Crowdrebate: How to get more rebate

Wangze Ni [†], Nian Chen [†], Peng Cheng [*], Lei Chen [†], Xuemin Lin [#,*]
[†]*The Hong Kong University of Science and Technology, Hong Kong, China*
{wniab, leichen}@cse.ust.hk, {nchenaf}@connect.ust.hk
[*]*East China Normal University, Shanghai, China*
pcheng@sei.ecnu.edu.cn
[#]*The University of New South Wales, Australia*
lxue@cse.unsw.edu.au

***Abstract*—With the rapid development of online shopping, there are various promotion operations encouraging people to shop online. To stimulate users to buy more products all at once, many online stores offer coupons with a high threshold. When a customer finds that the price of the products she/he wants to order is below the threshold of a coupon, she/he might want to place the order with others to meet this threshold and enjoy more instant rebates. However, to conduct these orders and deliver products to receivers, users may need to pay extra delivery costs. When an order comprises several receivers' requests, the products in the order should first be delivered from stores to an assigned place. The products will then be packed into different packages and delivered to the different receivers, which may be costly than directly delivering products from stores to receivers. Under this situation, we propose a platform, the Crowdrebate platform, which collects requests from users, group requests into a set of orders to help users get more rebates, and relay products to different receivers in an order. The platform charges a proportion of benefits (defined as the rebate minus the extra cost) of orders as its revenue. Thus, the platform is motivated to solve a problem, namely the Crowdrebate problem, about finding the desired assignment of requests to maximize the benefits of orders. Unfortunately, we prove the problem is NP-hard. In this paper, we first illustrate the Crowdrebate platform and formulate the Crowdrebate problem. We propose two effective and efficient approximation approaches with theoretical guarantees, namely the Order-First approach and the Warehouse-First approach. Moreover, we evaluate the effectiveness and efficiency of our algorithms via comprehensive experiments on both real and synthetic data sets.**

I. Introduction

With the development of e-commerce, nowadays, it is common for e-commerce platforms to make promotion operations to attract customers and stimulate consumption, such as Black Friday [1] and Singles' Day Global Shopping Festival [2]. One of the most common promotion operations issued by e-commerce is to provide customers with coupons. If the price of an order is higher than a certain amount during the promotion time, the user who places this order can get back some instant rebate with the coupons [3] (e.g., credit, cash) [4], [5].

Many works related to the application of coupons have been proposed. Some papers illustrated the effect of coupons in the promotion operations [6][7], some papers discussed how to design rules of coupons to maximize profits of retailers[8][9], and others studied how to dispatch coupons more effectively [10] [11] [12] [13]. However, in these works, the researchers just consider the scenario that a user buys her/his products individually. In other words, a user would not group the purchasing requests with other users into an order. However, in practice, since a single user's spending cannot qualify to get the maximum instant rebate, users usually refer their friends to purchase products together in one order to get more instant rebates. Besides, after grouping, the products in one order may need to be delivered to multiple different receivers. Since stores in current e-commerce platforms deliver products of one order to only one receiver, after grouping, the products in orders should be delivered from stores to an assigned location and then relayed to receivers, respectively. This process would cause extra delivery costs compared with the original case (i.e., buying products individually and delivering products from stores directly to users). Moreover, in practice, to encourage customers to purchase more products, some stores would pay the delivery fee for costumers if the total price of an order is higher than a threshold, which further enhances users' motivation to group their purchase requests into orders. For this scenario, existing works cannot guide how to group purchasing requests into orders to maximize total benefit (i.e., the rebate minus the extra delivery cost).

Inspired by this phenomenon, in this paper, we propose a platform, namely the Crowdrebate platform, to collect users' purchasing requests and group their requests into orders to maximize total benefit. The benefit that a request can get is the benefit of the order multiplied by the proportion of the price of the request to the total price of the order. The platform would charge a proportion of benefits as its revenue. Thus, the platform also is motivated to maximize the total benefit. In the sequel, we will illustrate a motivation example.

Example 1. *In an e-commerce platform, there are two stores s_1 and s_2, and two coupons, c_1 and c_2. A user can use c_1 x times in one order to get $x \cdot 30$ rebates if the total price of products in s_1 and s_2 in the order is higher than $x \cdot 200$. Similarly, a user can use c_2 x times in one order to get $x \cdot 90$ rebates if the total price of the products in s_1 and s_2 in the order is higher than $x \cdot 400$. The number of times that each coupon can use in one order is unlimited, but these two types of coupons cannot be used in one order simultaneously. If the total price of products in s_2 in one order is higher than 800, s_2 will pay the cost of delivering products from s_2 to the assigned warehouse. However, s_1 would not pay the delivery fee for users whatever the total price of products in s_1 in one order is. There are five requests r_i ($1 \leq i \leq 5$) that want to*

Request	Price	Store	Receiver
r_1	90	s_1	Beijing
r_2	110	s_1	Beijing
r_3	200	s_2	Shenzhen
r_4	290	s_2	Beijing
r_5	340	s_2	Beijing

(a) The requests

Order	Price	Rebate	Warehouse	Extra cost
$O_1 = \{r_3\}$	200	30		0
$O_2 = \{r_1, r_2\}$	200	30	Beijing	10
$O_3 = \{r_4, r_5\}$	630	90	Beijing	0
Total Benefit			**30+90+30-10=140**	

(b) Solution 1

Order	Price	Rebate	Warehouse	Extra cost
$O_1 = \{r_1, r_2, r_3\}$	400	90	Beijing	40
$O_2 = \{r_4, r_5\}$	630	90	Beijing	0
Total Benefit			**90+90-40=140**	

(c) Solution 2

Order	Price	Rebate	Warehouse	Extra cost
$O_1 = \{r_1, r_2\}$	200	30	Beijing	10
$O_2 = \{r_3, r_4, r_5\}$	830	180	Beijing	-10
Total Benefit			**30+180-10-(-10)=210**	

(d) Optimal Solution

Fig. 1: Motivation Example.

buy products in two different stores and delivers products to five different receivers. Suppose the active periods of these requests and coupons are the same. The details of the five requests are shown in Figure 1(a). For example, r_1 wants to buy a product whose price is 90 from the store s_1, and the receiver of r_1 locates in Beijing. The location of s_1 is Beijing, and the location of s_2 is Shenzhen. Assume the Crowdrebate platform has a warehouse in Beijing (w_1) and a warehouse in Shenzhen (w_2). Assume the delivery cost is only related to the pickup location and the destination location. The cost of delivering an express from a city to the same city is 10, and the cost of delivering from one city to another city is 20.

The first solution is presented in Figure 1(b), which tries to minimize the part of the price does not enjoy the rebate. In other words, the first solution greedily groups some requests as an order where the difference between the total price of the order and the threshold of the assigned coupon is minimized. Specifically, we first consider r_3 as an order o_1, since its price is the same as the threshold of c_1 and send to the receiver of r_1 directly. Then, we group r_1 and r_2 as an order o_2 whose price is the same as the threshold of c_1. The original delivery fee of r_1 and r_2 is 10 and 10, respectively. We assign o_2 to w_1. Since products should first be delivered from s_1 to w_1 and then be delivered from w_1 to two receivers in Beijing, the delivery fee is 10+10+10. In other words, the extra delivery fee is 10. Finally, we group r_4 and r_5 as an order o_3 and use c_1 three times. We assign o_3 to w_1, and the extra delivery fee is 0. With this solution, the total rebate we can get is 150, and the extra delivery cost is 10. Thus, the total benefit is 140.

The second solution is shown in Figure 1(c), which greedily groups some requests into an order to use a coupon with a high rebate. Specifically, we group requests r_1, r_2 and r_3 as an order o_1, whose total price is enough to use c_2 once. The original delivery fee of r_1, r_2, and r_3 is 10, 10, and 10, respectively. We assign o_1 to w_1. Since products should first be delivered from s_1 and s_2 to w_1, and then be delivered from w_1 to two receivers in Beijing and a receiver in Shenzhen, the delivery fee is 70. Thus, the extra delivery fee is 40. By this solution, the total rebate we can get is 180, and the extra delivery cost is 40. Thus, the total benefit is 140. Note that although the rebate gotten by the second solution is higher than the one of the first solution, the delivery cost of the second solution is higher than the first one, and the final benefits of these two solutions are the same.

The optimal solution is demonstrated in Figure 1(d). Particularly, r_3, r_4, and r_5 are grouped into an order which is assigned to the warehouse in Beijing. The original delivery fee of r_3, r_4, and r_5 is 10, 20, and 20, respectively. Since the total price of o_2 is higher than 800, the delivery fee from s_2 to w_1 would be paid by s_2. Thus, after grouping, the delivery fee is only 40, which saves 10 compared with the original delivery fees. Thus, by this solution, the total rebate we can get is 210, and the extra diversity cost is 0. Thus, the total benefit is 210.

Inspired by this example, we formalize the Crowdrebate problem in this paper, which aims to group a set of requests into a set of orders to maximize the total benefit satisfying 1) the time constraint (i.e., the requests and coupons should be assigned in their active period); 2) the threshold constraint (i.e., the price of products should be higher than the threshold of coupons); 3) the variety of coupons (i.e., the number of types of coupons cannot exceed a required value); and 4) the quota constraint (i.e., the number of times of a coupon used in an order cannot exceed a required value). Unfortunately, we prove the Crowdrebate problem is NP-hard. Thus, it is intractable, and to solve the Crowdrebate problem, there are some challenges that need to solve: 1) **High scalability and throughput.** In the e-commerce scenario where the number of transactions per second is very high, the algorithm should run very quickly; and 2) **Dependent rebate and delivery cost.** The rebate a request can get and the extra delivery cost a request needs to pay is dependent on the other requests that are grouped in the same order.

In this paper, to solve these challenges, we proposed two efficient approximation algorithms, the Order-First Algorithm and the Warehouse-First Algorithm, whose time complexity both are $\mathcal{O}(n^2)$. The Order-First Algorithm first greedily groups requests into orders to get a high rebate. Then, for each order, the algorithm assigns it to a warehouse to minimize the delivery cost. The Warehouse-First Algorithm first assigns each request to the nearest warehouse. Then the algorithm greedily groups requests which are assigned to the same warehouse into orders to get a high rebate. For each algorithm, we make the theoretical analysis and give proof of its approximation ratio.

Specifically, we make the following contributions:

- We formulate a new Problem, called the Crowdrebate problem, in Section II, and prove its NP-hardness;
- We design an approximation algorithm, the Order-First algorithm, with theoretical guarantees in Section III;
- We propose an efficient algorithm, the Warehouse-First algorithm, with theoretical guarantees in Section IV;
- We conduct extensive experiments on real and synthetic data sets, to show the efficiency and effectiveness of our approaches in Section V.

In addition, we review and compare the related works in Section VI and conclude in Section VII.

II. Problem Definition

In this section, we formally define some necessary concepts and formulate the Crowdrebate problem. Then we give the proof of its NP-hardness.

A. Warehouses

We first define the concept of warehouses.

Definition 1 (A Warehouse)**.** Let w_i denote a warehouse whose location is denoted by lw_i.

The Crowdrebate platform collaborates with some delivery companies with some warehouses to collect products of orders from stores and relay products to receivers. For example, the Sifang company [14] is the official transportation company of Taobao platform [15]. Users can ask stores to delivery products to Sifang. After receiving products, it can help users allocate products into several packages and deliver them to different destinations. The delivery fee would be introduced in Subsection II-E.

B. Stores

There are lots of stores on an e-commerce platform, where users can choose the products they want to buy.

Definition 2 (A Store)**.** A store s_i is denoted by a quadruple $\langle\ ls_i,\ C_i,\ a_i,\ fe_i\ \rangle$, where ls_i is the location of the warehouse of this store, C_i is the set of coupons issued by this store, a_i is the maximum number of coupon types that can be used in an order, fe_i is a threshold of the free-shipping service. When the total consumption of products in s_i in an order exceeds this threshold, the express fee will be paid by s_i.

is the threshold of the total price of products of s_i in an order that the cost of express would be paid by the store.

According to the e-commerce platform rules [15], although there may be multiple warehouses of a store, each store will only present one warehouse on the e-commerce platform as the origin of delivering their products to the users. If the products in the warehouse are sold out, and the store has to deliver products from a warehouse with a higher delivery fee, the difference between the delivery fee would be paid by the store. Besides, to attract customers, a store may issue a set of coupons, which we will formally define in the Subsection II-C. Usually, the number of types of coupons issued by the same store that can be used in one order is limited. For instance, the cosmetics brand retailer Kiehl's announced its online store promotions on black Friday 2019 [16], which provided three types of coupons. These three types of coupons could not be used in one order at the same time. Moreover, to encourage users to purchase more products, some stores would pay the delivery fee if the sum of prices of products in an order is higher than a threshold. For example, a store s_1 claims the threshold fe_1 is 68. If the sum of the products' prices in one order is higher than 68, the delivery fee would be paid by the store. Reaching the threshold of the free-shipping to save the delivery fee is one of the motivations that users want to group their requests in one order. If a store does not provide free-shipping services, its fe_i can be considered as ∞.

C. Coupons

In promotion operations, e-commerce platforms and stores would provide coupons to encourage users to purchase more.

Definition 3 (A coupon)**.** A coupon c_i is represented by a six-tuple $\langle\ S_i,\ q_i,\ th_i,\ d_i,\ bc_i,\ ec_i\ \rangle$, where S_i is the set of stores where c_i can be used, q_i is the upper bound of the times that c_i can be used in an order, bc_i is the time stamp when c_i can start to be used and ec_i is the expired time stamp of c_i. If the total price of products from stores in S_i in an order is higher than $x \times th_i$, the order can get $x \times d_i$ rebate by using c_i x times, $\forall\ x \leq q_i$.

A coupon c_i can be used in an order x times if and only if the sum of prices of products in the order whose stores in S_i are higher than $x \cdot th_i$. For example, a platform issues a coupon c_1 "\$40 off every \$300, up to \$400", and the coupon can be used in the stores s_1, s_2, and s_3 from 00:00, 2020-11-01 to 23:59, 2020-11-11. Thus, as given in Definition 3, we will consider that th_1 is \$300, d_1 is \$40, $S_1 = \{s_1, s_2, s_3\}$, q_1 is 10, bc_1 is 00:00, 2020-11-01, and ec_1 is 23:59, 2020-11-11. For a coupon which can be used an unlimited number of times in one order, we consider its q_i as ∞. If a user purchases \$320 products from s_1 and \$280 products from s_2 in an order, he can use c_1 twice and get \$80 rebate. But if a user purchases \$320 products from s_1 and \$280 products from s_4 in an order, he can use c_1 only once and get \$40 rebate.

D. Requests

In this subsection, we define the concept of a request.

Definition 4 (A request)**.** A request r_i, denoted by a tuple $\langle\ br_i,\ u_i,\ lr_i,\ p_i,\ wt_i,\ rs_i,\ er_i,\ of_i\ \rangle$, is sent to the Crowdrebate platform by a user at the time stamp br_i, where 1) u_i is the receiver who the product of r_i should be delivered to; 2) lr_i is the location of u_i; 3) p_i is the price of the product; 4) wt_i is the weight of the product; 5) rs_i is the store where the user wants to buy the product; 6) er_i is the expired time after which the request r_i is invalid; and 7) of_i is the original delivery fee of directly delivering the product from the store to the receiver.

If a request r_i is not grouped with others, and a user directly buys the product from the store, r_i would cost of_i delivery fee. If, after we help the group with others, the delivery fee is lower than of_i (because of the free-shipping services of stores mentioned in Subsection 2), the difference can be considered as a benefit of the grouping. On the other hand, if after we help to group with others, the delivery fee is higher than of_i (although it can get some rebate), the difference can be considered as a cost of the grouping. We will formally define the delivery and discuss the delivery fee in more detail in the Subsection II-E. Besides, each request is indivisible.

E. Delivery Cost

Initially, when a request is not grouped with other requests in one order, the products are packaged in expresses that are

directly delivered from stores to receivers. After grouping, if the products of an order may need to be delivered to multiple receivers, the products are delivered to a warehouse of the Crowdrebate platform first. Then, the Crowdrebate platform packages products of the same receiver in one order into an express and delivers it from the warehouse to the receiver. In this subsection, we formally define a delivery.

Definition 5. A delivery e_i, denoted by a tuple $\langle \kappa_i, \tau_i, RE_i, tw_i, ce_i \rangle$, delivers products of requests in RE_i from κ_i to τ_i, where tw_i is the total weight of products of requests in RE_i (i.e., $tw_i = \sum_{r_j \in RE_i} wt_j$), and ce_i is the delivery fee of e_i.

When a store receives an order, it would ask a delivery to deliver products to a Crowdrebate platform warehouse, which is claimed in the order. For delivery from a store, κ_i is the store, and τ_i is the warehouse claimed in the order. Once a warehouse of the Crowdrebate platform receives a delivery from a store, it would allocate products in the delivery into several packages, and each package only contains products that would be delivered to the same receiver. For delivery from a warehouse, κ_i is the warehouse, and τ_i is the receiver. For instance, for the order o_1 in the Figure 1(c), there are two deliveries, e_1 and e_2, which deliver the products of o_1 from stores to the warehouse w_1. The delivery e_1 delivers the products of r_1 and r_2 from s_1 to w_1. The delivery e_2 delivers the product of r_3 from s_2 to w_1. After receiving e_1, the Crowdrebate platform would conduct two deliveries e_3 and e_4 that deliver the product of r_1 to the receiver of r_1 and delivers the product of r_2 to the receiver of r_2, respectively.

On the delivery process, the Crowdrebate platform needs to pay an express company some delivery fee, which is related to the location of the pick-up location (i.e., κ_i), the destination (i.e., τ_i), and the weight of the package (i.e., tw_i). Since the delivery in the e-commerce scenario would cross cities, the express fee is not simply calculated by the distance between the pick-up location and the destination. For example, according to the average number of deliveries, Cainiao Guoguo, the official express company of the Taobao [15], divides cities in Mainland China into four categories. When a product of the same weight is delivered from the same place to different cities, it is cheaper to deliver them to cities with a high daily volume of expresses than cities with a low daily volume of expresses. Besides, the delivery fee is stepwise proportional to the weight of the package. For example, Cainiao GuoGuo charges a base fee for a package whose weight is less than 1kg and charges an extra fee for each kilogram added to the package's weight. Without loss of generality, we define the delivery fee measure function as following.

Definition 6 (Delivery Fee Measure Function)**.** A delivery fee measure function $F(\cdot)$ is a mapping from $(W \cup S) \times (W \cup U) \times \mathbb{R}$ to $\mathbb{R}$, where $\mathbb{R}$ is the set of real numbers, W is the set of warehouses of the Crowdrebate platform, S is the set of stores, U is the set of receivers.

Thus, given a delivery fee measure function $F(\cdot)$, the fee of e_i is $ce_i = F(\kappa_i, \tau_i, tw_i)$. For simplicity, when the fee of a delivery is paid by a store, we consider ce_i as 0. For example, in the Figure 1(d), suppose e_3 is the delivery that delivers products of o_2 from s_2 to w_1. Since the total price of e_3 is higher than 800, the delivery fee is paid by s_2, thus $ce_3 = 0$.

F. Orders

In this subsection, we formulate the concept of an order.

Definition 7 (An order)**.** An order o_i, denoted by a six-tuple $\langle R_i, SR_i, OC_i, wo_i, SE_i, RE_i \rangle$, contains a set of requests R_i, where SR_i is the set of stores of requests in R_i (i.e., $SR_i = \{sr_j | r_j \in R_i\}$), OC_i is the set of coupons used in o_i, wo_i is a warehouse where the products of requests in R_i are delivered from stores to, SE_i is a set of deliveries which deliver the products of requests in R_i from stores to wo_i, and RE_i is a set of deliveries which deliver the products of requests in R_i from wo_i to the receivers of requests.

For simplicity, we represent using a coupon c x times in an order o_i by adding x coupon c in the OC_i. For instance, for the order o_2 in the Figure 1(d), $R_2 = \{r_3, r_4, r_5\}$, $SR_2 = \{s_2\}$, $OC_2 = \{c_2, c_2\}$, $wo_2 = w_1$. Suppose e_1 is a delivery which delivers the products of r_3, r_4, and r_5 from s_2 to w_1. Suppose $e_2/e_3/e_4$ is a delivery which delivers the products of $r_3/r_4/r_5$ from w_1 to the corresponding receiver, respectively. Thus, $SE_2 = \{e_1\}$ and $RE_2 = \{e_2, e_3, e_4\}$.

G. Crowdrebate Problem

Based on the concepts mentioned above, we formulate the Crowdrebate problem.

Definition 8. Given a set of warehouses $W = \{w_1, \cdots, w_t\}$, a set of requests $R = \{r_1, \cdots, r_n\}$ and a set of coupons $C = \{c_1, \cdots, c_m\}$, the Crowdrebate problem aims to group requests in R into a set of orders O with the maximum utility $U = \sum_{o_i \in O} (\sum_{c_j \in OC_i} d_j + \sum_{r_j \in R_i} of_j - \sum_{e_j \in SE_i \cup RE_i} ce_j)$, satisfying following constraints:

- **Time constraint.** Each request $r_i \in R$ should be assigned during its active period $[rs_i, er_i]$, and each coupon $c_j \in C$ should be assigned during its active period $[bc_j, ec_j]$;
- **Threshold constraint.** To use a coupon c_j x times in an order o_i, the total price of product from some stores in S_j in o_i should be higher than the threshold $th_j \times x$;
- **Variety constraint.** For each order $o_i \in O$ and for each store $s_j \in \bigcup_{c_k \in OC_i} S_k$, the number of types of coupons issued by s_j in OC_i cannot exceed a_j; and
- **Quota constraint.** For each order $o_i \in O$ and for each coupon $c_j \in C$, the number of the coupon c_j in OC_i cannot exceed q_j.

For example, there are four requests and three coupons whose details are shown in the Table I. The store s_1 issues the coupon c_1, and the store s_2 issues the coupons c_2 and c_3. The number of types of coupons that are issued by s_2 and can be used in an order cannot exceed 1, i.e., $a_2 = 1$. In other words, c_2 and c_3 cannot be used in an order simultaneously.

TABLE I: Requests and Coupons.

Request	p_i	rs_i	br_i	er_i		
r_1	50	s_1	2:00	4:00		
r_2	150	s_2	6:00	8:00		
r_3	60	s_2	1:00	3:00		
r_4	200	s_2	5:00	7:00		
Coupon	S_i	q_i	th_i	r_i	bc_i	ec_i
c_1	$\{s_1\}$	1	120	30	2:00	3:00
c_2	$\{s_2\}$	5	50	5	1:00	7:00
c_3	$\{s_1, s_2\}$	1	100	20	2:00	5:00

We cannot group r_1 and r_2 in an order, since their active periods do not match, violating the **time constraint**. We group r_1 and r_3 into an order o_1. We group r_2 and r_4 into an order o_2. For o_1, we cannot assign c_1 to it, since the total price of products from s_1 is less than 120, not satisfying the **threshold constraint**. However, we can assign o_1 with a coupon c_3, since the total price of products from S_3 is higher than 100. Since we have assigned o_1 with a coupon c_3, we cannot assign o_1 with another coupon c_2. Otherwise, the number of types of coupons issued by s_2 in o_1 is larger than a_2, violating the **variety constraint**. For o_2, we cannot assign it with c_3, since their active periods do not match, violating the **time constraint**. For o_2, according to the **quota constraint**, we can use c_2 just 5 times, since $350 > 5 \times 50$ and $q_2 = 5$.

H. NP-hardness

In this subsection, we give the proof of the NP-hardness of the Crowdrebate problem.

Theorem II.1. *The Crowdrebate problem is NP-hard.*

Proof: Due to the space limitation, we do not show the details here. For the full proof, please refer to our technical report [17]. ■

TABLE II: Symbols and Descriptions.

Symbol	Description
W	The set of warehouses of the platform $\{w_1, \cdots, w_t\}$
lw_i	The location of a warehouse w_i
ls_i	The location of a store s_i
C_i	The set of coupons issued by a store s_i
a_i	The maximum of types of coupons from s_i can be used in an order
fe_i	The threshold of sum of consumption of s_i in an order for free delivery
R	The set of requests $\{r_1, \cdots, r_n\}$
u_i	The user who the product of r_i should be delivered to
p_i	The price of the product of r_i
wt_i	The weight of the product of r_i
rs_i	The store where product of r_i should be bought
(br_i,er_i)	The time slot when r_i should be ordered
lr_i	The location where the product of r_i should be delivered
κ_i	The pickup location of a delivery e_i
τ_i	The destination of a delivery e_i
tw_i	The total weight of products in e_i
ce_i	The cost of e_i
C	The set of coupons $\{c_1, \cdots, c_m\}$
th_i	The threshold of total price of an order to use c_i
S_i	The set of stores where the coupon c_i can be used
q_i	The upper bound of the times that c_i can be used in an order
d_i	The value of price can be deducted by using c_i each time
(bc_i,ec_i)	The time slot when c_i can be used
O	The set of orders $\{o_1, \cdots, o_k\}$
R_i	The set of requests in o_i
SR_i	The set of stores of requests in R_i
OC_i	The set of coupons used in o_i
wo_i	The warehouse where stores delivered products in R_i to
SE_i	The set of deliveries that deliver products in R_i from stores to wo_i
RE_i	The set of deliveries that deliver products in R_i from wo_i to receivers

III. Order-First Approach

In this section, we propose a novel approximation algorithm, the Order-First algorithm, to solve the Crowdrebate problem. Generally, the algorithm first greedily merges requests/orders to maximize the total rebate without considering the cost of delivery. After that, for each order, the algorithm assigns a warehouse to it to minimize the cost of delivery.

A. The Order-First Algorithm

Algorithm 1 illustrates the pseudo-code of our **Order-First Algorithm**. We first initialize a set of $|R|$ orders where each order refers a request in R (line 1). Then, we greedily merge orders to maximize the total rebate without considering the cost of delivery (line 2-14). For each coupon c_i in C, the algorithm greedily merges orders in O as new orders to use c_i to get more rebates until the algorithm cannot get a new order (line 3-14). We first initialize an empty order $o^{\#}$. Then, we greedily add orders to $o^{\#}$ (line 5-12). We enumerate each order o_j in O, and verify if o_j can be added to $o^{\#}$ to use c_i (line 6-7). Specifically, if the active period of o_j does not have overlap with the active period of c_i, because of the time constraint, o_j cannot be added to $o^{\#}$ to use c_i. In other words, o_j can not be added to $o^{\#}$ to use c_i if $bo_j > ec_i$ or $eo_j < bc_i$, where bo_j is the latest release time of requests in o_j (i.e., $bo_j = \max\{br_k | r_k \in R_j\}$), and eo_j is the earliest expired time of requests in o_j (i.e., $eo_j = \max\{er_k | r_k \in R_j\}$). Besides, if the active period of o_j does not have overlap with the active period of $o^{\#}$, because of the time constraint, o_j cannot be added to $o^{\#}$ either. In other words, o_j can not be added to $o^{\#}$ if $bo_j > eo^{\#}$ or $eo_j < bo^{\#}$. If an order o_j is eligible, the algorithm calculates the increment of rebate $\theta_{i,j}$ by the Dif(o_j, $o^{\#}$, c_i) procedure (line 8), which we will introduce shortly. Let o_{j^*} denote the order whose $\theta_{i,j}$ is the largest and let θ^* denote θ_{i,j^*} (line 9). If $\theta_{i,j^*} \geq 0$, it means that after adding o_j into $o^{\#}$, users can get more rebates. After adding o_j into $o^{\#}$, we remove o_j from O and update O (line 11). If θ_{i,j^*} is negative, it means that we cannot merge any more order o_j into $o^{\#}$ to get more rebates (line 12). After getting an $o^{\#}$, we add it to O (line 13). If we cannot get a nonempty $o^{\#}$, it means that we cannot get any more order to use c_i (line 14). Then, we turn to enumerate another coupon.

After enumerating all coupons, we would get a set of orders. Next, for each order, the algorithm greedily assigns a warehouse with the lowest extra delivery cost to it (line 15-21). Particularly, since there is only one receiver in some orders, products can be directly delivered to the receiver. For instance, in example 1, in the first solution, there is only one receiver in o_1. These orders do not need to be transported by warehouses (line 16). Otherwise, we enumerate all warehouses and calculate $\delta_{i,j}$ (line 17-18), which is defined as

$$\delta_{i,j} = \sum_{e_k \in SE_i \cup RE_i} ce_k - \sum_{r_k \in R_i} of_k. \quad (1)$$

For each order o_i where there are more than one receiver, we assign it with the warehouse w_{j^*} with the smallest $\delta_{i,j}$ and calculate the benefit of it (i.e., the rebate minus the extra delivery cost) (line 19). If the benefit of o_i is not positive, o_i is removed from O (line 20-21).

Algorithm 1: Order-First Approach

Input: A set of warehouses W, a set of requests R and a set of coupons C
Output: A set of orders O

1 Initialize O;
2 **foreach** *coupon* c_i *in* C **do**
3 **repeat**
4 initialize $o^{\#}$;
5 **repeat**
6 **foreach** o_j *in* O **do**
7 **if** o_j *is eligible* **then**
8 calculate $\theta_{i,j}$ = Dif(o_j,$o^{\#}$, c_i);
9 get o_{j^*} whose $\theta_{i,j}$ is the largest, $\theta^* = \theta_{i,j^*}$;
10 **if** $\theta^* \geq 0$ **then**
11 assign o_{j^*} to $o^{\#}$, update O;
12 **until** $\theta^* < 0$
13 add $o^{\#}$ to O;
14 **until** $o^{\#}$ *is empty*
15 **foreach** o_i *in* O **do**
16 **if** *there are more than one receivers in* o_i **then**
17 **foreach** *warehouse* w_j *in* W **do**
18 calculate $\delta_{i,j}$;
19 set wo_i as w_{j^*} with the smallest $\delta_{i,j}$, get the benefit of o_i;
20 **if** *the benefit of* o_i *is not positive* **then**
21 remove o_i;
22 **return** *the order set O;*

Algorithm 2: Dif() Procedure

Input: An order o_α, an order o_β, and a coupon c_ζ
Output: the increment of rebate

1 get s_γ, $vo_{\alpha,\gamma}$, $vo_{\beta,\gamma}$, add c_ζ to o_α and o_β;
2 **if** $vo_{\alpha,\gamma} > a_\gamma$ **then**
3 **for** *each* c_i *in* $CO_\alpha \cap C_\gamma$ **do**
4 calculate ro_i;
5 remove c_{i^*} from CO_α whose ro_i is the smallest;
6 **if** $vo_{\beta,\gamma} > a_\gamma$ **then**
7 **for** *each* c_i *in* $CO_\beta \cap C_\gamma$ **do**
8 calculate ro_i;
9 remove c_{i^*} from CO_β whose ro_i is the smallest;
10 get rb_α and rb_β, get $o_\omega = o_\alpha + o_\beta$, get SO_ω;
11 **foreach** $s_i \in SO_\omega$ **do**
12 calculate $vo_{\omega,i}$;
13 **if** $vo_{\omega,i} > a_i$ **then**
14 **for** *each* c_j *in* $CO_\omega \cap C_\gamma$ **do**
15 calculate ro_j;
16 **while** $vo_{\omega,i} > a_i$ **do**
17 remove c_{j^*} from CO_ω whose ro_j is the smallest;
18 get rb_ω;
19 **return** $rb_\omega - rb_\alpha - rb_\beta$;

Algorithm 2 illustrates the pseudo-code of our **Dif() Procedure**, which is used to calculate the increment of rebate after merging two orders to use a coupon c. Suppose s_γ is the store which issues the coupon c_ζ. Initially, we calculate $vo_{\alpha,\gamma}$ and $vo_{\beta,\gamma}$, where $vo_{\alpha/\beta,\gamma}$ is the number of types of coupons that are issued by s_γ and have been added to $o_{\alpha/\beta}$ (line 1). Besides, the procedure adds c_ζ to o_α $\min\{q_\zeta, \lfloor \frac{\sum_{r_i \in R_\alpha \wedge rs_i \in S_\zeta} p_i}{th_\zeta} \rfloor\}$ times. Similarly, the procedure adds c_ζ to o_β $\min\{q_\zeta, \lfloor \frac{\sum_{r_i \in R_\beta \wedge rs_i \in S_\zeta} p_i}{th_\zeta} \rfloor\}$ times. However, after adding c_ζ to o_α and o_β, these two orders may violate the variety constraint. In other words, the number of types of coupons which are issued by s_γ and have been added to $o_{\alpha/\beta}$ may exceed a_γ (line 2,6). Then, the procedure greedily removes a type of coupon which is issued by s_γ and contribute less rebates (line 3-5, 7-9). Specifically, the procedure enumerates all coupons which are issued by s_γ and have been added to o_α (line 3). For each coupon c_i, the procedure calculates ro_i (line 4), which is defined as

$$ro_i = \min\{q_i, \lfloor \frac{\sum_{r_j \in R_\alpha \wedge rs_j \in S_i} p_j}{th_i} \rfloor\} \cdot d_i. \quad (2)$$

After that, the procedure removes the coupon c_{i^*} from CO_α whose ro_i is the smallest. Similarly, the procedure processes o_β (line 6-9). Then, the procedure calculates the rebate of updated o_α and o_β, denoted by rb_α and rb_β (line 10). Specifically, the rebate of an order o_i is calculated as $rb_i = \sum_{c_j \in CO_i} d_j$.

Besides, the procedure merges o_α and o_β as a new order o_ω (line 10). Denote the set of stores of coupons that are assigned to o_ω as SO_ω, i.e., $SO_\omega = \{s_k | c_j \in s_k \wedge c_k \in CO_\omega\} = \{s_k | c_j \in s_k \wedge c_k \in CO_\alpha\} \cup \{s_k | c_j \in s_k \wedge c_k \in CO_\beta\}$. After merging, o_ω may violate the variety constraint. Thus, the procedure enumerates each store $s_i \in SO_\omega$ and greedily removes coupons to satisfy the variety constraint (line 11-17). For each store s_i, the procedure first calculates the number, $vo_{\omega,i}$, of types of coupons that are issued by s_i and have been added to o_ω, i.e., $vo_{\omega,i} = |\{c_k | c_k \in CO_\omega, c_k \in C_i\}|$. If $vo_{\omega,i}$ is larger than a_i, for each coupon c_i, by the Equation 2, the procedure calculates ro_j as $\min\{q_j, \lfloor \frac{\sum_{r_k \in R_\omega \wedge rs_k \in S_i} p_k}{th_j} \rfloor\} \cdot d_j$ (line 14-15). The procedure greedily removes c_{j^*} with the smallest ro_j from CO_ω until $vo_{\omega,i}$ is not larger than a_i (line 16-17). After that, the procedure calculates the rebate of o_ω (line 18). Finally, the procedure returns the increment of rebates as $rb_\omega - rb_\alpha - rb_\beta$ (line 19).

Example 2. *For instance, there are five requests and three coupons whose details are shown in the Table III. The store* s_1 *issues the coupon* c_1*, and the store* s_2 *issues the coupons* c_2 *and* c_3*. The number of types of coupons that are issued by* s_2 *and can be used in an order cannot exceed 1, i.e.,* $a_2 = 1$*. In other words,* c_2 *and* c_3 *cannot be used in an order simultaneously. The threshold of the free-shipping service of* s_1 *is 70, and the threshold of the free-shipping service of* s_2 *is 370. The region where* s_1 *locates is 1, and the region where* s_2 *locates is 2. The cost of delivering products from a region to another region is 10, and the cost of delivering products from a region to the same region is 5. There are two warehouses,*

TABLE III: Requests and Coupons in Example 2.

Request	p_i	rs_i	br_i	er_i	receiver	region
r_1	80	s_1	1:00	3:00	u_2	1
r_2	140	s_2	2:00	8:00	u_3	2
r_3	80	s_2	2:00	5:00	u_1	1
r_4	110	s_2	4:00	7:00	u_1	1
r_5	110	s_2	1:00	8:00	u_1	1
Coupon	S_i	q_i	th_i	r_i	bc_i	ec_i
c_1	$\{s_1\}$	1	200	60	2:00	3:00
c_2	$\{s_1, s_2\}$	1	90	10	2:00	5:00
c_3	$\{s_2\}$	3	120	30	1:00	7:00

where w_1 locates at region 1 and w_2 locates at region 2.

Suppose $o_{i,j}$ denote the j^{th} order in O when the Algorithm 1 conduct the i^{th} iteration of the first for-loop in line 2. Figure 2 shows the changes of some states during the Algorithm 1. After merging orders (i.e., finishing the first for-loop in line 14), the Algorithm 1 assigns a warehouse for each order. At that moment, there are two orders, $o_1 = \{r_2, r_4, r_5\}$ and $o_2 = \{r_1, r_3\}$. Thus, $\delta_{1,1}$ = 5, $\delta_{1,2}$ = 5, $\delta_{2,1}$ = 10, and $\delta_{2,2}$ = 15. Thus, the algorithm assigns o_1 with w_2 and assigns o_2 with w_1. Thus, the total benefit is 120-5-10 = 105.

B. Theoretical Analysis

In this subsection, we conduct some theoretical analysis of our Order-First Approach. We first give the poof of the time complexity of the Dif() Procedure in the Lemma III.1.

Lemma III.1. *The time complexity of the Dif() Procedure is $\mathcal{O}(|CO_\alpha| \cdot |RO_\alpha| + |CO_\beta| \cdot |RO_\beta|)$, where $|CO_{\alpha/\beta}|$ is the number of types of coupons that are added to $o_{\alpha/\beta}$ and $|RO_{\alpha/\beta}|$ is the number of requests that are added to $o_{\alpha/\beta}$.*

Proof: Due to the space limitation, we do not show the details here. For the full proof, please refer to our technical report [17]. ■

Based on the Lemma III.1, we give the proof of the time complexity of the Order-First Algorithm in the Theorem III.1.

Theorem III.1. *The time complexity of the Order-First Algorithm is $\mathcal{O}(n^2)$, where n is the number of requests.*

Proof: Due to the space limitation, we do not show the details here. For the full proof, please refer to our technical report [17]. ■

Next, we give the proof of the approximation ratio of the Order-First Algorithm in the Theorem III.2. Suppose $RT_{i,j}$ is a set of requests that are active at the time stamp j and are in the scope of the coupon c_i, i.e., $RT_{i,j} = \{r_k | br_k \leq j, er_k \geq j, rs_k \in S_i\}$. Suppose $\lambda_{i,j}$ is the total prices of requests that are active at the time stamp j and are in the scope of the coupon c_i, i.e., $\lambda_{i,j} = \sum_{r_k \in RT_{i,j}} p_k$.

Theorem III.2. *The approximation ratio of the Order-First Algorithm is $\sum_{c_i \in C} \frac{\min\{\lfloor \lambda_{min}(i) \rfloor, q_i \cdot th_i\} - \mu_i \cdot c_{max}}{n \cdot \lambda_{max}(i) + \mu_i \cdot of_{max}}$, where n is the number of requests in R, $\mu_i = \frac{n \cdot th_i}{d_i \cdot |C|}$, c_{max} is the highest delivery cost, of_{max} is the highest original delivery fee, $\lambda_{min}(i) = \min\{\lambda_{i,j} | \forall j \in [bc_i, ec_i]\}$, and $\lambda_{max}(i) = \max\{\lambda_{i,j} | \forall j \in [bc_i, ec_i]\}$.*

Proof: Due to the space limitation, we do not show the details here. For the full proof, please refer to our technical report [17]. ■

coupon	The order set at line 6	The $o^\#$ at line 13
c_1	$o_{1,1} = \{r_1\}, o_{1,2} = \{r_2\}, o_{1,3} = \{r_3\}, o_{1,4} = \{r_4\}, o_{1,5} = \{r_5\}$	$o^\# = o_{1,1}$
c_2	$o_{2,1} = \{r_2\}, o_{2,2} = \{r_3\}, o_{2,3} = \{r_4\}, o_{2,4} = \{r_5\}, o_{2,5} = \{r_1\}$	$o^\# = o_{2,1} = \{r_2\}$
	$o_{2,1} = \{r_3\}, o_{2,2} = \{r_4\}, o_{2,3} = \{r_5\}, o_{2,4} = \{r_1\}$	$o^\# = o_{2,1} + o_{2,4} = \{r_3, r_1\}$
	$o_{2,1} = \{r_4\}, o_{2,2} = \{r_5\}$	$o^\# = o_{2,1} = \{r_4\}$
	$o_{2,1} = \{r_5\}$	$o^\# = o_{2,1} = \{r_5\}$
c_3	$o_{3,1} = \{r_2\}, o_{3,2} = \{r_3, r_1\}, o_{3,3} = \{r_4\}, o_{3,4} = \{r_5\}$	$o^\# = o_{3,1} + o_{3,3} + o_{3,4} = \{r_2, r_4, r_5\}$
	$o_{3,1} = \{r_3, r_1\}$	$o^\# = o_{3,1} = \{r_3, r_1\}$

Fig. 2: The Changes of some states during the Algorithm 1

In other words, the difference between active periods of requests is larger, $\lambda_{min}/\lambda_{max}$ is smaller and it is harder to group requests into orders satisfying the time constraint. With the increment of q_i, it is easier to group requests into orders to get more rebates and the approximation ratio is higher. With the decrement of extra delivery cost, c_{max} is lower and the gap between the result of the Warehouse-First Algorithm and the optimal result is smaller.

IV. Warehouse-First Approach

Although the result of the Order-First Approach is guaranteed, it has two shortcomings. The first is that, when the Order-First Approach groups requests into orders, it does not consider the extra delivery cost. When the extra delivery cost, compared with the rebate, is much lower, it can get a good result. As shown in the Theorem III.2, when the extra delivery cost is higher, the result would be bad. The second is that, for each round of merging in the line 11 of the Algorithm 1, the Order-First Approach has to calculate $\theta_{i,j}$ for each order. In the e-commerce scenarios, the number of requests is extremely high at peak times [2], which would cause the running time of the Order-First Approach is high.

In this section, we propose an efficient approach, the Warehouse-First Approach. The approach first assigns requests to a warehouse that costs the lowest extra delivery cost, and then merges requests/orders that are assigned to the same warehouse into orders to get more rebates. Since requests are assigned to warehouses that cost the lowest extra delivery cost, the extra delivery cost would be low. Besides, since the algorithm merges requests/orders that are assigned to the same warehouse, the running time can be lower than the Order-First Approach.

A. The Warehouse-First Algorithm

Algorithm 3 illustrates the pseudo-code of our **Warehouse-First Algorithm**. We first assign each request to the warehouse that the extra delivery cost is the lowest (line 1-4). Specifically, for each request r_i and each warehouse w_j, the algorithm calculates $\delta_{i,j}$ (line 2-3) according to the Equation 1. Let WO_k denote the set of orders whose assigned warehouse is w_k. Initially, each request r_i is considered as an order and is added to WO_j where $\delta_{i,j}$ is the lowest (line 4).

Then, we greedily merge orders in each WO_k to maximize the total rebate without considering the cost of delivery (line

Algorithm 3: Warehouse-First Approach

Input: A set of warehouses W, a set of requests R and a set of coupons C
Output: A set of orders O

1 **foreach** *request* r_i *in* R **do**
2 **foreach** *warehouse* w_j *in* W **do**
3 calculate $\delta_{i,j}$;
4 add r_i to WO_j where $\delta_{i,j}$ is the lowest;
5 **foreach** *warehouse* w_k **do**
6 **foreach** *coupon* c_i *in* C **do**
7 **repeat**
8 initialize $o^{\#}$;
9 **repeat**
10 **foreach** o_j *in* WO_k **do**
11 **if** o_j *is eligible* **then**
12 calculate $\theta_{i,j}$ = Dif(o_j,$o^{\#}$, c_i);
13 get o_{j^*} whose $\theta_{i,j}$ is the largest, θ^* = θ_{i,j^*};
14 **if** $\theta^* \geq 0$ **then**
15 assign o_{j^*} to $o^{\#}$, update WO_k;
16 **until** $\theta^* < 0$
17 add $o^{\#}$ to WO_k;
18 **until** $o^{\#}$ *is empty*
19 **foreach** *warehouse* w_k *in* W **do**
20 **foreach** o_i *in* WO_k **do**
21 **if** *there is only one receiver* **then**
22 update the delivery cost.
23 **if** *the benefit of* o_i *is positive* **then**
24 $O \leftarrow O + o_i$;
25 **return** *the order set O;*

5-18). For a warehouse w_k, the algorithm enumerates each coupon c_i in C (line 6). For each coupon c_i, the algorithm greedily merges orders in WO_k as new orders to use c_i to get more rebate until the algorithm cannot get a new order (line 7-18). We first initialize an empty order $o^{\#}$. Then, we greedily add orders to $o^{\#}$ (line 9-16). We enumerate each order o_j in WO_k, and verify if o_j can be added to $o^{\#}$ to use c_i (line 10-11). Specifically, if the active period of o_j does not have overlap with the active period of c_i, because of the time constraint, o_j cannot be added to $o^{\#}$ to use c_i. In other words, o_j can not be added to $o^{\#}$ to use c_i if $bo_j > ec_i$ or $eo_j < bc_i$, where bo_j is the latest release time of requests in o_j (i.e., $bo_j = \max\{br_k | r_k \in R_j\}$), and eo_j is the earliest expired time of requests in o_j (i.e., $eo_j = \max\{er_k | r_k \in R_j\}$). Besides, if the active period of o_j does not have overlap with the active period of $o^{\#}$, because of the time constraint, o_j cannot be added to $o^{\#}$ either. In other words, o_j can not be added to $o^{\#}$ if $bo_j > eo^{\#}$ or $eo_j < bo^{\#}$. If an order o_j is eligible, the algorithm calculates the increment of rebate $\theta_{i,j}$ by the Dif(o_j, $o^{\#}$, c_i) procedure (line 12), which is shown in Algorithm 2. Let o_{j^*} denote the order whose $\theta_{i,j}$ is the largest and let θ^* denote θ_{i,j^*} (line 13). If $\theta_{i,j^*} \geq 0$, it means that after adding o_j into $o^{\#}$, users can get more rebates. After adding o_j into $o^{\#}$, we remove o_j from WO_k and update WO_k (line 15). If θ_{i,j^*} is negative, it means that we cannot merge any more order o_j into $o^{\#}$ to get more rebates (line 16). After getting an $o^{\#}$, we add it to WO_k (line 17). If we cannot get a nonempty $o^{\#}$, it means that we cannot get any more order to use c_i (line 18). Then, we turn to enumerate another coupon.

After that, we retrieve the set of orders whose benefit is positive (line 19-24). Particularly, since there is only one receiver in some orders, products can be directly delivered to the receiver. For instance, in example 1, in the first solution, there is only one receiver in o_1. These orders do not need to be transported by warehouses. Thus, we update the delivery cost of these orders (line 20-21). If the benefit of an order o_i is positive, we add it to O (line 23-24). Finally, we return the order set O as the result (line 25).

Example 3. *Back to our running example in Example 2, after conducting the first for-loop in line 1 of Algorithm 3, we get* $WO_1 = \{r_1, r_3, r_4, r_5\}$ *and* $WO_2 = \{r_2\}$. *When the algorithm terminates, we get two orders,* $o_1 = \{r_1, r_3, r_4, r_5\}$ *and* $o_2 = \{r_2\}$. *Particularly, since there are only two receivers in* o_1, *the extra delivery cost is 10+5+ 5 - 3 · 10 = -10. In other words, compared with originally users conduct requests individually,* o_1 *can save 10 in delivering. Since there is only a request in* o_2, *the products can be directly delivered to its receiver. Thus, the extra delivery cost of* o_2 *is 0. Thus, the total benefit is 130.*

B. Theoretical Analysis

In this subsection, we conduct some theoretical analysis of our Warehouse-First Approach. Based on the Lemma III.1, we give the proof of the time complexity of the Warehouse-First Algorithm in the Theorem IV.1.

Theorem IV.1. *The time complexity of the Warehouse-First Algorithm is* $\mathcal{O}(\sum_{w_i \in W} n_i^2)$, *where* n_i *is the number of requests that are assigned to the warehouse* w_i.

Proof: Due to the space limitation, we do not show the details here. For the full proof, please refer to our technical report [17]. ■

In the worst case when requests are assigned to the same warehouse, the time complexity of the Warehouse-First Algorithm is the same as the time complexity of the Order-First Algorithm. However, since $\forall k, \sum_{i=1}^{k} n_i^2 \leq (\sum_{i=1}^{k} n)^2$, in most cases, the time complexity of the Warehouse-First Algorithm is much lower than the one of the Order-First Algorithm.

Next, we give the proof of the approximation ratio of the Warehouse-First Algorithm in the Theorem III.2. Suppose RW_k is the set of requests of which w_k is the warehouse that can save its most extra delivery cost. Suppose $cw(i)$ is the warehouse that r_i is assigned to, and $c(i,k)$ is the extra delivery cost when assigning r_i with w_k. Suppose $RT_{i,j,k}$ is a set of requests that are assigned to the warehouse w_k, are active at the time stamp j and are in the scope of the coupon c_i, i.e., $RT_{i,j,k} = \{r_g | br_g \leq j, er_g \geq j, rs_g \in S_i, r_g \in RW_k\}$. Suppose $\lambda_{i,j,k}$ is the total prices of requests that are assigned

TABLE IV: Experimental Settings(Real).

Factors	Settings
the number, num_U, of users	6000, 8000, **10000**, 12000, 14000
the range of wait time, $[wr^-, wr^+]$	[1, 60], [1,180], **[1, 300]**, [1, 420], [1, 540]
the number of batches num_b	240, 180, **144**, 120, 102

to the warehouse w_k, are active at the time stamp j and are in the scope of the coupon c_i, i.e., $\lambda_{i,j,k} = \sum_{r_g \in RT_{i,j,k}} p_g$.

Theorem IV.2. *The approximation ratio of the Warehouse-First Algorithm is* $\sum_{w_k \in W} \sum_{c_i \in C} \frac{\min\{\lfloor \lambda_{min}(i,k) \rfloor, q_i \cdot th_i\} - \epsilon_{i,k} \cdot c^*_{min}}{n \cdot \lambda_{max}(i,k) + \epsilon_{i,k} \cdot of_{max}}$, *where n is the number of requests in R,* $\epsilon_{i,k} = \frac{n \cdot th_i}{d_i \cdot |C| \cdot |W|}$, c^*_{min} *is the maximum of minimum extra delivery cost of each request (i.e.,* $c^*_{min} = \max\{cw(g) | r_g \in R\}$*), of_{max} is the highest original delivery fee,* $\lambda_{min}(i,k) = \min\{\lambda_{i,j,k} | \forall j \in [bc_i, ec_i]\}$, *and* $\lambda_{max}(i,k) = \max\{\lambda_{i,j,k} | \forall j \in [bc_i, ec_i]\}$.

Proof: Due to the space limitation, we do not show the details here. For the full proof, please refer to our technical report [17]. ∎

In other words, the difference between active periods of requests is larger, $\lambda_{min}/\lambda_{max}$ is smaller and it is harder to group requests into orders satisfying the time constraint. With the increment of q_i, it is easier to group requests into orders to get more rebates and the approximation ratio is higher. With the decrement of extra delivery cost, c^*_{min} is lower and the gap between the result of the Warehouse-First Algorithm and the optimal result is smaller.

V. Experimental Study

A. Data Set

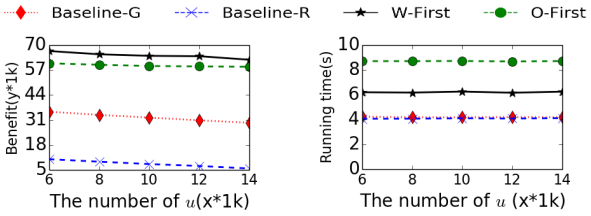
Real data. We collected the data set of requests from a mall in Wuhan. Except for the way of vising the mall physically, the mall also supports the way of making orders online. We extracted 117,509 requests from August 28 to 30, 2020. Each tuple of the request data has three attributes, price, weight, and the store. The total price of these requests was 24,82787.5 RMB, and the number of stores that sold these products was 114. During that period, the stores did not make promotion operations and did not issue any coupons. We crawled the coupon data set from the Taobao platform [15], which is the biggest e-commerce platform in China. For each store in the request data set, we find a store in the Taobao platform that sells the same products. In this way, we retrieve a store data set, which contains 114 stores. Each tuple of the store data set has three attributes: the store's location, the threshold for the free shipping service, and the upper bound of the variety (i.e., the maximum number of coupons types that are issued by this store and are used in the order). Besides, we crawled the coupons issued by stores in the store data set or the Taobao platform. In this way, we retrieve a coupon data set containing 286 coupons issued by stores and eight coupons issued by the Taobao platform. Each tuple of the coupon data set has six attributes, the threshold, the rebate, the quota, the scope of stores that the coupon can be used, the release time, and the expired time. Moreover, we retrieve the delivery cost measurement function from ZTO company [18], which is one of the top delivery companies in China. By this function, the cities in Mainland Chain is categorized into 31 regions. Besides, we crawled the warehouse data set from the Sifang company [14], which is the Taobao's official warehouse-delivery company. Users can ask stores to deliver products to Sifang company's warehouses and ask Sifang company to help allocate and repackage products into several expresses. In this way, we retrieve the warehouse data set, which contains eight warehouses in 8 different regions.

TABLE V: Experimental Settings(Syn).

Factors	Settings
number of stores	300, 400, **500**, 600, 700
range of a store's a_i	[1,2], [1,3], **[1, 4]**, [1, 5], [1, 6]
range of a store's fe_i	[0,50], [0,100], **[0,150]**, [0,200], [0,250]
number of users	30k, 40k, **50k**, 60k, 70k
number of requests	100k, 300k, **500k**, 700k, 900k
range of a request's price	[5, 200], [5, 300], **[5, 400]**, [5, 500], [5,600]
range of a request's weight	[0.01, 1], [0.01, 1.5], **[0.01, 2]**, [0.01, 2.5], [0.01, 3]
range of a request's $er_i - br_i$	[1, 60], [1,180], **[1,300]**, [1,420], [1,540]
number of coupons	0.5k, 1k, **1.5k**, 2k, 2.5k
range of c_i's threshold	[50,600],[50,800],**[50,1000]**,[50,1200],[50,1400]
range of c_i's scope $\|S_i\|$	[1,2], [1,4], **[1,6]**, [1,8], [1,10]
range of c_i's quota q_i	[1,3], [1,5], **[1,7]**, [1,9], [1,11]
range of c_i's wait time	[1s, 1h], [1s, 2h], **[1s, 3h]**, [1s, 4h], [1s, 5h]
range of c_i's rebate/threshold	[0.05,0.1],[0.05,0.2],**[0.05,0.3]**,[0.05,0.4],[0.05,0.5]
range of cost of first heavy	[1,6],[1,8],**[1,10]**,[1,12],[1,14]
range of cost of exceed heavy	[1,2], [1,3],**[1,4]**,[1,5],[1,6]
number of batches	1440,288, **144**, 96, 72

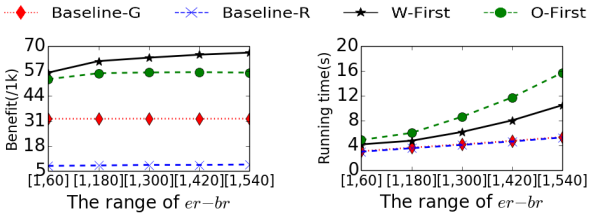
Meanwhile, according to the user analysis report by Taobao [19], we simulate the release time of a request. Specifically, we randomly set the release time of 30% requests between [0:00, 1:00], the release time of 30% requests between [1:00, 10:00], and the release time of 40% requests between [10:00, 24:00]. We simulate each request's waiting time with Uniform distribution with the range $[rw^-, rw^+]$. We generate a set of users as the receivers of requests. The number, num_U, of users is varied from 6000 to 14000. For each user, we randomly select one from the 31 regions as her/his location. We varied the number of batches from 240 to 102, i.e., the length of each batch is varied from 6 minutes to 15 minutes.

Synthetic data sets. To further detect each variable's effect on the algorithm, we generated the synthetic data set and ran the experiments on it. We generate a set of stores for synthetic data sets, whose number is varied from 300 to 700. For each store, we uniformly generate its variety upper bound within the range from [1,2] to [1,6]. We uniformly generate its threshold for each store, fe_i, for the free-shipping service within the range from [0,50] to [0,250]. For each store, we randomly select one among 31 regions as its location. We generate a set of users whose number is varied from 30k to 70k. For each user, we randomly select one among 31 regions as her/his location. We generate a set of requests, whose number is varied from 100k to 900k. For each request, we uniformly generate its price within the range from [5, 200] to [5, 600]. For each request, we uniformly generate its weight within the range from [0.01, 1] to [0.01, 3]. For each request, we randomly select a user as its receiver and uniformly generate its release time within [0:00, 24:00]. For each request, we uniformly generate its waiting time within the range from [1, 60] to [1, 540]. We generate a set of coupons, whose number is varied from 500 to 2500. For each coupon, we uniformly generate its threshold within the range from [50, 600] to [50, 1400]. For each coupon, we uniformly generate the cardinality of its store scope within the range from [1, 2] to [1, 10]. For each



(a) Benefit (b) Running Time

Fig. 3: Effect of the number of users (Real).



(a) Benefit (b) Running Time

Fig. 4: Effect of the range of requests' waiting time (Real).

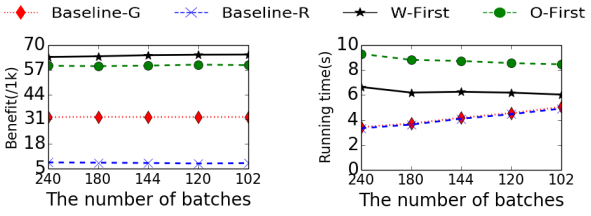
coupon, we uniformly generate its release time within [0:00, 24:00] and generate its waiting time within the range from [1, 60·60] to [1, 5·60·60]. For each coupon, we uniformly generate the ratio of its rebate to its threshold within the range from [0.05, 0.1] to [0.05, 0.5]. We uniformly generate the essential delivery cost between two regions within the range from [1,6] to [1,14]. We uniformly generate the overweight delivery cost between two regions within the range from [1,2] to [1,6]. We varied the number of batches from 1440 to 72.

B. Experiment Configuration

To evaluate the effectiveness and efficiency of our two approaches, the Order-First Approach (O-First) and the Warehouse-First Approach (W-First), we conduct experiments on both real and synthetic data sets. As proved in the Theorem II.1, the Corwdrebate problem is NP-hard. Thus, it is infeasible to calculate the optimal result as the ground truth in a large scale dataset. Alternatively, we compare our approaches with two baseline methods, the Baseline-G algorithm, and the Baseline-R algorithm. The Baseline-G algorithm first randomly assigns each request in order. For each order, the algorithm greedily assigns coupons until no coupon can be added. The Baseline-R algorithm first randomly assigns each request in one order. Then for each order, the algorithm randomly and repeatedly assigns coupons until there is no coupon that can be added.

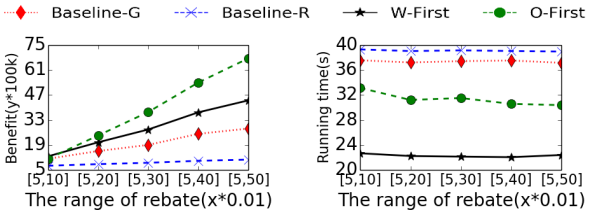
We experiment with a batch style. Specifically, we divide 24h into several batches with the same length. We first retrieve a set of requests and coupons for each batch, which is not assigned in the last batch and is not expired or is just released in this batch. Then, we run the four algorithms with the retrieved request set and the coupon set.

Table IV and Table V show experiment settings on two data sets, where the default values of parameters are in bold font. In each set of experiments, we vary one parameter while setting other parameters to their default values. For each experiment, we sample 10 problem instances to test approaches. We report the average value of the running time and the size of the RS. All experiments were run on an Intel CPU @2.2 GHz with 16GM RAM in Java.



(a) Benefit (b) Running Time

Fig. 5: Effect of the number of batches (Real).



(a) Benefit (b) Running Time

Fig. 6: Effect of rebate (Syn).

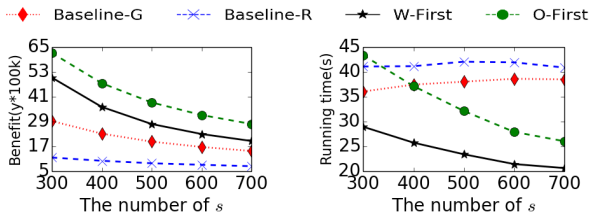
C. Experiment Results on Real Data Sets

To examine our two approximation algorithms' performance, we first conduct experiments on the real data sets.

Effect of the number of users. Figure 3 illustrates the experimental results on a different number of users, from 6k to 14k. In Figure 3(a), when the number of users increases, the benefits of orders generated by the four approaches all decrease. The reason is that, with the increment of the number of receivers, the products should be packaged in more express and should be delivered to more destinations, which increases the delivery cost. Compared with the two baseline algorithm, the benefits of orders generated by our approximation algorithm are much higher. Figure 3(b) illustrates the running time of four algorithms. As proved in Theorem III.1 and Theorem IV.1, the time complexities of our proposed two approaches are not dominated by the number of users, which satisfies the results of the running time.

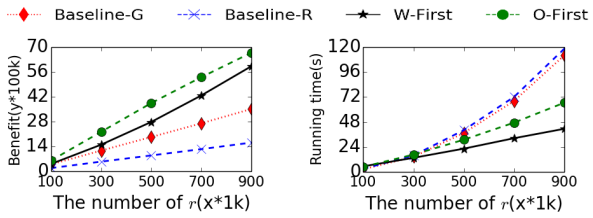
Effect of the range of requests' waiting time. Figure 4 illustrates the experimental results on different range of request' waiting time, from [1 s, 1 min] to [1 s, 9 min]. In Figure 4(a), when the average value of requests' waiting for time increases, the benefits of orders generated by the four approaches all increase. The reason is that the increment of the average value of requests' waiting time makes more valid requests can be grouped into orders, and it is easier to satisfy the time constraint. When the length of requests' waiting time is large enough, the benefits of orders is constrained by other constraints. Thus, when the length of requests' waiting time is large enough, the benefits of orders increase sightly with the increment of the length of waiting time. In Figure 4(b), with the increment of waiting time, the running time of our four algorithms increases. The reason is that the increment of waiting time makes more valid requests to generate more orders, which increases the running time of algorithms.

Effect of the number of batches. Figure 5 illustrates the experimental results on the different number of batches, from 240 to 102. In Figure 5(a), when the number of batches gets smaller, the benefits of orders generated by the four approaches all sightly increase. The reason is that, as the



(a) Benefit (b) Running Time

Fig. 7: Effect of s (Syn).



(a) Benefit (b) Running Time

Fig. 8: Effect of r (Syn).

number of batches gets smaller, the length of each batch is more considerable, and algorithms can get more requests and orders to make assignments, which increases the benefits of final results. In Figure 5(b), with the decrement of the number of batches, the running time of our two approximation algorithms decreases while the running time of two baseline algorithms increases. With the decrement of the number of batches, the number of requests in each batch is higher, making it easier to make assignments. Thus, the total running time of our two approximation algorithms decreases. However, since two baseline algorithms randomly group requests into orders, the number of requests in each batch cannot help save much running time. On the contrary, since the running time of two baseline algorithms is dominated by the process of retrieving a set of valid orders whose active periods are matched for requests, the increment of the number of requests in each batch significantly increases the running time of two baseline algorithms. Thus, combining these two effects, the running time of two baseline algorithms increases.

In the experiments on the entire data set, we find that the benefit of orders generated by our proposed two approaches is much higher than the benefit of orders generated by two baseline approaches. In the case of the entire data sets where the price of requests in the real data set is not high, the coupons used in each order are minimal. The delivery fee varies significantly between different regions. The Warehouse-First approach performs better than the Order-First approach.

D. Experiment Results on Synthetic Data Sets

To further examine the range of each coupon's rebate ratio (i.e., the ratio of the rebate to the threshold), the effects of the number of stores, and the number of requests, we generate the synthetic data set and run the experiments on it. We also test the effects of the range of prices, the range of weight, the number of coupons, the cardinality of each coupon's scope, the range of each coupon's quota, the range of each coupon's active period, the range of each store's variety bound, the range of the threshold of the free-shipping service, the number of receivers, the range of each request's waiting time, the range of each coupon's threshold, the number of batches, the range of the delivery price of the first massive, and the range of the delivery price of the exceed heavy on the synthetic data sets. Due to the space limitation, please refer to our technical report [17] for details.

Effect of the range of each coupon's rebate ratio. Figure 6 illustrates the experimental results on different range of each coupon's rebate ratio, from [0.05, 0.1] to [0.05, 0.5]. In Figure 6(a), when the average value of each coupon's rebate ratio gets larger, the benefits of orders generated by the four approaches all increase. The reason is that the increment of each coupon's rebate ratio's average value lets the rebates of coupons are higher, and orders can use the same coupons to get more rebate. In the case of the synthetic data set, rebates of orders are larger than the delivery cost. Thus, in this case, the benefit of orders generated by the Order-First Algorithm is higher than the benefit of orders generated by the Warehouse-First Algorithm. Figure 6(b) illustrates the running time of four algorithms. As proved in Theorem III.1 and Theorem IV.1, the time complexities of our proposed two approaches are not dominated by the value of coupons' rebate, which satisfies the results of the running time. Since the running time of two baseline algorithms is dominated by the process of retrieving a set of valid orders whose active periods are matched for requests. Thus, in the case of synthetic data sets where the number of requests is huge, the running time of two baseline algorithms is higher than the running time of our two approximation algorithms.

Effect of the number of stores s. Figure 7 illustrates the experimental results on different numbers of stores, from 300 to 700. In Figure 7(a), when the number of stores gets larger, the benefits of orders generated by the four approaches all decrease. The increment of the number of stores would make requests more widely distributed across the stores. When the cardinality of each coupon's scope does not change, it would be more challenging to satisfy the threshold constraint to use coupons. Despite this, the Order-First algorithm also gets the highest benefits, and the Warehouse-First algorithm follows closely. In Figure 7(b), when the number of stores gets larger, the running time of our two approximation approaches decreases sharply while the running time of two baseline algorithms keeps stable. The reason is that when the number of stores gets larger, it is more challenging to satisfy the threshold constraint, and the number of orders generated by our two approximation approaches decreases. The running time of two baseline algorithms is dominated by the process of retrieving a set of valid orders whose active periods are matched for requests. Thus, the running time of two baseline algorithms is not affected by the number of stores' change.

Effect of the number of requests r. Figure 8 illustrates the experimental results on the different number of requests, from 100k to 900k. In Figure 8(a), when the number of requests gets larger, the benefits of orders generated by the four approaches all increase dramatically. Compared with the benefit of orders generated by two baseline algorithms, with the increment of the number of requests, the superiorities of our two approximation algorithms are more obvious. In Figure 8(b), when the number of requests gets larger, the running time of four

algorithms increases dramatically. For each algorithm, the relation between its running time and the number of requests is quadratic, which satisfies our Theorem III.1 and Theorem IV.1.

We summarize our finds as follows:

- Our two approximate approaches can generate orders with much higher benefit compared with that f two baseline algorithms.
- In the cases where rebates are much higher than the delivery cost, the result of the Order-First Approach is higher than the Warehouse-First Approach.
- The Warehouse-First Approach is more efficient than the Order-First Approach. In the cases where the rebate is not much higher than the delivery cost, the benefit of orders that generated by the Warehouse-First Approach is higher than the Order-First Approach.

VI. Related Work

In the field of e-commerce, researchers have proposed many papers about the usage of coupons. These works can be classified into three categories. The works in the first category focus on the effects of coupons on e-commerce platforms. These paper [6][7] illustrate the effect of coupons in the promotion operations from the perspective of economics and marketing. The works in the second category focus on how to design coupons to maximize the profits of retailers. In the early days, the studies in this category mainly rely on marketing experts' experience and insight. Recently, due to the development of data science, many algorithms [8] [9], which can automatically analyze the profiles of users and design a good coupon, have been proposed by computer scientists. For example, in [8], researcher proposed an algorithm, QLFP(Q-learning with forwarding projection), to maximize the retailers' profits by predicting the optimal coupons from historical transaction data. The works in the third category focus on how to allocate coupons. As social networks become more and more popular, social coupons (i.e., a set of coupons, one for self-use and others for social sharing) [11] has been hot research in those years. By sharing social coupons, not only does the person sharing the coupon get the benefit, but the person receiving the coupon also gets the benefit. Simultaneously, the popularity of the retailer who produced the coupons is also increasing during the distribution [13] [11], such that social coupons are widely used in e-commerce platforms. Seed Selection and SC allocation for Redemption Maximization (S3CRM) [10] is an algorithm that aimed to help the online stores allocate the social coupons more efficiently optimize the redemption rate.

Although these excellent works contribute a lot in the research field of coupons, they consider the scenario where a user places her/his purchasing requests in order. In practice, to get more rebate or catch the threshold of free-shipping service, users are motivated to group their requests into orders. In this scenario, where users are collaborating, existing works cannot guide how to group requests to maximize the benefit. Our work makes up for the lack in this scenario.

VII. Conclusion

In this paper, we fist define the Crowdrebate problem, which aims to group a set of requests into orders and assign them with appropriate coupons to maximize the total benefit (the rebate minus the extra delivery cost). We prove that the crowdrebate problem is a NP-hard problem and propose two efficient algorithms: the Order-First algorithm and the Warehouse-First algorithm. We evaluate the effectiveness and efficiency of our algorithms via valid experiments on both synthetic and real data sets. Our work is the first study to group requests to maximize total benefit, which provides new insights into the research of the coupon usage.

References

[1] E. Swilley and R. E. Goldsmith, "Black friday and cyber monday: Understanding consumer intentions on two major shopping days," *Journal of Retailing and Consumer Services*, vol. 20, no. 1, pp. 43–50, 2013.

[2] G. Huang, X. Cheng, J. Wang, Y. Wang, D. He, T. Zhang, F. Li, S. Wang, W. Cao, and Q. Li, "X-engine: An optimized storage engine for large-scale e-commerce transaction processing," in *Proceedings of the 2019 International Conference on Management of Data*, pp. 651–665, 2019.

[3] "[online] Rebate (marketing)." https://en.wikipedia.org/wiki/Rebate_(marketing).

[4] "[online] Every single Black Friday beauty deal worth knowing about." https://www.cosmopolitan.com/uk/beauty-hair/a47386/black-friday-beauty-deals/.

[5] "[online] Double 11 campaign techniques & how to get ready." https://walkthechat.com/double-11-campaign-techniques-how-to-get-ready/.

[6] J. He and W. Jiang, "Understanding users' coupon usage behaviors in e-commerce environments," in *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*, pp. 1047–1053, IEEE, 2017.

[7] K. Jung and B. Y. Lee, "Online vs. offline coupon redemption behaviors," *International Business & Economics Research Journal (IBER)*, vol. 9, no. 12, 2010.

[8] H. Xie, Y. Li, and J. C. Lui, "Optimizing discount and reputation trade-offs in e-commerce systems: Characterization and online learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 7992–7999, 2019.

[9] J. Guo, T. Chen, and W. Wu, "Budgeted coupon advertisement problem: Algorithm and robust analysis," *IEEE Transactions on Network Science and Engineering*, 2020.

[10] T.-C. Chang, Y. Shi, D.-N. Yang, and W.-T. Chen, "Seed selection and social coupon allocation for redemption maximization in online social networks," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pp. 410–421, IEEE, 2019.

[11] S. Hanson and H. Yuan, "Friends with benefits: social coupons as a strategy to enhance customers social empowerment," *Journal of the Academy of Marketing Science*, vol. 46, no. 4, pp. 768–787, 2018.

[12] S. Tang, "Stochastic coupon probing in social networks," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pp. 1023–1031, 2018.

[13] R. M. Gubela, S. Lessmann, and S. Jaroszewicz, "Response transformation and profit decomposition for revenue uplift modeling," *European Journal of Operational Research*, vol. 283, no. 2, pp. 647–661, 2020.

[14] "[online] Sifang." http://express.4px.com/article/detail/id/500063/cid/29.

[15] "[online] Taobao." https://www.taobao.com/.

[16] "[online] Kiehl's Black Friday Event." https://www.dealmoon.com/en/35-off-kiehl-s-skincare-on-black-friday-sale/1637648.html.

[17] "[online] Technical Report ." cspcheng.github.io/pdf/Corwdrebate-ICDE.pdf.

[18] "[online] ZTO." https://www.zto.com/guestservice/bill.

[19] "[online] China Double 11 shopping festival statistics 2019." https://www.chinainternetwatch.com/29999/double-11-2019/.