


ON-DEMAND ORDER ASSIGNMENT IN THE RIDE-SHARING PROBLEM

A PREPRINT

 **Jiachen Wang**
SIST 2018533087
wangjch2@shanghaitech.edu.cn

 **Zhongling Xu**
SIST 2018533139
xuzhl2@shanghaitech.edu.cn

January 3, 2021

ABSTRACT

Order-assignment problem is an important part in real-world ride-sharing problem. In this paper, we consider the static order-assignment problem: assigning the packed orders (requests) to vehicles while reaching the minimum cost. We first formulate the above problem as an MIP (Mixed Integer Programming) problem, and propose greedy methods to accelerate the solving process. Also, we propose stochastic algorithms to solve it in parallel. Experiments are done on 120 real-world tasks, and our methods performs better than directly calling MIP solvers on both cost and time.

*code is available at https://github.com/Peppacat/2020_numerical_optimization_project

Keywords Ride-sharing · Order-assignment · Mixed Integer Programming · Greedy algorithm

1 Introduction

Ride sharing services are transforming urban mobility by providing timely and convenient transportation to anybody, anywhere, anytime. These services present enormous potential for positive social impacts with respect to pollution, energy consumption, congestion, etc. A special problem of interest is the order-assignment problem. At any time stamp, an online decision system is responsible for assigning some requests from users to some agents. The problem is more complex in the ride-sharing context, as some request may be first packed together and then be distributed. This setting also gives rise to a key feature: the cost of combining some requests may be much lower than the sum of the individuals.

Many researchers have tried to solve these problems. One paper Alonso-Mora et al. [2017] published in 2017 formulated the above static problem as a Mixed Integer Programming problem. Another paper Qin et al. [2019] used Deep Reinforcement Learning to solve the dynamic version of the above problem (modeling the order-assignment problem in the continuous time).

In our project, we tried to solve the static version. We assume that the requests have already been packed into trips by some ad-hoc rules. Then given the requests, trips and vehicles, our goal is to find a distribution strategy such that each request is assigned to exactly one vehicle while reaching the minimum total cost.

2 Problem Formulation

We model the above problem with a graph and explicitly give the objective functions and constraints.

In each task, we are given a set of requests $\mathcal{R} = \{r_1, \dots, r_n\}$, and a set of vehicles $\mathcal{V} = \{v_1, \dots, v_m\}$, and the set of trips $\mathcal{T} = t_1, \dots, t_q$ where $t_s = \{r_i, r_j\}$ or $\{r_i\} \subset \mathcal{R}$. We then construct two graphs with the above information. The first is an undirected graph called "RT-graph", which links the requests and trips. The vertices $V = R \cup T$, and $E = \{e_{i,j} | r_i \in t_j, \forall r_i \in R, t_j \in T\}$. In brief, there exist an edge $e_{i,j}$ in the RT-graph if request i is contained in the trip j (by some ad-hoc rules).

The second graph is an weighted undirected graph called the "TV-graph", which links the trips and vehicles. The vertices $V = T \cup V$, and $E = \{e_{i,j} | v_j \text{ is accessible to } t_i, \forall v_j \in V, t_i \in T\}$. In brief, there exists an edge $e_{i,j}$ in TV-graph if it is possible to assign (possible but not necessary) the trip i to vehicle j . The weights $c_{i,j}$ indicates the cost of assigning trip t_i to vehicle v_j .

Combine the above two graphs, we have the RTV-graph, as illustrated below. Recall that our goal is to find the optimal assignment of vehicles to trips. We also take the following assumptions:

1. Each vehicle can be assigned with at most 1 trip. It is possible for a vehicle to be idle.
2. Each request should only be assigned to one vehicle at one time, meaning that it can not appear in multiple trips that are served by any vehicles.

We then give the explicit optimization problem in the above context. Define binary variable: A binary variable $\epsilon_{i,j} \in \{0, 1\}$ is introduced for each edge $e(T_i, v_j)$ between a trip $T_i \in \mathcal{T}$ and a vehicle $v_j \in \mathcal{V}$ in the RTV-graph. If $\epsilon_{i,j} = 1$ then vehicle v_j is assigned to trip T_i . Denote by \mathcal{E}_{TV} the set of $\{i, j\}$ indexes for which an edge $e(T_i, v_j)$ exists in the RTV-graph.

An additional binary variable $\chi_k \in \{0, 1\}$ is introduced for each request $r_k \in \mathcal{R}$. These variables are active (i.e. $\chi_k = 1$) if the associated request r_k can not be served by any vehicle and is ignored.

Objective function:

Our goal is to optimize the following cost function:

$$\mathcal{C}(\mathcal{X}) := \sum_{i \in T, j \in V} c_{i,j} \epsilon_{i,j} + \sum_{k \in \{1, \dots, n\}} c_{ko} \chi_k$$

, where c_{ko} is an penalty for requests that fail to be served by any vehicles.

Constraints:

There are two type of constraints: First, each request should only appear once in all the trips that are served by any vehicles.

$$\sum_{i \in \mathcal{I}_j^V} \epsilon_{i,j} \leq 1 \quad \forall v_j \in \mathcal{V}$$

where \mathcal{I}_j^V denotes the indexes i for which an edge $e(T_i, v_j)$ exists in the RTV-graph.

Second, each vehicle should only be assigned with at most one trip:

$$\sum_{i \in \mathcal{I}_k^R} \sum_{j \in \mathcal{I}_i^T} \epsilon_{i,j} + \chi_k = 1 \quad \forall r_k \in \mathcal{R}$$

where \mathcal{I}_k^R denotes the indexes i for which an edge $e(r_k, T_i)$ exists in the RTV-graph and \mathcal{I}_i^T denotes the indexes j for which an edge $e(T_i, v_j)$ exists in the RTV-graph.

In conclusion, we have the following optimization problem:

$$\begin{aligned} \Sigma_{\text{optim}} &:= \arg \min_{\mathcal{X}} \mathcal{C}(\mathcal{X}) := \sum_{i \in T, j \in V} c_{i,j} \epsilon_{i,j} + \sum_{k \in \{1, \dots, n\}} c_{ko} \chi_k \\ \text{s.t.} \quad &\sum_{i \in \mathcal{I}_j^V} \epsilon_{i,j} \leq 1, \quad \forall v_j \in \mathcal{V} \\ &\sum_{i \in \mathcal{I}_k^R} \sum_{j \in \mathcal{I}_i^T} \epsilon_{i,j} + \chi_k = 1, \quad \forall r_k \in \mathcal{R} \end{aligned}$$

3 Method

We propose two methods to solve the above problem: mixed integer programming with greedy initialization and a Swarm (stochastic) algorithm based on simulated annealing.

3.1 MIP with greedy initialization

First, a greedy solution is computed, which serves as an initial point for the MIP optimization. Here we give two greedy initialization algorithm, a basic one and a modified one.

- *Greedy assignment 1*: Trips are assigned to vehicles iteratively in decreasing size of the trip and increasing cost. The idea is to maximize the amount of requests served while minimizing cost. The method to greedily maximize the cost function \mathcal{C} is described in **Algorithm1**.
- *Greedy assignment 2*: The modified greedy assignment is similar to the former one but differs in that the trips with two requests are assigned greedily first, then trips with one request are assigned greedily. Intuitively, this strategy should give better results than the former one in most cases since total cost can not be linearly added and trips with two requests usually require less cost than the cost of separately combining two requests. The modified method is described in **Algorithm2**.

Algorithm 1 Greedy assignment 1

```

 $\mathcal{R}_{ok} = \emptyset; \mathcal{V}_{ok} = \emptyset$ 
 $\mathcal{S}_k := \text{sort } e(T, v) \text{ in increasing cost, } \forall T \in \mathcal{T}, v \in \mathcal{V}$ 
while  $\mathcal{S}_k \neq \emptyset$  do
  pop  $e(T, v) \leftarrow \mathcal{S}_k$ 
  if  $\forall r \in \mathcal{T}, r \notin \mathcal{R}_{ok} \text{ and } v \notin \mathcal{V}_{ok}$  then
     $\mathcal{R}_{ok} \leftarrow \{\forall r \in T\}; \mathcal{V}_{ok} \leftarrow v$ 
     $\Sigma_{greedy} \leftarrow e(T, v)$ 
  end if
end while

```

Algorithm 2 Greedy assignment 2

```

 $\mathcal{R}_{ok} = \emptyset; \mathcal{V}_{ok} = \emptyset$ 
 $\mathcal{S}_{k1} := \text{sort } e(T_1, v) \text{ in increasing cost, } \forall T_1 \in \mathcal{T}, v \in \mathcal{V}, \text{ where } T_1 = \{t \mid |t| = 1\}$ 
 $\mathcal{S}_{k2} := \text{sort } e(T_2, v) \text{ in increasing cost, } \forall T_2 \in \mathcal{T}, v \in \mathcal{V}, \text{ where } T_2 = \{t \mid |t| = 2\}$ 
while  $\mathcal{S}_{k2} \neq \emptyset$  do
  pop  $e(T, v) \leftarrow \mathcal{S}_{k2}$ 
  if  $\forall r \in \mathcal{T}, r \notin \mathcal{R}_{ok} \text{ and } v \notin \mathcal{V}_{ok}$  then
     $\mathcal{R}_{ok} \leftarrow \{\forall r \in T\}; \mathcal{V}_{ok} \leftarrow v$ 
     $\Sigma_{greedy} \leftarrow e(T, v)$ 
  end if
end while
while  $\mathcal{S}_{k1} \neq \emptyset$  do
  pop  $e(T, v) \leftarrow \mathcal{S}_{k1}$ 
  if  $\forall r \in \mathcal{T}, r \notin \mathcal{R}_{ok} \text{ and } v \notin \mathcal{V}_{ok}$  then
     $\mathcal{R}_{ok} \leftarrow \{\forall r \in T\}; \mathcal{V}_{ok} \leftarrow v$ 
     $\Sigma_{greedy} \leftarrow e(T, v)$ 
  end if
end while

```

There might also be other variants of initialization, whether greedy or not. However, the reason why we apply the principle of greediness is that it can save huge amounts of computing time while satisfying the constraints at the same time. As an initial solution, greedy algorithm is desired since it is efficient and easy to understand.

3.2 simulated annealing

Simulated annealing belongs to heuristic algorithms. Though it still apply the principle of greediness, stochastic factor is introduced in the search procedure. When we generate new feasible solutions iteratively, a solution worse than the current one might be accepted with a certain probability. Therefore, chances are that we can jump out of the local optimal point and eventually reach the global optimal point.

Here we give a general procedure of simulated annealing in **Algorithm 3**.

Notations:

- $J(y)$: the evaluation function under state y

- $Y(i)$: the current state
- $Y(i + 1)$: the next state
- r : a factor used to control the speed of annealing
- T : the temperature of the system (the system should be at a high temperature initially)
- T_{min} : the lower limit of temperature (when T reaches T_{min} , the searching procedure should be stopped.)

Algorithm 3 Simulated Annealing

```

while  $T > T_{min}$  do
   $dE \leftarrow J(Y(i + 1)) - J(Y(i))$ 
  if  $dE \geq 0$  then
     $Y(i + 1) \leftarrow Y(i)$ 
  else
    if  $e^{\frac{dE}{T}} > \text{random}(0,1)$  then
       $Y(i + 1) \leftarrow Y(i)$ 
    end if
  end if
   $i \leftarrow i + 1$ 
end while

```

4 Experiments

For details on experiment scripts, please refer to readme.md in the code files

We test the following 5 algorithms on the 120 tasks:

1. MIP: directly use Gurobi MIP solver
2. greedy1: use the above greedy1 method
3. greedy2: use the above greedy2 method
4. MIP+greedy1: Gurobi MIP solver initialized with a guess from greedy1 method
5. MIP+greedy2: Gurobi MIP solver initialized with a guess from greedy2 method
6. (Simulated annealing): due to time limit, we did not implement it

For MIP solvers, we use Gurobi 9.1 with python API and set "model.Params.method = 4 (deterministic parallel)" which exploits all threads available on CPU (in our task: 8). For greedy method, we only implement the non-parallelism version.

We based on the following metrics to evaluate each method:

1. Infeasible: number of solutions given by each method that are infeasible (violate the constraints)
2. Time Out: number of solutions that are generated beyond 1.2s. We take this into account as in real-world problems, companies have to truncate the solving process in limited time.
3. Total Times: the total time cost in running all the tasks. We exclude those solutions that exceed time limit (1.2s)
4. Optimal Solutions: number of solutions that are optimal among all 5 methods and are generated in the time limit (1.2s)

5 Discussion

From the above table, we can see that MIP methods produce feasible solutions if it is possible, but it is not the case for greedy methods, as they did not guarantee the global optimal.

For the time consumed, we conclude that MIP itself is quite slow as it is dealing with an NP-hard problem. Greedy methods all work almost two times faster than MIP, but it will not give the optimal solutions in some cases as mentioned

method	Infeasible	Time Out ($\geq 1.2s$)	Total Time (s)	Optimal Solutions (in time limit)
MIP	1*	6	27.40	32
greedy1	13	0	12.56	0
greedy2	4	0	12.55*	84*
MIP+greedy1	1*	2	28.02	32
MIP+greedy2	1*	0*	24.37*	32

Table 1: Comparison of 5 methods

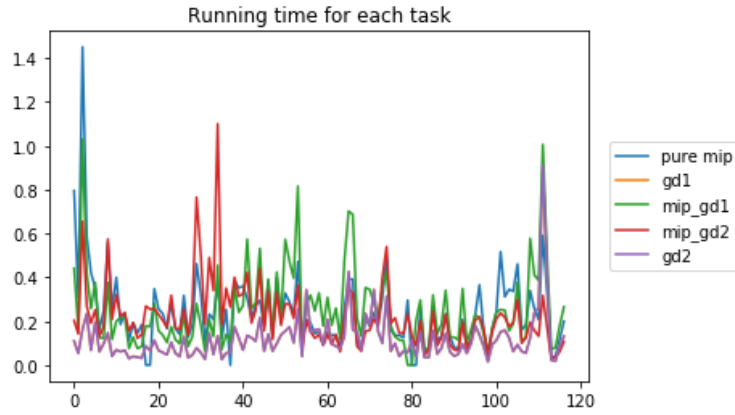


Figure 1: Time Consumed on each Task

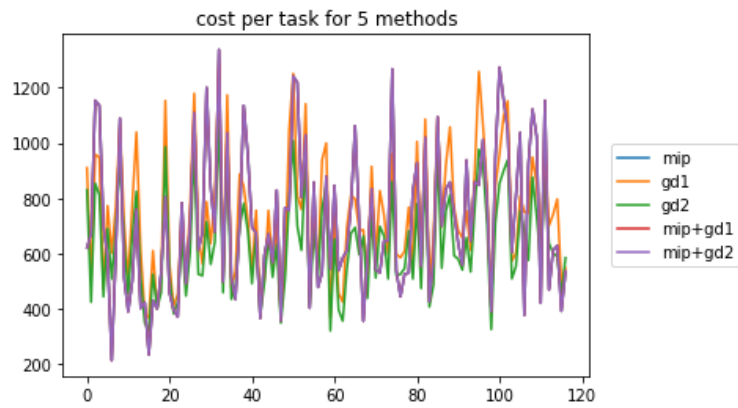


Figure 2: Cost for each Task

above. The method we propose to accelerate the MIP solving process with a initial guess from greedy methods do work! Moreover, we can see that the second greedy rule performs much better than the first one.

For the optimal solutions given in the time limit (1.2s), we find that greedy2 method performs the best. Although from 2, MIP methods give much lower cost than greedy-based methods, but they run much slower. Thus greedy2 method is acceptable in real industrial settings.

In conclusion, for real-world ride-sharing and order-assignment problems, it would be acceptable to use MIP solver with Proper initialization (greedy or some ad-hoc rules). Another branch of algorithms may also be applicable: swarm intelligent algorithms like simulated annealing (MCMC). These methods suffer from the slow-convergence problem (though typical variants of them may relatively faster, but still quite slow compared to deterministic algorithms). But they are a potential solution in medium-scale problems because they are easy to run in parallel and get linearly-acceleration.

References

- Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3):462–467, 2017.
- Zhiwei Tony Qin, Xiaocheng Tang, Yan Jiao, Fan Zhang, Chenxi Wang, and Qun Tracy Li. Deep reinforcement learning for ride-sharing dispatching and repositioning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 6566–6568. AAAI Press, 2019.