A Practical Distributed ADMM Solver for Billion-Scale Generalized Assignment Problems

Jun Zhou
College of Computer Science and Technology
Ant Group
Hangzhou, China
jun.zhoujun@antgroup.com

Feng Qi
Ant Group
Sunnyvale, USA
feng.qi@antgroup.com

Zhigang Hua
Ant Group
Sunnyvale, USA
z.hua@antgroup.com

Daohong Jian
Ant Group
Beijing, China
daohong.jdh@antgroup.com

Ziqi Liu
Ant Group
Hangzhou, China
ziqiliu@antgroup.com

Hua Wu
Ant Group
Beijing, China
wu.hua.wh@antgroup.com

Xingwen Zhang
Ant Group
Sunnyvale, USA
xingwen.zhang@antgroup.com

Shuang Yang†
Ant Group
Sunnyvale, USA
shuang.yang@antgroup.com

ABSTRACT
Assigning items to owners is a common problem found in various real-world applications, for example, audience-channel matching in marketing campaigns, borrower-lender matching in loan management, and shopper-merchant matching in e-commerce. Given an objective and multiple constraints, an assignment problem can be formulated as a constrained optimization problem. Such assignment problems are usually NP-hard [21], so when the number of items or the number of owners is large, solving for exact solutions becomes challenging. In this paper, we are interested in solving constrained assignment problems with hundreds of millions of items. Thus, with just tens of owners, the number of decision variables is at billion-scale. This scale is usually seen in the internet industry, which makes decisions for large groups of users. We relax the possible integer constraint, and formulate a general optimization problem that covers commonly seen assignment problems. Its objective function is convex. Its constraints are either linear, or convex and separable by items. We study to solve our generalized assignment problems in the Bregman Alternating Direction Method of Multipliers (BADMM) framework where we exploit Bregman divergence to transform the Augmented Lagrangian into a separable form, and solve many subproblems in parallel. The entire solution can thus be implemented using a MapReduce-style distributed computation framework. We present experiment results on both synthetic and real-world datasets to verify its accuracy and scalability.

CCS CONCEPTS
• Mathematics of computing → Integer programming; • Theory of computation → MapReduce algorithms.

KEYWORDS
Assignment Problem; ADMM; Large-Scale Optimization; Distributed Algorithms

ACM Reference Format:
Jun Zhou, Feng Qi, Zhigang Hua, Daohong Jian, Ziqi Liu, Hua Wu, Xingwen Zhang, and Shuang Yang. 2022. A Practical Distributed ADMM Solver for Billion-Scale Generalized Assignment Problems. In Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM ’22), October 17–21, 2022, Atlanta, GA, USA. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3511808.3557148

1 INTRODUCTION
Many real-world scenarios can be abstracted as assigning “items” to “owners” (or “bins”). To name a few, assigning customers to channels in marketing campaigns [19, 32], assigning borrowers to lenders in loan management [6], and assigning shoppers to merchants in e-commerce [30] are assignment examples that are designed to maximize or minimize some metrics while simultaneously satisfying requirements on other metrics. The assignment problems in internet industry could have multiple variants, and the scale of decision variables could be in billions. Existing literature either study particular forms of assignment problems or cannot solve problems in billion scale. In this paper, we formulate a more general assignment problem, and propose a practical distributed solver to decompose the Augmented Lagrangian of the original large-scale
problem into small pieces via Bregman divergence under BADMM framework [29]. We show the quality of the solution is promising in real-world applications.

1.1 Problem Formulation
To assign $I$ items to $J$ owners, we use $X \in [0, 1]^{I \times J}$ or $X \in [0, 1]^{J \times I}$ to represent the decision variables. In the former case, the element in row $i$ and column $j$ $x_{ij} \in [0, 1]$ is 1 if item $i$ is assigned to owner $j$, and 0 if not. In the latter case, $x_{ij} \in [0, 1]$ indicates that a portion of item $i$ is assigned to owner $j$. We also use $x_i \in [0, 1]^J$ or $x_i \in [0, 1]^I$ to represent the $i$-th row of $X$.

We study the following generalized constrained assignment problem: (1)–(5),

$$\min_X f(X)$$

subject to

$$X^T u_m \leq b_m, \quad \forall m \in [M] \tag{2}$$

$$X^T c_n = e_n, \quad \forall n \in [N] \tag{3}$$

$$h_i(x_i) \leq 0, \quad \forall i \in [I] \tag{4}$$

$$x_{ij} \in [0, 1] \quad \text{or} \quad 0 \leq x_{ij} \leq 1 \quad \forall i \in [I], \forall j \in [J]. \tag{5}$$

We require that when $X$ is continuous on $[0, 1]^{I \times J}$, the objective function $f(X)$ is convex. The first set of constraints (2) are $J \times M$ inequalities, with coefficients $u_m \in \mathbb{R}^J$ and $b_m \in \mathbb{R}$. They allow each of the $J$ owners to have one linear inequality constraint on each of a set of $M$ item features. The second set of constraints (3) are $J \times N$ equations, with coefficients $c_n \in \mathbb{R}^J$ and $e_n \in \mathbb{R}$. They allow each of the $J$ owners to have one linear equality constraint on each of another set of $N$ item features. The third set of constraints (4) are $I$ inequalities each on one item $x_i$. We allow each item to have its unique $h_i(x_i)$. The most commonly used such constraint is $h_i(x_i) = \sum_j x_{ij} - 1 \leq 0$. We distinguish between inequality (2) and equality (3) constraints because the former needs an extra treatment in our algorithm. For item-level constraints (4), we only consider inequality without loss of generality. Constraint (5) either requires each $x_{ij}$ to be a binary decision, or allows a portion of each item $i$ to be allocated to an owner $j$.

1.2 Our Work
There exist three main challenges in solving the above assignment problems:

- The scale of billions of decision variables is unprecedented. We implemented distributed algorithm on top of the MapReduce framework, and verified its scalability and convergence in experiments and production.

- The assignment problems are NP-hard integer programming (IP) problems. We solve the linear programming (LP) relaxation of the original IP problems and project the solutions back to integers. We empirically verify this practical approach yields results that are close to optimal and acceptable.

- The complexity of solving the assignment problems is further increased when the objective function is nonlinear and non-separable. Though the BADMM is a general framework to deal with quadratic penalties, the use of Bregman divergence [29] to derive decomposable forms for our general assignment problems is still non-trivial.

As far as we know, this has been one of the first works to solve the assignment problems in billion-scale. The experiments have shown promising results in terms of both accuracy and convergence.

2 RELATED WORK
2.1 Generalized Assignment Problem (GAP)
GAP [21] is a well studied problem, and can be viewed as a generalization of knapsack problem or bin-packing problem. It solves for the minimum cost when assigning $I$ items to $J$ owners. Each owner has a capacity $r_j$. Each item is assigned to one owner. Its mathematical formulation is

$$\min_X \sum_i \sum_j p_{ij} x_{ij} \tag{6}$$

subject to

$$\sum_i q_{ij} x_{ij} \leq r_j, \quad \forall j \in [J] \tag{7}$$

$$\sum_j x_{ij} = 1, \quad \forall i \in [I] \tag{8}$$

$$x_{ij} \in [0, 1], \quad \forall i \in [I], \forall j \in [J]. \tag{9}$$

In (6)-(9), $p_{ij} \in \mathbb{R}_{\geq 0}$ is the cost of assigning item $i$ to owner $j$, and $q_{ij} \in \mathbb{R}_{\geq 0}$ is the consumption of capacity if assigning item $i$ to owner $j$.

GAP is a linear IP problem, and is NP-hard. To solve for exact solutions, branch-and-bound and branch-and-price methods can be used [26].

When exact solution is not necessary, one class of algorithms solves GAP by relaxing constraints. Ross and Soland [25] proposed an algorithm that first relaxes (7) to get a lower bound result, and then solves $J$ knapsack problems. Martello et al. [22] relaxes (8), and solves $J$ knapsack problems, while a penalty is applied to alleviate constraint violation. Jörnsten and Näsberg [15] proposed a Lagrangian relaxation algorithm, where the optimal Lagrangian multipliers are obtained by a subgradient method. Fisher et al. [11] developed a multiplier adjusting method to determine good Lagrangian multipliers. Linear programming relaxation (LP-relaxation) can also be applied by replacing (9) with $x_{ij} \geq 0$. In “practical” problems where $I >> J$, this relaxation leads to solutions that are within 0.1% of their LP lower bounds, while in settings where $\frac{J}{I}$ equals 3 or 4, reassigning the fractional jobs can turn out to be infeasible [4].

Another class of algorithms solve GAP by heuristic optimization, for example, genetic algorithms [7][10], simulated annealing and tabu search [23], heuristic search and ant system [20].

Variants of GAP are also studied. In Multi-Resource GAP, each owner has multiple resource constraints [12]. In Multilevel GAP, a third dimension “level” is added to each item and owner combination, thus each assignment $x_{ij}$ operating at level $k$ is represented as $x_{ijk}$. [1, 5]. Standard GAP considers resource upper bounds, whereas [17] considered lower bounds. [31] discussed the scenario when multiple items from each of $K$ categories are required by each owner, and each item belongs to a subset of the $K$ categories.

Our formulation (1)-(5) is different from GAP. First, we allow nonlinear objective (1) and nonlinear per-item constraints (4). Second, we allow each item feature $u_m$ and $v_n$ to interact with each
owner $j$ in (2) and (3). GAP is actually a special case of (1)-(5). To see this:

- Notice (6) and (8) are specified (1) and (4), respectively.
- To convert (7) to (2), set $M = J$ to get $Jx_j$ inequality constraints. When $m \neq j$, set the $j$-th element in $b_m$ to infinity.
- To remove (3), set all elements of $c_n$ to infinity.

In fact, the constraint structure of GAP, Multi-Resource GAP, and our formulation are all just $Ax \leq b^*$ with different block matrix patterns, when we use a vector $x \in \mathbb{R}^{J \times 1}$ to represent decision variables. ADMM discussed in Section 2.2 can solve all these systems.

### 2.2 ADMM

ADMM [3] is an optimization framework for constrained optimization. It uses the Augmented Lagrangian function as objective. For an equality-constrained optimization, the objective adds quadratic penalties on constraint violation to the Lagrangian function. ADMM can be extended to work with inequality constraints by adding slack variables.

An ADMM method iteratively solves primal and dual variables until a stopping criterion is met. To solve for primal variables, it separates variables into two or more blocks, and solves them sequentially with Gauss-Seidel routine [2]. Due to the quadratic penalty in the Augmented Lagrangian, solving each block can be viewed as a proximal problem [24]. To solve for dual variables, ADMM uses a subgradient method with a predetermined learning rate. It was shown that when the objective function of the primal problem is a summation of convex functions, ADMM enjoys a linear convergence rate on primal and dual variables [14], and $O(1/T)$ convergence rate on objective [13], where $T$ is the number of iterations. ADMM can also be used in non-convex optimization, such as mixed-integer quadratic programming, though there is a lack of convergence guarantee, and hence the usage needs to be justified by experiments [27].

When the Augmented Lagrangian is difficult to solve, each block of variables may be constructed easier to solve. For a large-scale optimization problem, if we can make each block separable on decision variables, we can solve sub-problems in parallel with Jacobi routine [2]. The non-separability of Augmented Lagrangian may be due to either the quadratic penalty or the primal objective. Proximal Jacobi ADMM [8, 9] adds to the Augmented Lagrangian a Mahalanobis distance between the primal variables and their values in the previous iteration. By choosing the Mahalanobis distance carefully, any quadratic penalties due to the general form of equality constraints $Ax = b^*$ can be reduced to separable terms. Generalized Bregman ADMM (BADMM) [29] generalizes the quadratic penalty to a Bregman divergence. The proximal term between primal variables and their values in the previous iteration is modeled as another Bregman divergence. By constructing these Bregman divergences carefully, any non-separable terms in the Augmented Lagrangian can be reduced to separable terms. BADMM enjoys $O(1/T)$ convergence on objective, when the objective is convex, and the penalty Bregman divergence is defined on a $\alpha$-strongly convex function, where $\alpha > 0$. Linearized ADMM [18] uses first-order approximation and second order proximal terms to approximate the Augmented Lagrangian in ADMM iterations, and hence making the problem separable.

In this paper, since the use of Bregman divergence could help us effectively exploit the structure of problems, we follow the framework of BADMM [29] and use Bregman divergence to derive decomposable sub-problems for our generalized assignment problems with even non-separable objectives so that it can be solved in parallel.

### 3 OUR APPROACH

To handle large scale items ($I$) and owners ($J$), we solve (1) - (5) in the BADMM framework. This framework updates primal and dual variables iteratively, as ADMM does, and decomposes the objective function $f(X)$ into a separable form via replacing non-separable terms with Bregman divergence terms. Therefore the primal update step in ADMM can be divided into many subproblems and solved parallelly. To discuss the detailed routine, we first form the Augmented Lagrangian. We rewrite (1) - (5) by moving (4) and (5) into the objective. We also convert (2) into equality constraints by introducing additional positive decision variables $\xi_m \in \mathbb{R}^{M\times 1}$:

$$
\min_{X, \xi} f(X) + \sum_i \Phi(h_i(x_i)) + \sum_{m,j} \Phi(-\xi_{m,j}) + \sum_{i,j} \Psi(x_{i,j})
$$

(10)

$$
s.t. \quad X^Tu_m + \xi_m = b_m, \quad \forall m \in [M]
$$

(11)

$$
X^Tv_n = c_n, \quad \forall n \in [N].
$$

(12)

In (10), $\Phi$ and $\Psi$ are defined as below.

$$
\Phi(y) = \begin{cases} 
0, & y \leq 0 \\
+\infty, & y > 0 
\end{cases}
$$

(13)

$$
\Psi(y) = \begin{cases} 
0, & y \text{satisfies (5)} \\
+\infty, & \text{otherwise}
\end{cases}
$$

(14)

Therefore (10) - (12) are equivalent to (1)-(5). The Augmented Lagrangian based on (10) - (12) is as below.

$$
L(X, \xi, \lambda, \mu) = f(X) + \sum_i \Phi(h_i(x_i)) + \sum_{m,j} \Phi(-\xi_{m,j}) + \sum_{i,j} \Psi(x_{i,j})
$$

$$
+ \frac{\rho}2 \sum_m \|X^Tu_m + \xi_m - b_m + \frac{\lambda_m}{\rho}\|_2^2
$$

$$
+ \frac{\rho}2 \sum_n \|X^Tv_n - c_n + \frac{\mu_n}{\rho}\|_2^2.
$$

(15)

Here we use $\lambda_m \in \mathbb{R}^J$ to represent the Lagrangian multipliers of (11), and $\mu_n \in \mathbb{R}^J$ to represent the Lagrangian multipliers of (12). $\rho$ is a predetermined constant for the quadratic penalty terms. We also use $\xi$ to represent the collection $\xi_{1:M}$, $A$ to represent the collection $\lambda_{1:M}$, and $\mu$ to represent the collection $\mu_{1:N}$.

Using ADMM, we can solve (15) by iteratively running the following four steps described in Algorithm 1. We use superscript to label the iteration in which a variable is determined.

#### 3.1 ADMM Routine

Here we introduce the following steps in each ADMM iteration as per line 4-7 in Algorithm 1.
Algorithm 1 The ADMM routine for solving (10) - (12) by (15)
1: Input: Iteration $t = 0$, and initialize $X, \xi, \lambda, \mu$ with 0.
2: Output: $X$
3: while Not converged do
4: \[ X^{(t+1)} = \arg \min_{X} L(X, \xi^{(t)}, \lambda^{(t)}, \mu^{(t)}) \]
5: \[ \xi^{(t+1)} = \arg \min_{\xi} L(X^{(t+1)}, \xi, \lambda^{(t)}, \mu^{(t)}) \]
6: \[ \lambda^{(t+1)} = \lambda^{(t)} + \rho ((X^{(t+1)})^T \mu^{(t+1)} - b_m) \quad \forall m \]
7: \[ \mu^{(t+1)} = \mu^{(t)} + \rho ((X^{(t+1)})^T \nu - c_n) \quad \forall n \]
8: \[ t = t + 1 \]
9: end while

Step 1: Update $X$. We first update the decision variables $X$ as follows:
\[ X^{(t+1)} = \arg \min_{X} f(X) + \sum_{i,j} \Phi(h_i(x_i)) + \sum_{i,j} \Psi(x_{i,j}) \]
\[ + \frac{\rho}{2} \sum_{m} \|X^T u_m + \xi^{(t)} - b_m + \frac{\lambda^{(t)}}{\rho} \|^2 \quad \forall m \]
\[ + \frac{\rho}{2} \sum_{n} \|X^T v_n - c_n + \frac{\mu^{(t)}}{\rho} \|^2 \quad \forall n. \]

This problem is equivalent to solving $M \times F$ 1-variable constrained quadratic programming problem:
\[ \min_{\xi_{m,j}} (\xi_{m,j} - l_{m,j})^2 \]
\[ s.t. \quad l_{m,j} \geq 0. \]

Here $l_{m,j} = -(X^{(t+1)})^T u_m + b_m - \frac{\xi^{(t)}}{\rho}$, and $l_{m,j}$ is the j-th element of $l_m$. The solution to the problem is
\[ \xi_{m,j} = \max(0, l_{m,j}). \]

Step 3-4: Update $\lambda$ and $\mu$. The last two steps are to update the Lagrangian multipliers.
\[ \lambda_{m}^{(t+1)} = \lambda_{m}^{(t)} + \rho ((X^{(t+1)})^T u_m + \xi^{(t+1)} - b_m) \]
\[ \mu_{n}^{(t+1)} = \mu_{n}^{(t)} + \rho ((X^{(t+1)})^T v_n - c_n). \]

According to (20), if $\xi_{m}^{(t+1)}$ is larger than 0, $\lambda_{m}^{(t+1)}$ will be set to 0 in the $t + 1$-th iteration.

3.2 Decomposition via Bregman Divergence

(17) is still a difficult problem given that the scale of $X$ is large and that the exact formula of $f(X)$ is nonseparable by elements of $X$. We derive to convert (17) into a separable function of $X$ under the BADMM framework. Basically, we introduce a Bregman divergence term into the Augmented Lagrangian.
\[ X^{(t+1)} = \arg \min_{X} f(X) + \sum_{i} \Phi(h_i(x_i)) + \sum_{i,j} \Psi(x_{i,j}) \]
\[ + \frac{\rho}{2} \sum_{s} \|X^T w_s - d_s + \frac{v_s}{\rho} \|^2 \quad + B_{\Omega}(X, X^{(t)}) \]

The Bregman divergence $B_{\Omega}(x, y)$ defined on a convex function $\omega(z) \in \mathbb{R}$ is $\omega(x) - \omega(y) - \omega'(y)(x - y)$. Our decision variable is $X$ as a matrix in $\mathbb{R}^{K \times F}$. For convenience we slightly abuse the derivative symbol $\Omega'(X)$ to let it represent a $\mathbb{R}^{K \times F}$ matrix, the element on the i-th row and j-th column of which is the derivative of $\Omega(X) \in \mathbb{R}$ to $x_{i,j}$. We use $\circ$ to represent Hadamard product. We also use the operation $1^T A 1$ to sum up all elements of a matrix $A \in \mathbb{R}^{K \times F}$. In this case the first 1 is a vector of ones in $\mathbb{R}^K$, and the second 1 is a vector of ones in $\mathbb{R}^F$. We use the same symbol for these two vectors for simplicity, since their dimensions are known once $A$'s dimension is known. The Bregman divergence term in (22) can be expressed as:
\[ B_{\Omega}(X, X^{(t)}) = \Omega(X) - \Omega(X^{(t)}) - 1^T \Omega'(X^{(t)}) \circ (X - X^{(t)})1 \]

In our design, $\Omega$ is composed of two components $\Omega_1$ and $\Omega_2$.
\[ \Omega = \Omega_1 + \Omega_2 \]
\[ \Omega_1 = g(X) - f(X) \]
\[ \Omega_2 = \beta \text{tr}(X^T X) - \frac{\rho}{2} \sum_{s} w_s^T XX^T w_s \]

The solution can be obtained as follows:
\[ X^{(t+1)} = \arg \min_{X} \sum_{k} g(X_k) - 1^T (g'(X^{(t)}) - f'(X^{(t)})) \circ X1 \]
\[ + \sum_{i} \Phi(h_i(x_i)) + \sum_{i,j} \Psi(x_{i,j}) \]
\[ + \beta \text{tr}(X - X^{(t)})^T (X - X^{(t)})1 \]
\[ + \rho \text{tr}(X^T) (\sum_{s} w_s(X^{(t)})^T w_s - d_s + \frac{v_s}{\rho}) \circ X1 \]

Symbols $g'$ and $f'$ are used in a similar way as $\Omega'$, each element of both matrices represents the derivative with respect to the corresponding element of $X$. $\beta$ is chosen such that $\beta \geq \frac{\rho}{2} (M + N)$. The function $g(X)$ is constructed such that it is separable on the first dimension of $X$, i.e. $g(X) = \sum_k g_k(X_k)$, where $|k|$ is a mutually exclusive and collectively exhaustive set of $X$ row index clusters. We assume such $g(X)$ can be found by choosing a convex $\Omega_1$. In Section 4 we provide concrete examples.
and form a $\mathbb{R}^{[k] \times J}$ matrix. (27) can be divided into $|k|$ independent optimization problems.

$$\min_{X_k} \quad g(X_k) + 1^T B_k \circ X_k 1 + \beta \operatorname{tr} ((X_k - X_k^{(t)})^T (X_k - X_k^{(t)}))$$

s.t.  \hspace{0.5cm} h_i(x_i) \leq 0 \quad \forall i \in k \hspace{0.5cm} \text{or} \hspace{0.5cm} 0 \leq x_{i,j} \leq 1 \quad \forall i \in k, j \hspace{0.5cm} \text{where} \hspace{0.5cm} B = -g'(X^{(t)}) + f'(X^{(t)}) + \rho \sum_s w_s (X^{(t)} T w_s - d_s + \frac{\gamma}{\rho})^T$$

We have moved the non-differentiable components $\Psi$ and $\Phi$ to constraints to illustrate that (28) is a constrained convex optimization problem. In practice $X_k$ can usually be made much smaller than $X$ in size, and the $|k|$ problems of (28) can be solved in parallel using existing convex optimization techniques.

### 3.3 Acceleration For Quadratic Objective On Simplex Region

In practice, we found that $g(X)$ can often chosen to be a quadratic function that is separable over $x_{i,j}$, i.e., $g(X) = \sum_i \sum_j a_{ij} x_{ij}$ where $a_{ij} \in \mathbb{R}$. Two concrete examples are presented in Section 4. The per-item constraint $h_i(x_i) \leq 0$ is usually $\sum_j x_{i,j} \leq 1$. Thus each block $k$ in (28) can be further separated into many optimization problems for each item $i$.

$$\min_{x_{ij}} \quad \sum_j a_{ij} x_{ij}^2 + B_{ij} x_{ij} + \beta (x_{ij} - x_{ij}^{(t)})^2$$

s.t.  \hspace{0.5cm} \sum_j x_{ij} \leq 1 \quad \text{and} \quad x_{i,j} \geq 0 \quad \forall j. \quad (29)$$

$B_{ij}$ is the element of $B$ on the $i$-th row and $j$-th column. Transform (29) to simplify its formula.

$$\min_{x_{ij}} \quad \sum_j y_{ij} x_{ij}^2 + \eta_{ij} x_{ij}$$

s.t.  \hspace{0.5cm} \sum_j x_{ij} \leq 1 \quad \text{and} \quad x_{i,j} \geq 0 \quad \forall j \quad (30)$$

where $y_{ij} = a_{ij} + \beta$. $\eta_{ij} = B_{ij} - 2 \beta x_{ij}^{(t)}$.

This sub-problem can be solved with $O(J \ln J)$ time complexity. We formulate its partial Lagrangian

$$\max_{\pi} \min_{x_{ij}} \quad \sum_j y_{ij} x_{ij}^2 + \pi \sum_j x_{ij} - 1$$

s.t. $x_{ij} \geq 0 \quad \forall j, \pi \geq 0. \quad (31)$

The solution to the inner minimization problem, considering the constraint on $x_{ij}$, is $x_{ij} = \max(0, -\frac{\eta_{ij} + \pi}{2y_{ij}}), \forall j$. Substituting $x_{ij}$ with this solution gives

$$\max_{\pi} \quad \sum_j (y_{ij} \max(0, -\frac{\eta_{ij} + \pi}{2y_{ij}})^2 - (\frac{\eta_{ij} + \pi}{2y_{ij}})^2 - \pi$$

where $\pi \geq 0. \quad (32)$

Let $\tilde{x}_{ij}$ be the $\pi_{\tilde{x}}$ with the maximum objective (35), thus the corresponding solutions to $x_{ij}$ is $x_{ij}^{*} = \max(0, -\frac{\eta_{ij} + \pi_{\tilde{x}}}{2y_{ij}})$.

Consider the entire procedure, an outer loop running a $O(\ln J)$ binary search over the concave (32) is used to search for a range of $\pi$. At each range of $\pi$, $\pi_{\tilde{x}}$ and its objective are evaluated with $O(J)$ time. It is possible to pre-compute $\pi_{\tilde{x}}$ and the three coefficients for its objective with $O(J)$ time, though an initial sort of $\eta_{ij}$ is required for this pre-computation, so the overall complexity is still $O(J \ln J)$.

### 3.4 Post-processing for Integer Solutions

When integer solution is desired, we use post-process to convert continuous solutions to integers. As discussed in Section 2, when the number of decision variables is small, post-process like rounding can lead to infeasibility or suboptimality, but in large scale problems it empirically generates near-optimal solutions. Given local constraints like $\sum_j x_{ij} \leq 1, \forall i \in [I]$, our post-processing treats each $x_{ij}$ as the probability of assigning item $i$ to owner $j$, and the probability of not assigning $i$ to any $j$ is $1 - \sum_j x_{ij}$. We sample a binary solution from such probability simplex for each item, to obtain integer solutions.
3.5 Distributed System Design

Figure 1 shows the system construction of our distributed solver, which consists of an orchestrator, an aggregator, a farm of workers, and a permanent data store.

Coefficients of \( g \), \( u_m \), and \( v_n \) are stored in the permanent data store, and are partitioned and indexed by sub-problems. Coefficients \( b \) and \( c \), parameters \( \rho \), and other configurations are sent to all servers during task initialization.

Each worker obtains a list of sub-problem IDs from the orchestrator, then retrieves the corresponding coefficient partitions from the data store, and solves these sub-problems to update \( X \). Each worker keeps its partitions of coefficients and intermediate result \( X_k \) in memory, so as to reuse these values in subsequent iterations. Instead of sending \( X_k \) to the aggregator, each worker aggregates its partition solution locally. The local aggregation results are partial left-hand-side values of constraints. Workers report them to the orchestrator.

The orchestrator keeps track of each worker’s sub-problem partitions, and whether a partition has been solved in current iteration. It assigns unsolved partitions to available workers if either these partitions have not been assigned, or the assignee worker fails health check.

Once all sub-problems are solved in an iteration, the orchestrator sends partially aggregated constraint left-hand-sides to the aggregator. The aggregator is a specialized worker that knows how to update \( \xi \), \( \lambda \), and \( \mu \).

The system tolerates failures on workers and the aggregator, by health-checking and maintaining redundancy on servers. If a worker fails, the orchestrator will assign its job to another worker, which can retrieve coefficients and \( X_k \) from the permanent data store. The aggregator is stateless, so if it fails, the aggregation task can just be sent to its backup.

4 RESULTS ON SYNTHETIC DATA

In this section, we set up two classes of non-linear non-separable problems. We aim to answer the following questions: (1) how can our approach approximate the optimal solution in small assignment problems? (2) how does our post-processing of sampling from simplex \( x_i \) affect the solution of integer programming when we vary the problem scales? (3) The scalability of our approach with 1 billion decision variables.

4.1 Objective Functions

If the objective function \( f \) is separable by item, for example linear, we can set \( g \) to \( f \). The problems could be more challenging when we have non-separable (by item) cases. We specify two non-separable objective functions, and give their corresponding choices of \( \Omega_1 \). Their convexity can be proven by taking the derivatives and observe the function values are always positive.

4.1.1 Quadratic objective.

\[
\min_X \frac{1}{2} \sum_j (\omega_j^T x_j + \alpha_j)^2 \tag{36}
\]

s.t. \( X^T u_m \leq b_m, \forall m \in [M], \quad X^T v_n = c_n, \forall n \in [N], \)
\[
\sum_j x_{ij} \leq 1, \quad \forall i \in [I], \quad 0 \leq x_{ij} \leq 1, \quad \forall i \in [I], \forall j \in [J] \tag{37}
\]

Let convex function \( \Omega_1 \) be
\[
\Omega_1 = g(X) - f(X) \tag{38}
\]

\[
= \sum_j N \omega_j^T \text{diag}(x_j \circ x_j)\omega_j - \frac{1}{2} \sum_j (\omega_j^T x_j + \alpha_j)^2 \tag{39}
\]

4.1.2 Logarithmic objective.

\[
\min_X -\sum_j \ln(\omega_j^T x_j + \alpha_j) \tag{40}
\]

s.t. \( X^T u_m \leq b_m, \forall m \in [M] \)
\[
X^T v_n = c_n, \forall n \in [N], \quad \sum_j x_{ij} \leq 1, \quad \forall i \in [I] \tag{41}
\]
\[
0 \leq x_{ij} \leq 1, \quad \forall i \in [I], \forall j \in [J] \tag{42}
\]

Let convex function \( \Omega_1 \) be
\[
\Omega_1 = g(X) - f(X) \tag{43}
\]

\[
= \sum_j N \omega_j^T \text{diag}(x_j \circ x_j)\omega_j + \sum_j \ln(\omega_j^T x_j + \alpha_j) \tag{44}
\]

4.2 Coefficients and Parameters

If not otherwise stated, coefficients and parameters used in the subsequent experiments are generated as follows. For both (4.1.1) and (4.1.2), each element of coefficient vectors \( \omega_j \) and \( v_n \) are drawn from uniform distribution \([0, 1]\). Each element of vectors \( u_m \) is drawn from uniform distribution \([-1, 0]\). Each element of vector \( b_m \) is set to 0.3. Each element of vector \( c_n \) is set to be \( 0.3 \frac{I}{J} \). In (4.1.1), each element of vector \( \alpha_j \) is set to 10^{-4}I, and \( \rho \) is set to 10^{-3}. In (4.1.2), each element of vector \( \alpha_j \) is set to be 10^{-1}I, and \( \rho \) is set to 10^{-2}. \( \beta \) is set to \( \frac{2}{3}I(M + N) \).

4.3 Algorithm Correctness

4.3.1 Evenly distributed coefficients. We compared the objective values of the solution obtained by our implementation and that of the open-sourced solver IPOPT [28]. IPOPT is a commonly used non-linear constrained optimization solver. We use it as a reference, and argue that an optimal solution should be close to an IPOPT solution. We use absolute percentage difference (APD) to quantify the difference between the two solutions. We use sol to represent the objective value of ADMM solution, and sol* to represent the objective value of IPOPT.

\[
\text{APD} := \left| \frac{\text{sol} - \text{sol}^*}{\text{sol}^*} \right| \tag{46}
\]

To quantify the violation of an inequality constraint, we use the left-hand-side subtract the right-hand-side of the inequality, and
ADMM run 2500 iterations, and found the max objective APD is \(\text{MAPD} \), and the mean of the APD for all its equality constraints to.

\[
\text{Table 1: Results of quadratic objective problems}
\]

| \(M\) | \(N\) | \(J\) | Obj. APD | Ineq. MAPD | Eq. MAPD |
|---|---|---|---|---|---|
| 3 | 2 | 10 | 0.0032 | 0.0006 | 0.0006 |
| 5 | 2 | 10 | 0.0069 | 0.0008 | 0.0011 |
| 10 | 2 | 10 | 0.0250 | 0.0006 | 0.0014 |
| 3 | 5 | 10 | 0.0385 | 0.0009 | 0.0010 |
| 3 | 10 | 10 | 0.0256 | 0.0009 | 0.0010 |
| 3 | 2 | 15 | 0.0351 | 0.0009 | 0.0006 |

\[
\text{Table 2: Results of logarithmic objective problems}
\]

| \(M\) | \(N\) | \(J\) | Obj. APD | Ineq. MAPD | Eq. MAPD |
|---|---|---|---|---|---|
| 3 | 2 | 10 | 0.0421 | 0.0 | 0.0004 |
| 5 | 2 | 10 | 0.0420 | 0.0 | 0.0004 |
| 10 | 2 | 10 | 0.0414 | 0.0 | 0.0004 |
| 3 | 5 | 10 | 0.0379 | 0.0 | 0.0005 |
| 3 | 10 | 10 | 0.0353 | 0.0 | 0.0006 |
| 3 | 2 | 15 | 0.0326 | 0.0 | 0.0008 |

set it to zero if this difference is negative. For example, the \(j\)-th constraint for a specific \(m\) of (2) is \(x_j^T u_m \leq b_{mj}\), and its APD is

\[
\text{APD} := \frac{\max(x_j^T u_m - b_{mj}, 0)}{b_{mj}}
\]

To quantify the violation of an equality constraint, we use the relative difference between the left-hand-side and right-hand-side. For example, the \(j\)-th constraint for a specific \(n\) of (3) is \(x_j^T v_n = c_{nj}\), and the APD is

\[
\text{APD} := \frac{|x_j^T v_n - c_{nj}|}{c_{nj}}
\]

With the solution to one problem, we take the mean of the APD for all its inequality constraints to get the inequality mean APD (MAPD), and the mean of the APD for all its equality constraints to get the equality MAPD.

The results of the correctness metrics are summarized in Table 1 and Table 2. To solve the problems with IPOPT, which runs on single machine, we set \(l = 3000\). We tested on various values of \(M, N, J\). To solve the quadratic objective problems, we let ADMM run 2500 iterations, and found the max objective APD is less than 0.04, the max inequality MAPD is less than 0.001, and the max equality MAPD is less than 0.0015. To solve the logarithmic objective problems, we let ADMM run 1000 iterations, and found the max objective APD is less than 0.05, the max inequality MAPD is 0, and the max equality MAPD is less than 0.0008. These difference metrics indicate the ADMM solutions are close to optimal with very small constraint violation.

### 4.3.2 Unevenly distributed coefficients

When different sub-problems have different coefficient distributions, solving the optimization problems become more difficult. To test the quality of our ADMM solution in such scenarios, we mix 1500 items which have coefficient values drawn from the distributions described in 4.2, with another 1500 items which have coefficients scaled 10 times, i.e. \(u_j\) and \(v_n\) are drawn from uniform distribution \([0, 10]\), and \(u_m\) is drawn from uniform distribution \([-10, 0]\). Other coefficients and parameters are the same as in 4.3.1.

The results are summarized in Table 3 and Table 4. The quadratic objective problems have max objective APD less than 0.05, max inequality MAPD less than 0.002, and max equality MAPD less than 0.0003. The logarithmic objective problems have max objective APD less than 0.09, max inequality MAPD as 0, and max equality MAPD less than 0.0009. These difference metrics indicate that when items have drastically (10x) different coefficient distributions, constraint violation remains at the same level, while solution optimality decreased, at least in the logarithmic problem.

### 4.4 Effects of Integer Relaxation

We evaluate constraint MAPD as described in 4.3.1. We evaluate the solution objective by comparing it to the continuous solution objective, which can be viewed as a lower bound. If we represent the continuous solution objective as \(lb\), the objective APD is

\[
\text{APD} := \frac{\text{sol} - \text{lb}}{\text{lb}}
\]

With coefficients and parameters set the same as in 4.3.1, Table 5 and Table 6 summarize the solution quality. The quadratic objective problems have max objective APD less than 0.025, max inequality MAPD less than 0.015, and max equality MAPD less than 0.015. The logarithmic objective problems have max objective APD less than 0.002, max inequality MAPD as 0, and max equality MAPD less than 0.05. These difference metrics indicate that with tens of thousands of decision variables (30,000–45,000), integer post-processing generates objectives close to their relaxation lower bounds, and that constraint violation increases, though still at low level.
Table 5: Results of quadratic objective problems with integer post-processing

| M  | N  | J  | Obj. APD | Ineq. MAPD | Eq. MAPD |
|----|----|----|----------|------------|----------|
| 3  | 2  | 10 | 0.0080   | 0.0030     | 0.0046   |
| 5  | 2  | 10 | 0.0119   | 0.0087     | 0.0084   |
| 10 | 2  | 10 | 0.0113   | 0.0131     | 0.0136   |
| 3  | 5  | 10 | 0.0053   | 0.0064     | 0.0111   |
| 3  | 10 | 10 | 0.0118   | 0.0065     | 0.0086   |
| 3  | 2  | 15 | 0.0211   | 0.0049     | 0.0128   |

Table 6: Results of logarithmic objective problems with integer post-processing

| M  | N  | J  | Obj. APD | Ineq. MAPD | Eq. MAPD |
|----|----|----|----------|------------|----------|
| 3  | 2  | 10 | 0.0001   | 0.0        | 0.0386   |
| 5  | 2  | 10 | 0.0      | 0.0        | 0.0429   |
| 10 | 2  | 10 | 0.0      | 0.0        | 0.0423   |
| 3  | 5  | 10 | 0.0001   | 0.0        | 0.0321   |
| 3  | 10 | 10 | 0.0002   | 0.0        | 0.0423   |
| 3  | 2  | 15 | 0.0002   | 0.0        | 0.0345   |

4.5 Scalability and Convergence

We then scale up the problem sizes to $I = 10^8$. We set $J = 10$, $M = 3$, and $N = 2$. Thus the number of decision variables is 1 billion. This scale corresponds to the scenario of making one-out-of-ten choices for 100 million items, which is not uncommon for internet companies.

Figure 2 shows objective (log10 scale), inequality MAPD, and equality MAPD versus ADMM iterations, for the quadratic objective problem. At 2000 iterations, inequality MAPD is 0.00036 and equality MAPD is 0.000036. After integer post-processing by sampling probability simplex, inequality MAPD is still 0.00036 and equality MAPD is still 0.000036. The objective curve has converged, and the APD between continuous and integer solution is $3.6 \times 10^{-6}$. We only show the result of the first 300 iterations as the remaining curves look flat.

Figure 3 shows the same metrics for the logarithmic objective problem. At 100 iterations, inequality MAPD is zero, equality MAPD is $10^{-5}$. After integer post-processing by sampling probability simplex, inequality MAPD is still zero, and equality MAPD is still $10^{-5}$. The objective curve has converged, and the APD between continuous and integer solution is $7.2 \times 10^{-5}$. We only show the result of the first 50 iterations as the remaining curves look flat.

We implemented the algorithm using the programming language Python. When the software runs on 2000 CPUs, the average time consumed per iteration is 100 seconds. This includes all steps in Section 3.1. The implementation features “horizontal scalability”, so can further scale out.

5 REAL-WORLD APPLICATIONS

Solving assignment problems at scale is crucial for many real world businesses. We developed our distributed ADMM solver based on the AntOpt platform [33], and have made it in production at Alipay\(^1\) since mid 2021. It has been used to power crucial decisions such as allocation and reallocation of several hundred million users in loans, capital management of financial institutes, portfolio management, and marketing, etc. Our billion-scale ADMM solver has supported running of both daily and on-demand tasks in our business.

5.1 Financial Product Marketing

In this section, we describe a concrete application in marketing. There are $J$ equity funds that are managed by different owners. We aim to assign $J$ funds to $I$ users while each user can be exposed with at most one equity fund. The owners themself require that the population of the users to be assigned should satisfy specific demographics. That is, we are given $K$ groups of users $\{1, ..., k, ..., K\}$ that each group defines an exclusive set of users. For example, group $k = 1$ denotes a set of users who are sensitive to the risk of equity funds and contribute to high volume and open interest, and group $k = 2$ denotes a set of users who are sensitive to the risk but contribute low volume and open interest, and so on so forth. In reality, each owner $j$ would specify a desired population of users denoted as the distribution $q_{j,k}$. We are required to make the assignment to satisfy each owner’s desired population. Typically we use population stability index (PSI) \(^2\) to measure the matching of the population, i.e., $\sum_{k=1}^{K} (p_{j,k} - q_{j,k}) \ln \frac{p_{j,k}}{q_{j,k}}$, where we denote $p_{j,k}$ as the population as we do the assignment for each owner $j$. We aim to make the assignment to minimize the PSI of each owner. Besides, each owner has an inventory of supply that is prepared

\(^1\)https://en.wikipedia.org/wiki/Alipay
\(^2\)PSI is an important metric to identify a shift in population for retail credit score-cards [16].
in advance. Our assignment should help meet the supply of each owner. As such, we have the following assignment problem:

\[
\min \sum_{j} \sum_{k} \left( \sum_{i \in I_k} c_{ij} x_{ij} \right) \frac{b_j}{q_{j,k}} - q_{j,k} \ln \left( \frac{b_j}{q_{j,k}} \right)
\]

\[
\text{s.t. } \sum_{i} c_{ij} x_{ij} = b_j, \ \forall j \quad \text{and} \quad \sum_{j} x_{ij} \leq 1, \ \forall i,
\]

where we denote \(c_{ij}\) as the probability of purchase of fund \(j\) by user \(i\), \(b_j\) as the inventory of fund \(j\), and \(I_k\) as the set of users in the \(k\)-th group. In practice, each day we assign around 1.1 hundred of million users to tens of funds to meet the demand of fund owners via notification in Alipay app. Note that the objective is nonseperable due to the log of a sum.

5.2 Results

5.2.1 Numeric Results. Following the case of financial product marketing, we have nearly 1.1 hundred of million users and 21 equity fund owners each day. We show the results of our algorithm for solving this real-world large scale problem in Figure 5. We first show the objectives w.r.t. real number solution and integer solution and their APD as our algorithm converges. It shows that the objective gap between integer solution and real number solution is large at the first few round of iterations but converges to 0.00231 as the real number solutions approximate to either 0 or 1. We also show the MAPD of equality constraint that the violation being consistently optimized up to 0.00081.

5.2.2 Deployment and Online Results. We first describe the deployment of our distributed solver, then present the online results. We demonstrate the overall workflow in Figure 4. We have fund owners and Alipay users. We allow fund owners to config parameters such as inventory of supply and desired population of users. The Alipay notification marketing system maintains the overall workflow. Note that the overall decision making is data-driven based, i.e., it relies on machine learning components to predict the scores such as the probability of purchase \(c_{ij}\). The system first requests the “ML scoring” component to retrieve the scores (i.e., coefficients of our assignment problem) for final decision making. The “ML scoring” component requests “Feature store” to access the necessary features. After returning the scores back to “Notification system”, the system sends such coefficients to our distributed solver (see details in Figure 1). After solving the assignment problem, the solver returns the decisions and finally completes the notification to Alipay users. We further have monitors to record the quality of our solutions.

The fund owners prepare the inventory and configure the numbers each day. They may also periodically adjust the desired population \(q_{j,k}\). Given those parameters, we solve the problem each day. Note that the lower objective value the better, and the lower mean equality violation (MAPD) the better. We report the average relative improvements w.r.t. the quality of our solution compared with a heuristic approach in 7 days in Table 7.

6 CONCLUSIONS

We introduced the formulation for a class of assignment problems. The formulation supports nonlinear objective, convex per-item constraints, and feature-owner cross constraints. It is more general than GAP, and covers many real-world business problems. In the era of big data, there are scenarios when solving such problems with billions of decision variables is essential to business. To achieve such scale, we adapted the Bregman ADMM framework to our problem and obtained a distributed algorithm. In experiments, we specified the full Bregman ADMM construction for two example problems, with quadratic and logarithmic objectives, respectively. Using synthetic data, we showed the correctness of our implementation, and scaled the solver to problems of one billion decision
variables. We deploy our distributed solver to production and show its application to business decision making in an internet scale.

REFERENCES

[1] Pasquale Avella, Maurizio Boccia, and Igor Vasilyev. 2013. A branch-and-cut algorithm for the multilevel generalized assignment problem. *IEEE Access* 1 (2013), 475–479.

[2] Dimitri P Bertsekas and John N Tsitsiklis. 1989. *Parallel and distributed computation: numerical methods*. Vol. 23. Prentice hall Englewood Cliffs, NJ.

[3] Stephen Boyd, Neal Parikh, and Eric Chu. 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Now Publishers Inc.*

[4] Dirk G Cattrysse and Luk N Van Wassenhove. 1992. A survey of algorithms for the generalized assignment problem. *European Journal of Operational Research* 60, 3 (1992), 260–272.

[5] Alberto Ceselli and Giovanni Righini. 2006. A branch-and-price algorithm for the multilevel generalized assignment problem. *Operations Research* 54, 6 (2006), 1172–1184.

[6] Dawei Cheng, Zhishun Niu, and Yiyi Zhang. 2020. Contagious chain risk rating for networked-guarantee loans. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2715–2723.

[7] Paul C Chu and John E Beasley. 1997. A genetic algorithm for the generalised assignment problem. *Computers & Operations Research* 24, 1 (1997), 17–23.

[8] Wei Deng, Ming-Jun Lai, Zhimin Peng, and Wotao Yin. 2017. Parallel multi-block ADMM with O (1/k) convergence. *Journal of Scientific Computing* 71, 2 (2017), 712–736.

[9] Wei Deng and Wotao Yin. 2016. On the global and linear convergence of the generalized alternating direction method of multipliers. *Journal of Scientific Computing* 66, 3 (2016), 889–916.

[10] Harald Feilh and Gunther R Raidl. 2004. An improved hybrid genetic algorithm for the generalized assignment problem. In *Proceedings of the 2004 ACM symposium on Applied computing*. 990–995.

[11] Marshall L Fisher, Ramchandran Jaikumar, and Luk N Van Wassenhove. 1986. A multiplier adjustment method for the generalized assignment problem. *Management Science* 32, 9 (1986), 1095–1103.

[12] Bezalel Gavish and Hasan Pirkul. 1991. Algorithms for the multi-resource generalized assignment problem. *Management Science* 37, 6 (1991), 695–713.

[13] Bingzheng He and Xiaoming Yuan. 2012. On the O(1/n) convergence rate of the Douglas–Rachford alternating direction method. *SIAM J. Numer. Anal.* 50, 2 (2012), 700–709.

[14] Mingyi Hong and Zhi-Quan Luo. 2017. On the linear convergence of the alternating direction method of multipliers. *Mathematical Programming* 162, 1-2 (2017), 163–199.

[15] Kurt Jörnsten and Mikael Ni berg. 1986. A new Lagrangian relaxation approach to the generalized assignment problem. *European Journal of Operational Research* 27, 3 (1986), 313–323.

[16] Grigorios Karakoulas. 2004. Empirical validation of retail credit-scoring models. *RMA Journal* 87 (2004), 56–60.

[17] Sven O Krumke and Clemens Thielen. 2013. The generalized assignment problem with minimum quantities. *European Journal of Operational Research* 228, 1 (2013), 46–55.

[18] Qinghua Liu, Xinyue Shen, and Yuantao Gu. 2019. Linearized admm for nonconvex nonsmooth optimization with convergence analysis. *IEEE Access* 7 (2019), 76331–7644.

[19] Ziqi Liu, Dong Wang, Qianyu Yu, Zhiqiang Zhang, Yue Shen, Jian Ma, Wenliang Zhong, Jinjie Gu, Jun Zhou, Shuang Yang, et al. 2019. Graph representation learning for merchant incentive optimization in mobile payment marketing. In *Proceedings of the 28th ACM Conference on Information and Knowledge Management*. 2577–2584.

[20] H Ramalhinho Lourenço and Daniel Serra. 1998. Adaptive approach heuristics for the generalized assignment problem. *IEEE-UPF, Economic Working paper Series* 288 (1998).

[21] Silvano Martello. 1990. *Knapsack problems: algorithms and computer implementations*. Wiley-Interscience series in discrete mathematics and optimization (1990).

[22] Silvano Martello et al. 1981. An algorithm for the generalized assignment problem. (1981).

[23] Ibrahim H Osman. 1995. Heuristics for the generalised assignment problem: simulated annealing and tabu search approaches. *Operations-Research-Spektrum* 17, 4 (1995), 211–225.

[24] Neal Parikh and Stephen Boyd. 2014. *Proximal algorithms*. *Foundations and Trends in optimization* 1, 3 (2014), 127–239.

[25] G Terry Ross and Richard M Soland. 1975. A branch and bound algorithm for the generalized assignment problem. *Mathematical programming* 8, 1 (1975), 91–103.

[26] Martin Savelsbergh. 1997. A branch-and-price algorithm for the generalized assignment problem. *Operations research* 45, 6 (1997), 831–841.

[27] Reza Takapoui, Nicholas Moehle, Stephen Boyd, and Alberto Bemporad. 2020. A simple effective heuristic for embedded mixed-integer quadratic programming. *Internat. J. Control* 93, 1 (2020), 2–12.

[28] Andreas Wächter and Lorenz T Biegler. 2006. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming* 106, 1 (2006), 25–57.

[29] Huahua Wang and Arindam Banerjee. 2013. Bregman alternating direction method of multipliers. *arXiv preprint arXiv:1306.3203* (2013).

[30] Yijun Wang, Le Wu, Zongda Wu, Enhong Chen, and Q. Liu. 2016. Selecting valuable customers for merchants in e-commerce platforms. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 1281–1286.

[31] Sheng Xiong, Yuqi Yang, and Kaizhan Huai. 2017. A new model of generalized assignment problem and its method. In *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. IEEE, 2379–2384.

[32] Li Yu, Zhengwei Wu, Tianchi Cai, Ziqi Liu, Zhiqiang Zhang, Lihong Gu, Xiaodong Zeng, and Jinjie Gu. 2021. Joint Incentive Optimization of Customer and Merchant in Mobile Payment Marketing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35, 15000–15007.

[33] Jun Zhou, Yang Bao, Hua Wu, and Zhigang Hua. 2021. AntOpt: A Multi-functional Large-scale Decision Optimization Platform. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 4833–4837.