Alleviating Users' Pain of Waiting: Effective Task Grouping for Online-to-Offline Food Delivery Services

Shenggong Ji
Southwest Jiaotong University, China
shenggongji@163.com

Yu Zheng*
JD Urban Computing Business Unit, China
msyuzheng@outlook.com

Zhaoyuan Wang
Southwest Jiaotong University, China
wang_zhaoyuan@foxmail.com

Tianrui Li*
Southwest Jiaotong University, China
trli@swjtu.edu.cn

ABSTRACT

Ordering take-out food (a.k.a. takeaway food) on online-to-offline (O2O) food ordering and delivery platforms is becoming a new lifestyle for people living in big cities, thanks to its great convenience. Web users and mobile device users can order take-out food (i.e. obtain *online food ordering services*) on an O2O platform. Then the O2O platform will dispatch food carriers to deliver food from restaurants to users, i.e. providing users with *offline food delivery services*. For an O2O food ordering and delivery platform, improving food delivery efficiency, given the massive number of food orders each day and the limited number of food carriers, is of paramount importance to reducing the length of time users wait for their food. Thus, in this paper, we study the food delivery task grouping problem so as to improve food delivery efficiency and alleviate the pain of waiting for users, which to the best of our knowledge has not been studied yet. However, the food delivery task grouping problem is challenging, given two reasons. First, the food delivery efficiency is affected by multiple factors, which are non-trivial to formulate and jointly consider. Second, the problem is a typical NP-hard problem and to find near-optimal grouping results is not easy. To address these two issues, we propose an effective task grouping method. On one hand, we provide formal formulations for the factors affecting the food delivery efficiency, and provide an objective to organically combine these factors such that it can better guide the task grouping. On the other hand, we propose heuristic algorithms to efficiently obtain effective task grouping results, consisting of a greedy algorithm and a replacement algorithm. We evaluate our task grouping method using take-out food order data from web users and mobile device users on a real-world O2O food ordering and delivery platform. Experiment results demonstrate that our task grouping method can save ∼16% (87 seconds) of average waiting time for each user, comparing with many baseline methods. It indicates that our method is able to significantly improve the food delivery efficiency and can provide better food delivery services for users.

*Yu Zheng and Tianrui Li are the correspondence authors of this paper. Yu Zheng is also with JD Intelligent City Research, China, and Xidian University, China.

CCS CONCEPTS

• Information systems → Spatial-temporal systems; **Data mining**; **• Human-centered computing → Ubiquitous and mobile computing**;

KEYWORDS

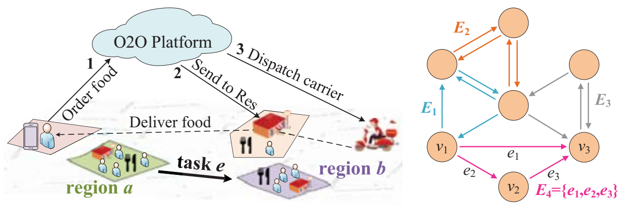
Mobile applications, O2O food delivery services, task grouping, graph edge partition, optimization

ACM Reference Format:
Shenggong Ji, Yu Zheng, Zhaoyuan Wang, and Tianrui Li. 2019. Alleviating Users' Pain of Waiting: Effective Task Grouping for Online-to-Offline Food Delivery Services. In *Proceedings of the 2019 World Wide Web Conference (WWW '19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3308558.3313464

1 INTRODUCTION

Advance in modern web and mobile device technologies is spurring the prevalence of ordering take-out food (takeaway food) on online-to-offline (O2O) food ordering and delivery platforms (O2O platforms for short). As shown in Figure 1(a), web users or mobile device users can order take-out food on an O2O platform (step 1), through a web or an App. After receiving the food orders, the O2O platform will send them to corresponding restaurants (step 2), and will dispatch food carriers to deliver the food from restaurants to users (step 3), i.e. providing offline food delivery services for users. Thus, users are not necessary to go to restaurants in person, just waiting and enjoying the delivery of food from food carriers. Thanks to its great convenience, there are massive people ordering take-out food on O2O platforms every day. For example, in China, the number of daily food orders from users on one O2O platform has already reached 10 million [3, 4]. As a result, for O2O food ordering and delivery platforms, given the limited number of food carriers and the massive number of users each day, how to improve the food delivery efficiency such that the waiting time of the massive users can be reduced has become an urgent problem.

A feasible way to improve the food delivery efficiency is grouping delivery tasks in a city into groups (i.e. the task grouping problem). One delivery task is not one specific food order from a user, but all food orders from a region to another. For the example in Figure 1(a), there is a delivery task e from region a to region b, since there exist users in region b ordering food from restaurants in region a. A region contains many users and restaurants. If we see regions as vertices and delivery tasks as directed edges, we can construct a delivery task graph, like the one in Figure 1(b). In Figure 1(b), tasks (edges) are grouped into four task groups, i.e. E_1, E_2, E_3 and E_4, which are highlighted using different colors. A food carrier will be assigned to only one task group such that he/she can be very familiar with the locations of users and restaurants in the task group. In other words, by task grouping, food carriers in each task group can avoid wasting time on finding the locations of users and restaurants, and the routes between them, thus improving the food delivery efficiency and reducing users' waiting time.



(a) Online-to-Offline Food Delivery Services (b) Task Graph

Figure 1: Food delivery task grouping problem.

It should be noted that the task grouping problem is not a real-time problem. For an O2O food ordering and delivery platform, delivery tasks are grouped just from time to time, e.g. once per month. For example, the task grouping result of this month can be obtained based on users' food orders in the last few months. Having the task grouping result, we can allocate food carriers to each task group and the food orders (arriving in real time) can be delivered by food carriers in corresponding task group. Clearly, *the food carrier allocation problem* (i.e. how many food carriers are needed for each task group?) and *the real-time food carrier dispatching problem* (i.e. which food carrier should be dispatched when a food order arrives in real time?) also significantly affect the food delivery efficiency. In this work, we focus on the task grouping problem since it is the foundation of the allocation and dispatching problem. These two problems will be studied in our future work.

In fact, our task grouping problem can be seen as a graph edge partition problem, i.e. a graph vertex-cut problem [9, 19, 30]. However, previous graph edge partition algorithms (heuristic algorithms mainly) are designed for distributed graph computation platforms [9, 14, 15], whose aim is to minimize the number of total replicated vertices over different edge groups. Therefore, due to the different scenarios and the different objectives, previous graph edge partition algorithms cannot work well to our task grouping problem. As a consequence, a new task grouping method is in great need for the O2O food ordering and delivery platforms.

However, our task grouping problem is challenging, due to the following two reasons.

First, to improve the food delivery efficiency, our task grouping should consider multiple factors, e.g. the *shareability* and the *empty run time* of each task group. Besides the joint consideration of these two factors, to mathematically formulate and derive the shareability and the empty run time is also non-trivial. Specifically, the *shareability* indicates to what extent food orders in a task group can be shared (jointly delivered). The *empty run time* refers to the empty cruising time before carriers deliver next food orders. For example, the task group E_4 marked by the pink color in Figure 1(b) may have high shareability, since carriers can jointly deliver orders in task e_1 together with orders in task e_2 or e_3, saving the total delivery time, if these orders come at close time. However, the task group has high empty run time. For instance, after a food carrier delivers an order to region v_3, if the next food order is in task e_1 (or e_2), the food carrier needs to go back to region v_1, *in empty*, to deliver the food order. Apparently, to improve the food delivery efficiency, we expect task groups with *high* shareability and *low* empty run time.

Second, the task grouping problem can be easily proved to be a NP-hard problem [9, 19, 30]. As a result, it is also non-trivial to design an effective and efficient algorithm to obtain near-optimal task grouping solutions.

To address these issues, in this paper, we propose a method to do the task grouping for O2O food ordering and delivery platforms. Specifically, our contributions are as below.

- To the best of our knowledge, we are the first to study the delivery task grouping problem for O2O food ordering and delivery platforms in real world. We formally define the food delivery task grouping problem.
- We provide formal formulations for the shareability and the empty run time. Besides, we propose a method to organically incorporate the shareability and the empty run time into one objective such that it can better guide our task grouping.
- We propose effective heuristic algorithms to solve the task grouping problem, including a greedy algorithm and a replacement algorithm. The greedy algorithm is to output a fast and good task grouping solution, while the replacement algorithm is to further refine the obtained grouping solution.
- We evaluate our task grouping method using the food order data from web users and mobile device users on an O2O food ordering and delivery platform in real world. Compared with previous graph edge partition methods, our task grouping method is able to save at least 16% (87 seconds) of average travel time for each food order, significantly reducing the waiting time of users.

2 OVERVIEW

2.1 Preliminary

Definition 1 (*Region*): A region v is a small piece of a city, containing some restaurants and users. Note that considering each individual restaurant (or user) is not necessary, given that some restaurants (or users) are geographically very close and thus can be seen as a unity. A city is comprised of many regions and we denote all regions in a city by V. Regions V in a city are obtained by region segmentation using road networks, which will be detailed later in Section 3.1. An example for the obtained regions V in a city can be found in Figure 8(b).

Definition 2 (*Delivery Task*): A delivery task, denoted by $e = (o, d, n^1, \cdots, n^P)$, is a directed edge from region o to region d, containing all food orders from restaurants of region o to users of region d. $n^p, p = 1, \cdots, P$ denotes the number of food orders in the delivery task in time period p, as demonstrated in Figure 2. A period can be with 15 minutes. P is the number of total time periods under consideration for the task grouping (e.g. one month). We can denote all delivery tasks in a city by E. The extraction of all delivery tasks E is based on the food orders of users in the city, which will be detailed in Section 3.2.

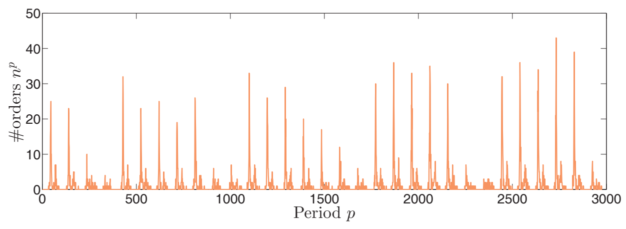


Figure 2: An example for a food delivery task's $n^1, \cdots, n^P$.

Different with the edges on distributed graph computation platforms [9, 14, 15], our edges (tasks) contain clear spatial and temporal properties. Specifically, o and d refer to the spatial locations of a task, while $n^1, \cdots, n^P$ denote the temporal frequency of food orders (from web users and mobile device users) from region o to region d (Figure 2). Thus, the task grouping problem in our scenario can be more difficult.

Definition 3 (*Task Graph*): A task graph is a directed graph, denoted by $G = \langle V, E\rangle$, with all regions V as vertices and all delivery tasks E as directed edges. G is naturally obtained once V and E have been extracted.

2.2 Problem Definition

M-task grouping problem: given a constructed task graph $G = \langle V, E\rangle$ and the number M of task groups, the task grouping problem aims to group all delivery tasks in E into M groups, i.e. $E_m, m = 1, \cdots, M$, such that the following conditions (or objectives) can be satisfied:

(1) Each task group E_m is a subset of E, i.e. $E_m \subset E, \forall m$;
(2) All tasks are grouped into the M groups, i.e. $\bigcup_{m=1}^{M} E_m = E$;
(3) Task groups do not overlap, i.e. $E_{m_1} \cap E_{m_2} = \emptyset, \forall m_1 \neq m_2$;
(4) Each task group E_m has high shareability;
(5) Each task group E_m has low empty run time.

As mentioned in the Introduction, our task grouping is not conducted in real time, but just from time to time. For the examples in Figure 3, the task grouping is conducted every month. Actually, since the distribution of food orders won't change dramatically, the task grouping can even be conducted every quarter/half of a year. In Figure 3(a), the task groups in October are obtained based on the task graph constructed using food order data in the previous three months, i.e. July, August and September. The obtained task groups are applied in October, i.e. each arriving food order are delivered by a food carrier in the corresponding task group. That is, the food carrier allocation and real-time dispatching (defined in the Introduction) in October are based on the obtained task groups. Similarly, in Figure 3(b), at the beginning of November, the task grouping is re-conducted based on the task graph constructed using food order data in August, September and October.

Our task grouping problem is actually a graph edge partition problem, which has already been proved to be NP-hard [6, 30]. As mentioned before, due to the spatial and temporal properties of the edges (tasks) in our task graph and the different objectives 4 and 5, our task grouping problem can be more difficult.

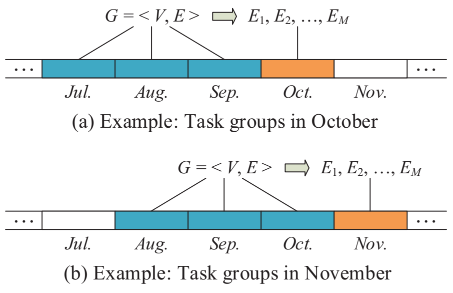


Figure 3: Examples for task grouping.

2.3 Framework

Figure 4 demonstrates the framework of our task grouping method, consisting of three components: pre-processing, shareability and empty run time, and task grouping algorithm. Below, we introduce each component in detail.

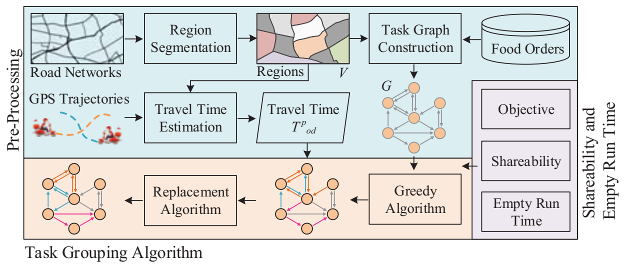


Figure 4: The framework of our task grouping method.

As shown in Figure 4, the pre-processing component has three small components: region segmentation, task graph construction, and travel time estimation. Firstly, the region segmentation is to segment a city into non-overlapping regions V. In detail, a city is segmented based on the road structure of the city (i.e. road networks), e.g. Figure 8(a, b). Next, with the segmented regions V, we construct the task graph $G = \langle V, E\rangle$. The construction process uses the historical food order records collected from web users and mobile device users in real world. From these food order records, we can know whether there are food orders from one region to another and the number of the food orders in each time period. That is, all delivery tasks E between any two regions can be extracted. Then, based on V and E, task graph G can be constructed. The constructed task graph G will be transferred to the task grouping algorithm component for grouping. The third small component is the travel time estimation, which is to estimate the travel time of food carriers between any two regions in V in each time period p. The estimation is based on the GPS trajectories of food carriers collected from real world. The estimated travel time are also necessary in the task grouping algorithm component.

The shareability and empty run time component is to formally formulate the shareability and the empty run time, which are then combined as one objective. The objective will be used to guide the grouping of tasks. In general, given a grouping result, we utilize the total delivery time of food orders to indicate the shareability of the grouping result. The smaller the total delivery time, the higher shareability the grouping result. Thus, our objective is formulated as the sum of the delivery time and the empty run time. The delivery time and the empty run time of a grouping result will consider the number of food orders in each task in each period.

The task grouping algorithm component details our algorithms for grouping tasks into M task groups, including a greedy algorithm and a replacement algorithm. The greedy algorithm assigns tasks to each task group E_m in a greedy manner based on the objective, and will output a task grouping result $E_1, \cdots, E_m$. The replacement algorithm is to further improve the grouping result of the greedy algorithm. Specifically, we try to replace the owner (one task group) of a task with another task group, with the expectation that the total delivery time and the total empty run time can be reduced after the replacement. The replacement algorithm stops when the objective cannot be improved any more, and then our ultimate task grouping result is returned.

3 PRE-PROCESSING

3.1 Region Segmentation

In this subsection, we segment a city into non-overlapping regions V, which includes two steps. At the first step, we segment the city into regions according to the structure of road networks, using the region segmentation method proposed in [29]. By this method, the segmented regions have semantic meanings, e.g. a region can be a small block of a city. This is an important advantage of this segmentation method over the grid-based method [13, 20]. However, since road networks in a city are too dense, some regions obtained from the first step are very small, e.g. the widths and lengths of some regions are less than 200 meters. Therefore, at the second step, we recursively merge those small and geographically adjacent regions into relatively big regions, until the size of the smallest region reaches a threshold (e.g. 400 meters $\times$ 400 meters).

3.2 Task Graph Construction

With segmented regions V, we can construct the task graph based on users' food order records over a long time in real world. Specifically, for each food order (see the data format in the Section 6), we can get the location of the restaurant and the location of the user, which can be mapped into regions. For example, if the restaurant is in region o and the user is in region d, we have a task $e = (o, d, n^1, \cdots, n^P)$. Iterating over the entire food order records, we can obtain all tasks E. Besides the location information, we also have the time stamp at which a food order is ordered, which can be mapped to a corresponding time period. Thus, for each task $e \in E$, we can get the number of food orders in each period p, i.e. the $e.n^p$. Finally, based on the constructed V and E, we can then obtain the task graph $G = \langle V, E \rangle$ in a city.

3.3 Travel Time Estimation

In this subsection, we estimate the travel time $T^p_{o,d}$ of food carriers traveling from region $o \in V$ to region $d \in V$ in each period p, using the GPS trajectories of food carriers in real world. Since a period is too short (e.g. 15 minutes), we estimate the travel time from o to d in each hour, based on which we can get $T^p_{o,d}$. In detail, our estimation process has the following steps. First, we detect and remove stay points from the GPS trajectories of food carriers, using the stay-point detection algorithm proposed in [32]. In real world, food carriers may stay still to wait for users to fetch their food, or to wait for restaurants to prepare the food. Thus, we need to remove these stay points, before estimating the travel time of food carriers. Second, we map the GPS trajectories (without stay points) onto road segments, using the map-matching algorithm proposed in [28]. After the map-matching, we can obtain the travel time of carriers on each road segment. Note that there may exist some missing data, we can fill them based on the spatial and temporal correlation of travel time between road segments [8, 10, 26, 31]. Finally, the travel time between any two regions can be estimated as the minimum travel time of a path (a sequence of road segments), concatenating the centers of these two regions [26]. Note that the estimated travel time is necessary for the calculation of the shareability and the empty run time, when we conduct the task grouping.

4 SHAREABILITY AND EMPTY RUN TIME

4.1 Objective

The shareability of a grouping result $E_1, \cdots, E_M$ can be indicated by the total delivery time needed to deliver all food orders in all tasks. The smaller the total delivery time, the higher the shareability. For a simple example in Figure 5, there are four tasks. Tasks e_1, e_2 can be easily shared since their origins and destinations are both highly close, and food orders in e_1, e_2 have a similar arrival pattern (Figure 5(c)). That is, food orders in e_1, e_2 that come in the same time period can be delivered jointly, using just around one share of delivery time. So can tasks e_3, e_4. Thus, if we group e_1, e_2 as a task group and e_3, e_4 as another group, as Figure 5(a), the shareability of the grouping is high. In the meantime, the total delivery time to deliver all food orders is small, since jointly delivering food orders in e_1, e_2 (or e_3, e_4) can save a lot of delivery time. In contrast, as Figure 5(b), if we group e_1, e_3 as a group and e_2, e_4 as another group, the shareability of the grouping is low. Meanwhile, the total delivery time is large, since jointly delivering food orders in e_1, e_3 (or e_2, e_4) cannot save the total delivery time. Therefore, we can use the total delivery time to indicate the shareability. It should be noted that for a task group, in addition to the tasks' origins and destinations, the shareability is also affected by tasks' number of food orders in each time period (Figure 5(c)). The formal formulation of the shareability is in Section 4.2.

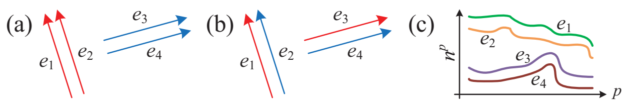


Figure 5: An example for the shareability.

The total delivery time for delivering all food orders is denoted by $DT(E_1, \cdots, E_M)$ and is actually the aggregation of delivery time in each task group E_m, i.e.

$$DT(E_1, \cdots, E_M) = \sum_{m=1}^{M} DT(E_m), \tag{1}$$

where $DT(E_m)$ refers to the total delivery time needed to deliver food orders in task group E_m.

The total empty run time of a grouping result $E_1, \cdots, E_M$ is denoted by $ET(E_1, \cdots, E_M)$, which is the aggregation of empty run time in each task group E_m:

$$ET(E_1, \cdots, E_M) = \sum_{m=1}^{M} ET(E_m), \tag{2}$$

where $ET(E_m)$ denotes the empty run time of task group E_m.

For our task grouping problem, we expect a grouping result with smaller delivery time (i.e. higher shareability) and smaller empty run time. Therefore, we can combine them as one objective, i.e. the total travel time, which is the sum of the total delivery time and the total empty run time. Thus, the total travel time $TT(E_1, \cdots, E_M)$ is

$$TT(E_1, \cdots, E_M) = \sum_{m=1}^{M} TT(E_m), \tag{3}$$

where $TT(E_m)$ is the aggregation of $DT(E_m)$ and $ET(E_m)$:

$$TT(E_m) = DT(E_m) + ET(E_m). \tag{4}$$

To obtain our objective function in Equation 3, we need to mathematically formulate the delivery time (shareability) $DT(E_m)$ and the empty run time $ET(E_m)$ in each task group E_m. Thus, for the rest of this section, we detail how to mathematically formulate the delivery time $DT(E_m)$ and the empty run time $ET(E_m)$ for each task group m.

4.2 Delivery Time $DT(E_m)$ (Shareability)

When calculating the delivery time of food orders in a task group E_m, we will consider jointly delivering (sharing) food orders, such that the delivery time can denote the shareability. However, in real world, only food orders that arrive within a time period (e.g. 15 minutes) can be shared, since jointly delivering food orders in different time periods would let users wait too long for their food. That is, jointly delivering food in different time periods is not user-friendly and will lead to poor user experience. Thus, the $DT(E_m)$ is actually the aggregation of the delivery time of food orders in each time period p. That is, $DT(E_m)$ is

$$DT(E_m) = \sum_{p=1}^{P} DT(E_m, p), \tag{5}$$

where P denotes the number of total time periods, and $DT(E_m, p)$ refers to the delivery time of food orders in task group E_m in period p. A period considered in this paper is with 15 minutes.

For food orders in a task group E_m in a period p, there may exist many strategies to jointly deliver them. For example, as shown in Figure 6, there are three tasks in task group E_m, i.e. e_1, e_2, e_3. In period p, e_1 is with one order, e_2 with two and e_3 with one. The possible joint delivery solutions include: 1) jointly delivering the order in e_1 with an order in e_2 and jointly delivering another order in e_2 with the order in e_3; 2) jointly delivering the order in e_1 with the order in e_3 and jointly delivering the two orders in e_2; etc. The problem now is to find a delivery solution such that it can be used for the calculation of our needed delivery time.

Definition 4 (*Delivery Time*): The delivery time $DT(E_m, p)$ is defined as the minimum delivery time among all possible delivery

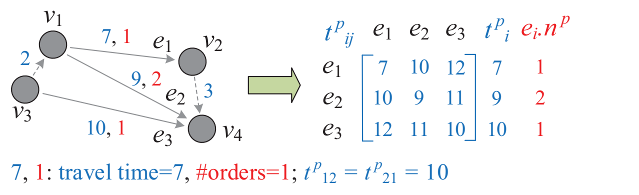

v_1
v_2
v_3
v_4
2
7, 1
9, 2
3
10, 1
e_1
e_2
e_3
$t^p{}_{ij}$ e_1 e_2 e_3 $t^p{}_i$ $e_i.n^p$
e_1 7 10 12 7 1
e_2 10 9 11 9 2
e_3 12 11 10 10 1
7, 1: travel time=7, #orders=1; $t^p{}_{12} = t^p{}_{21} = 10$

Figure 6: A simple example for delivery time $DT(E_m, p)$.

solutions such that it can indicate the shareability. As a consequence, to obtain the $DT(E_m, p)$, we need to find the optimal delivery solution. In math, it is

$$DT(E_m, p) = \min_{y_i, x_{ij}} \sum_{i=1}^{|E_m|} \left(y_i \cdot t_i^p + \sum_{j=i}^{|E_m|} x_{ij} \cdot t_{ij}^p \right)$$
$$s.t. \begin{cases} 1 \cdot y_i + \left(2 \cdot x_{ii} + 1 \cdot \sum_{j=1, j \neq i}^{|E_m|} x_{ij}\right) = e_i.n^p, \forall i \\ x_{ij} = x_{ji}, \forall i, j \\ y_i, x_{ij} \in \mathbb{Z}^+ \cup \{0\}, \forall i, j \end{cases} \tag{6}$$

y_i and x_{ij} are the decision variables. y_i refers to the number of food orders in task e_i being delivered *individually*. x_{ij} refers to the number of food orders in task e_i and e_j that are delivered jointly. t_i^p denotes the travel time needed for delivering an order in task e_i, i.e. $t_i^p = T^p_{e_i.o, e_i.d}$. For example, in Figure 6, $t_1^p = T^p_{v_1, v_2} = 7$. t_{ij}^p is the minimum travel time needed to jointly deliver an order in e_i and an order in e_j, and $t_{ij}^p = t_{ji}^p$ holds. For example, in Figure 6, $t_{12}^p = t_{21}^p = 10$ means that the minimum travel time to jointly deliver an order in e_1 and an order in e_2 is 10 minutes (by taking the route $v_1 \rightarrow v_2 \rightarrow v_4$, i.e. picking up two orders in region v_1 and then delivering the one to region v_2 and another one to v_4). $e_i.n^p$ denotes the number of orders in e_i in period p. $|E_m|$ denotes the number of tasks in group E_m.

In this optimization problem, the objective is to find the optimal delivery solution, i.e. y_i and x_{ij}, with the minimum delivery time. The first constraint requires that all orders in each task e_i should be delivered. Note that if $x_{ii} = 1$, two orders in task e_i are delivered (denoted by $2 \cdot x_{ii}$). The second constraint indicates that the sharing of orders between two tasks is symmetric. The last constraint means that y_i, x_{ij} are nonnegative integers.

From the optimization problem 6, we can find that the shareability of a task group is affected both by tasks' origins and destinations and by tasks' number of food orders in a time period, as aforementioned. The more tasks having close origins and destinations, the higher the shareability. The more food orders that come at the same time period, the higher the shareability.

The optimization problem can be efficiently solved using some off-the-shelf optimization solvers, e.g. Gurobi Optimizer [1], Microsoft Solver Foundation [2], etc. For the example in Figure 6, the solution is $x_{21} = x_{12} = 1, x_{23} = x_{32} = 1$ and thus $DT(E_m, p) = (T^p_{v_1, v_2} + T^p_{v_2, v_4}) + (T^p_{v_3, v_1} + T^p_{v_1, v_4}) = 21$. That is, the optimal delivery solution is to jointly deliver an order in e_2 with the order in e_1 (taking route $v_1 \rightarrow v_2 \rightarrow v_4$) and to jointly deliver another order in e_2 with the order in e_3 (taking route $v_3 \rightarrow v_1 \rightarrow v_4$).

4.3 Empty Run Time $ET(E_m)$

The empty run time for task group E_m is denoted by $ET(E_m)$. Similar to $DT(E_m)$, $ET(E_m)$ is also the aggregation of the empty run time $ET(E_m, p)$ in each period p, i.e.

$$ET(E_m) = \sum_{p=1}^{P} ET(E_m, p). \tag{7}$$

The empty run time $ET(E_m, p)$ is the total time intervals between the delivery units in period p.

Definition 5 (*Delivery Unit*): A delivery unit contains one food order in task e_1^s and one food order in e_2^s, and is denoted by $du = (e_1^s, e_2^s, o, d)$, where o and d denote the origin and destination of the delivery unit, respectively. For the example in Figure 6, the solution for $DT(E_m, p)$ contains two delivery units. The first unit is to jointly deliver an order in e_1 and e_2 by taking the route $v_1 \to v_2 \to v_4$. Therefore, $du_1 = (e_1, e_2, v_1, v_4)$. The second unit is to jointly deliver an order in e_2 and e_3 by taking the route $v_3 \to v_1 \to v_4$, that is, $du_2 = (e_2, e_3, v_3, v_4)$. Note that if a delivery unit contains only one food order, e_2^s is null and we denote $e_2^s = *$.

After a carrier finishes a delivery unit du_i, to deliver next delivery unit du_j, the carrier needs to spend some time traveling from du_i's destination $du_i.d$ to du_j's origin $du_j.o$, *in empty* (without carring any food). That is, the travel time from $du_i.d$ to $du_j.o$ is the empty run time. For example, as the first row in Figure 7, from delivery unit du_1 to du_2, there is empty run time $T^p_{v_4,v_3}$.

Definition 6 (*Empty Run Time*): Given the last delivery unit du_0 in period $p-1$ and the n delivery units $du_1, \cdots, du_n$ obtained from the solution for $DT(E_m, p)$ in this period p, the empty run time $ET(E_m, p)$ is the minimum time needed to travel between delivery units $du_1, \cdots, du_n$, starting from du_0. In math, it is

$$ET(E_m, p) = \min_{(i_1, i_2, \cdots, i_n) \in \mathrm{Perm}(1,2,\cdots,n)} \sum_{j=0}^{n} T^p_{du_{i_j}.d,\, du_{i_{j+1}}.o} \tag{8}$$

where $\mathrm{Perm}(1, 2, \cdots, n)$ denotes all permutations of set $\{1, 2, \cdots, n\}$ and $i_0 = 0$. $T^p_{du_{i_j}.d,\, du_{i_{j+1}}.o}$ is the travel time from du_{i_j}'s destination $du_{i_j}.d$ to $du_{i_{j+1}}$'s origin $du_{i_{j+1}}.o$, i.e. the empty run time between the i_j-th delivery unit and i_{j+1}-th delivery unit.

In summary, the delivery time $DT(E_m, p)$ is the total travel time of all delivery units in period p, while the empty run time $ET(E_m, p)$ is the minimum travel time needed to travel between all delivery units. For example, for the delivery units obtained in Figure 6, i.e. the $du_1 = (e_1, e_2, v_1, v_4)$ and $du_2 = (e_2, e_3, v_3, v_4)$ in Figure 7, $DT(E_m, p) = (T^p_{v_1,v_2} + T^p_{v_2,v_4}) + (T^p_{v_3,v_1} + T^p_{v_1,v_4})$. Assume that the last delivery unit in period $p-1$ is $du_0 = (e_1, *, v_1, v_2)$. We have $\mathrm{Perm}(1, 2) = \{(1, 2), (2, 1)\}$, corresponding to the two rows in Figure 7. If the optimal solution is $(i_1, i_2) = (2, 1)$, i.e. the second row, then $ET(E_m, p) = T^p_{v_2,v_3} + T^p_{v_4,v_1}$.

The optimization problem 8 is a NP-hard problem. In fact, the problem can be seen as a traditional traveling salesman problem [21], which is to find the shortest route to visit a given list of cities. Specifically, a delivery unit in our problem can be seen as a city in the traveling salesman problem and the distance between two cities is the travel time from the destination of a delivery unit to the origin of another delivery unit. Thus, to solve the NP-hard optimization problem 8, we can apply an efficient and effective nearest neighbour

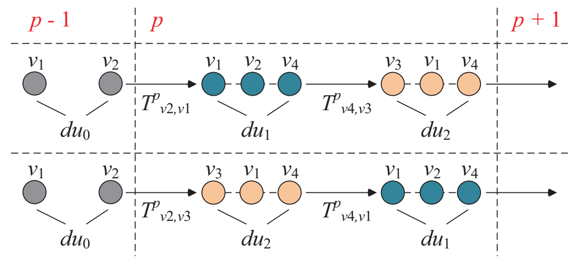


Figure 7: A simple example for empty run time $ET(E_m, p)$.

algorithm [11, 21, 22], which has been widely used in the traveling salesman problem. Specifically, assuming the current delivery unit is du_i, the algorithm selects the nearest (in terms of travel time) unvisited delivery unit du_j as the next delivery unit. That is, we select du_j such that

$$T^p_{du_i.d,\, du_j.o} \le T^p_{du_i.d,\, du_k.o}, \text{ for each unvisited } du_k.$$

In this way, starting from du_0, we can obtain the solution for the optimization problem 8 efficiently and the empty run time $ET(E_m, p)$. The time complexity is $O(n)$, where n is the number of delivery units in period p.

An assumption underlying the above formulation is that there exists only one carrier in each task group E_m. This is due to that when conducting task grouping, we don't know the number of carriers in each task group E_m. The allocation of carriers to task groups is after the task grouping. Actually, if we are given the number of carriers in each task group, the problem can be formulated as a multiple travelling salesman problem [5] and corresponding empty run time can be similarly calculated.

5 TASK GROUPING ALGORITHM

5.1 Greedy Algorithm

The greedy algorithm is demonstrated in Algorithm 1. In general, the greedy algorithm is comprised of M rounds (Line 3-11) and at each round m, we assign tasks to task group E_m. Below, we detail the processes of assigning tasks to E_m at each round m.

Algorithm 1 Greedy Algorithm

```
1:  procedure GREEDY($E, M$)
2:      $E^u = E$                                   ▷ Initial unassigned tasks
3:      for $m = 1, \cdots, M$
4:          $E_m = \emptyset$
5:          $e_* \leftarrow$ a random task from $E^u$
6:          $E_m = E_m \cup \{e_*\}$,  $E^u = E^u \backslash \{e_*\}$
7:          while $|E_m| < \alpha|E|/M$ or $E^u \cap Nei(E_m) \ne \emptyset$
8:              for $e \in E^u \cap Nei(E_m)$
9:                  Compute $MTR(e|E_m)$                ▷ Equation 9
10:             $e_{**} = \arg\min_{e \in E^u \cap Nei(E_m)} MTR(e|E_m)$
11:             $E_m = E_m \cup \{e_{**}\}$, $E^u = E^u \backslash \{e_{**}\}$
12:     return $E_1, \cdots, E_M$
```

Initially, E_m is an empty set and is assigned with a random task e_* selected from unassigned tasks E^u (Line 3-6). E^u is initialized as E (Line 2) and tasks assigned will be removed from E^u. *Second*, we search each unassigned task e from E_m's neighborhood tasks $Nei(E_m)$ and compute the *marginal travel time ratio* $MTR(e|E_m)$ (Equation 9) of assigning e to E_m (Line 8-9). *Third*, the task e_{**} with the minimum marginal time ratio is assigned to E_m (Line 10-11), making E_m be with high shareability and low empty run time. We repeat the Step 2 and Step 3, until the number of tasks in E_m surpasses $\alpha|E|/M$ or there does not exist unassigned task nearby E_m (Line 7). The first stop criterion is to balance the size of each task group (e.g. $\alpha = 1.1$). The reason that we only consider the neighborhood tasks of E_m is that usually tasks not in $Nei(E_m)$ have much bigger marginal travel time ratios than tasks in $Nei(E_m)$. Thus, tasks not in $Nei(E_m)$ can be pruned and the total running time can be largely reduced. Below, we introduce the definitions of the marginal travel time ratio and the neighborhood tasks.

Definition 7 (*Marginal Travel Time Ratio*): For task group E_m and a task e, the marginal travel time ratio of assigning task e to task group E_m is defined as:

$$MTR(e|E_m) = \frac{TT(\{e\} \cup E_m) - TT(E_m)}{TT(\{e\})}, \tag{9}$$

where $TT(E_m)$ denotes the total travel time needed (including delivery time and empty run time) to deliver food orders in task group E_m as defined in Equation 4. Similarly, $TT(\{e\} \cup E_m)$ denotes the total travel time for delivering food orders in tasks $\{e\} \cup E_m$. That is, $MTR(e|E_m)$ is the marginal travel time of food orders after assigning e to E_m, divided by the original travel time needed to deliver food orders in task e. Apparently, to minimize the total travel time (our objective), we assign the task with the minimum $MTR(e|E_m)$ to the task group E_m each time, i.e. Line 10-11 in Algorithm 1.

Definition 8 (*Neighborhood Task*): A task e is a neighborhood task of task group E_m, if both e's origin $e.o$ and e's destination $e.d$ are geographically close to E_m, i.e.

$$T^p_{e.o,E_m} \le t_{nei} \text{ and } T^p_{e.d,E_m} \le t_{nei}.$$

$T^p_{e.o,E_m}$ denotes the minimum of the travel time from e's origin $e.o$ to regions in E_m. t_{nei} is a given threshold, e.g. 5 minutes.

5.2 Replacement Algorithm

The replacement algorithm, shown in Algorithm 2, is to improve the grouping result of the greedy algorithm. The main idea of the replacement algorithm is to replace the owner of a task from one task group with another task group, with the expectation that the total travel time can be reduced (our objective). The replacement algorithm stops (Line 3) when the number of continuous failed replacement attempts n_{cur} reaches the threshold n_{th} (e.g. $n_{th} = 100$). For each replacement process (Line 4-11), we have the following four steps.

First, we randomly select a task e from a random task group E_{m_1} and we remove e from E_{m_1} (Line 4-5). *Second*, for each task group E_m, we compute the *marginal travel time* $MT(e|E_m)$ (Equation 10) if assigning task e to group E_m (Line 6-7). *Third*, we find the task group E_{m^*} with the minimum marginal travel time, and assign task e to group E_{m^*} (Line 8-9). *Fourth*, if the task group E_{m^*} is exactly the task group E_{m_1}, this replacement process fails, since the total travel time is not reduced. As a result, the number of continuous failed attempts increases, i.e. $n_{cur} = n_{cur} + 1$ (Line 10). If the task group

Algorithm 2 Replacement Algorithm

```
1:  procedure REPLACE($E_1, \cdots, E_M$)
2:      $n_{cur} = 0$                                  ▷ Initial failed times
3:      while $n_{cur} < n_{th}$
4:          $e \leftarrow$ a random task from a random $E_{m_1}$
5:          $E_{m_1} = E_{m_1} \backslash \{e\}$
6:          for $m = 1, \cdots, M$
7:              Compute $MT(e|E_m)$                    ▷ Equation 10
8:          $m^* = \arg\min_{m=1,\cdots,M} MT(e|E_m)$
9:          $E_{m^*} = E_{m^*} \cup \{e\}$
10:         if $m^* = m_1$: $n_{cur} = n_{cur} + 1$
11:         else: $n_{cur} = 0$
12:     return $E_1, \cdots, E_M$
```

E_{m^*} is not the task group E_{m_1}, the replacement process succeeds and the total travel time has been reduced successfully. Thus, we reset the n_{cur} as $n_{cur} = 0$ (Line 11).

Definition 9 (*Marginal Travel Time*): The marginal travel time $MT(e|E_m)$ of assigning e to E_m is defined as:

$$MT(e|E_m) = TT(\{e\} \cup E_m) - TT(E_m), \tag{10}$$

where $TT(E_m)$ denotes the total travel time for task group E_m as defined in Equation 4. Thus, selecting the task group E_{m^*} with the minimum travel time for the task e is helpful for minimizing the total travel time (our objective).

6 EVALUATION

6.1 Datasets and Pre-processing Results

To evaluate our task grouping method, we use datasets collected from a real-world O2O food ordering and delivery platform. Specifically, our datasets include road network data, food order record data, and food carriers' GPS trajectory data. Below, we detail these datasets and our pre-processing results.

6.1.1 Road network. We use the road network data in city of Shanghai, China. The network contains 333,766 intersections and 440,922 road segments, as shown in Figure 8(a). As detailed before, our region segmentation is based on the road network. Specifically, using the road network, we segment the area in Figure 8(a) into regions, and after merging small and adjacent regions, we obtain 1,000 regions (i.e. $|V| = 1000$), as demonstrated in Figure 8(b). Besides, the road network data is also needed when we estimate the travel time of food carriers between any two regions.

6.1.2 Food order records. We collect one month of food order record data from users in this area. The food order records start from December 1 to December 31, 2016. For each food order record, we have the following information: 1) the time stamp at which the user ordered the food; 2) the restaurant information (ID, latitude, longitude); 3) the user information (ID, latitude, longitude); 4) the ID of the food carrier delivering the food order; and etc.

Temporal distribution of food orders. In total, our food order record dataset contains 1,852,439 food orders in total. In average, each day is with around 59,756 orders. Note that a time period is with 15 minutes, thus we have 2,976 time periods for the 31 days of food order records, i.e. $P = 2976$. The number of food orders in

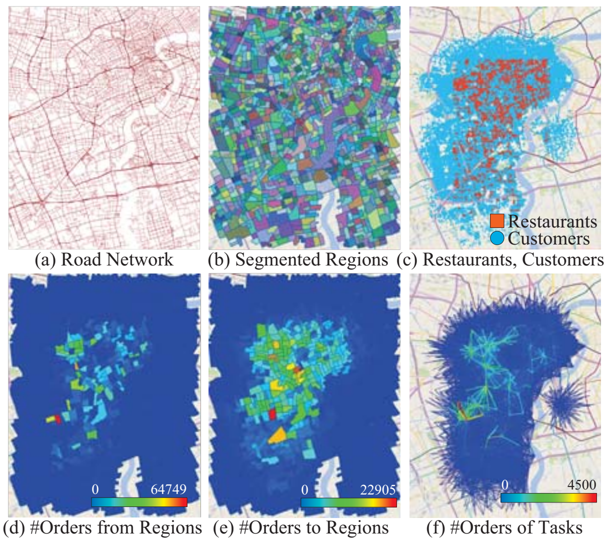


Figure 8: Regions, tasks, and food orders in an area of Shanghai city, China.

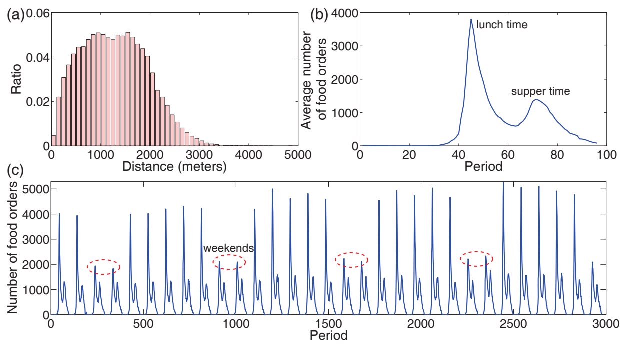


Figure 9: Statistics of food orders from real-world users.

each time period is demonstrated in Figure 9(c), and Figure 9(b) presents the average number of food orders in each period of a day. Clearly, the number of food orders has a clear daily pattern, i.e. there exist two peaks (a big one and a small one) each day. The big peak appears during the lunch time while the small peak appears during the supper time. On the other hand, there also exists a weekly pattern, i.e. weekends have much less food orders than weekdays, as shown in Figure 9(c). It implies that people more often order take-out food during their working hours, especially at noon on weekdays.

Spatial distribution of food orders. Together with the segmented regions V, the food order record data can be used to construct the task graph. Based on the food order records, using the method proposed in Section 3.2, we extract 2,980 tasks, i.e. $|E| = 2980$. And then we can construct the task graph $G = \langle V, E \rangle$, based on the obtained V and E.

The tasks (edges) are demonstrated in Figure 8(f). Task (edges) are denoted by lines in different colors based on the number of food orders in each task. That is, different colors correspond to different numbers of food orders, as the color bar shows. For example, a task denoted by red color has around 4500 food orders in the 31 days, while a task in blue color has only a few food orders.

Similar to the Figure 8(f), we can also draw the spatial distribution of food orders in each region, as shown in Figure 8(d) and (e). Specifically, Figure 8(d) demonstrates the number of food orders from each region (restaurants), while Figure 8(e) shows the number of food orders to each region (users).

Figure 8(c) shows the geographical locations of restaurants and users. Figure 9(a) presents the distribution of the geographical distances between restaurants and users. As depicted in the figure, for most food orders, the distances between restaurants and users are within 3,000 meters. This is due to that the O2O platform usually recommends restaurants within 3 kilometers to users on its web and App.

6.1.3 GPS trajectories of food carriers. The GPS trajectories of food carriers are also collected in December 2016. For each GPS record, we have a food carrier ID, a latitude, a longitude, and a time stamp. In total, there are 4,774 food carriers with 115,602,018 GPS records collected. Based on the GPS trajectory data, using the method proposed in Section 3.3, we can estimate the travel time of food carriers traveling between any two regions in V, in each time period p.

6.2 Metrics

To evaluate our task grouping method, we use four metrics, which are the average delivery time $\overline{DT}$, the average empty run time $\overline{ET}$, the average travel time $\overline{TT}$ for each food order, and the average number of regions that each task group covers. Formally, the four metrics can be formulated as

$$\overline{DT} = \frac{DT}{n_{total}}, \quad \overline{ET} = \frac{ET}{n_{total}}, \quad \overline{TT} = \frac{TT}{n_{total}}, \quad \overline{\#R} \tag{11}$$

where DT, ET and TT correspond to the total delivery time in Equation 1, the total empty run time in Equation 2 and the total travel time in Equation 3, respectively. n_{total} denotes the number of total food orders in all tasks. The smaller the first three metrics, the better a grouping result. For a task grouping result with a smaller $\overline{\#R}$, carriers in a task group can be more familiar with the locations of users and restaurants. Thus, the smaller the $\overline{\#R}$, the better.

6.3 Baselines

To demonstrate the effectiveness of our task grouping method, we compare our method with many state-of-the-art baseline methods, including NE [30], DBH [27], HDRF [19], RAND [9], and Oblivious [9, 19]. Although these baseline methods are originally designed for undirected graph edge partition problems for distributed graph computation platforms [9, 14, 15], they can be used or can be slightly modified to partition directed graph edges for our task grouping problem.

6.4 Effectiveness

We conduct extensive experiments to compare our task grouping method with the above baselines. Specifically, we use the task graph G constructed from our datasets and set $M = 300, 400, 500,$

Table 1: Comparisons with baselines (unit for $\overline{DT}, \overline{ET}, \overline{TT}$: seconds).

Task Grouping Method	$M = 300$				$M = 400$				$M = 500$			
	$\overline{DT}$	$\overline{ET}$	$\overline{TT}$	$\overline{\#R}$	$\overline{DT}$	$\overline{ET}$	$\overline{TT}$	$\overline{\#R}$	$\overline{DT}$	$\overline{ET}$	$\overline{TT}$	$\overline{\#R}$
NE	275	260	535	9.9	277	264	541	7.9	279	268	547	6.6
DBH	276	294	570	8.8	276	280	556	6.6	276	276	552	5.4
HDRF	283	267	550	7.3	285	264	549	6.0	285	258	543	5.1
RAND	292	611	903	18.4	292	603	895	13.9	292	606	898	11.3
Oblivious	284	261	545	7.5	285	252	537	5.9	285	250	535	4.9
Greedy (ours)	270	191	461	8.8	273	198	471	6.8	276	200	476	6.3
Greedy+Replace (ours)	268	180	448	8.5	271	185	456	7.0	273	186	459	6.0

i.e. grouping all tasks into 300, 400, 500 groups. To better evaluate the performance of the greedy algorithm and the replacement algorithm in our method, we first consider only the greedy algorithm (Greedy) and then add the replacement algorithm (Greedy+Replace). Experiment results are summarized in Table 1.

In terms of the first three metrics, our task grouping method defeats all baseline methods. The best baseline method is the NE method [30]. The improvement of our method over baseline methods is significant. For example, when $M = 300$, our greedy algorithm (461 seconds) has already defeated the baseline algorithms by at least 74 seconds (comparing with 535 seconds of NE method). If combined with the replacement algorithm, our method needs just 448 seconds for each food order, saving 87 seconds for each order. That is, our method can save at least 16% of average travel time for each order, comparing with the baseline methods, i.e. significantly reducing the waiting time of users.

In terms of the average number $\overline{\#R}$ of regions covered by each task group, each task group of our method covers less than 9 regions, which is a proper value. From Table 1, we can find that the smaller the M, the bigger the $\overline{\#R}$. Thus, to make carriers familiar with the locations of users and restaurants in a task group, grouping more task groups (a relatively bigger M) is better.

In the meantime, we can find that the average delivery time, empty run time, and travel time will slightly increase with the increasing of M. This is due to that the larger the M, the smaller the size of each task group, and the more difficult to improve shareability and to reduce empty run time. However, this does not mean a smaller M is better since we expect a bigger M to reduce $\overline{\#R}$. Thus, the selection of M should be a trade-off between $\overline{TT}$ and the $\overline{\#R}$ for O2O platforms in real world.

6.5 Time Efficiency

In this subsection, we study the time efficiency of our task grouping method. Our task grouping method is realized in C# and is performed on a single computer with 4 Intel Xeon E3-1225 3.31 GHz cores and 8 GB RAM. The running time of our task grouping method is presented in Figure 10. We can see that our task grouping method is efficient, using just at most 1.5 hours. It should be noted that given a city, the task grouping is only conducted from time to time, e.g. once per month or even per quarter/half of a year, as mentioned in the Introduction. Besides, our task grouping method can easily run in parallel. When more machines are used, the running time can be significantly reduced. Thus, our task grouping method is efficient enough for the applications of real-world O2O food ordering and delivery platforms.

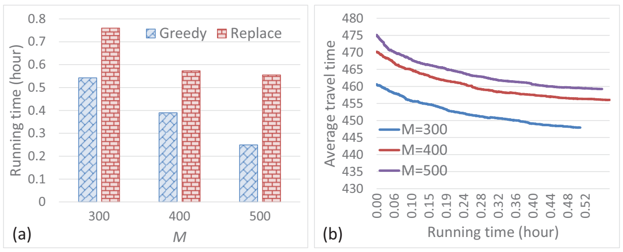


Figure 10: Running time of our task grouping method.

6.6 Balance of Task Grouping Results

The balance of each task group is also an important index. The amounts of tasks, regions, and food orders in different task groups are expected to be balancing. To this end, for a task grouping result, we use entropy to indicate the balance of tasks, regions and food orders in each task group. More specifically, we define the entropy of tasks for a task grouping result as $Ent^t = -\sum_{m=1}^{M} p_m^t \log_2 p_m^t$. p_m^t denotes the ratio of tasks in group m, i.e. the number of tasks in group m divided by the total number of tasks. Similarly, in terms of regions and food orders, we can define $Ent^r = -\sum_{m=1}^{M} p_m^r \log_2 p_m^r$ and $Ent^o = -\sum_{m=1}^{M} p_m^o \log_2 p_m^o$. p_m^r and p_m^o refer to the ratio of regions and the ratio of food orders in group m, respectively. Entropies of different task grouping methods are presented in Figure 11. As shown in the figure, our method, together with HDRF, RAND and Oblivious, achieves balancing task grouping results, with entropies approaching the optimal entropy $\log_2 300 \approx 0.829$.

7 RELATED WORK

7.1 Previous Research on Take-out Food

Previous research on take-out food (takeaway food) is conducted mainly from the socioeconomic perspective [17, 18, 25] or from the perspective of human health [23, 24]. In general, previous research mainly focuses on the influence of take-out food consumption on health. For example, in [18], Miura et al. studied the relationship between the consumption of take-out food with the socioeconomic differences in fruit and vegetable intake. Based on their analysis, they found that less educated people are more likely to consume

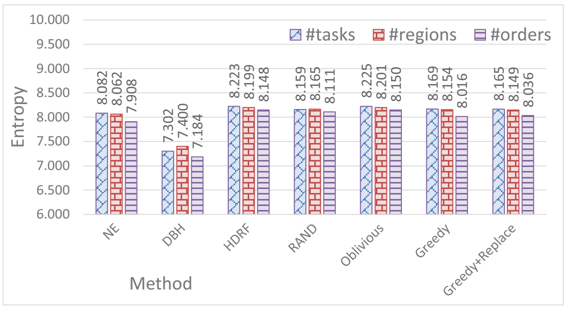


Figure 11: Entropies of tasks, regions, and orders. $M = 300$**.**

less healthy take-out food, containing fewer fruit and vegetables, and that consuming less healthy take-out food may result in the socioeconomic differences in fruit and vegetable intake. For another example, in literature [24], Smith et al. investigated the relationship between the take-out food consumption with diet quality and abdominal obesity. This work concluded that consuming take-out food twice a week or more may lead to poor diet quality and moderate abdominal obesity in young men and women.

Different with the above work, our work aims to improve the take-out food delivery efficiency for O2O take-out food ordering and delivery platforms, from the perspective of computer science, using data-driven methods. Consuming take-out food frequently may lead to some health concerns, however more and more healthy take-out food can be ordered. Since massive food orders are issued each day, to improve the food delivery efficiency is essential for O2O platforms. To the best of our knowledge, we are the first to study the take-out food from this perspective.

7.2 Graph Edge Partition

From the perspective of the problem formulation, our task grouping problem is formulated as a graph edge partition problem (a.k.a. graph vertex-cut problem) [6, 9, 16, 19, 27, 30]. Previous graph edge partition problems are mainly studied for large-scale distributed graph computation platforms, e.g. PowerGraph [9], GraphLab [14], Pregel [15]. Specifically, they partition a large-scale computation graph to different machines and each machine runs a part of the graph in parallel, so as to speed up the whole computation process. Their objective is to minimize the number of replicated vertices in different machines so as to lower the communication cost between machines. Many heuristic algorithms [9, 16, 19, 27, 30] have been proposed in recent years for graph edge partition problems. For example, in [30], Zhang et al. proposed a neighborhood expansion-based heuristic algorithm to do the graph edge partition. In [9], Gonzalez et al. proposed a random algorithm and some greedy algorithms. In [19], Petroni et al. presented a high-degree replicated first algorithm. Besides the graph edge partition problems, graph vertex grouping problems (a.k.a. graph edge-cut problems) [6, 7, 12] were also widely studied, which is to partition vertices (instead of edges) in a graph into a given number of groups.

As discussed before, although our task grouping problem is a graph edge partition problem, due to the different scenarios and objectives, previous graph edge partition algorithms cannot apply well to our problem. Actually, our experiment results have also validated that our proposed task grouping algorithm can achieve much better performance than the previous graph edge partition algorithms (see Table 1).

8 CONCLUSION AND FUTURE WORK

In this paper, we proposed a task grouping method for O2O take-out food ordering and delivery platforms, consisting of two main contents. First, we formulated the factors affecting the food delivery efficiency and provided a combined objective to guide our task grouping. Second, we presented effective heuristic algorithms for grouping tasks, including a greedy algorithm and a replacement algorithm. Using our task grouping method, the food delivery efficiency can be highly improved. According to the experiment results, our method is able to group tasks into task groups with high shareability (i.e. low delivery time) and low empty run time. Comparing with the previous graph edge partition algorithms, our method is able to save at least 16% (87 seconds) of average travel time for each food order, which is a significant improvement. As a result, with our research, O2O take-out food ordering and delivery platforms can largely improve their food delivery efficiency, thus providing better food delivery services for users. In the future, as mentioned in the Introduction, we plan to study the food carrier allocation problem and the real-time food carrier dispatching problem. Note that these two problems also significantly affect the food delivery efficiency of O2O food ordering and delivery platforms.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (Grant Nos. 61672399, U1609217, 61773324), the Fundamental Research Funds for the Central Universities (Nos. 220710004005040177), and Doctoral Innovation Fund Program of Southwest Jiaotong University.

REFERENCES

[1] [n. d.]. *Gurobi optimizer reference manual.* http://www.gurobi.com.
[2] [n. d.]. *Microsoft solver foundation reference manual.* https://msdn.microsoft.com/en-us/library/ff524499(v=vs.93).aspx.
[3] [n. d.]. *Online take-out food ordering platform 1 in China.* https://en.wikipedia.org/wiki/Ele.me.
[4] [n. d.]. *Online take-out food ordering platform 2 in China.* https://en.wikipedia.org/wiki/Meituan-Dianping.
[5] Tolga Bektas. 2006. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega* 34 (2006), 209–219.
[6] Florian Bourse, Marc Lelarge, and Milan Vojnovic. 2014. Balanced graph edge partition. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD.* 1456–1465.
[7] Aydin Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. 2016. Recent Advances in Graph Partitioning. In *Algorithm Engineering - Selected Results and Surveys.* 117–158.
[8] C. De Fabritiis, R. Ragona, and G. Valenti. 2008. Traffic estimation and prediction based on real time floating car data. In *Proceedings of the 11th International IEEE Conference on Intelligent Transportation Systems.*
[9] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. 2012. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation.* 17–30.
[10] E. Jenelius and H. N. Koutsopoulos. 2013. Travel time estimation for urban road networks using low frequency probe vehicle data. *Transportation Research Part B: Methodological* 53 (2013), 64–81.
[11] David S. Johnson and Lyle A. McGeoch. 1997. The Traveling Salesman Problem: A Case Study in Local Optimization. *Local Search in Combinatorial Optimisation* (1997), 215–310.

[12] George Karypis and Vipin Kumar. 1998. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Scientific Computing* 20, 1 (1998), 359–392.

[13] John Krumm and Eric Horvitz. 2007. Predestination: Where Do You Want to Go Today? *IEEE Computer* 40 (2007), 105–107.

[14] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. 2012. Distributed GraphLab: A Framework for Machine Learning in the Cloud. *PVLDB* 5, 8 (2012), 716–727.

[15] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: a system for large-scale graph processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD*. 135–146.

[16] Daniel W. Margo and Margo I. Seltzer. 2015. A Scalable Distributed Graph Partitioner. *PVLDB* 8, 12 (2015), 1478–1489.

[17] Kyoko Miura, Katrina Giskes, and Gavin Turrell. 2009. Socioeconomic differences in takeaway food consumption and their contribution to inequalities in dietary intakes. *Journal of Epidemiology & Community Health* 63, 10 (2009), 820–826.

[18] Kyoko Miura, Katrina Giskes, and Gavin Turrell. 2011. Contribution of take-out food consumption to socioeconomic differences in fruit and vegetable intake : a mediation analysis. *Journal of The American Dietetic Association* 111, 10 (2011), 1556–1562.

[19] Fabio Petroni, Leonardo Querzoni, Khuzaima Daudjee, Shahin Kamali, and Giorgio Iacoboni. 2015. HDRF: Stream-Based Partitioning for Power-Law Graphs. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management, CIKM 2015*. 243–252.

[20] Jason W. Powell, Yan Huang, Favyen Bastani, and Minhe Ji. 2011. Towards Reducing Taxicab Cruising Time Using Spatio-Temporal Profitability Maps. In *Advances in Spatial and Temporal Databases - 12th International Symposium, SSTD*. 242–260.

[21] César Rego, Dorabela Gamboa, Fred Glover, and Colin Osterman. 2011. Traveling salesman problem heuristics: Leading methods, implementations and latest advances. *European Journal of Operational Research* 211, 3 (2011), 427–441.

[22] Daniel J. Rosenkrantz, Richard Edwin Stearns, and Philip M. Lewis II. 1974. Approximate Algorithms for the Traveling Salesperson Problem. In *15th Annual Symposium on Switching and Automata Theory*. 33–42.

[23] Helmut Schröder, Montserrat Fïto, and Maria Isabel Covas. 2007. Association of fast food consumption with energy intake, diet quality, body mass index and the risk of obesity in a representative Mediterranean population. *British Journal of Nutrition* 98 (2007), 1274–1280.

[24] Kylie J Smith, Sarah A McNaughton, Seana L Gall, Leigh Blizzard, Terence Dwyer, and Alison J Venn. 2009. Takeaway food consumption and its associations with diet quality and abdominal obesity: a cross-sectional study of young adults. *International Journal of Behavioral Nutrition and Physical Activity* 6, 29 (2009).

[25] Gavin Turrell and Katrina Giskes. 2008. Socioeconomic disadvantage and the purchase of takeaway food: a multilevel analysis. *Appetite* 51, 1 (2008), 69–81.

[26] Yilun Wang, Yu Zheng, and Yexiang Xue. 2014. Travel time estimation of a path using sparse trajectories. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 25–34.

[27] Cong Xie, Ling Yan, Wu-Jun Li, and Zhihua Zhang. 2014. Distributed Power-law Graph Computing: Theoretical and Empirical Analysis. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014*. 1673–1681.

[28] Jing Yuan, Yu Zheng, Chengyang Zhang, Xing Xie, and Guangzhong Sun. 2010. An Interactive-Voting Based Map Matching Algorithm. In *Eleventh International Conference on Mobile Data Management, MDM*. 43–52.

[29] Nicholas Jing Yuan, Yu Zheng, and Xing Xie. 2012. Segmentation of Urban Areas Using Road Networks. *MSR-TR* 65 (July 2012).

[30] Chenzi Zhang, Fan Wei, Qin Liu, Zhihao Gavin Tang, and Zhenguo Li. 2017. Graph Edge Partitioning via Neighborhood Heuristic. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD*. 605–614.

[31] Yu Zheng. 2015. Trajectory Data Mining: An Overview. *ACM Transaction on Intelligent Systems and Technology* 6, 3 (2015), 29:1–29:41.

[32] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. 2012. Mining interesting locations and travel sequences from GPS trajectories. In *Proceedings of the 18th International Conference on World Wide Web, WWW*. 17–30.