MANAGING PRODUCT-DISCOUNTS USING LOWER BOUND APPROXIMATION ALGORITHM

C.Vineeth Kumar1 K.Satish Kumar2 K.Kavya3 B.M.Lakshmipathi4
Dr.D.Sumathi5
Department of Computer Science and Engineering
Kuppam Engineering College
Kuppam, AP, India

Abstract: Recently, online shopping marketplaces usually hold some price promotion campaigns to attract customers and increase their purchase intention. E-com web site provides best time consumption and efficient results. Considering the requirements of customers in this practical application scenario, we are concerned about product selection under price promotion. To make the machines to understand the data by using the semantic web, it is used in the connection of information as used to link the data from one source to another meanwhile it also makes the system to understand the data[10]. We formulate a Constrained Optimal Product Combination (COPC) problem. It aims to find out the Product discounts 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 we first reviewed 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 the 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 .customers under price promotion, which is certified by a customer study. To process the COPC problem effectively.

Index Terms: **Semantic Web, Online marketing, Constrained Optimal Product Algorithm, Two List Exact algorithm, Incremental Greedy algorithm**

I: INTRODUCTION

Most products are created with a single product variant. But some products may be best suited for a single variant but multiple product variants attributes, as is the case with a shirt that comes in different colors and sizes. The steps shown below will assist in creating that scenario. See 'Understanding products' for more information on all options for creating products. with the development of e-commerce, a growing number of customers choose to go shoppinonline because it saves time and effort. However, it always contraries to the expectations of customers. This is because they may need to pick up one choice among thousands of products. To help customers identify attractive products, an Product discounts query[1] is admittedly a common and effective methodology. According to the definition of the Product discounts query, a product that is not dominated by any other product is said to be an Product discounts product or it is in the Product discounts. The products in the Product discounts are the best possible trade-offs between all the factors that customers care about. The Product discounts query is useful in identifying attractive products.

In marketing, product bundling is offering several products or services for sale as one combined product or service package. It is a common feature in many imperfectly competitive product and service markets. Industries engaged in the practice include telecommunications services, financial services, health care, information, and consumer electronics. A software bundle might include a word processor, spreadsheet, and presentation program into a single office suite. The cable television industry often bundles many TV and movie channels into a single tier or package. The fast-food industry combines separate food items into a "meal deal" or "value meal".

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. 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 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 Product discounts queries that 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 that 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 Product discounts query is powerful to compute the Product discounts products that have a strong appeal to customers. However, it is inadequate to help customers select Product discounts product combinations with the greatest benefit.

E-commerce is the activity of buying or selling products on online services or over the Internet. Electronic commerce draws on technologies such as mobile commerce, electronic funds transfer, supply chain management, Internet marketing, online transaction processing, electronic data interchange (EDI), inventory management systems, and automated data collection systems. Modern electronic commerce typically uses the World Wide Web for at least one part of the transaction's life cycle although it may also use other technologies such as e-mail. Typical e-commerce transactions include the purchase of online books (such as Amazon) and music purchases (music download in the form of digital distribution such as iTunes Store), and to a less extent, customized/personalized online liquor store inventory services.[1] There are three areas of e-commerce: online retailing, electric markets, and online auctions. E-commerce is supported by electronic business.

II: OBJECTIVES

Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.

It is achieved by creating user-friendly screens for the data entry to handle a large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.

When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as to when needed so that the user will not be in maize of instant. Thus the objective of the input design is to create an input layout that is easy to follow.

3:LITERATURE SURVEY

Suppose you are going on holiday, and you are looking for a hotel that is cheap and close to the beach. Unfortunately, these two goals are complementary as the hotels near the beach tend to be more expensive. The database system at your travel agents' is unable to decide which hotel is best for you, but it can at least present you all interesting hotels. Interesting is all hotels that are not worse than any other hotel in both dimensions. We call this set of interesting hotels the Product discounts. From the Product discounts, you can now make your final decision, thereby weighing your personal preferences for price and distance to the beach. Computing the Product discounts is known as the maximum vector problem [1]. We use the term Product discounts because of its graphical representation (see below). More formally, the Product discounts is defined as those points which are not dominated by any other point. A point dominates another point if it is as good or better in all dimensions and better in at least one dimension. For example, a hotel with price = $50 and distance = 0.8 miles dominates a hotel with price = $100 and distance = 1 .O miles[3].

This index covers all technical items — papers, correspondence, reviews, etc. — that appeared in this periodical during 2016, and items from previous years that were commented upon or corrected in 2016. Departments and other items may also be covered if they have been judged to have archival value. The Author Index contains the primary entry for each item, listed under the first author's name. The primary entry includes the coauthors' names, the title of the paper or another item, and its location, specified by the publication abbreviation, year, month, and inclusive pagination. The Subject Index contains entries describing the item under all appropriate subject headings, plus the first author's name, the publication abbreviation, month, and year, and inclusive pages. Note that the item title is found only under the primary entry in the Author Index[2]

The importance of dominance and Product discounts analysis has been well recognized in multi-criteria decision-making applications. Most previous works study how to help customers find a set of "best" possible products from a pool of given products. In this paper, we identify an interesting problem, creating competitive products, which have not been studied before. Given a set of products in the existing market, we want to study how to create a set of "best" possible products such that the newly created products are not dominated by the products in the existing market. We refer such products as competitive products. A straightforward solution is to generate a set of all possible products and check for dominance relationships. However, the whole set is quite large. In this paper, we propose a solution to generate a subset of this set effectively. An extensive performance study using both synthetic and real datasets is reported to verify its effectiveness and efficiency.

There have been great interests in representing data using compact binary codes in recent developments. Compact binary codes not only facilitate storage of large-scale data but also benefit fast similarity computation, so that they are applied to the fast nearest neighbor search, as it only takes a very short time (generally less than a second) to compare a query with millions of data points. For learning compact binary codes, a number of hash function learning algorithms have been developed in the last five years. There are two types of hashing methods: the data independent ones and the data dependent ones. Typical data independent hash models include Locality Sensitive Hashing (LSH) and its variants like `p-stable hashing, min-hash and kernel LSH (KLSH). Since using the information of data distribution or class labels would make a significant improvement in fast search, more efforts are devoted to the data-dependent approach. For the data dependent hashing methods, they are categorized into unsupervised-based, supervised-based, and semi-supervised-based hash models. In addition to these works, multi-view hashing, multimodal hashing, and active hashing have also been developed.

Lemma2.1: Given a customer's payment willingness WTP and the price promotion that is getting $β off every $α purchase, the maximum discount number the customer can obtain is MaxDisNum=⌊WTP α−β⌋.Proof: Given a discount product combination SP′⊆SP, its original price is OriPri(SP′)=t×α+r, where the discount number t=⌊OriPri(SP′) α ⌋, and the discountDiscount (SP′)=⌊OriPri(SP′) α ⌋×β. It holds thatActPay (SP′)=OriPri(SP′)−Discount(SP′) =OriPri(SP′)−⌊OriPri(SP′) α ⌋×β= t×α+r−t×β. Since pay(SP′)≤WTP, we have t×α+r−t×β≤WTP. Therefore, it holds that t≤WTP−r α−β . Because 0≤r<α, we gain t≤WTP−r α−β ≤WTP α−β . Therefore, it holds that MaxDisNum=⌊WTP α−β⌋ and this lemma holds.

Theorem2.1: 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 [7], [8], [9]. The subset sum problem is defined as follows: Subset sum problem. Given a positive integer set W={w1,w2,...,wn} and a positive integer M, is there a subset W′⊆W such that∑w∈W′ w=M? For a skyline product combination SP′⊆SP, assume that OriPri(SP′)=∑p∈SP′ OriPri(p)=t×α+r where t=⌊OriPri(SP′) α ⌋, and r=OriPri(SP′)modα. On the basis of Property 2, we can get the maximum discount rate when OriPri(SP′) is a multiple of α, and OriPri(SP′)=∑pi∈SP′ OriPri(pi)=t×α. Here t is a positive integer and 1≤t≤⌊WTP α−β⌋due to Lemma 3.1. In the subset sum problem, let element wi∈W represent the original price of a skyline product pi∈SP, and M=t×α. Due to Property 2, if ∑w∈W′ w=t×α, 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 into 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 [8]. Furthermore, our COPC problem is more complex than the subset sum problem.

4:TWO_LIST_EXACT ALGORITHM

Due to Theorem 2.1, 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 [4], [5], and present a two list exact algorithm for the COPC problem.

Lemma 3.1: Given a price promotion campaign “get $β off every $α purchase”, and skyline product combinations SP ′,SP″⊆SP with i×α≤OriPri(SP′)<OriPri(SP″)< (i+1)×α for I is an integer and i∈[0,MaxDisNum], it holds that DisRate(SP′)>DisRate(SP″). Proof: Since i×α≤OriPri(SP′)<(i+1)×α, it holds that i=⌊OriP(SP′) α ⌋for 1≤i≤⌊WTP α−β⌋according to Lemma 3.1. Due to Equation (1), we have DisRate(SP′)=⌊OriP(SP′) α ⌋×βOriP (SP′) = i×β OriP(SP′) . Similarly, we have DisRate(SP″)= i×β OriP(SP″). Since OriPri(SP′)<OriPri(SP″),it has DisRate(SP′)>DisRate(SP″), and this lemma holds.

Algorithm:

Input: The Product discounts product set SP, a price promotion campaign

"get $β off every $α purchase", and a customer's payment

willingness WTP

Output: A result set SP∗ of the COPC problem

1: Divide SP into two parts: SP1={sp1, sp2, . . . , spNS/2} andSP2={spNS/2+1, spNS/2+2, . . . , spNS}
2: Generate all the product combinations SP′⊆SP1 with ActPay(SP′)≤WTP, sort them in increasing order of OriPri(SP′), and store OriPri(SP′) as the list A={a1, a2, . . . , aN1}
3: Compute a∗∈A which is with the maximum discount rate
4: Generate all the product combinations SP′⊆SP2 with ActPay(SP′)≤WTP, sort them in a descending order of OriPri(SP′), and store OriPri(SP′) as the list B={b1, b2, . . . , bN2
}

5: Compute b∗∈B which is with the maximum discount rate
6:SP∗=argmaxOriPri(SP′)∈{a∗,b∗}DisRate(SP′)
7: Set the maximum discount number MaxDisNum=⌊WTPα−β⌋
due to Lemma 3.1
8: for k=1 to MaxDisNum do
9: Initialize i=1, flag=0 and y∗k=(k+1)×α
10: for ai∈A do
11: j=flag+1
12: for bj∈B do
13: if ai+bj is equal to k×α then
14: y∗k=k×α and Break
15: else
16: if ai+bj>k×α then
17: j=j+1
18: y∗k=min{y∗k, ai+bj}
19: else
20: i=i+1
21: flag=j
22.AddSP″=argmaxOriPri(SP′)=y∗jDisRate(SP′) for 1≤j≤MaxDisNum to SP∗ and refresh SP∗ by removing the combinations whose discount rates are less than that of SP″
23: Return SP∗

In two list exact algorithm, we mainly deal with the products which provide more discount when we purchase a combination of products and if we purchase one product is going to provide another product for free. For example, if we purchase toothpaste is going to provide toothbrush freely.

Example 1

Going back to the example in, by the Product discounts query, we have a wine set W={w4;w5;w6;w8}where each wine is in the offered bench. Dividing the set W into two partsW1={w4;w5} andW2={w6;w8}. Line 2 generates the wine combinations {w4}; {w5}, and {w4;w5} over W1.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 {w4;w5} with OriPri({w4;w5})=430 brings the maximum discount rate.Line 4 generates the wine combinations {w6}; {w8}, and{w6;w8} over W2. After sorting them in descending order of their original prices, we have the list B={390; 210; 180}, and b∗=210. Line 6 gets the wine combination {w6} which matches the current maximum discount rate 0.286. Line 7 computes the maximum discount number due to Lemma1 have MaxDisNum=400/[200−60]=2. Lines 8 to 22are utilized to combine the elements within A and B. Firstly, 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×α=200. By combining a1=190 with bjεB, there is not any combination whose sum is just equal to 200, and we get the combination {a1; b3}={190; 180} whose sum is no less than but nearest to 200. We also get flag=3. Considering the second element a2=240, it only needs to be combined with the element b3 and other elements ranked after it within B.We have y*1=a1+b3=370, SP″={w5;w8}, and SP*={w6}.Similarly, for k=2, we get y*2=a1+b2=400, SP″={w5;w6},and SP*={w5;w6}. Finally, the wine combination {w5;w6}is returned as the final result of COPC.

5:LOWER BOUND APPROXIMATE (LBA) ALGORITHM

Based on Lemmas 2.1 and 3.1, and Theorem 2.1, we design a lower bound approximate algorithm for the COPC problem, which is depicted in Algorithm.

The LBA algorithm first removes each product p′εSP whose actual payment is larger than WTP (Line 1). Line2 initializes a list L with a set that contains an element"0".

Thereafter, the list L stores original prices of candidates Product discounts product combinations. Lines 3-10 are applied to find candidates Product discounts product combinations which may bring the maximum discount rate. In Line 3, it computes.

Algorithm

Input: The Product discounts product set SP with |SP|=NS, a price promotion campaign "get $β off every $α purchase", a customer's payment willingness WTP, and a trimming

parameter ϵ for 0<ϵ<1

Output: A result set SP* of the COPC problem

1: Remove each product p′€SP with Act Pri(p′)>WTP
2: Initialize L={0}
3: Set the maximum discount number Max DisNum=[WTP/α−β]due to Lemma 3.1
4: Initialize y*j=∞ for 1≤j≤MaxDisNum
5: while SP is not empty do
6: L=L∪{y+Ori(p) : y€L} for p€SP
7: Sort all the elements in L in an increasing order an remove each element y from L ify−⌊ yα[y/α]×β>WTP
8: Compute y*jεL where @y′εL−{yj*}, y′<yj* withy′, y*jε[j×α, (j+1)×α) for 1≤j≤MaxDisNum and j is an integer
9: Remove each element y from L which is larger thany*MaxDisNum
10:L=Trim(L−yj*ϵ2NS)for1≤j≤MaxDisNum
11: Return SP*=arg maxOriPri(SP′)=yj*
DisRate(SP′) for j is an integer
and 1≤j≤MaxDisNum
12: Function: Trim(L, δ)
13: Initialize L′={y1}
14: last=y1
15: for i=2 to |L| do
16: if yi>last×(1+δ) then
17: Append yi onto the end of L′
18: last=yi
19: Return L′

in the lower bound approximate algorithm, we are going to deal with a combination of products which has provided more discounts among all possible combinations of products.

Example 2.

Continuing with the example in 1, after getting the Product discounts set {w4;w5;w6;w8}, Max Dis Num is computed by[400/200−60]=2. Assume that "=0:6 and"2NS= 0:62×4=0:075. The LBA generates the combinations whose original prices are as small as possible but not less than j×α for jε{1; 2}. Firstly, we have a listL1={0; 240}, y1*=240, and initialize y2=∞. Considering the wine w5, we have L2= {0; 190; 240; 430} by merging L1 and{y+OriPri(w5) for yεL1}, y1*=240 and y 2*=430. By invoking the Trim function over the list L2−{y1*; y2*}={0; 190},we have L2−{y1*; y2*}={0; 190} and L2={0; 190; 240; 430}.This is since 190>0×(1+ 0:62×4)=0. And then, we obtainL3={0; 190; 210; 240; 400; 430; 450; 640} by mergingL2 and {y+OriPri(w6) for yεL2}, y1*=210, and y2=400.Through removing the elements 430, 450, and 640are larger than y2*=400 from L3, we gain L3={0; 190; 210; 240; 400}. After trimming L3−{y1*; y2*},we have L3={0; 190; 210; 240; 400}. Considering the wine w8, we obtain L4={0; 180; 190; 210; 240; 370; 390;400; 420; 580}, y1*=210, and y2*=400. By removing the elements 420 and 580 which are larger than y2*=400 from L4, we gain L4={0; 180; 190; 210; 240; 370; 390; 400}.After trimming L4−{y1*; y2*}, it holds that L4={0; 180;210; 240; 370; 400} by removing the elements 190 and390.This is because 190<180×(1+0:62×4)=193:5and390<370×(1+ 0:62×4)=397:75. Now, we have y1*=210 andy2*=400 which are as small as possible but not less than j×αε{200; 400}. The original prices of the wine combinations {w6} and {w5;w6} are equal to y1*=210and y2*=400 respectively. Lastly, the wine combination

{w5;w6} is reported as the final result of COPC sinceDis Rate({w5;w6})=0:300>Dis Rate({w6})=0:289.

Theorem 4.1: LBA is a (1+")-approximation algorithm for the COPC problem.

Proof: Let Oi denote the set of all values obtained by selecting a subset of {pεSP|OriPri(p)}, summing its members.

The list Li contains a suitably trimmed version of the set Oi. After removing all the elements that ensure not to be the final results from Li and trimming the list Li, each element of Li is also the element of Oi which represents the original prices of some Product discounts product combinations. For each element yεOi, there is an element y′εOi∩Li such that(y1+ "2n)≤y′≤y; where n=NS=|SP|. Let y*jεOn represent the element which is as small as possible but not less than j×αfor 1≤j≤MaxDisNum.There is an element zjεLn such that y*j/(1+ε/2n)(j1+"2n)n≤zj≤y*j : Thus yj*/*zj≤(1 +ε/2n)n Therefore for the approximate optimal solutionzj*εLn which is as small as possible but not less thanj×_, we also have yj*zj*≤(1+"2n)n:Since (1 + "2n)n≤1+",it holds that yj*/zj*≤1+".Finally, after computing the discount rates of the Product discounts product combinations whose original prices are equal to yj* , we obtain the ones having the maximum discount rate as the final results.

6:INCREMENTAL GREEDY ALGORITHM

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

Input: The Product discounts set SP of a product dataset P, a price promotion campaign "getting $β off every $α purchase", and a customer's payment willingness WTP

Output: A result set SP* of the COPC problem

1: Remove each product pεSP with Act Pay(p)>WTP
2: Initialize Pre P=θ
3: Compute product combinations {p} where pεSP and p are with the highest discount rate, and add them to PreP
4: Initialize SP*=PreP
5: Initialize Max R=D is Rate({p}) for {p}εPre P
6: while PreP is not empty do
7: TempMax R=0 and a set CandSet=θ
8: for each candidate product combination SP′=PreP do
9: PreP=PreP−SP′
10: for each product pεSP−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 C and Set
16: Add SP″ to C and Set
17: else
18: if DisRate(SP″)=TempMax R then
19: Add SP″ to C andSet
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*∪C andSet
26: PreP=CandSet
27: Return SP*

As depicted in Algorithm 3, the IG algorithm first removes all the Product discounts 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 many more products.

It improves the performance of an application which deals with customers module, it optimizes the user interaction phase or path.

Example 3:

Going back to the example in , by the Product discounts query over the wine set, it gets the Product discounts set SP={w4;w5;w6;w8}. Since the wine w6 is with the largest discount rate compared to other wines w4;w5, and w8, {w6} is inserted into Pre P. We have SP*={{w6}} and Max R=0:286. Next, by combining{p} with {w6}εPre P for each pε{w4;w5;w8},we get new wine combinations {w4;w6}, {w5;w6} and{w8;w6}. Since {w5;w6} gets the local maximum discount rate, it holds that CandSet={{w5;w6}} and TempMax R=DisRate({w5;w6})=0:300. Here, we have SP*={{w5;w6}},Max R=0:300, and PreP={{w5;w6}}. Then, we generate two new wine combinations {{w4;w5;w6}} and {{w5;w6;w8}}. For Act Pay ({w4;w5;w6})=460>400, {w4;w5;w6}is not a legal result of the COPC problem. Similarly SP″={{w5;w6;w8}} is pruned as an unqualified result. Finally, {w5;w6}εSP* is returned as the final result.

7: EXPERIMENTAL EVALUATION

In this section, similar to [6], 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.

7.1 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.

7.2 Experiment results on customer's willingness to pay WTP.

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×⌊AvePri(P)⌋ to 25×⌊AvePri(P)⌋by a step of 5×⌊AvePri(P)⌋. As shown in Fig. 1, PT of the LBA and IG algorithms increases with the growth of WTP. This is because of them. 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.

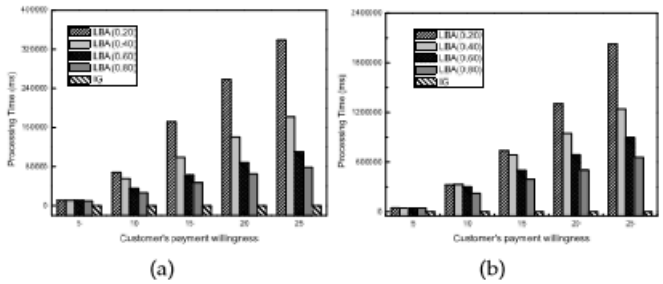

Fig 1: Performance vs. customers willingness to pay

WTP (a) Ind (b) Ant

7.3 Experiment results on maximum discount rate UD is rate

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×(15×⌊AvePri(P)⌋) off every 15×⌊AvePri(P)⌋purchase". Obviously, the price promotion campaigns vary with the change of UD is Rate as shown in Fig. 3. As the growth of UD is Rate, customers could buy many 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 UD is Rate. Additionally, IG needs much less PT compared to LBA, PT of LBA decreases with the growth of ε.

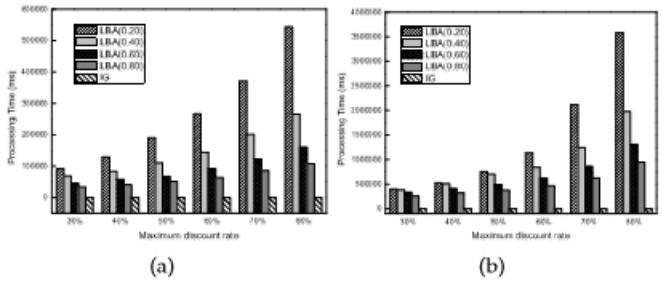

Fig. 2: Performance vs. Maximum Discount rate

UDisRate (a) Ind (b) Ant

CONCLUSION

In this system, we formulate the COPC problem to retrieve optimal Product discounts product combinations that satisfy the customer's payment constraints 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 signature of our COPC problem. Additionally, the experimental results on both real and synthetic datasets illustrate the effectiveness and efficiency of the proposed algorithms.

REFERENCES:

[1]. S. B¨orzs¨onyi, D. Kossmann, and K. Stocker, "The skyline operator," in Proc. Intel Conf. Data Eng. (ICDE), pp. 421–430, 2001.

[2]. Q. Wan, R. C.-W. Wong, I. F. Ilyas, M. T.O¨zsu, and Y. Peng, "Creating competitive products, "Proc.ofthe VLDB Endowment, vol. 2, no. 1, pp. 898–909, 2009.

[3]. X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang, "Selecting stars: The k most representative skyline operator," in Proc. 23th Int'l Conf. Datang. (ICDE), pp. 86–95, IEEE, 2007.

[4]. F. B. Chedid, "A note on developing optimal and scalable parallel two-list algorithms," in International Conference on Algorithms and Architectures for Parallel Processing, pp. 148–155, 2012.

[5]. C. A. A. Sanches, N. Y. Soma, and H. H. Yanasse, "Observations on optimal parallelizations of a two-list algorithm," Parallel Computing, vol. 36, no. 1, pp. 65–67, 2010.

[6]. J. Liu, L. Xiong, J. Pei, J. Luo, and H. Zhang, "Finding Pareto optimal groups: Group-based skyline," Proc. of the VLDB Endowment, vol. 8, no. 13, 2015.

[7]. 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. on Information and Knowledge Management, pp. 417–426, 2017.

[8]. T. H. Cormen, Introduction to Algorithms. MIT Press, 2009.

[9]. D. Sumathi and P. Poongodi, 'Improved Scheduling Strategy in Cloud using Trust-Based Mechanism', International Journal of Computer and System Engineering, Vol-9, No-2, 2015.

[10]. Pascal Hitzler and Krzysztof Janowicz,'About the Semantic Web Journal', IOS Press, 2019.