\text{-}MPP_{dep}$. The result is shown in Fig. 19. We observe that the execution time of each approach decreases for the increased grid size and it stabilizes after

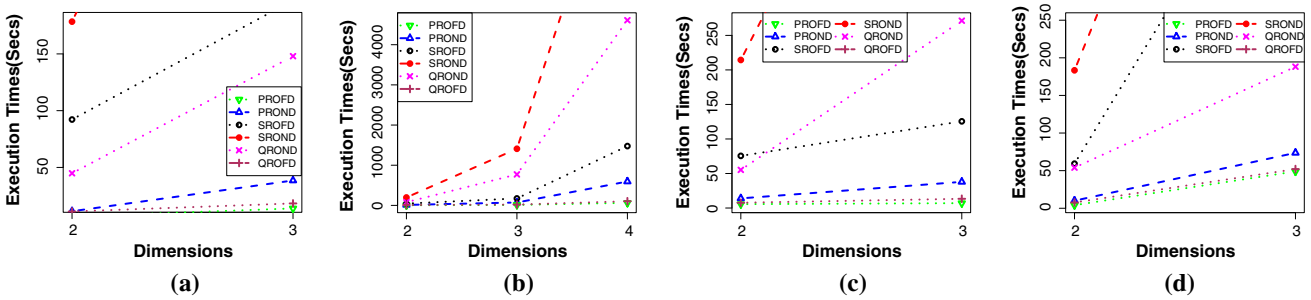


Fig. 15 Effect of dimensions on execution time in **a** CarDB, **b** UN, **c** CO and **d** AC datasets

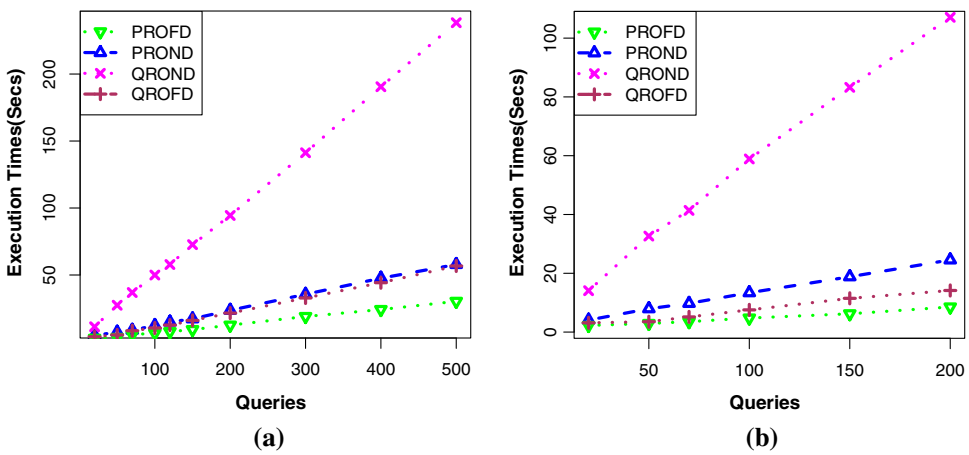


Fig. 16 Effect of queries ($|Q|$) on execution time in **a** CarDB and **b** CO datasets

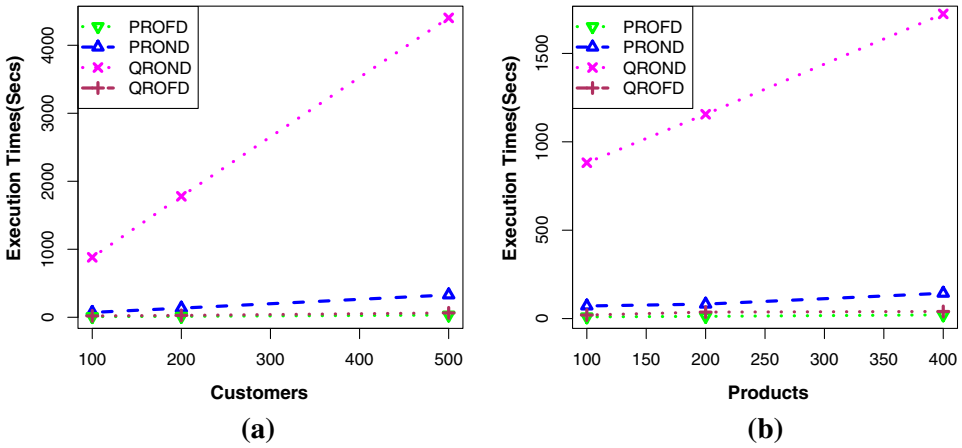


Fig. 17 Effect of cardinality on execution time in the UN dataset, **a** Customers versus time, **b** Products versus time

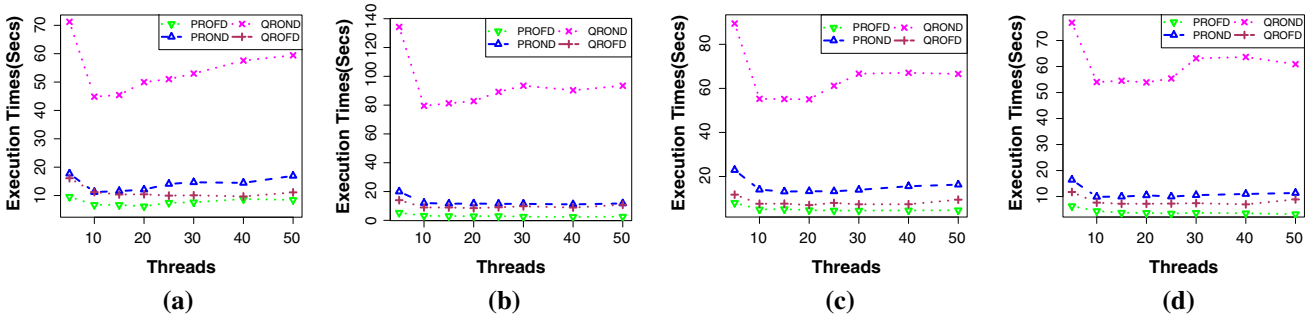


Fig. 18 Effect of threads on execution time in **a** CarDB, **b** UN, **c** CO and **d** AC datasets

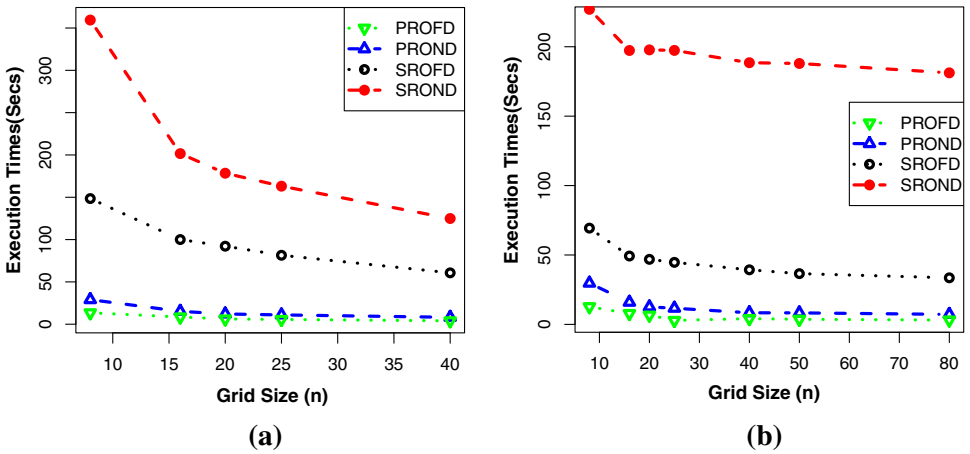


Fig. 19 Effect of grid size (n) on execution time in **a** CarDB and **b** UN datasets

certain limit. However, the index-building time and its size on disk also increase as explained in Sect. 6.3.

6.4.6 *The k-*MPP$_{ind}$ *versus k-*MPP$_{dep}$ *query*

This section evaluates the performance of our approaches of processing *k-MPP*$_{ind}$ and *k-MPP*$_{dep}$ queries by conducting two experiments. In the first experiment, we run 100 queries for each dataset in two dimensions by setting the grid size (n) empirically, $|P| = 1 \times 10^5$, $|C| = 1 \times 10^5$, the number of

Table 4 The efficiency (millisecs) of *k-MPP* variants

Dataset	PROFD		PROND	
	k-MPP$_{ind}$	*k-MPP*$_{dep}$	*k-MPP*$_{ind}$	*k-MPP*$_{dep}$
CarDB	6689	6192	12,017	12,093
UN	2772	2847	11,823	11,661
CO	4775	4734	13,360	13,302
AC	3860	3720	10,484	10,471

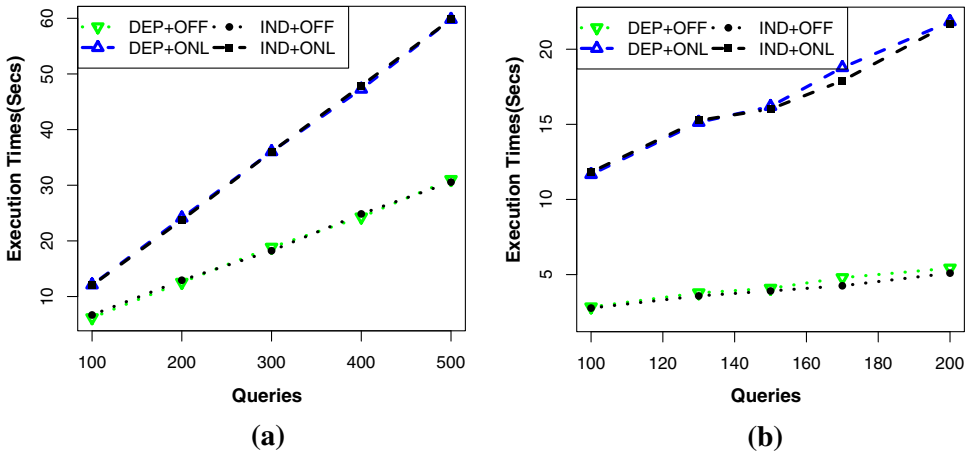


Fig. 20 Effect of queries ($|Q|$) on k-MPP_{ind} (IND) and k-MPP_{dep} (DEP) in **a** CarDB and **b** UN datasets, where OFF is Offline DSL and ONL is Online DSL

threads to 20 and the k to 25 for k-MPP. The results are shown in Table 4. We conduct another experiment on CarDB and UN datasets in two dimensions by varying $|Q|$ from 100 to 500 and 100 to 200, respectively, with similar parameter settings and the result are shown in Fig. 20. In these experiments, we do not observe any significant difference in k-MPP_{ind} and k-MPP_{dep}'s execution times. This is because, we firstly compute the midpoint skyline of q based on P only and then, we update it by performing the dominance tests with Q to finalize it for k-MPP_{dep}. We also follow similar approach for computing $DSL(c)$ of $c \in RSL(q)$ for k-MPP_{dep}. The above significantly reduces the pairwise dominance tests performed for k-MPP_{dep}. We may also have different $RSL(q)$ for k-MPP_{dep} than k-MPP_{ind} and therefore, different number of DSL computations. However, the difference in their efficiencies may become significant for large $|Q|$.

6.4.7 Evaluation of optimized strategy

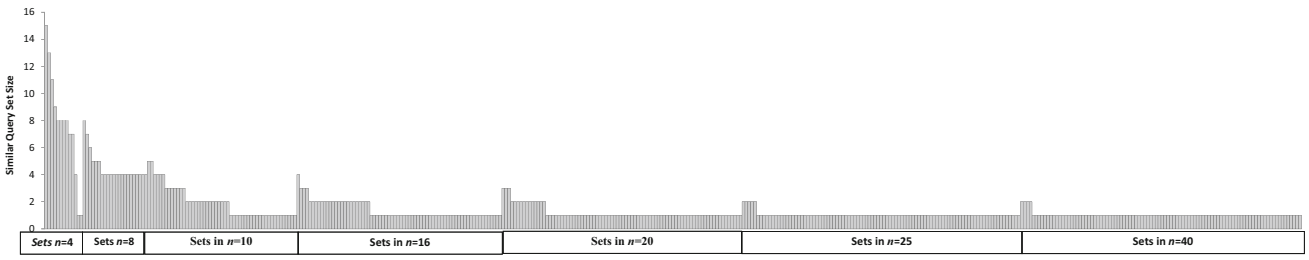
This section evaluates the efficiency of the optimized strategy for processing k-MPP queries in parallel. More specifically, we compare the performances of the following approaches: *PMROND* (a variation of the optimized strategy) and *PMROFD* (the optimized strategy) as summarized in Table 3. We create a number of clustered queries (to increase the similarities of their search spaces) by setting $|Q| = 100$ for UN and $|Q| = 200$ for real CarDB datasets in two dimensions and varying the grid size from 4 to 40. However, the distributions of $q \in Q$ in the resultant clusters (i.e., group of similar queries) are not uniform as shown in Fig. 21. We conduct two experiments to evaluate the performance of the optimized strategy as given below.

Effect of data indexing We study the effect of grid size (n) on the efficiency of the optimized strategy by setting k to 25 for k-MPP_{dep}, $|P| = 1 \times 10^5$, $|C| = 1 \times 10^5$ and the number of threads to 20. The results are shown in Fig. 22. We see that the optimized strategies PMROND and PMROFD take less time compared to the straightforward strategies PROND and PROFD for lower grid sizes (n), except when the distributions of $q \in Q$ in the clusters are too skewed, e.g., the first few clusters contain most of the query products $q \in Q$ in CarDB for $n = 8$ as Fig. 21b and the optimized strategies perform worst (i.e., the responsible thread is heavily loaded). However, the optimized strategy tends to perform similarly to the straightforward strategy for the increased grid sizes (n) as we cannot form clusters of similar queries to share their search spaces and thereby, computations to expedite the efficiency.

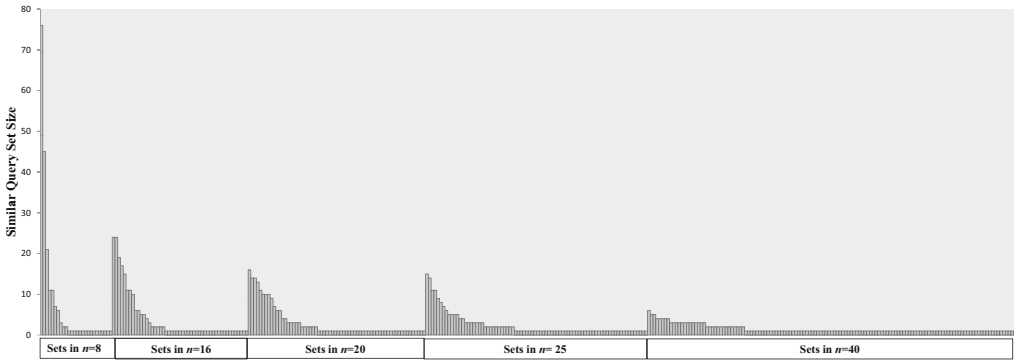
Effect of threads We study the effect of threads in Java on the efficiency of the optimized strategy again in UN and CarDB datasets by setting the grid size (n) for them to 8 and 25, respectively. We set the other parameters similar to the first experiment. The results are shown in Fig. 23. We see that the optimized as well as the straightforward strategies achieve the best efficiency when the number of threads set in the range 10 to 20 for the CarDB dataset. However, we see an exception in the UN dataset where the optimized strategy PMROND stabilizes when the number of threads set in the range 10–20, then again offers a further improvement on the efficiency when the number of threads set to 40 as shown in Fig. 23a. We leave the optimization of these parameters as future research direction.

7 Related work

The first work in dominance-based data retrieval was the skyline query proposed by Börzsönyi et al. [2]. Since then, this work has received lots of attention among the community and is studied extensively [4,18,22,23,29] due to its potential in multicriteria decision making applications. Our product adoption model and the product selection strategy is dominance-based and the basic computational units comprise of the proposed k-MPP query are dynamic and reverse skylines. This section briefly describes the works that either



(a)



(b)

Fig. 21 Effect of grid size (n) on similar query set (cluster) sizes in **a** UN and **b** CarDB datasets

study the dominance-based customer–product relationships [1,8,9,11,13,29] or offer techniques for parallelizing the skylines [10,17,19,21,24,31,32].

7.1 Customer–product relationship

There are a number of works that study the computational aspects of customer–product relationships. Li et al. [11] propose a data cube framework called DADA to analyze dominance relationships from a microeconomic perspective. The framework is aimed to provide insights about the dominant relationships between the products and their potential buyers and supports three types of dominant relationship queries, e.g., (a) Linear Optimization Queries, (b) Subspace Analysis Queries, and (c) Comparative Dominant Queries. Given a set of products P and a set of customers C, Wu et al. [29] propose an improved algorithm for computing the influence set of a query product q. The influence of the query product q is measured as the cardinality of the reverse skyline of q termed as influence set for q, i.e., $IS(q) = |RSL(q)|$. In [1], Arvanitis et al. propose an approach for computing k-Most Attractive Candidates (k-*MAC*). Given a set of candidate query products Q and an integer $k > 1$ (as well as P and C as in [29]), k-MAC query discovers the k-most attractive candidate set $Q' \subseteq Q$ such that $|Q'| = k$ and the joint influence score of Q', defined as $IS(Q') = |\bigcup_{q \in Q'} RSL(q)|$, is maximized. In these two works, every customer c appearing in the $RSL(p)$ is assumed to contribute 100 % for the sustainment of the product p in the market.

In [13], Lin et al. propose an approach for selecting k products from a set of candidate products such that the expected number of the total customers is maximized known as k-most demanding products (k-*MDP*) discovering. The authors propose an exact algorithm for k-MDP by estimating the upper bound of the expected number of the total customers. They also offer a greedy algorithm which is scalable w.r.t. k. In [9], Koh et al. presents an approach of computing k-most favorite products based on reverse top-t queries, which is NP-hard. They design an incremental greedy approach to find an approximate solution with guaranteed quality exploiting the properties of the top-t queries and skyline queries. In [30], Xu et al., propose a product adoption model and a greedy-based approximation algorithm for selecting k products that can maximize the sales for the manufacturer. In [28] Wu et al. propose approaches for discovering the promotive subspaces in which the product objects becomes prominent.

In [27] Wu et al. propose efficient approaches for processing region-based promotion queries that can discover the top-k-most interesting regions for effective promotion of a product object in which it is top-ranked. In [14,15], Miah et al. propose approaches for finding the best set of attributes of a new product so that it can stand out in the crowd of existing competitive products. Given a set of existing products P and a set of given products Q, Wan et al. [25,26] and Peng et al. [20] propose approaches for finding a set of k best possible

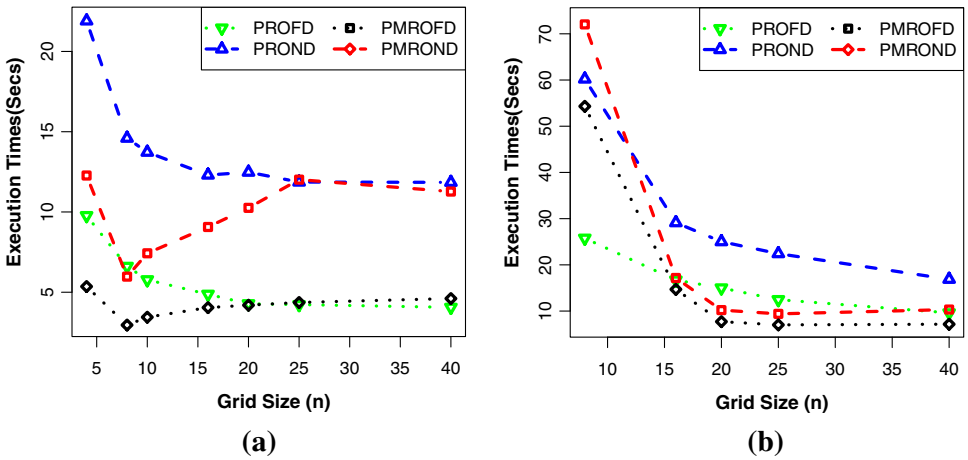


Fig. 22 Effect of grid size (n) on the efficiencies of the optimized strategy in **a** UN and **b** CarDB datasets

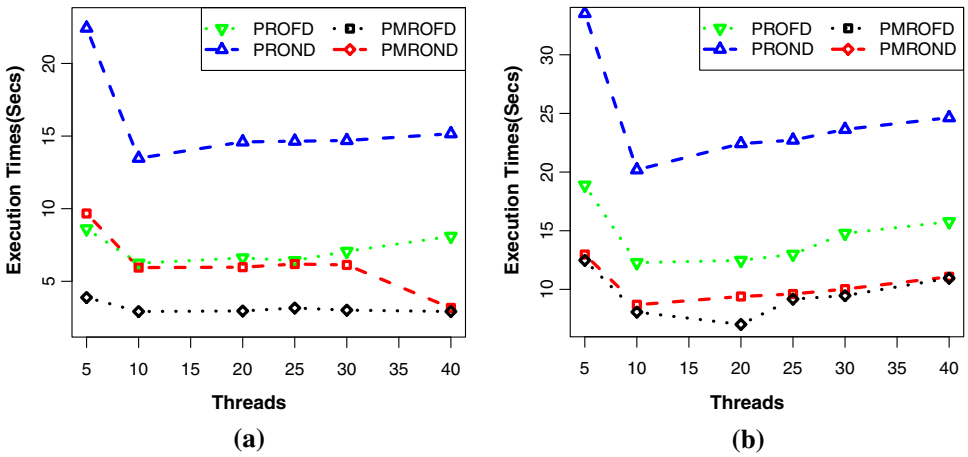


Fig. 23 Effect of threads on the efficiencies of the optimized strategy in **a** UN and **b** CarDB datasets

products from Q such that this subset of products of Q are not dominated by the products in P. They describe several variants of this problem and provide solutions for them. In [8] Islam et al. propose an approach of establishing an automatic negotiation between a customer and a product by modifying some of the product's attributes (e.g., price) and customer preferences for those attributes with minimum penalty.

In our work, we show that a customer does not contribute 100 % for the sustainment of a product in the market, rather only a fraction of it. Considering this, we propose a novel probability-based product adoption model for the customer and a product selection strategy for the manufacturer with maximum expected number of attracted customers based on dynamic and reverse skylines. We also propose a new type of query called finding *k-MPP* and a solution approach for it.

7.2 Computing skylines in parallel

There are a number of works on parallelizing the processing of skyline queries. However, none of them are suitable for parallelizing the execution of *k-MPP* query. In [24], Vlachou et al. propose an angle-based space partitioning for skyline query processing in a parallel using the hyperspherical coordinates of the data points. In [32], Zhang et al. present an object-based space partitioning approach for processing skyline queries in parallel. In [10], Kohler et al. propose a hyperplane data projection technique for computing skyline in parallel which is independent of data distribution. In [31], Zhang et al. propose MR-BNL algorithm by partitioning the data space into 2^d subspaces according to the median of each dimension and then, computing the local skyline of every subspace in parallel. Finally, the global skyline is computed in a single machine from all the local skylines. In [17], Mullesgaard et al. propose grid-based data partitioning scheme for computing skyline in MapReduce. In [21], Pertesis and Doulkeridis propose an approach of processing skyline query in SpatialHadoop. However, these approaches are not suitable for dynamic/reverse skyline query processing and thereby, cannot solve our problem.

The work in [19] parallelizes a single dynamic/reverse skyline query using quad-tree index, not multiple reverse skylines. The quad-tree index is also query dependent and does not allow multiple queries to be grouped together. Therefore, this approach is not suitable for processing *k-MPP* query in parallel. In our work, we propose an approach for comput-

ing *k-MPP* query in parallel by designing a simple yet very efficient query-independent grid-based index structure. We also provide an approach for grouping queries based on their similarities and processing them together with parallelize the processing of *k-MPP*.

8 Conclusion

This paper presents an efficient approach for processing *k-MPP* product selection query and its variants. We design a simple yet very efficient query-independent grid-based index structure to partition the data space. We also establish the partitionwise dominances and several theoretical properties to reduce the search spaces of the basic computational units and filter the non-resultant data objects for computing *k-MPP* query in parallel. We also show how to improve the efficiency further by grouping queries based on their extended search spaces and processing them together. The effectiveness and efficiency of the proposed product selection model are demonstrated by conducting extensive experiments. Our model adds another level of assurance to identify the valued customers in the influence-based market research queries, which can be studied further.

Acknowledgments The work is supported by the ARC Discovery Grant DP140103499. We would like to thank Dr. Robin Humble for his help on optimizing the performance of our programs in Swinburne HPC system and the anonymous reviewers for their insightful feedback.

References

1. Arvanitis, A., Deligiannakis, A., Vassiliou, Y.: Efficient influence-based processing of market research queries. In: CIKM, pp. 1193–1202 (2012)
2. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: ICDE, pp. 421–430 (2001)
3. Chang, K.C., Hwang, S.: Minimal probing: supporting expensive predicates for top-k queries. In: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, June 3–6, 2002, pp. 346–357 (2002)
4. Dellis, E., Seeger, B.: Efficient computation of reverse skyline queries. In: VLDB, pp. 291–302 (2007)
5. Deshpande, P.M., Padmanabhan, D.: Efficient reverse skyline retrieval with arbitrary non-metric similarity measures. In: EDBT, pp. 319–330 (2011)
6. Ester, M., Ge, R., Jin, W., Hu, Z.: A microeconomic data mining problem: customer-oriented catalog segmentation. In: KDD, pp. 557–562 (2004)
7. Frischknecht, B., Gonzalez, R., Papalambros, P., Reid, T.: A design science approach to analytical product design. In: International Conference on Engineering Design, 24–27 (2009)
8. Islam, M.S., Zhou, R., Liu, C.: On answering why-not questions in reverse skyline queries. In: ICDE, pp. 973–984 (2013)
9. Koh, J.-L., Lin, C.-Y., Chen, A.L.P.: Finding k most favorite products based on reverse top-t queries. VLDB J. **23**(4), 541–564 (2014)
10. Köhler, H., Yang, J., Zhou, X.: Efficient parallel skyline processing using hyperplane projections. In: SIGMOD, pp. 85–96 (2011)
11. Li, C., Ooi, B.C., Tung, A.K.H., Wang, S.: DADA: a data cube for dominant relationship analysis. In: SIGMOD, pp. 659–670 (2006)
12. Lian, X., Chen, L.: Monochromatic and bichromatic reverse skyline search over uncertain databases. In: SIGMOD, pp. 213–226 (2008)
13. Lin, C.-Y., Koh, J.-L., Chen, A.L.P.: Determining (k)-most demanding products with maximum expected number of total customers. IEEE Trans. Knowl. Data Eng. **25**(8), 1732–1747 (2013)
14. Miah, M., Das, G., Hristidis, V., Mannila, H.: Standing out in a crowd: selecting attributes for maximum visibility. In: ICDE, pp. 356–365(2008)
15. Miah, M., Das, G., Hristidis, V., Mannila, H.: Determining attributes to maximize visibility of objects. IEEE Trans. Knowl. Data Eng. **21**(7), 959–973 (2009)
16. Michalek, J.J., Feinberg, R.M., Papalambros, P.Y.: An optimal marketing and engineering design model for product development using analytical target cascading. In: Proceedings of the TMCE, April 12–16 (2004)
17. Mullesgaard, K., Pederseny, J.L., Lu, H., Zhou, Y.: Efficient skyline computation in MapReduce. In: EDBT, pp. 37–48 (2014)
18. Papadias, D., Tao, Y., Fu, G., Seeger, B.: An optimal and progressive algorithm for skyline queries. In: SIGMOD, pp. 467–478 (2003)
19. Park, Y., Min, J.-K., Shim, K.: Parallel computation of skyline and reverse skyline queries using MapReduce. PVLDB **6**(14), 2002–2013 (2013)
20. Peng, Y., Wong, R.C., Wan, Q.: Finding top-k preferable products. IEEE Trans. Knowl. Data Eng. **24**(10), 1774–1788 (2012)
21. Pertesis, D., Doulkeridis, C.: Efficient skyline query processing in SpatialHadoop. Inf. Syst. **54**, 325–335 (2015)
22. Sharifzadeh, M., Shahabi, C.: The spatial skyline queries. In: VLDB, VLDB Endowment, pp. 751–762 (2006)
23. Tao, Y., Ding, L., Lin, X., Pei, J.: Distance-based representative skyline. In: ICDE, pp. 892–903 (2009)
24. Vlachou, A., Doulkeridis, C., Kotidis, Y.: Angle-based space partitioning for efficient parallel skyline computation. In: SIGMOD, pp. 227–238 (2008)
25. Wan, Q., Wong, R.C., Ilyas, I.F., Özsu, M.T., Peng, Y.: Creating competitive products. PVLDB **2**(1), 898–909 (2009)
26. Wan, Q., Wong, R.C., Peng, Y.: Finding top-k profitable products. In: ICDE, pp. 1055–1066 (2011)
27. Wu, T., Sun, Y., Li, C., Han, J.: Region-based online promotion analysis. In: EDBT, pp. 63–74 (2010)
28. Wu, T., Xin, D., Mei, Q., Han, J.: Promotion analysis in multi-dimensional space. PVLDB **2**(1), 109–120 (2009)
29. Wu, X., Tao, Y., Wong, R.C.-W., Ding, L., Yu, J.X.: Finding the influence set through skylines. In: EDBT, pp. 1030–1041 (2009)
30. Xu, S., Lin, Y., Xie, H., Lui, J.C.S.: A provable algorithmic approach to product selection problems for market entry and sustainability. In: SSDBM, p. 19 (2014)
31. Zhang, B., Zhou, S., Guan, J.: Adapting skyline computation to the MapReduce framework: Algorithms and experiments. In: DASFAA, pp. 403–414, Springer, Berlin (2011)
32. Zhang, S., Mamoulis, N., Cheung, D.W.: Scalable skyline computation using object-based space partitioning. In: SIGMOD, pp. 483–494 (2009)