Finding Optimal Skyline Product Combinations under Price Promotion

Xu Zhou, Kenli Li, *Senior Member, IEEE*, Zhibang Yang, and Keqin Li, *Fellow, IEEE*

Abstract—Nowadays, with the development of e-commerce, a growing number of customers choose to go shopping online. To find attractive products from online shopping marketplaces, the skyline query is a useful tool which offers more interesting and preferable choices for customers. The skyline query and its variants have been extensively investigated. However, to the best of our knowledge, they have not taken into account the requirements of customers in certain practical application scenarios. Recently, online shopping marketplaces usually hold some price promotion campaigns to attract customers and increase their purchase intention. Considering the requirements of customers in this practical application scenario, we are concerned about product selection under price promotion. We formulate a constrained optimal product combination (COPC) problem. It aims to find out the skyline product combinations which both meet a customer's willingness to pay and bring the maximum discount rate. The COPC problem is significant to offer powerful decision support for customers under price promotion, which is certified by a customer study. To process the COPC problem effectively, we first propose a two list exact (TLE) algorithm. The COPC problem is proven to be NP-hard, and the TLE algorithm is not scalable because it needs to process an exponential number of product combinations. Additionally, we design a lower bound approximate (LBA) algorithm that has a guarantee about the accuracy of the results and an incremental greedy (IG) algorithm that has good performance. The experiment results demonstrate the efficiency and effectiveness of our proposed algorithms.

Index Terms—Data management, price promotion, skyline query, NP-hard

1 Introduction

With the development of e-commerce, a growing number of customers choose to go shopping online because it saves time and effort. However, it always contraries to expectations of customers. This is because they may need to pick up one choice among thousands of products. To help customers identify attractive products, a skyline query is admittedly a common and effective methodology. According to the definition of the skyline query [1], a product which is not dominated by any other product is said to be a skyline product or it is in the skyline. The products in the skyline are the best possible trade offs between all the factors that customers care about. The skyline query is useful in identifying attractive products.

In Jingdong and Alibaba's Taobao Mall which are the most famous online shopping malls in China, there are many online stores that specialize in one category of products such as red wine, watches, television, laptop, to name just a few. During the weekends or holidays, these stores usually hold some price promotion campaigns to boost consumption. Under the price promotion campaigns of these stores, a customer could select an optimal product combination by himself. Besides, the customer is common to participate in cooperation with his families or friends for group-buying.

The present price promotion campaigns can be classified into two categories due to whether products can be chosen independently. The first category, namely, independent-product selection, includes the campaigns such as "buy one product and get another product for free" and "25% discount for two pics" etc. Under these campaigns, customers can pick out the products meeting their demands independently and directly, and skyline queries could offer powerful decision support. The second category, namely, dependent-product selection, consists of the campaigns such as "get \$60 off every \$200 purchase" and "\$100 coupon every \$500 purchase" etc. In these scenarios, customers always expect to select products which are attractive and bring the greatest benefit. Moreover, it needs to take into consideration the customer's willingness to pay which is an important issue that affects the customer's purchasing behavior. The skyline query is powerful to compute the skyline products that have a strong appeal to customers. However, it is inadequate to help customers select skyline product combinations with the greatest benefit.

Considering the requirements of customers in this practical application scenario, we are concerned about a new problem of identifying optimal product combinations under price promotion campaigns. In this paper, we focus on the dependent-product selection campaigns that are much more popular but complicated with comparison to the independent-product selection campaigns.

Assume that Jingdong offers a price promotion campaign which is "get \$60 off every \$200 purchase" (we will use this price promotion campaign in all the remaining examples). It has a French wine set $W = \{w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8\}$

- *X. Zhou, K. Li, and K. Li are with the College of Information Science and Engineering, Hunan University, and the National Supercomputing Center in Changsha, Hunan 410082, China. E-mail: zhouxu2006@126.com, lkl@hnu.edu.cn, lik@newpaltz.edu.*
- *Z. Yang is with the Department of Mathematics and Computer Science, Changsha University, Changsha, Hunan 410082, China. E-mail: yangzhibang2006@126.com.*

Manuscript received 22 June 2017; revised 19 Feb. 2018; accepted 28 Mar. 2018. Date of publication 6 Apr. 2018; date of current version 5 Dec. 2018. (Corresponding author: Kenli Li and Keqin Li.)
Recommended for acceptance by J. Levandoski.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2018.2823707

TABLE 1
The Candidate Wines

Wine	Class	Praise Degree	Original Price($)
w_1	4	50%	450
w_2	3	60%	340
w_3	1	65%	300
w_4	4	75%	240
w_5	2	85%	190
w_6	3	80%	210
w_7	3	55%	360
w_8	1	90%	180

for sale as illustrated in Table 1. We take three attributes of each wine, which are class, praise degree, and original price, into account. The French wines are usually divided into four classes which are 1. Vin de France (VDF), 2. Vin de Pays (VDP), 3. Vin Delimitesde Qualite Superieure (VDQS), and 4. Appellation d'Origine Protegee (AOP). Without loss of generality, for the two attributes, class and praise degree, of the French wines, large values are considered to be preferable over small ones. For the original price, small value is better than large one.

In order to find out the wines, which are attractive to customers, the skyline query is one of the most useful tools. In table 1, the wine w_4 dominates the wine w_2 since its class and praise degree are larger and the original price is smaller. Similarly, the wines w_1 and w_3 are dominated by the wine w_4. The wine w_7 is dominated by the wine w_6. After the skyline query over the wine dataset in Table 1, we get a skyline set $\{w_4, w_5, w_6, w_8\}$, where each wine is not dominated by any other one. All these wines in the skyline set offer more interesting and preferable choices for customers.

Although the skyline query is helpful to compute attractive wines, it is inadequate to gain the optimal wine combinations under price promotion campaigns. In practice, the customer always chooses product combinations with a payment constraint. Continuing with the example in Table 1, if the customer's willingness to pay is \$400 at most, then it only needs to process the wine combinations in Table 2 whose actual payments are not more than \$400. The other wine combinations including $\{w_4, w_5, w_6\}$, $\{w_4, w_5, w_8\}$, $\{w_4, w_6, w_8\}$, $\{w_5, w_6, w_8\}$, and $\{w_4, w_5, w_6, w_8\}$ are out of consideration since their actual payments exceed the customer's willingness to pay. In Table 2, discount denotes the discount of each wine combination, actual payment is computed as $Original\ Price - Discount$, and discount rate is $\frac{Discount}{Original\ Price}$. For an example, the discount of wine combination $\{w_5, w_6\}$ is $\lfloor\frac{190+210}{200}\rfloor \times 60 = 120$, the actual payment is $400-120 = 280$, and the discount rate is $\frac{120}{400} = 0.300$.

Customer demands are diversification and individuation. When selecting products under price promotion, apart from payment constraints, customers are common to have demands such as maximum discount rate, spending or saving the most money, to name just a few. As illustrated in Table 2, the wine combination $\{w_5, w_6\}$ is with the maximum discount rate. Moreover, the customer can save the most money when selecting from the wine combinations $\{w_4, w_5\}, \{w_4, w_6\}, \{w_4, w_8\}$, and $\{w_5, w_6\}$. If the customer wants to spend money as much as possible, the wine combinations $\{w_4, w_6\}$ and $\{w_6, w_8\}$ are great choices.

In this paper, we measure the product combinations by their discount rates, and focus on how to combine homogenous products under price promotion campaigns. Besides, in some cases, customers may need to select products from different categories, this is out of the scope in this paper.

TABLE 2
The Candidate Wine Combinations

Wine Combination	Original Price ($)	Discount ($)	Actual Payment ($)	Discount Rate
$\{w_4\}$	240	60	180	0.250
$\{w_5\}$	190	0	190	0.000
$\{w_6\}$	210	60	150	0.286
$\{w_8\}$	180	0	180	0.000
$\{w_4, w_5\}$	430	120	310	0.279
$\{w_4, w_6\}$	450	120	330	0.267
$\{w_4, w_8\}$	420	120	300	0.286
$\{w_5, w_6\}$	400	120	280	0.300
$\{w_5, w_8\}$	370	60	310	0.162
$\{w_6, w_8\}$	390	60	330	0.154

In the literature, the closely related researches to our problem are group skyline queries [2], [3], [4], [5], [6], [7], [8], [9] and skyline queries under constraints [10], [11], [12], [13], [14], [15], [16]. However, they cannot be applied to our problem directly as analyzed in Section 2. To the best of our knowledge, we study product selection under price promotion for the first time in the literature. Our contributions are briefly summarized as follows.

- We devise the COPC problem. This problem aims to find skyline product combinations which meet a customer's payment willingness and bring the maximum discount rate. We prove the COPC problem is NP-hard.
- We propose an exact algorithm, namely two list exact algorithm, for the COPC problem. Besides, we design a lower bound approximate algorithm, which has guarantee about the accuracy of the results. To get better performance, we develop the incremental greedy algorithm for the COPC problem.
- We introduce how to extend the proposed approaches to handle the corresponding problem under other price promotions and discuss two variants of the COPC problem by taking into account different customer demands.
- We conduct a small customer study to verify the significant of our COPC problem and perform an extensive experimental study to clarify the effectiveness and efficiency of all the proposed algorithms.

The rest of the paper is organized as follows. In Section 2, we review the related work. In Section 3, we formulate the COPC problem. In Section 4, we propose effective algorithms to solve the COPC problem. In Section 5, we discuss the way of extending the proposed approaches to handle the corresponding problem under other price promotions and introduce two variants of the COPC problem. In Section 6, we evaluate the performance of the proposed algorithms by extensive experiments. In Section 7, we conclude the paper and also expatiate the directions for future work.

2 Related Work

As an important data management operator [17], the skyline query and its variants has received a great attention in the literature [10], [18], [19], [20], [21]. In our COPC problem, it computes the optimal skyline product combinations with a constraint, which is the customer's willingness to pay.

The closely related problems are group skyline queries and skyline queries under constraints, and the related works are reviewed in this section.

2.1 Group Skyline Queries

The skyline query aims to return the points that are not-dominated by any other point [1]. However, most of the works about the skyline query just analyze individual points, and they are inappropriate to many applications that call for analysis of groups of different points. Motivated by this, group skyline queries are developed and paid growing attention.

In most of the group skyline queries, optimal groups are computed by the dominance relationship between corresponding aggregate-based points of different groups. Su et al. [3] formulated top k combinatorial skyline query (k-CSQ). It returns those combinatorial skyline tuples whose aggregate values for a certain attribute are maximum. Since only the first k combinations are required, the k-CSQ query process can be simplified [4]. Chung et al. [4] extended the traditional skyline queries and formulated a combinatorial skyline query, namely CSQ, which is to find the outstanding skyline combinations. Im et al. [5] studied the group skyline query which is based on the dominance relationship between the groups of the same size. The dominance relationship is checked according to the aggregate values of attributes. Magnani et al. [6] introduced aggregate skylines, where the skyline works as a filtering predicate on sets of records. The aggregate skyline queries merge the functionalities of two basic database operators, skyline and group by. Zhang et al. [7] focused on a novel problem of groups of k tuples, which are not dominated by any other group of equal size, based on aggregate-based group dominance relationship. They also identified two anti-monotonic properties to filter out candidate groups. Wu et al. [2] researched the work which is similar to the group skyline computation. They focused on a problem of creating competitive products which are not dominated by the products in the existing market. Here each new product is generated by combing products from different source tables. To reduce the search space, it only combines the skyline products from source tables to generate new products. In addition, they also presented an approach which divides similar products into groups and processes them as a whole.

In the group skyline queries aforementioned, the aggregate values of corresponding attributes are taken into account when checking the dominance relationship between different groups. However, it is difficult for customers to specify an appropriate aggregate function. Moreover, it overlooks many significant groups that may contain non-skyline points [8]. Motivated by the problems which the group skyline queries face, Liu et al. [8] presented a Pareto group-based skyline (G-skyline) query which retrieves G-skyline groups. These G-skyline groups are not g-dominated by any other group of the same size. They presented a directed skyline graph which captures the dominant relationship of tuples within the first k skyline layers. Furthermore, two heuristic algorithms are designed to solve the G-skyline query effectively. Recently, Yu et al. [9] defined the Multiple Skyline Layers, and presented two fast algorithms to compute the G-skylines due to the observation that skyline points contribute more to skyline groups compared to non-skyline points.

The above approaches for group skyline queries are inappropriate for our problem. This is since they filter out the optimal groups by the dominance relationship, but the skyline product combinations in our COPC problem are measured by the maximum discount rate.

2.2 Skyline Queries under Constraints

Skyline query is a useful tool to find out attractive products which offer more interesting and preferable choices for customers. However, the size of the skyline query results cannot be controlled flexibly. Accordingly, many research efforts have been devoted to contend with this problem. The existing approaches to address this problem are developed to identify k representative skylines which have the maximum dominant capacity or the maximum diversification.

Lu et al. [10] were concerned about the case when the actual cardinality of skyline results is less than the desired result cardinality k. They proposed a new approach, namely skyline ordering, which forms a skyline-based partitioning of a given dataset. Then they applied a set-wide maximization technique, which is used to find an object set dominating the largest number of points, to process each partition. Bai et al. [19] focused on how to choose k representative skylines over data streams. Gao et al. [18] formulated the most desirable skyline object query which reports the most preferable k skylines based on a new ranking criterion. Lin et al. [12] studied the problem of selecting k skyline points such that the number of points, which are dominated by at least one of these k skyline points, is maximized. Papadias et al. [11] proposed the K-dominating query which retrieves K points dominating the largest number of other points. This query does not necessarily contain skyline points but has the advantages of both ranking queries and skyline queries, which are with the control on the size of the answer set and without users' efforts to specify ranking functions. Chan et al. [15] formulated a top k ranking problem which retrieves points appearing frequently in the subspace skylines. Wan et al. [14] identified the problem of finding top-k profitable products. Given a set of packages in the existing market and a set of potential new packages, they wanted to select k new packages such that the sum of the profits of the selected packages is maximized and each selected package is not dominated by any package in the existing market and any selected new package. Lin et al. [13] proposed a k-most demanding products (k-MDP) discovering problem that helps the company to select k products from the candidate products with the maximum expected number of the total customers. In [20], we developed a top k favorite probabilistic product query. It is utilized to select k products which can meet the needs of a customer set at the maximum level.

The representative skyline queries aforementioned only highlight some features which are stability, scale invariance, diversification of the results, and partial knowledge of the record scoring function [16]. Magnani et al. [16] focused on the representative skyline queries in terms of both the significance and diversity of results. This representative skyline query can satisfy all the features above. Moreover, there are also some other approaches to avoid returning many skyline query results. In [22], Chen et al. studied a constrained skyline query. It first filters out the points which do not satisfy range constraints of each attributes, and then processes the skyline query over the remaining points.

To our best knowledge, this is the first attempt to research product selection under price promotion. The COPC problem aims to find optimal skyline product combinations whose actual payments are not beyond the customer's willingness to pay. Moreover, these optimal skyline product combinations

TABLE 3
The Summary of Frequently Used Notations

Notation	Definition
P	the product dataset
N	the size of the product dataset
SP	the skyline product set over P
N_S	the size of SP
$SP^{'}$	the skyline product combination with $SP^{'} \subseteq SP$
WTP	a customer's willingness to pay
β, α	the price promotion campaign is getting $\$\beta$ off each $\$\alpha$ purchase
$MaxSize$	the maximum size of $SP^{'}$
$MaxDisNum$	the maximum discount number
$MaxDisRate$	the maximum discount rate
OriPri(p)	the original price of the product p
Discount(p)	the discount by purchasing the product p
ActPay(p)	the actual payment of the product p
DisRate(p)	the discount rate of the product p

could bring the maximum discount rate. Noticeably, the methods for the skyline queries under constraints cannot be applied to process our problem straightly.

The frequently used symbols are summarized in Table 3.

3 The Constrained Optimal Product Combination (COPC) Problem

In the COPC problem, it needs to compute the skyline products by the skyline query which is a useful tool for decision support.

The skyline query over all the attributes may give rise to loose some important product combinations. Assume that there are three products p_1, p_2, and p_3 whose prices are \$190, \$210, and \$200, respectively, the other attributes of p_2 and p_3 are the same, and the price promotion campaign is "get \$60 off every \$200 purchase". In the skyline query over all the attributes, p_2 is dominated by p_1 and pruned since p_2 has a lower price and the other attributes of them are the same. However, the discount rates of $\{p_1, p_2\}$ and $\{p_1, p_3\}$ are equal to 0.300 and 0.154 separately, and $\{p_1, p_2\}$ is obvious a great choice with the maximum discount rate. For p_2 is not in the skyline, the important product combination $\{p_1, p_2\}$ is overlooked in the process of product selection.

In [8], Liu et al. formulated a new G-Skyline query that aims to return optimal point groups, namely G-Skylines. Different from other group skyline queries, it reports a more comprehensive results and may return optimal point groups (G-Skylines) that contain non-skyline points [8]. In essence, for a G-Skyline G, each non-skyline point $p \in G$ is only dominated by some other points $p' \in G$.

In this paper, we introduce the grouping campaign introduced in [8] to avoid missing important products that are not skyline query results. For a given dataset P, we part products $p \in P$ into different groups G and products $p \in G$ are with the same values in term of the attributes except the original price. After that, we adapt the definition of dominance operator in [1].

Given a nonempty set of products $P = \{p_1, p_2, \ldots, p_n\}$ which stores the information of different products, for each product $p' \in P$, it can be represented by a multi-dimensional point $< p'[1], p'[2], \ldots, p'[d] >$. Here $p'[i]$ for $1 \leq i \leq d$ denotes the ith attribute value of p'. For ease to description, $p'[d]$ is used to represent the original price of p', denoted as $OriPri(p')$.

Definition 1 (Dominance). *For two products p and p' that are included in different groups, p' is said to dominate p, denoted as $p' \prec p$, if it holds that for all i, $p'[i] \leq p[i]$ and for at least one i, $p'[i] < p[i]$ for $1 \leq i \leq d$.*

In the above definition of the dominance operator, products which have equal values of the attributes except the price are divided into a same group. A product p in a group G is considered non-skyline if it is dominated by some points $p \notin G$. By this adjustment, the skyline query over the product set could get a more comprehensive results.

Definition 1 (Skyline Query [1]). *The skyline query returns all the products $p \in P$ such that there does not exist any other product $p' \in P - p$ satisfying $p' \prec p$.*

In Section 1, we classify the present price promotion campaigns into two categories which are independent-product and dependent-product selections. In this paper, we focus on the dependent-product selection, which includes the campaigns such as "get $\$\beta$ off every $\$\alpha$ purchase" and "$\$\beta$ coupon every $\$\alpha$ purchase" etc. This is because these campaigns are widely adopted by online shopping malls and much more complicated than the ones of the independent-product selection campaigns. In addition, under the campaigns of independent-product selection, the skyline query could offer powerful decision support. But, the skyline query only does little help when selecting products under the campaigns of dependent-product selection.

In the following, we first research our problem under the price promotion campaign as "get $\$\beta$ off every $\$\alpha$ purchase". In particular, by investigating from Jingdong and Alibaba's Taobao Mall, the two most famous online shopping malls in China, β and α are usually set to two- and three-digit numbers, respectively. The popular price promotion campaign is getting $\beta' \times 10$ off every $\alpha' \times 100$ purchase where β' and α' are integers and $\beta', \alpha' \in [1, 9]$.

Noteworthily, the approaches can also be adjusted to handle product selection under other campaigns of the dependent-product selection as analyzed in Section 5.

Definition 2 (Original Price). *Given a product combination P', its original price is computed as*

$$OriPri(P') = \sum_{p \in P'} OriPri(p),$$

where $OriPri(p)$ is original price of a product $p \in P'$.

Definition 3 (Actual Payment). *Given a price promotion campaign "get $\$\beta$ off every $\$\alpha$ purchase ", the actual payment of a product $p \in P$ is*

$$ActPay(p) = OriPri(p) - \left\lfloor \frac{OriPri(p)}{\alpha} \right\rfloor \times \beta.$$

Moreover, the actual payment of a product combination P' is computed by

$$ActPay(P') = \sum_{p \in P'} OriPri(p) - \left\lfloor \frac{\sum_{p \in P'} OriPri(p)}{\alpha} \right\rfloor \times \beta.$$

Going back to the example in Table 2, the actual payment of the wine combination $\{w_4, w_5\}$ is

$$ActPay(\{w_4, w_5\}) = OriPri(\{w_4, w_5\}) - \left\lfloor \frac{OriPri(\{w_4, w_5\})}{200} \right\rfloor \times 60 = 430 - 120 = 310.$$

Definition 4 (Discount Rate). *Suppose that the price promotion campaign is getting $\$\beta$ off every $\$\alpha$ purchase. The discount rate gaining by selecting a product combination P' is computed by*

$$DisRate(P') = \frac{\left\lfloor \frac{OriPri(P')}{\alpha} \right\rfloor \times \beta}{OriPri(P')}. \quad (1)$$

Definition 5 (The Constrained Optimal Product Combination (COPC) problem). *Given a set of products P, a customer's willingness to pay WTP, the COPC problem is to find the skyline product combinations SP', such that they bring the maximum discount rate without exceeding the customer's payment willingness. The COPC problem can be formulated as*

$$maximize\ DisRate(SP')$$
$$subject\ to\ ActPay(SP') \leq WTP,$$
$$for\ SP' \subseteq SP.$$

Here SP denotes the skyline product set over P.

Consider the wine dataset depicted in Table 1. Suppose the customer's willingness to pay *WTP* is equal to \$400. In the COPC problem, it first executes a skyline query and gains the skyline set $SP = \{w_4, w_5, w_6, w_8\}$. Then it generates all the nonempty wine combinations and chooses the ones whose actual payments are no more than \$400. Therefore, we gain the candidate wine combinations as shown in Table 2. Since the wine combination $\{w_5, w_6\}$ has the maximum discount rate, which is equal to 0.300, it is returned as the final result of COPC.

Property 1. *Given a price promotion campaign "get $\$\beta$ off every $\$\alpha$ purchase", the upper bound of discount rate is $\frac{\beta}{\alpha}$.*

The results of COPC can be identified and reported to customers progressively due to Property 1. Based on Property 1, each skyline product combination SP' with $DisRate(SP') = \frac{\beta}{\alpha}$ is a result of COPC, and it can be reported to customers as soon as possible.

Considering the price promotion campaign that is "getting \$60 off each \$200 purchase", the upper bound of the discount rate is equal to $\frac{60}{200} = 0.300$. This means that customers can get the discount rate that is no more than 0.300. In the computation of the COPC problem, if getting some wine combinations whose discount rates are just equal to 0.300, then they could be returned to customers as final results.

Property 2. *Given a skyline product combination SP' and a price promotion campaign "get $\$\beta$ off every $\$\alpha$ purchase", it can get the upper bound of the discount rate iff $OriPri(SP')$, which denotes the original price of SP', is an integral multiple of α.*

Definition 6 (Discount Number). *Suppose that the price promotion campaign is getting $\$\beta$ off every $\$\alpha$ purchase. The discount number by selecting a skyline product combination SP' is computed by*

$$DisNum(SP') = \left\lfloor \frac{OriPri(SP')}{\alpha} \right\rfloor.$$

In Table 2, the original price of the wine combination $\{w_5, w_6\}$ is \$400. Under the price promotion strategy that is getting \$60 off each \$200 purchase the discount number of the wine combination $\{w_5, w_6\}$ is $DisNum(\{w_5, w_6\}) = \lfloor \frac{400}{200} \rfloor = 2$.

Lemma 3.1. *Given a customer's payment willingness WTP and the price promotion that is getting $\$\beta$ off every $\$\alpha$ purchase, the maximum discount number the customer can obtain is $MaxDisNum = \lfloor \frac{WTP}{\alpha - \beta} \rfloor$.*

Proof. Given a skyline product combination $SP' \subseteq SP$, its original price is $OriPri(SP') = t \times \alpha + r$, where the discount number $t = \lfloor \frac{OriPri(SP')}{\alpha} \rfloor$, and the discount $Discount(SP') = \lfloor \frac{OriPri(SP')}{\alpha} \rfloor \times \beta$. It holds that

$$ActPay(SP') = OriPri(SP') - Discount(SP') = OriPri(SP') - \left\lfloor \frac{OriPri(SP')}{\alpha} \right\rfloor \times \beta = t \times \alpha + r - t \times \beta.$$

Since $ActPay(SP') \leq WTP$, we have $t \times \alpha + r - t \times \beta \leq WTP$. Therefore, it holds that $t \leq \frac{WTP - r}{\alpha - \beta}$. Because $0 \leq r < \alpha$, we gain $t \leq \frac{WTP - r}{\alpha - \beta} \leq \frac{WTP}{\alpha - \beta}$.

Therefore, it holds that $MaxDisNum = \lfloor \frac{WTP}{\alpha - \beta} \rfloor$ and this lemma holds. □

Theorem 3.2. *The COPC problem is an NP-hard problem.*

Proof. The NP-hardness proof can be achieved by transforming the subset sum problem, which is an NP-hard problem, to a special case of the COPC problem [9], [23].

The subset sum problem is defined as follows:

Subset Sum Problem. Given a positive integer set $W = \{w_1, w_2, \ldots, w_n\}$ and a positive integer M, is there a subset $W' \subseteq W$ such that $\sum_{w \in W'} w = M$?

For a skyline product combination $SP' \subseteq SP$, assume that $OriPri(SP') = \sum_{p \in SP'} OriPri(p) = t \times \alpha + r$ where $t = \lfloor \frac{OriPri(SP')}{\alpha} \rfloor$, and $r = OriPri(SP') \bmod \alpha$. On the basis of Property 2, we can get the maximum discount rate when $OriPri(SP')$ is a multiple of α, and $OriPri(SP') = \sum_{p_i \in SP'} OriPri(p_i) = t \times \alpha$. Here t is a positive integer and $1 \leq t \leq \lfloor \frac{WTP}{\alpha - \beta} \rfloor$ due to Lemma 3.1.

In the subset sum problem, let element $w_i \in W$ represent the original price of a skyline product $p_i \in SP$, and $M = t \times \alpha$. Due to Property 2, if $\sum_{w \in W'} w = t \times \alpha$, the subset W' represents a final result of our COPC problem. The result of the corresponding subset sum problem is also the result of this instance of the COPC problem.

From the above analysis, any instance of the subset sum problem can be transformed to an instance of the COPC problem. Since the subset problem has been proven to be an NP-hard problem, the COPC problem is also NP-hard [23]. Furthermore, our COPC problem is more complex than the subset sum problem. □

4 Algorithms for the COPC Problem

To process the COPC problem, a naive exact algorithm is to generate all the skyline product combinations which are not beyond the customer's payment willingness, compute the discount rate of each candidate combination, and identify the ones that bring the maximum discount rate.

Consider each combination of the skyline products within SP, which contains t products for $1 \leq t \leq MaxSize$. The number of these combinations that contain t products is $\binom{N_S}{t}$, where N_S represents the cardinality of the skyline set SP, $MaxSize$ denotes the maximum size of the skyline product combinations. Therefore, the total number of candidate product combinations is

$$\binom{N_S}{1} + \binom{N_S}{2} + \cdots + \binom{N_S}{MaxSize} = \sum_{i=1}^{MaxSize} \binom{N_S}{i} \leq 2^{N_S}.$$

The time complexity of this native algorithm is $O(2^{N_S})$.

As analyzed above, the number of skyline product combinations can be intimidating, and the computational complexity of the native algorithm is unavoidable and unacceptable. To enhance the performance of solving the COPC problem, we develop a two list exact algorithm, a lower bound approximate algorithm, and an incremental greedy algorithm in this section.

4.1 The Two List Exact Algorithm

Due to Theorem 3.2, the COPC problem is closely related to the subset sum problem. Moreover, our COPC problem is much more complicated, and the approaches for the subset problem cannot be utilized to our problem directly. In this section, we develop the two-list algorithm, which is a famous algorithm for the subset sum problem [24], [25], and present a two list exact algorithm for the COPC problem.

As introduced in the proof of Theorem 3.2, we can get the results of the COPC problem through computing several subset sum problems whose sums are equal to $t \times \alpha$ for $1 \leq t \leq MaxDisNum$. Here $MaxDisNum$ represents the maximum discount number and can be computed due to Lemma 3.1. Furthermore, when there are not any product combination SP' with $OriPri(SP') = t \times \alpha$, we will conserve the combinations whose sums are as small as possible but not less than $t \times \alpha$ due to the following lemmas.

Lemma 4.1. *Given a price promotion campaign "get \$$\beta$ off every \$$\alpha$ purchase", and skyline product combinations $SP', SP'' \subseteq SP$ with $i \times \alpha \leq OriPri(SP') < OriPri(SP'') < (i+1) \times \alpha$ for i is an integer and $i \in [0, MaxDisNum]$, it holds that $DisRate(SP') > DisRate(SP'')$.*

Proof. Since $i \times \alpha \leq OriPri(SP') < (i+1) \times \alpha$, it holds that $i = \lfloor \frac{OriP(SP')}{\alpha} \rfloor$ for $1 \leq i \leq \lfloor \frac{WTP}{\alpha - \beta} \rfloor$ according to Lemma 3.1. Due to Equation (1), we have

$$DisRate(SP') = \frac{\left\lfloor \frac{OriP(SP')}{\alpha} \right\rfloor \times \beta}{OriP(SP')} = \frac{i \times \beta}{OriP(SP')}.$$

Similarly, we have $DisRate(SP'') = \frac{i \times \beta}{OriP(SP'')}$. Since $OriPri(SP') < OriPri(SP'')$, it has $DisRate(SP') > DisRate(SP'')$, and this lemma holds. □

Based on the two list algorithm for the subset sum problem [24], we develop an exact algorithm as depicted in Algorithm 1. Similarly to the two-list algorithm, Line 1 divides SP into two parts SP_1 and SP_2 equally. After that, it generates all the subsets of SP_1 and SP_2. These subsets, which represent different skyline product combinations, satisfy the conditions that actual payments do not exceed the customer's willingness to pay WTP. Thereafter Lines 2 and 4 get lists A and B that store the sums of these subsets (original prices). Specially, the elements in A are sorted in increasing order while the elements in B are sorted in decending order. Here, we generate subsets and sort them all at once through merging. This can further improve the performance of TLE [24]. Lines 3 and 5 compute the elements of A and B that are equal to the original prices of the skyline product combinations with current maximum discount rate. Line 6 initializes SP^* with the product combinations that have the current maximum discount rate. Lines 7 to 22 are executed to merge the two lists, A and B, to get the final product combinations SP^* that bring the maximum discount rate. Line 7 computes the maximum discount number $MaxDisNum$ due to Lemma 3.1. Due to Property 2, if there are some combinations whose original prices are just an integral multiple of α, the corresponding skyline product combinations can bring the maximum discount rate. Otherwise, the combinations whose original prices are as small as possible but no less than an integral multiple of α may be good choices. Thereafter, Lines 8-22 are an iteration process to compute elements y_k^* for $1 \leq k \leq MaxDisNum$. These elements represent the original prices of product combinations that may bring the maximum discount rate. Suppose element $a_i \in A$ is combined with the first j elements within B, the next element a_{i+1} only needs to be combined with b_{j+1} and the elements ranked after it. Hence, Line 9 defines $flag$ to store the location of the last element within B that is merged with a_i. Lines 13 and 14 are used to identify the combinations SP with $OriPri(SP) = k \times \alpha$. Lines 16 to 18 are utilized to find out the skyline product combinations whose original prices are no less than but nearest to $k \times \alpha$. Line 22 chooses the product combinations SP'' that can bring the maximum discount rate from the ones whose original prices are equal to y_k^*. Then SP^* is updated by only reserving the combinations that bring the maximum discount rate and satisfy the customer's payment willingness. At last, Line 23 returns SP^* as the final result of COPC.

Example. Going back to the example in Table 1, by the skyline query, we have a wine set $W = \{w_4, w_5, w_6, w_8\}$ where each wine is in the skyline. Dividing the set W into two parts $W_1 = \{w_4, w_5\}$ and $W_2 = \{w_6, w_8\}$. Line 2 generates the wine combinations $\{w_4\}$, $\{w_5\}$, and $\{w_4, w_5\}$ over W_1. After sorting these combinations in increasing order of their original prices, we have the list $A = \{190, 240, 430\}$ and $a^* = 430$ since the wine combination $\{w_4, w_5\}$ with $OriPri(\{w_4, w_5\}) = 430$ brings the maximum discount rate. Line 4 generates the wine combinations $\{w_6\}$, $\{w_8\}$, and $\{w_6, w_8\}$ over W_2. After sorting them in decending order of their original prices, we have the list $B = \{390, 210, 180\}$, and $b^* = 210$. Line 6 gets the wine combination $\{w_6\}$ which matches the current maximum discount rate 0.286. Line 7 computes the maximum discount number due to Lemma 3.1, and we have $MaxDisNum = \lfloor \frac{400}{200-60} \rfloor = 2$. Lines 8 to 22 are utilized to combine the elements within A and B. First, the parameter k is set to 1, we combine the elements within A and B to get the combinations whose sums are just equal to $k \times \alpha = 200$. By combining $a_1 = 190$ with $b_j \in B$, there is not any combination whose sum is just equal to 200, and we get the combination $\{a_1, b_3\} = \{190, 180\}$ whose sum is no less than but nearest to 200. We also get $flag = 3$. Considering the second element $a_2 = 240$, it only needs to be combined with the element b_3 and other elements ranked after it within B. We have $y_1^* = a_1 + b_3 = 370$, $SP'' = \{w_5, w_8\}$, and $SP^* = \{w_6\}$. Similarly, for

$k=2$, we get $y_2^* = a_1 + b_2 = 400$, $SP'' = \{w_5, w_6\}$, and $SP^* = \{w_5, w_6\}$. Finally, the wine combination $\{w_5, w_6\}$ is returned as the final result of COPC.

Algorithm 1. Two_List_Exact(TLE) Algorithm

Input: The skyline product set SP, a price promotion campaign "get \$$\beta$ off every \$$\alpha$ purchase", and a customer's payment willingness WTP

Output: A result set SP^* of the COPC problem

1: Divide SP into two parts: $SP_1 = \{sp_1, sp_2, \ldots, sp_{N_S/2}\}$ and $SP_2 = \{sp_{N_S/2+1}, sp_{N_S/2+2}, \ldots, sp_{N_S}\}$
2: Generate all the product combinations $SP' \subseteq SP_1$ with $ActPay(SP') \leq WTP$, sort them in an increasing order of $OriPri(SP')$, and store $OriPri(SP')$ as the list $A = \{a_1, a_2, \ldots, a_{N_1}\}$
3: Compute $a^* \in A$ which is with the maximum discount rate
4: Generate all the product combinations $SP' \subseteq SP_2$ with $ActPay(SP') \leq WTP$, sort them in a decending order of $OriPri(SP')$, and store $OriPri(SP')$ as the list $B = \{b_1, b_2, \ldots, b_{N_2}\}$
5: Compute $b^* \in B$ which is with the maximum discount rate
6: $SP^* = argmax_{OriPri(SP') \in \{a^*, b^*\}} DisRate(SP')$
7: Set the maximum discount number $MaxDisNum = \lfloor \frac{WTP}{\alpha - \beta} \rfloor$ due to Lemma 3.1
8: **for** k=1 to $MaxDisNum$ **do**
9: Initialize $i = 1$, $flag = 0$ and $y_k^* = (k+1) \times \alpha$
10: **for** $a_i \in A$ **do**
11: $j = flag + 1$
12: **for** $b_j \in B$ **do**
13: **if** $a_i + b_j$ is equal to $k \times \alpha$ **then**
14: $y_k^* = k \times \alpha$ and Break
15: **else**
16: **if** $a_i + b_j > k \times \alpha$ **then**
17: $j = j + 1$
18: $y_k^* = \min\{y_k^*, a_i + b_j\}$
19: **else**
20: $i = i + 1$
21: $flag = j$
22: Add $SP'' = argmax_{OriPri(SP') = y_j^*} DisRate(SP')$ for $1 \leq j \leq MaxDisNum$ to SP^* and refresh SP^* by removing the combinations whose discount rates are less than that of SP''
23: Return SP^*

Complexity. In Algorithm 1, it first divides SP into two parts SP_1 and SP_2 equally. Each part has $\frac{N_S}{2}$ elements. Thereafter, it generates all the subsets of $SP' \subseteq SP_1$ and $SP' \subseteq SP_2$ with $ActPay(SP') \leq WTP$. The number of these combinations that contain t products is $\binom{N'}{t}$ where $N' = \frac{N_S}{2}$. Therefore, the total number of candidate skyline product combinations that consist of the products within SP_1 or SP_2 is calculated by

$$\binom{N'}{1} + \binom{N'}{2} + \cdots + \binom{N'}{MaxSize} = \sum_{i=1}^{MaxSize} \binom{N'}{i} \leq 2^{N'} = \sqrt{2}^{N_S}.$$

Here $MaxSize$ is the maximum size of the skyline product combination.

Algorithm 1 contains three stages, which are generation of all subsets of SP_1 and SP_2, sorting these subsets by their sums, and searching the final result of our problem by merging the subsets of SP_1 and SP_2. By merging the generation and sorting stage, the time complexity is $O(\sum_{i=1}^{MaxSize} \binom{N'}{i})$ due to [26]. In the searching stage, the time complexity is $O(MaxDisNum \times \sum_{i=1}^{MaxSize} \binom{N'}{i})$. Here $MaxDisNum$ denotes the maximum discount number which can be computed due to Lemma 3.1.

As mentioned above, the time complexity of Algorithm 1 is $O\big((MaxDisNum+1) \times \sum_{i=1}^{MaxSize} \binom{N'}{i}\big) = O\big((MaxDisNum+1) \times \sqrt{2}^{N_S}\big)$ where N_S represents the cardinality of the skyline set SP and $MaxDisNum$ is the maximum discount number which can be computed due to Lemma 3.1.

Since the COPC problem is NP-hard, although the TLE algorithm has better performance than the native algorithm, it is not scalable for large skyline product sets. In the experiments, the TLE algorithm can handle the skyline product sets whose sizes are at most 35 while the naive algorithm can only deal with the skyline product sets whose sizes are at most 25. Therefore, we develop an approximate algorithm with an approximate bound and a greedy algorithm which have nice scalability for large skyline product sets in the following subsections.

4.2 The Lower Bound Approximate Algorithm

Based on Lemmas 3.1 and 4.1, and Theorem 3.2, we design a lower bound approximate algorithm for the COPC problem, which is depicted in Algorithm 2.

The LBA algorithm first removes each product $p' \in SP$ whose actual payment is larger than WTP (Line 1). Line 2 initializes a list L with a set that contains an element "0". Thereafter, the list L stores original prices of candidate skyline product combinations. Lines 3-10 are applied to find candidate skyline product combinations which may bring the maximum discount rate. In Line 3, it computes $MaxDisNum$ which represents the maximum discount number based on Lemma 3.1. Thereafter, Line 4 initializes y_j^*, which are the original prices of skyline product combinations that may bring the maximum discount rate without exceeding the customer's payment willingness, with ∞. The while loop (Lines 5-10) refreshes the list L (Line 6) by generating new elements $y + \{Ori(p)\}$ for $y \in L$ and $p \in SP$ at a time, and adding these elements to L. Line 7 sorts the elements in L in an increasing order, and each element y with $y - \lfloor \frac{y}{\alpha} \rfloor \times \beta > WTP$ is removed from L. This is because if $y - \lfloor \frac{y}{\alpha} \rfloor \times \beta > WTP$, the skyline product combinations whose original prices are equal to y are beyond the customer's payment willingness. This helps to reduce the search space by pruning the skyline product combinations which are beyond the customer's payment willingness as soon as possible. Line 8 computes elements $y_j^* \in [j \times \alpha, (j+1) \times \alpha)$ which is as small as possible but not less than $j \times \alpha$. The elements ranked after $y_{MaxDisNum}^*$ are removed from L (Line 9). This is because for a skyline product combination SP' with $OriPri(SP') > y_{MaxDisNum}^*$, its discount rate is less than the ones whose original prices are equal to $y_{MaxDisNum}^*$ due to Lemma 4.1. Moreover, the skyline product combinations SP'' with $SP' \subseteq SP''$ cannot be good choices for customers also. Hence, in the next iteration, it is not necessary to generate new product combinations based on SP'. Line 10 employs a function Trim to trim some similar elements within $L - y_j^*$. Since y_j^* contains the present optimal results, we always maintain it in L without trimming.

The trimming function is similar to that utilized in the full polynomial-time approximate algorithm for the subset

sum problem [23]. As described in Lines 12-19, an element is appended onto the list L' if it is the first element of L or it cannot be represented by any other element. In particular, if we trim the list L by δ, for two elements y and y' within L, y' is removed from L if it holds that $y' \leq y \times (1+\delta)$. This means that y and y' are closed to each other enough, and y could be used to represent y'.

Algorithm 2. Lower_Bound_Approximate (LBA) Algorithm

Input: The skyline product set SP with $|SP| = N_S$, a price promotion campaign "get \$$\beta$ off every \$$\alpha$ purchase", a customer's payment willingness WTP, and a trimming parameter ϵ for $0 < \epsilon < 1$

Output: A result set SP^* of the COPC problem

1: Remove each product $p' \in SP$ with $ActPri(p') > WTP$
2: Initialize $L = \{0\}$
3: Set the maximum discount number $MaxDisNum = \left\lfloor \frac{WTP}{\alpha-\beta} \right\rfloor$ due to Lemma 3.1
4: Initialize $y_j^* = \infty$ for $1 \leq j \leq MaxDisNum$
5: **while** SP is not empty **do**
6: $L = L \cup \{y + Ori(p) : y \in L\}$ for $p \in SP$
7: Sort all the elements in L in an increasing order and remove each element y from L if $y - \lfloor \frac{y}{\alpha} \rfloor \times \beta > WTP$
8: Compute $y_j^* \in L$ where $\nexists y' \in L - \{y_j^*\}, y' < y_j^*$ with $y', y_j^* \in [j \times \alpha, (j+1) \times \alpha)$ for $1 \leq j \leq MaxDisNum$ and j is an integer
9: Remove each element y from L which is larger than $y^*_{MaxDisNum}$
10: $L = \text{Trim}(L - y_j^*, \frac{\epsilon}{2N_S})$ for $1 \leq j \leq MaxDisNum$
11: Return $SP^* = argmax_{OriPri(SP')=y_j^*} DisRate(SP')$ for j is an integer and $1 \leq j \leq MaxDisNum$
12: **Function: Trim(L, δ)**
13: Initialize $L' = \{y_1\}$
14: $last = y_1$
15: **for** $i = 2$ to $|L|$ **do**
16: **if** $y_i > last \times (1+\delta)$ **then**
17: Append y_i onto the end of L'
18: $last = y_i$
19: Return L'

After the while loop (Lines 5-10), we gain y_j^* which are as small as possible but not less than $j \times \alpha$ for $1 \leq j \leq MaxDisNum$. Line 11 finds out the skyline product combinations with the maximum discount rate from the ones whose original prices are equal to y_j^* for $1 \leq j \leq MaxDisNum$, and stores them in SP^*. Finally, Line 11 returns SP^* as the final results of COPC.

It is worth to notice that for large skyline product datasets, it may return many results that bring the same discount rate. In this case, we will choose one result at random.

Example. Continuing with the example in Table 1, after getting the skyline set $\{w_4, w_5, w_6, w_8\}$, $MaxDisNum$ is computed by $\lfloor \frac{400}{200-60} \rfloor = 2$. Assume that $\epsilon = 0.6$ and $\frac{\epsilon}{2N_S} = \frac{0.6}{2\times 4} = 0.075$. The LBA generates the combinations whose original prices are as small as possible but not less than $j \times \alpha$ for $j \in \{1, 2\}$. First, we have a list $L_1 = \{0, 240\}$, $y_1^* = 240$, and initialize $y_2^* = \infty$. Considering the wine w_5, we have $L_2 = \{0, 190, 240, 430\}$ by merging L_1 and $\{y + OriPri(w_5)$ for $y \in L_1\}$, $y_1^* = 240$ and $y_2^* = 430$. By invoking the Trim function over the list $L_2 - \{y_1^*, y_2^*\} = \{0, 190\}$, we have $L_2 - \{y_1^*, y_2^*\} = \{0, 190\}$ and $L_2 = \{0, 190, 240, 430\}$. This is since $190 > 0 \times (1 + \frac{0.6}{2\times 4}) = 0$. And then, we obtain $L_3 = \{0, 190, 210, 240, 400, 430, 450, 640\}$ by merging L_2 and $\{y + OriPri(w_6)$ for $y \in L_2\}$, $y_1^* = 210$, and $y_2^* = 400$. Through removing the elements 430, 450, and 640 which are larger than $y_2^* = 400$ from L_3, we gain $L_3 = \{0, 190, 210, 240, 400\}$. After trimming $L_3 - \{y_1^*, y_2^*\}$, we have $L_3 = \{0, 190, 210, 240, 400\}$. Considering the wine w_8, we obtain $L_4 = \{0, 180, 190, 210, 240, 370, 390, 400, 420, 580\}$, $y_1^* = 210$, and $y_2^* = 400$. By removing the elements 420 and 580 which are larger than $y_2^* = 400$ from L_4, we gain $L_4 = \{0, 180, 190, 210, 240, 370, 390, 400\}$. After trimming $L_4 - \{y_1^*, y_2^*\}$, it holds that $L_4 = \{0, 180, 210, 240, 370, 400\}$ by removing the elements 190 and 390. This is because $190 < 180 \times (1 + \frac{0.6}{2\times 4}) = 193.5$ and $390 < 370 \times (1 + \frac{0.6}{2\times 4}) = 397.75$. Now, we have $y_1^* = 210$ and $y_2^* = 400$ which are as small as possible but not less than $j \times \alpha \in \{200, 400\}$. The original prices of the wine combinations $\{w_6\}$ and $\{w_5, w_6\}$ are equal to $y_1^* = 210$ and $y_2^* = 400$ respectively. Lastly, the wine combination $\{w_5, w_6\}$ is reported as the final result of COPC since $DisRate(\{w_5, w_6\}) = 0.300 > DisRate(\{w_6\}) = 0.289$.

Complexity. The LBA algorithm first removes the skyline products whose actual payments are larger than WTP. The cost of this stage is $O(N_S)$.

Then, it generates the list L_i which stores the original prices of candidate skyline product combinations. Here L_i represents the list generated at the end of the ith iteration. After sorting the elements in L_i and removing the unqualified elements, it invokes the Trim function to trim L_i. Since the time complexity of Trim is $O(|L_i|)$, the time complexity of Lines 5-10 of the LBA algorithm is $O(N_S \times |L_i|)$.

Consider the successive elements y and y' of L_i. Due to the Tim function, it holds that $\frac{y'}{y} > 1 + \frac{\epsilon}{2N_S}$ and $|L_i| = \frac{\ln t}{-\ln(1-\epsilon/2N_S)} \leq \frac{t}{\epsilon/2N_S} = \frac{2N_S \times t}{\epsilon}$, where $t = j \times \alpha$ for $1 \leq j \leq MaxDisNum$. Here N_S represents the cardinality of skyline products and $MaxDisNum$ is the maximum discount number that can be computed due to Lemma 3.1.

As analyzed above, the complexity of the LBA algorithm is

$$N_S + N_S \times \sum_{i=1}^{MaxDisNum} |L_i| < N_S + N_S \times \sum_{i=1}^{MaxDisNum} \left(\frac{2N_S \times t}{\epsilon}\right) = \frac{\alpha}{\epsilon}\left(MaxDisNum^2 \times N_S^2 + MaxDisNum \times N_S^2\right) + N_S.$$

Therefore, the complexity of the LBA is $O(\frac{\alpha}{\epsilon} MaxDisNum^2 \times N_S^2)$, and it is apparent to be a pseudo-polynominal time algorithm.

Theorem 4.2. *LBA is a $(1+\epsilon)$-approximation algorithm for the COPC problem.*

Proof. Let O_i denote the set of all values obtained by selecting a subset of $\{p \in SP | OriPri(p)\}$, summing its members. The list L_i contains a suitably trimmed version of the set O_i. After removing all the elements that ensure not to be the final results from L_i and trimming the list L_i, each element of L_i is also the element of O_i which represents the original prices of some skyline product combinations. For each element $y \in O_i$, there is an element $y' \in O_i \cap L_i$ such that $\frac{y}{\left(1+\frac{\epsilon}{2n}\right)^i} \leq y' \leq y$, where $n = N_S = |SP|$.

Let $y_j^* \in O_n$ represent the element which is as small as possible but not less than $j \times \alpha$ for $1 \leq j \leq MaxDisNum$. There is an element $z_j \in L_n$ such that $\frac{y_j^*}{\left(1+\frac{\epsilon}{2n}\right)^n} \leq z_j \leq y_j^*$. Thus $\frac{y_j^*}{z_j} \leq \left(1+\frac{\epsilon}{2n}\right)^n$. Therefore for the approximate optimal solution $z_j^* \in L_n$ which is as small as possible but not less than $j \times \alpha$, we also have $\frac{y_j^*}{z_j^*} \leq (1+\frac{\epsilon}{2n})^n$. Since $(1+\frac{\epsilon}{2n})^n \leq 1+\epsilon$, it holds that $\frac{y_j^*}{z_j^*} \leq 1+\epsilon$.

Finally, after computing the discount rates of the skyline product combinations whose original prices are equal to y_j^*, we obtain the ones having the maximum discount rate as the final results. □

4.3 The Incremental Greedy Algorithm

In this section, to further improve the performance of processing the COPC problem, we propose an incremental greedy (IG) algorithm.

As depicted in Algorithm 3, the IG algorithm first removes all the skyline products whose actual payments are more than WTP. Due to the property of the COPC problem, it does not always bring a greater benefit (larger discount rate) by selecting much more products. As shown in Table 2, we have $DisRate(\{w_6\}) = 0.286 > DisRate(\{w_4, w_6\}) = 0.267$ and $DisRate(\{w_6\}) > DisRate(\{w_6, w_8\}) = 0.154$. Obviously, $\{w_6\}$ is better than $\{w_4, w_6\}$ or $\{w_6, w_8\}$ in terms of discount rate. Therefore, besides a set SP^* to store the final optimal product combinations. Line 2 initializes a set $PreP$ to store local optimal skyline product combinations. Line 3 computes the product p with the highest discount rate and adds $\{p\}$ to the set $PreP$. The set SP^* is initialized as $PreP$ and the final maximum discount rate Max_R is set as $DisRate(\{p\})$. By combining each product combination $SP' \in PreP$ with other skyline products $p \in SP - SP'$, we get new skyline product combinations of larger size (Lines 5 to 25). Lines 5 to 25 are a process of iteration. It generates the skyline product combinations incrementally and maintains the ones that have the current maximum discount rate in SP^*. In each iteration, the $TempMax\_R$ stores the local maximum discount rate which is the maximum discount rate of new skyline product combinations in the current iteration. Lines 7 to 18 compute a set $CandSet$ that contains the product combinations whose discount rates are equal to $TempMax\_R$. Lines 19 to 24 update SP^* if $TempMax\_R$ exceeds Max_R. Otherwise, SP^* and Max_R are maintained without changing. Line 25 updates $PreP$ as $CandSet$. New combinations will be generated based on $PreP$ in the next iteration. This process of iteration proceeds until $PreP$ is empty. Finally SP^* is returned as the final result set of the COPC problem.

Example 5. Going back to the example in Table 1, by the skyline query over the wine set, it gets the skyline set $SP = \{w_4, w_5, w_6, w_8\}$. Since the wine w_6 is with the largest discount rate compared to other wines w_4, w_5, and w_8, $\{w_6\}$ is inserted into $PreP$. We have $SP^* = \{\{w_6\}\}$ and $Max\_R = 0.286$. Next, by combining $\{p\}$ with $\{w_6\} \in PreP$ for each $p \in \{w_4, w_5, w_8\}$, we get new wine combinations $\{w_4, w_6\}$, $\{w_5, w_6\}$ and $\{w_8, w_6\}$. Since $\{w_5, w_6\}$ gets the local maximum discount rate, it holds that $CandSet = \{\{w_5, w_6\}\}$ and $TempMax\_R = DisRate(\{w_5, w_6\}) = 0.300$. Here, we have $SP^* = \{\{w_5, w_6\}\}$, $Max\_R = 0.300$, and $PreP = \{\{w_5, w_6\}\}$. Then, we generate two new wine combinations $\{\{w_4, w_5, w_6\}\}$ and $\{\{w_5, w_6, w_8\}\}$. For $ActPay(\{w_4, w_5, w_6\}) = 460 > 400$, $\{w_4, w_5, w_6\}$ is not a legal result of the COPC problem. Similarly, $SP'' = \{\{w_5, w_6, w_8\}\}$ is pruned as an unqualified result. Finally, $\{w_5, w_6\} \in SP^*$ is returned as the final result.

Algorithm 3. Incremental_Greedy (IG) Algorithm

Input: The skyline set SP of a product dataset P, a price promotion campaign "getting \$$\beta$ off every \$$\alpha$ purchase", and a customer's payment willingness WTP
Output: A result set SP^* of the COPC problem

```
1: Remove each product p ∈ SP with ActPay(p) > WTP
2: Initialize PreP = ∅
3: Compute product combinations {p} where p ∈ SP and p are with the highest discount rate, and add them to PreP
4: Initialize SP* = PreP
5: Initialize Max_R = DisRate({p}) for {p} ∈ PreP
6: while PreP is not empty do
7:   TempMax_R = 0 and a set CandSet = ∅
8:   for each candidate product combination SP' ∈ PreP do
9:     PreP = PreP − SP'
10:    for each product p ∈ SP − SP' do
11:      Generate a new product combination SP'' = SP' ∪ {p}
12:      if ActPay(SP'') ≤ WTP then
13:        if DisRate(SP'') > TempMax_R then
14:          TempMax_R = DisRate(SP'')
15:          Remove the product combinations within CandSet
16:          Add SP'' to CandSet
17:        else
18:          if DisRate(SP'') = TempMax_R then
19:            Add SP'' to CandSet
20:  if TempMax_R > Max_R then
21:    SP* = CandSet
22:    Max_R = TempMax_R
23:  else
24:    if TempMax_R = Max_R then
25:      SP* = SP* ∪ CandSet
26:  PreP = CandSet
27: Return SP*
```

Complexity. The IG algorithm is composed of two stages. In the first stage (Line 1), it checks and removes the products whose actual payments are larger than WTP. Besides, the discount rates of each product $p \in SP$ are computed. The cost of this stage is $O(N_S)$. In the second stage (Lines 5 to 25), it is a while loop which generates skyline product combinations incrementally. During each iteration, it needs to combine each product combination $SP' \in PreP$ with a product in $SP - SP'$, the complexity of this stage is

$$O\big(|PreP| \times |SP - SP'|\big) = O\big(|PreP| \times |SP|\big).$$

As analysed above, the complexity of the IG algorithm is equal to $O(|PreP| \times N_S)$ where $N_S = |SP|$.

5 Discussions

In this section, we discuss variants of the COPC problem with considering other price promotion campaign and different customer's demands.

5.1 Other Price Promotion Campaigns

In this paper, we pay attention on the price promotion campaigns of the dependent-product selection. In the proposed algorithms, we consider one typical campaign as

getting $\$\beta$ off every $\$\alpha$ purchase. It is worth to notice that our approaches can also be used to handle the COPC problem under other campaigns of the dependent-product selection.

Consider the popular price promotion campaign "$\$\beta$ coupon every $\$\alpha$". Some definitions and lemmas in Section 3 need to be modified as follows. At first, in Definition 3, the actual payment of a skyline product combination P is computed by $ActPay(P) = \sum_{p \in P} OriPri(p)$. Then, the discount rate of the skyline product combination P in Definition 4 is

$$DisRate(P) = \frac{\left\lfloor \frac{OriPri(P)}{\alpha} \right\rfloor \times \beta}{OriPri(P) + \left\lfloor \frac{OriPri(P)}{\alpha} \right\rfloor \times \beta}.$$

Last, due to Property 1, the upper bound of the discount rate is $\lfloor \frac{\beta}{\alpha+\beta} \rfloor$. The maximum discount number in Lemma 3.1 is computed by $MaxDisNum = \lfloor \frac{WTP}{\alpha} \rfloor$. Additionally, under the price promotion that is getting $\$\beta$ coupon every $\$\alpha$ purchase, customers can get the maximum discount rate if the original price of P' is an integral multiple of α. Therefore, it does not need to change Property 2.

Based on the modified definitions and lemmas, the approaches proposed in Section 4 could also be employed to the COPC problem under the price promotion campaigns such as "$\$\beta$ coupon every $\$\alpha$ purchase".

5.2 Different Customer's Demands

In this paper, the COPC problem is to find the skyline product combinations SP', such that they bring the maximum discount rate without exceeding the customer's payment willingness. However, when selecting products under price promotion, apart from the maximum discount rate, customers are common to have other two popular demands that are spending or saving the most money. When customers want to spend the most money, it only needs to redefine the COPC problem in this paper by modifying the objection function as "$maximize\ ActPay(SP')$" in Definition 5. In addition, for saving the most money (maximize the discount), the new objection function is "$maximize\ Discount(SP')$".

6 Experimental Evaluation

In this section, similar to [8], we first conduct a small customer study to certify the significance of our problem in the product-selection under price promotion. We then evaluate the performance of the proposed algorithms.

The naive exact algorithm of COPC is to generate all the skyline product combinations which are not beyond the customer's payment willingness, compute the actual discount rate of each skyline product combination, and identify the ones that bring the maximum discount rate.

Since we investigate the COPC problem for the first time, we also take the naive exact algorithm as a baseline algorithm. Admittedly, this exact algorithm, which needs to enumerate all the skyline product combinations, is not scalable. Similar to the ways in [14], we first conduct some experiments to compare all the proposed algorithms, TLE in Section 4.1, LBA in Section 4.2, and IG in Section 4.3, over several small skyline product sets. Besides, we compare the LBA and IG algorithms for the COPC problem over large skyline product sets.

All the proposed algorithms aforementioned were implemented in C++ to evaluate their effectiveness and efficiency. In particular, we evaluate the algorithms in term of processing time (PT) which is the time to compute the optimal skyline product combinations. Besides, number of skyline products (N_S) is illustrated for evaluating the relationship between PT and N_S.

TABLE 4
The Results of the Customer Study

Wine Combination	Discount Rate	Picked Number
$\{w_5, w_6\}$	0.300	28
$\{w_4, w_6\}$	0.267	16
$\{w_6, w_8\}$	0.154	14
$\{w_5, w_8\}$	0.162	8
$\{w_4\}$	0.250	7
$\{w_4, w_8\}$	0.286	6
$\{w_8\}$	0.000	5
$\{w_6\}$	0.286	3
$\{w_4, w_5\}$	0.279	2

The experiments were performed on a PC with Intel Core I7-6700T 2.81 GHz CPU (contains 4 cores), 8 GB main memory, and under the Microsoft Windows 7 operation system.

6.1 A Customer Study

In [8], Liu et al. presented a small customer study to verify the motivation of the G-skyline query. Similarly, we also perform a small customer study using the wine set in Table 1. This study is based on questionnaire investigation for 47 undergraduates, 38 master and doctorate candidates, and 16 staff members in our university. They are asked to pick out the best wine combinations when the price promotion campaign is getting $60 off each $200 purchase and the customer's willingness to pay is $400. Besides, they are also required to provide the reasons of their selections. Table 4 shows the picked number of each wine combination from 89 feedbacks The wine combinations are ranked in descending order of their picked numbers.

From the questionnaires, it holds that customers mainly select the wines in the skyline which have advantages in the attributes, class and praise degree. Here, the great attention is common to be paid to the attributes except the original prices. Additionally, we have the following conclusions: First of all, most of the customers choose the wine combination $\{w_5, w_6\}$ because it does not exceed the customer's willingness to pay $400 and brings the maximum benefit (discount rate) for the customers. More, some customers who should not spend too much time on the selection tend to choose the wine combinations $\{w_4, w_5\}$, $\{w_4, w_6\}$, and $\{w_4, w_8\}$. This is since these wine combinations could bring good benefits, which are close to the maximum one, for the customers. Furthermore, some customers who do not drink wine select the wine combinations $\{w_4\}$ and $\{w_6\}$ that only contain one wine and bring good benefit. Last but not least, the customers who favorite the wine combinations $\{w_8\}$, $\{w_5, w_8\}$, and $\{w_6, w_8\}$ are most concerned about the praise degree.

Customer requirements are diversified in the product-selection under price promotion, but most of the customers tend to find the wine combinations whose original prices are just equal to integral multiples of $\$\alpha$ and the actual prices are not beyond the customer's willingness to pay. This means that these wine combinations bring the maximum discount rate due to Property 2. As shown in Table 4, in the process of product selection, about 70 percent of the customers pick out the wine combinations with large discount rates. We place these wine combinations gray backgrounds in Table 4.

On the basis of the analysis above, our COPC problem is significant for customers to select products under price promotion.

TABLE 5
The Parameter Settings on Synthetic Datasets

Parameter	Range	Description
N	64K, 128K, **256K**, 512K	the cardinality of the product dataset P
$UDisRate$	30%, 40%, **50%**, 60%, 70%, 80%	the upper bound of the discount rate which the merchant could offer
WTP	$\{5, 10, \mathbf{15}, 20\} \times \lfloor AvePri(P) \rfloor$	the customer's willingness to pay
ϵ	0.20, 0.40, 0.60, 0.80	the trimming parameter utilized in the LBA algorithm
d	3, **4**, 5	the number of attributes

TABLE 6
PT(ms) of the Proposed Algorithms versus Cardinality of the Skyline Product Set N_S (Ant)

N_S	$AvePri$	α	β	Naive Exact	TLE	LBA (0.20)	LBA (0.40)	LBA (0.60)	LBA (0.80)	IG
5	420.80	400	200	1.077	0.780	0.917	0.874	0.782	0.761	0.622
10	474.50	400	200	4.534	3.312	1.337	1.091	1.032	0.912	0.764
15	465.27	400	200	22.524	4.209	2.848	1.710	1.264	1.237	0.742
20	425.50	400	200	691.879	18.591	7.280	3.901	2.966	2.056	0.821
25	422.04	400	200	34701.793	810.472	20.962	7.254	4.947	3.432	0.863
30	448.77	400	200	$--$	26880.209	47.419	13.501	7.571	5.852	0.925
35	427.20	400	200	$--$	4223055.338	95.598	24.216	12.641	8.770	0.911

6.2 Experimental Setup

In order to test the performance of the proposed algorithms for the COPC problem on different datasets, we adjusted the publicly available data generator provided by [27] to generate synthetic datasets used in the following experiments. We applied the modified generator to generate two types of datasets distributions: Independent (Ind) and Anti-correlated (Ant), respectively. Each dataset is indexed by an R-tree with 4 KB page size.

We use two real datasets, namely Beer and Smartphone. The beer dataset that is extracted from jx.tmall.com contains 2001 tuples, and we take into account three attributes which are the weight, average monthly sales, and price. Besides, the smartphone dataset is gained form jd.com. It contains 4476 tuples with seven attributes which are weight, comments, resolution, number of pixel of the camera, size of memory, number of cores of the CPU, and price.

In practice, merchants usually compute the upper bound of discount rate $UDisRate$ on the basis of product's historical transaction data. Here $UDisRate$ is different from the $MaxDisRate$. $UDisRate$ is the upper bound of the discount rate the online merchants can offer while $MaxDisRate$ represents the maximum discount rate the customers can gain. Since larger $UDisRate$ brings more sales, the total profits, which the merchants can obtain, increase in turn. Therefore, online merchants trend to provide an appropriate $UDisRate$ that can bring them the maximum total profits. Particularly, the price promotion campaign is designed due to the $UDisRate$ which the merchants could offer.

There are numerous approaches to design the price promotion campaign. Consider the concrete approaches have little effect on performance of the proposed algorithms. In this paper, we only take one approach, which is due to $UDisRate$ and the hundreds of $\lfloor AvePri(P) \rfloor$, into account. Here, the average price of products in P is computed by $AvePri(P) = \frac{OriPri(P)}{N}$. If $UDisRate$ is equal to 30% and the hundreds of $\lfloor AvePri(P) \rfloor$ is 5, then the price promotion campaign is "get $\$500 \times 30\% = \$150$ off every \$500 purchase".

The parameter settings of the synthetic datasets are summarized in Table 5 where default values of the parameters are highlighted in bold.

6.3 Experimental Results for the COPC Problem

In this section, since the exact algorithms cannot be utilized to process large skyline product sets, we first evaluate all the proposed algorithms for the COPC problem over several small skyline product sets. Then, we compare the LBA and IG algorithms over some large skyline product sets.

6.3.1 Performance on Small Skyline Product Sets

The cardinalities of the skyline product sets which the exact algorithms can deal with depend on the memory capacity. The more memory capacity, the large skyline product sets the exact algorithms can process. Table 6 shows PT of the proposed algorithms over some small skyline product sets where each product is selected from the skyline set of an Ant dataset with size 256K and $d = 4$ at random. Since the experimental results of the proposed algorithms over an Ind dataset are close to those of an Ant dataset when processing the same number of skyline products. We only illustrate the experimental results over the Ant dataset in this section.

As shown in Table 6, in our experiments, the TLE algorithm can handle the skyline product set with the cardinality $N_S \leq 35$ while the naive algorithm can only deal with the skyline product sets with $N_S \leq 25$. The proposed algorithms need much more PT with the growth of N_S. When processing small skyline product sets, the naive exact and TLE algorithms may have better performance than the LBA and IG algorithms. This is since PT of the algorithms are close but the exact algorithms could gain accurate results. Besides, LBA and IG have advantages in processing large skyline product sets. LBA needs less PT with the growth of ϵ. IG outperforms LBA in term of PT and offers better scalability.

6.3.2 Performance on Large Skyline Product Sets

In this section, we vary the cardinality N, the customer's payment willingness WTP, the upper bound of the discount rate $UDisRate$, and the dimensionality d, respectively, and evaluate the performance of the LBA and IG algorithms for the COPC problem.

Experimental Results on Cardinality N. We research the performance of the LBA and IG algorithms with N varying from 64 K to 512 K, and the other parameters are kept to their default values. The experimental results of LBA and IG

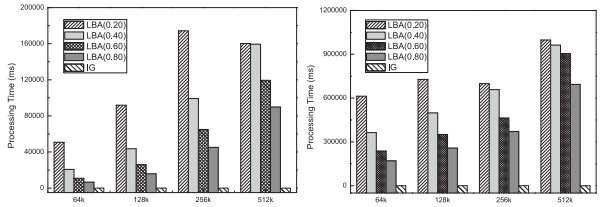


Fig. 1. Performance versus cardinality N (a) Ind and (b) Ant.

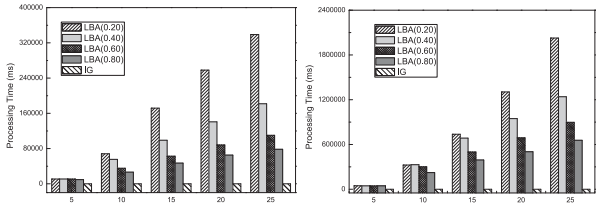


Fig. 2. Performance versus customer's willingness to pay WTP (a) Ind and (b) Ant.

TABLE 7
Running Information versus Cardinality N

N	Ind ($d = 4$)				Ant ($d = 4$)			
	N_S	$AvePri$	α	β	N_S	$AvePri$	α	β
64K	278	264.63	200	100	932	296.23	200	100
128K	399	233.66	200	100	1117	294.23	200	100
256K	635	197.74	100	50	1406	260.71	200	100
512K	1004	156.03	100	50	2047	249.57	200	100

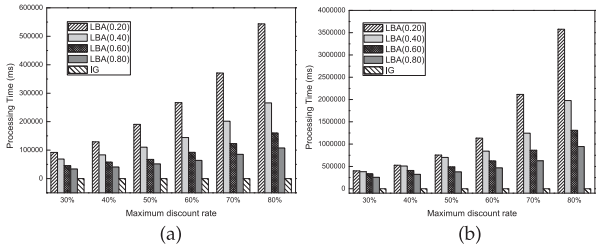


Fig. 3. Performance versus maximum discount rate $UDisRate$ (a) Ind and (b) Ant.

under various N are depicted in Fig. 1. Table 7 shows the running information as varying N. For the Ind dataset with $N = 64$K, it reports a skyline product set with $N_S = 278$, the average price of the skyline products is $AvePri = 264.63$, and the price promotion campaign is "get $\$\beta = \$200 \times UDisRate = \$200 \times 50\% = \$100$ off each \$200 purchase". Similarly, we could get the running information over other datasets as shown in Table 7. The increase of N brings much more skyline product combinations to be considered in general. This also brings the growth of PT. Moreover, IG outperforms LBA by many orders of magnitudes in term of PT. Considering LBA, it needs less PT when ϵ increases in most cases. It is an interesting phenomena that for the LBA algorithm with $\epsilon = 0.2$, PT decreases when processing the Ind datasets with N varying from 256K to 512K. This depends on the values of the price in the datasets. In the LBA algorithm, after pruning unqualified product combinations, the solution space over a large dataset may be less than that over a small dataset. Thus, in some cases, it may need less time to process the larger dataset than the small dataset by the LBA algorithm with the same ϵ.

Experiment Results on Customer's Willingness to Pay WTP. Fig. 2 shows the results of the LBA and IG algorithms for the COPC problem with varying the customer's willingness to pay WTP, and the other parameters are kept to their default values. Here WTP varies from $5 \times \lfloor AvePri(P) \rfloor$ to $25 \times \lfloor AvePri(P) \rfloor$ by a step of $5 \times \lfloor AvePri(P) \rfloor$.

As shown in Fig. 2, PT of the LBA and IG algorithms increases with the growth of WTP. This is because the growth of WTP makes the customer have much more choice. Therefore, much more skyline combinations require to be considered. Furthermore, IG also needs much less PT than LBA. Besides, PT of the LBA algorithm decreases with the growth of ϵ.

Experiment Results on Maximum Discount Rate UDisRate. Furthermore, we also do experiments on varying the maximum discount rate $UDisRate$, which a merchant can offer, from 30% to 80%, and the other parameters are kept to their default values. The price promotion is "get $UDisRate \times (15 \times \lfloor AvePri(P) \rfloor)$ off every $15 \times \lfloor AvePri(P) \rfloor$ purchase". Obviously, the price promotion campaigns vary with the change of $UDisRate$ as shown in Fig. 3.

As the growth of $UDisRate$, customers could buy much more products with the same WTP. This is similar to increase the customer's payment willingness WTP. Because it requires to compute much more skyline product combinations in turn, the LBA and IG algorithms require much more PT as the growth of $UDisRate$. Additionally, IG needs much less PT compared to LBA, PT of LBA decreases with the growth of ϵ.

Experiment Results on Dimensionality d. Tables 8 and 9 show the results of the LBA and IG algorithms for the COPC problem with varying d from 3 to 5 by a step of 1. The dimensionality d has a significant effect on the proposed algorithms. As d grows, PT significantly increases as illustrated in Tables 8 and 9. This is since the cardinality of skyline product set N_S increases with the growth of d. Moreover, IG four orders of magnitude faster than LBA. Considering the LBA algorithm, PT decreases with the growth of ϵ.

Experiment Results on α. We evaluate the performance of the LBA and IG algorithms under different price promotion campaigns. Tables 10 and 11 show the results of LBA and IG for the COPC problem with varying α, and the other parameters are kept to their default values. We generate α randomly and choose values due to the average prices of products in the datasets.

As shown in Tables 10 and 11, PT of the LBA and IG algorithms decrease with the growth of α. This is because the larger α is, the less products the customer can buy. Therefore, less skyline product combinations need to be generated and checked. The IG algorithm requires less PT and has better scalability than the LBA algorithm. As ϵ increases, PT of the LBA algorithm decreases.

6.4 Performance on Real Datasets

In this subsection, we report experimental results on the real datasets, Beer and Smartphone. Fig. 4 shows PT of the LBA and IG algorithms for the COPC problem over the above datasets, respectively.

We apply the price promotion campaigns "get 100 yuan off 199 yuan" and "get 400 yuan off 1999 yuan", which are offered by Jingdong and Alibabas Taobao Mall, for the beer and smartphone, respectively.

The results on the real datasets are consistent with the ones obtained from the experiments on the synthetic datasets. As illustrated in Fig. 4, PT of the LBA and IG algorithms

TABLE 8
PT (ms) versus Dimensionality d (Ind)

d	N_S	LBA (0.20)	LBA (0.40)	LBA (0.60)	LBA (0.80)	IG
3	381	28928.412	16891.914	10112.268	6539.403	1.182
4	635	174322.639	99205.364	65055.745	45076.783	1.679
5	1713	1137468.790	1077165.967	758256.252	605025.628	2.873

TABLE 9
PT (ms) versus Dimensionality d (Ant)

d	N_S	LBA (0.20)	LBA (0.40)	LBA (0.60)	LBA (0.80)	IG
3	422	61233.599	30273.884	17912.103	11322.974	1.547
4	1406	699495.325	657629.099	463531.221	371199.523	2.225
5	6099	5812650.716	5602174.722	4785533.809	4063466.844	8.663

TABLE 10
PT(ms) of the LBA and IG Algorithms versus α (Ind)

α	β	LBA (0.20)	LBA (0.40)	LBA (0.60)	LBA (0.80)	IG
222	110	179086.773	97580.890	62211.985	43840.969	0.4904
495	240	160603.879	93662.425	58523.776	41709.408	0.2880
676	330	159672.526	92828.485	57096.462	39977.643	0.3205
893	440	152736.130	90040.361	56501.649	39902.512	0.2651

TABLE 11
PT(ms) of the LBA and IG Algorithms versus α (Ant)

α	β	LBA (0.20)	LBA (0.40)	LBA (0.60)	LBA (0.80)	IG
265	130	720305.956	675961.955	495234.720	378460.696	1.6512
435	210	712640.120	664243.563	451091.051	356581.461	0.8205
634	310	706395.286	612276.516	431516.169	335209.025	0.7913
848	420	696462.452	604887.070	424758.962	333665.256	0.5948

increases as WTP grows. Here for the Beer, the customer's payment willingness WTP is changed from $5 \times \lfloor AvePri(P) \rfloor$ to $25 \times \lfloor AvePri(P) \rfloor$ by a step of $5 \times \lfloor AvePri(P) \rfloor$. Consider the smartphone, we vary WTP from $2 \times \lfloor AvePri(P) \rfloor$ to $6 \times \lfloor AvePri(P) \rfloor$ by a step of $\lfloor AvePri(P) \rfloor$. The IG algorithm outperforms the LBA algorithm in term of PT. For the LB algorithm, PT reduces with the growth of ϵ.

6.5 Summary

As analyzed above, the naive exact and TLE algorithms are appropriate to process small skyline product sets. The LBA and IG algorithms have advantages in dealing with large skyline product sets. The IG algorithm always requires far less PT with comparing to LBA. Compared to the exact algorithms and the LBA algorithm, the IG algorithm has the best scalability. For the LBA algorithm, it results in the degradation of PT with the increase of ϵ in most cases.

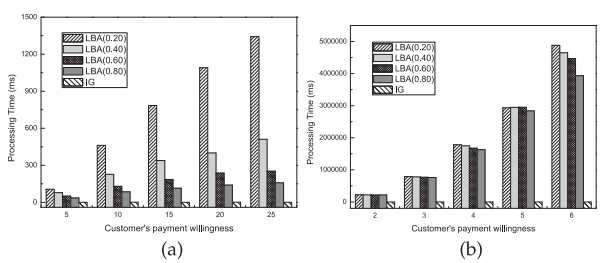


Fig. 4. Performance versus customer's willingness to pay WTP (a) Beer and (b) Smartphone.

It always needs more PT to process the Ant datasets than the Ind datasets. This is reasonable because N_S of the Ant datasets is larger than that of the Ind datasets with equal cardinality. As N, WTP, $UDisRate$ or d grows, it faces much more candidate skyline product combinations, and the proposed algorithms need much more PT.

7 Conclusions

In this paper, we formulate the COPC problem to retrieve optimal skyline product combinations that satisfy the customer's payment constraint and bring the maximum discount rate. To tackle the COPC problem, we propose an exact algorithm, design an approximate algorithm with an approximate bound, and develop an incremental greedy algorithm to boost the performance. We conduct a customer study to verify the significant of our COPC problem. Additionally, the experimental results on both real and synthetic datasets illustrate the effectiveness and efficiency of the proposed algorithms.

This work opens to some promising directions for future work. First, in addition to combinations of homogeneous products, we will focus on the COPC problem over products of different categories. After that, in reality, the customer's demands are diversification and individuation, and it is significant and interesting to compute optimal product combinations that meet different customer demands such as save or spend the most money under their budgets. Last but not least, we could also research top k COPC problem that aims to compute k optimal product combinations due to customer demands based on the work in [17], [28], [29].

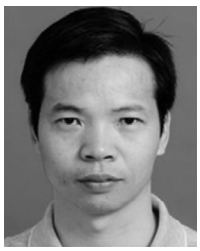
ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable and helpful comments on improving the manuscript. The research was partially funded by the National Key R&D Program of China (Grant No. 2016YFB0201303), the Key Program of National Natural Science Foundation of China (Grant No. 61432005), the International (Regional) Cooperation and Exchange Program of National Natural Science Foundation of China (Grant No. 61661146006), the National Outstanding Youth Science Program of National Natural Science Foundation of China (Grant No. 61625202), the National Natural Science Foundation of China (Grant Nos. 61472126, 61602170, 61772182), the National High-tech R&D Program of China (Grant No. 2015AA015303), China Postdoctoral Science Foundation (Grant No. 2016M602410), and the Science and Technology Plan of Changsha (K1705032).

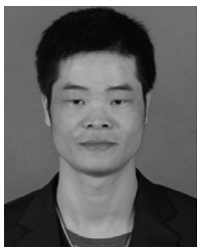
REFERENCES

[1] S. Börzsönyi, D. Kossmann, and K. Stocker, "The skyline operator," in *Proc. Int. Conf. Data Eng.*, 2001, pp. 421–430.
[2] Q. Wan, R. C.-W. Wong, I. F. Ilyas, M. T. Özsu, and Y. Peng, "Creating competitive products," *Proc. VLDB Endowment*, vol. 2, no. 1, pp. 898–909, 2009.
[3] I.-F. Su, Y.-C. Chung, and C. Lee, "Top-k combinatorial skyline queries," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 2010, pp. 79–93.
[4] Y.-C. Chung, I.-F. Su, and C. Lee, "Efficient computation of combinatorial skyline queries," *Inf. Syst.*, vol. 38, no. 3, pp. 369–387, 2013.
[5] H. Im and S. Park, "Group skyline computation," *Inf. Sci.*, vol. 188, pp. 151–169, 2012.
[6] M. Magnani and I. Assent, "From stars to galaxies: skyline queries on aggregate data," in *Proc. 16th Int. Conf. Extending Database Technol*, 2013, pp. 477–488.
[7] N. Zhang, C. Li, N. Hassan, S. Rajasekaran, and G. Das, "On skyline groups," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 4, pp. 942–956, Apr. 2014.
[8] J. Liu, L. Xiong, J. Pei, J. Luo, and H. Zhang, "Finding pareto optimal groups: Group-based skyline," *Proc. VLDB Endowment*, vol. 8, no. 13, pp. 2086–2097, Sep. 2015.
[9] W. Yu, Z. Qin, J. Liu, L. Xiong, X. Chen, and H. Zhang, "Fast algorithms for pareto optimal group-based skyline," in *Proc. Int. Conf. Inf. Knowl Manage.*, 2017, pp. 417–426.
[10] H. Lu, C. S. Jensen, and Z. Zhang, "Flexible and efficient resolution of skyline query size constraints," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 7, pp. 991–1005, Jul. 2011.
[11] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 41–82, 2005.
[12] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang, "Selecting stars: The k most representative skyline operator," in *Proc. 23th Int. Conf. Data Eng.*, 2007, pp. 86–95.
[13] C.-Y. Lin, J.-L. Koh, and A. L. Chen, "Determining k-most demanding products with maximum expected number of total customers," *IEEE Trans. Knowl. Data Eng*, vol. 25, no. 8, pp. 1732–1747, Aug. 2013.
[14] Q. Wan, R.-W. Wong, and Y. Peng, "Finding top-k profitable products," in *Proc. 27th Int. Conf. Data Eng.*, 2011, pp. 1055–1066.
[15] C.-Y. Chan, H. Jagadish, K.-L. Tan, A. K. Tung, and Z. Zhang, "On high dimensional skylines," in *Proc. Adv. Database Technol.*, 2006, pp. 478–495.
[16] M. Magnani, I. Assent, and M. L. Mortensen, "Taking the big picture: representative skylines based on significance and diversity," *VLDB J.*, vol. 23, no. 5, pp. 795–815, 2014.
[17] G. Xiao, K. Li, X. Zhou, and K. Li, "Efficient monochromatic and bichromatic probabilistic reverse top-k query processing for uncertain big data," *J. Comput. Syst. Sci.*, vol. 89, pp. 92–113, 2016.
[18] Y. Gao, Q. Liu, L. Chen, G. Chen, and Q. Li, "Efficient algorithms for finding the most desirable skyline objects ," *Knowl.-Based Syst.*, vol. 89, no. C, pp. 250–264, 2015.
[19] M. Bai, J. Xin, G. Wang, and L. Zhang, "Discovering the k representative skyline over a sliding window," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 8, pp. 2041–2056, Aug. 2016.
[20] X. Zhou, K. Li, G. Xiao, Y. Zhou, and K. Li, "Top k favorite probabilistic products queries," *IEEE Trans. Knowl. Data Eng*, vol. 28, no. 10, pp. 2808–2821, Oct. 2016.
[21] X. Zhou, K. Li, Y. Zhou, and K. Li, "Adaptive processing for distributed skyline queries over uncertain data," *IEEE Trans. Knowl. Data Eng*, vol. 28, no. 2, pp. 371–384, Feb. 2016.
[22] L. Chen, B. Cui, and H. Lu, "Constrained skyline query processing against distributed data sites," *IEEE Trans. Knowl. Data Eng*, vol. 23, no. 2, pp. 204–217, Feb. 2011.
[23] T. H. Cormen, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 2009.
[24] F. B. Chedid, "A note on developing optimal and scalable parallel two-list algorithms," in *Proc. Int. Conf. Algorithms Archit. Parallel Process.*, 2012, pp. 148–155.
[25] C. A. A. Sanches, N. Y. Soma, and H. H. Yanasse, "Observations on optimal parallelizations of two-list algorithm," *Parallel Comput.*, vol. 36, no. 1, pp. 65–67, 2010.
[26] A. G. Ferreira, "A parallel time/hardware tradeoff t.h=o(2/sup n/2/) for the knapsack problem," *IEEE Trans. Comput.*, vol. 40, no. 2, pp. 221–225, Feb. 1991.
[27] Y. Tao, X. Xiao, and J. Pei, "Efficient skyline and top-k retrieval in subspaces," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 8, pp. 1072–1088, Aug. 2007.
[28] J.-L. Koh, C.-Y. Lin, and A. L. Chen, "Finding k most favorite products based on reverse top-t queries," *VLDB J.*, vol. 23, no. 4, pp. 541–564, 2014.
[29] M. S. Islam and C. Liu, "Know your customer: Computing k-most promising products for targeted marketing," *VLDB J.*, vol. 25, no. 4, pp. 545–570, 2016.

Xu Zhou received the PhD degree in computer science from Hunan University, Changsha, China, in 2016. She is currently a postdoctoral with the Department of Information Science and Engineering, Hunan University. Her research interests include parallel computing and data management.



Kenli Li received the PhD degree in computer science from Huazhong University of Science and Technology, China, in 2003. His major research interests include parallel computing, high-performance computing, and grid and cloud computing. He has published more than 200 research papers in international conferences and journals such as the *IEEE Transactions on Computers*, *IEEE Transactions on Parallel and Distributed Systems*, and ICPP. He is a senior member of the IEEE and serves on the editorial board of the *IEEE-TC*.



Zhibang Yang received the PhD degree in computer science from Hunan University, Changsha, China, in 2012. His major research interests include data management and parallel computing. He is currently an associate professor with Changsha University.

Keqin Li is a SUNY distinguished professor of computer science with the State University of New York. His current research interests include parallel computing and distributed computing. He has published more than 550 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently serving or has served on the editorial boards of the *IEEE Transactions on Computers*, the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Cloud Computing*, the *IEEE Transactions on Services Computing*, and the *IEEE Transactions on Sustainable Computing*. He is a fellow of the IEEE.