Task Selection and Scheduling for Food Delivery: A Game-Theoretic Approach

Mingzhe Li*†, Jin Zhang*, Wei Wang†
*†Computer Science and Engineering Department
*Southern University of Science and Technology
†Hong Kong University of Science and Technology
†{mlibn, weiwa}@cse.ust.hk, *zhangj4@sustc.edu.cn

***Abstract*—With the development of embedded sensors in smartphones and the ever-growing number of mobile users, more and more spatial crowdsourcing applications come into our daily lives. Food delivery, a specific application of spatial crowdsourcing, emerged and quickly proliferates in recent years. However, task selection and scheduling remains a challenging problem in food delivery. In this paper, we aim to address the task selection and scheduling problem from a game-theoretic perspective. Specifically, the riders are allowed to select a set of tasks as well as the task completion order, by taking account of their locations, speed, traveling costs and capacities. The objective of each rider is to maximize his own utility. We formulate a food delivery game and prove the existence of a Nash equilibrium. Experimental results show that our scheme outperforms existing solutions in terms of the social welfare, average utility, task completion ratio, and fairness. Moreover, the social welfare attained by our scheme comes close to that of the centralized optimal solution.**

I. Introduction

Today's smartphones are equipped with various embedded sensors, such as accelerometers, GPS, cameras, microphones, and light sensors [6], [7], [8]. Together with the proliferation of smartphones, the ever-growing number of mobile users make the spatial crowdsourcing possible and commonplace in daily life [5]. Spatial crowdsourcing is a specific type of crowdsourcing in which tasks are often interrelated to spatial information, and mobile users must physically move to the task's location in order to get the task done.

With the accelerated pace of life, more and more people want to enjoy their meal by ordering take-away food either in workplace or at home. Thus a specific type of spatial crowdsourcing called food delivery emerged and quickly proliferates in recent years [11]. Typical food delivery applications work as follows. The customers order the take-away food in the platform; the platform collects orders from the customers, notifies the restaurants, and publishes or assigns food delivery tasks to riders. A rider, after getting tasks from the platform, picks up the food at the restaurants and then travels to customer's location to deliver the order. The restaurants should prepare the meals once noticing customers' orders from the platform.

A main problem for general spatial crowdsourcing systems and also the main problem for food delivery systems is how to decide which workers (riders) are assigned which tasks given the workers' locations, capacities, traveling costs, and traveling speeds. Several researches have focused on this problem in a more general spatial crowdsourcing setting. For instance, Kazemi *et al.* [2] formulate this task selection problem in spatial crowdsourcing in a centralized model, where the server positively assigns each task to users using a matching approach with the objective of maximizing the number of assigned tasks. Deng *et al.* [3] proposed a centralized task matching and scheduling scheme in general spatial crowdsourcing in order to maximize the number of completed tasks, which first matches tasks to users then schedules users to complete their corresponding tasks in a certain order. Cheung *et al.* [4] considered a distributed game theoretical mechanism in which heterogeneous users distributively select time-sensitive and location-dependent tasks by their own in order to maximize his own utility.

However, none of these works are a good fit for food delivery systems. Food delivery has its own characteristics as follows. First, food delivery has strict time constraints. Thus it is a spatial-temporal joint problem, whereas most spatial crowdsourcing problems mainly focus on the spatial domain. Second, the task in traditional spatial crowdsourcing system usually is associated with one location. However in food delivery, every task has two locations, one for the riders to pick up tasks and the other for delivering the tasks. What's more, a task is associated with a time window. That is, a task is successfully finished only if the riders pick up the task after the task start time and deliver it before the task end time. Besides, to save the travel distance, the riders can have multiple tasks to be taken together in one ride. All of the aforementioned complex spatio-temporal constraints make the task scheduling problem even more difficult. In addition, in food delivery, one task can only be finished by one rider, as cooperation among riders to finish one task together is not applicable in food delivery. This constraint increases the difficulty of task selection.

The aforementioned task selection and scheduling problem for food delivery system can be solved using a centralized algorithm. However, these centralized mechanisms often aim at maximizing social welfare, thus causing strong users (e.g. a rider with extremely high speed or infinite capacity) finishing most of the tasks and weak users getting none. Consequently, users with weaker ability will leave the system with a high probability. Another problem of centralized mechanisms is the privacy leakage [1]. In order to assign the right tasks to certain users, the servers must know the users' locations, which are the private information. Moreover, in practice, people are often selfish, they care more about their own utility. However, in the centralized system users do not have the authorities to choose tasks and hence will sometimes get sub-optimal benefit.

In this paper, we consider a game-theoretical approach to model this case in which platform publishes food delivery tasks and riders with heterogeneous locations, capacities, traveling costs, and traveling speeds distributively make their decisions to select tasks to complete. The interactions between riders are formulated as a task selection and scheduling game, where each rider aims to maximize his own profit. In addition, we propose an algorithm using dynamic programming to help riders to make the decision of task selection and scheduling.

To the best of our knowledge, our work is the first to focus on spatio-temporal task selection and scheduling problem in food delivery setting with a game theoretical form. The main contributions are summarized as follows:

- Motivated by food delivery systems, we focus on the selection and scheduling of spatio-temporal tasks by the number of riders and formulate it into a game. Riders here are assumed to have different locations, capacities, traveling speed and cost. We show the existence of the Nash equilibrium in our model.
- We propose an algorithm using dynamic programming to help riders to decide the tasks they should choose and the path to finish the selected tasks.
- Through extensive simulation studies, we show that our task selection and scheduling algorithm outperforms existing solutions with better fairness, while at the same time achieving the near-optimal social welfare.

II. System Model

We consider a specific spatial crowdsourcing system called food delivery system. A complete food delivery system consists of four agents and five processes as shown in Figure 1. First, customers order meals using mobile apps of the platform (e.g. Deliveroo [10]). Upon receiving the orders, the restaurant starts to prepare meals. At the meantime, riders can check food delivery tasks on their smartphones. After selecting tasks by themselves (or being assigned tasks by the platform), riders pick up tasks at the restaurant locations and finally deliver the tasks to the customers' locations. In this paper, we focus on solving the task selection and scheduling problem in this system. By using our scheme, the riders should be able to select a suitable set of tasks to deliver and decide the sequence of tasks to finish.

For each food delivery task $k, k \in \mathcal{K}$, where $\mathcal{K}$ is the task set containing all the available tasks $\{1, \cdots, K\}$, it should be picked at the start location l_k^+ and be delivered to the end location l_k^-. Besides, there are also time constraints on each task. Task k can only be picked up after its start time a_k and be delivered successfully before its end time b_k, where the

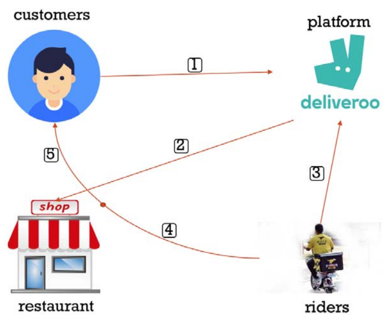

Fig. 1. A food delivery system.

start time corresponds to the time when the food is ready for deliver in the restaurant and the end time is the deadline set by the customer. Once the task is successfully delivered on time, the rider can achieve a reward r_k, otherwise, the reward is zero for the rider. For task k, the load is d_k which defines the weight of the food to be delivered.

For each rider n, $n \in \mathcal{N}$, where $\mathcal{N} = \{1, \cdots, N\}$ is the rider set, the initial location is w_n. The traveling speed of rider n is v_n , which defines the maximum speed the rider n travels between two locations. The cost the rider n spends for a distance d is defined as $c_n = c_0^n d$, where c_0^n is the cost for travelling unit distance. The capacity of rider n is z_n, which defines the maximum weight of all the tasks the rider is carrying. Each rider is associated with an available task set $\mathcal{K}_n \subseteq \mathcal{K}$, a rider n can choose a subset of tasks $\mathcal{K}_n^{'} \subseteq \mathcal{K}_n$ along with a path p_n to finish the corresponding tasks, where the path p_n is the sequence to visit the start locations and end locations of the selected tasks from rider n's initial location.

We formulate the task selection and scheduling problem as a on-line game. The tasks and riders arrive randomly along the time. We divide the whole procedure into multiple rounds. At the beginning of each round, the platform runs our proposed scheme to achieve the suitable task selection and scheduling result, using the current task set $\mathcal{K}$ and rider set $\mathcal{N}$ as the input, where task set $\mathcal{K}$ contains all the tasks arrived in the previous rounds which have not been allocated to any rider, and rider set $\mathcal{N}$ contains all active riders at that time. For each round, one task can only be assigned to one rider and one rider can claim multiple tasks. The platform randomly generates a sequence for the riders in each round, and according to the sequence, the riders will decide their best task set and task sequence (path) one by one, trying to maximize their own utility when make the decisions. Therefore, we can formulate the problem as a non-cooperative sequential game, where each rider is the player and their strategy space is all possible task sets and paths.

Although the aforementioned task selection and scheduling problem can also be solved using a centralized optimization scheme, game has more superior properties than the centralized optimization problem. Firstly, each rider is an individual who only cares about his own profit, instead of the overall performance of the system. Therefore, he only has motivation to maximize his own revenue instead of follow the platforms task allocation. Besides, the centralized mechanism, which targets at maximizing social welfare, usually achieves an unfair result which favors strong users (e.g. a rider with high speed, high capacity and low cost) but is unfriendly to weak users. By formulating the problem as a game, we can achieve more reasonable and fair results. We will analyze the game in detail in the next section.

III. Game Theoretical Analysis

In this section, we formulate the riders' task selection and scheduling problem as a task selection and scheduling game, in which riders are players, and the goal for each of them is to achieve his best response (maximize his own utility). What's more, we prove the existence of Nash equilibrium in this game. We also formulate the finding best response process as a graph and solve it by using dynamic programming algorithm.

A. Task Selection and Scheduling Game

Before analyzing the task selection and scheduling game, some basic game theory definitions should be noticed. For a rider n, the strategy space $\mathcal{S}_n = (2^{\mathcal{K}_n}, \mathcal{P}_n)$ is all subsets of his available task set $2^{\mathcal{K}_n}$ along with his available path set $\mathcal{P}_n$. A rider's strategy $s_n = (\mathcal{K}^{'}_n, p_n)$ is thus consists of a subset $\mathcal{K}^{'}_n$ of his available task set and a path p_n in his available path set to finish the tasks. A rider n's best response $s^*_n = (\mathcal{K}^*_n, p^*_n)$ is the best task set $\mathcal{K}^*_n$ and best path p^*_n that can achieve his maximum utility. The strategy profile for the game is denoted as $\boldsymbol{s} = (s_n, s_{-n})$, where $s_{-n} = (s_1, \cdots, s_{n-1}, s_{n+1}, \cdots, s_N)$ is the strategy profile of all riders except rider n. The utility for a given rider n is defined as the total payment of completed tasks for rider n minus the traveling cost occurred along the path when doing the tasks, which is

$$U_n = \sum_{k \in \mathcal{K}'_n} r_k - \sum_{l_k, l^{'}_{k'} \in \mathcal{L}, k, k' \in \mathcal{K}'_n} c^n_0 \cdot d_{l_k, l^{'}_{k'}}$$

where l_k, $l^{'}_{k'}$ means the locations (start location or end location) of task k and $k^{'}$, $d_{l_k, l^{'}_{k'}}$ means the distance between locations l_k and $l^{'}_{k'}$.

Our proposed scheme for task selection and scheduling is shown in Algorithm 1. The platform first generate a random sequence on all the N riders, then riders sequentially make their best responses by selecting their best task sets and task paths using algorithm 2. Before calculating their best response strategies, they should first remove the tasks which are already selected by the previous riders from the available task set, because one task can only be allocated to one rider.

Definition 1 (Nash equilibrium)**.** A strategy profile $\mathbf{s}^*$ is a Nash equilibrium (NE) if for any player, he can not be profitable when he unilaterally deviate from strategy, that is $U_n(s^*_n, s^*_{-n}) \geq U_n(s_n, s^*_{-n})$ for any s_n.

Theorem 1. *Every task selection and scheduling game has a Nash equilibrium.*

Proof: We prove this theorem by contradiction. Suppose a rider n has a better strategy $s^{'}_n$ that can get more utility than using the strategy s^*_n, thus will cause the game could not achieve Nash equilibrium. However, the strategy s^*_n is got by making the best response, and after making the best response the output is a set of tasks and a path that can maximize a rider's utility, which means the strategy s^*_n got by making the best response is the strategy that maximize one's utility. Thus there couldn't exist any strategy which can earn more utility than s^*_n, and thus a contradiction occurs. ■

Algorithm 1 task selection and scheduling game

```
1: randomly order the $N$ riders as $\{1, 2, \cdots, N\}$
2: Initialization: set the initial available task set as the total
   task set ($\mathcal{K}_0 = \mathcal{K}$) and the optimal set to be empty set
   ($\mathcal{K}^*_0 = \emptyset$)
3: for each ordered rider $n = 1 : N$ do
4:     calculate the available task set ($\mathcal{K}_n = \mathcal{K}_{n-1} - \mathcal{K}^*_{n-1}$)
5:     calculate rider $n$'s best task set $\mathcal{K}^*_n$ and best path $p^*_n$
       using Algorithm 2, aiming at maximizing rider $n$'s
       utility
6: end for
```

B. Best Response for Each Rider

In the task selection and scheduling game shown in Algorithm 1, one key process is for each rider to calculate his best response in terms of best task set and task path. However, these two problems are interconnected with each other, which can not be separated. Besides, each task consists of two locations and one time window, and each rider has his own capacity constraint. Therefore, we can not simply model the best response problem as a shortest path problem and solve it using Dijkstra or Bellman-Ford algorithm. To solve this challenging problem, we proposed to use a dynamic programming-based algorithm to construct all possible sets and paths gradually, trying to find the optimal set and path eventually. To reduce the complexity of the algorithm, we adopt branch pruning by removing the paths which are not good or that does not satisfy the constraints during the path construction, which dramatically reduces the number of feasible path and overall complexity.

To achieve the best response strategy on task set and task path, the first thing for rider n is to construct a graph basing on the location of the tasks and his own initial location.

In the graph $\mathcal{G}_n = (\mathcal{O}_n, \mathcal{O}^2_n)$, the set of vertices $\mathcal{O}_n = \mathcal{L}^+_n \cup \mathcal{L}^-_n \cup w_n$ is made up with the task start locations $\mathcal{L}^+_n$ and task end locations $\mathcal{L}^-_n$ of $\mathcal{K}_n$, and with the rider's initial location w_n. $\mathcal{O}^2_n$ is the set of all edges connecting any two vertices (e.g. for two vertices i and j, the edges are $(i,j), (j,i), (i,i), (j,j)$).

Each vertex $i \in \mathcal{O}_n$ is associated with a weight of goods d_i at vertex i, where the load for the task start location d_{i^+} is larger than zero, and the load for the task end location is set to be $d_{i^-} = -d_{i^+}$, the load for the rider's initial location d_s is equal to zero. $[a_i, b_i]$ is the time window for $i \in \mathcal{O}_n$, the time window for a task start location $[a_{i^+}, b_{i^+}]$ should be equal to the time window for the task end location of the same task $[a_{i^-}, b_{i^-}]$, and we set the start time for the initial location a_w to be zero and the end time for the initial location b_w to be infinite.

Every edge $(i,j) \in \mathcal{O}_n^2$ is associated with a traveling time t_{ij}^n, which is related to a rider's velocity v_n and the distance between i and j, and a traveling cost c_{ij}^n.

Before the solution algorithm, we eliminate some edges which don't satisfy certain constraints of the problem to reduce the complexity of computation, and thus produce a new gragh $\mathcal{G}_n^{'} = (\mathcal{O}_n, \mathcal{E}_n)$, where $\mathcal{E}_n \in \mathcal{O}_n^2$ is the set of feasible edges generated after the elimination rules. The elimination rules are as follows:

1) for $i \in \mathcal{O}_n$, (w_n, i^-) and (i^-, i^+) can be eliminated, because that a rider must first go to the task start location to pick up the task then deliver the task at the task end location.
2) for $i \in \mathcal{O}_n$, (i,i) can be eliminated, the reason is that one task cannot be picked up or delivered twice. Thus the path a rider finally get is an elementary path (e.g. no cycle).
3) for $i,j \in \mathcal{O}_n$, $i \neq j$, if $d_{i^+} + d_{j^+} > z_n$, then (i^+, j^+), (j^+, i^+), (i^+, j^-), (j^+, i^-), (i^-, j^-), (j^-, i^-) can be eliminated, because a rider can not take the goods whose weight is more than his capacity.
4) for $(i,j) \in \mathcal{O}_n^2$, $i \neq j$, if $a_i + t_{ij} > b_j$, then (i,j) can be eliminated, because if a rider wants to finish a task, that task must be picked up and delivered on time.
5) for $(i^+, i^-) \in \mathcal{O}_n^2$, if $a_i + t_{i+i-} > b_i$, then (all, i^+), (i^-, all), (i^+, i^-) can be eliminated, the reason is similar as rule no. 4.

Now we solve the problem by using a dynamic programming algorithm on the constructed graph $\mathcal{G}_n^{'}$, as shown in algorithm 2. The algorithm can be divided into several iterations. In each iteration, we use the paths which go through $k-1$ vertices to generate paths with k vertices. The generated paths must satisfy several constraints. After the path generation, some paths can be eliminated according to certain criteria. After the elimination, we temporarily save the paths for the next round computation and delete the paths in the $(k-1)$th iteration to save space. For the generated paths in which all picked tasks along that path has been finished, we call them complete paths and permanently save them to a complete path set $\mathcal{P}_n^{'} \subseteq \mathcal{P}_n$. Finally, we can find the best path p_n^* and the best task set $\mathcal{K}_n^*$. The path generation rules (line 3) and elimination rules (line 4) will be explained in detail in the next few paragraphs.

The path generation process is illustrated as follows. First, some extra attributes of a path is needed to be defined. Every path p is associated with a tuple $(\mathcal{H}_p, o_p, T_p, U_p, L_p)$, where $\mathcal{H}_p \subseteq \mathcal{O}_n$ denotes the set of past vertices that the path has passed, o_p denotes the end vertex of path p, T_p is the arrival time at vertex o_p, U_p is the total utility which equals to the reward of all finished tasks along path p minus the total traveling cost on that path, and L_p denotes the load of the rider at vertex o_p. Given these attributes, a path p including $k-1$ vertices connects the neighbors of o_p to generate paths go through k vertices. Paths while generating must satisfy following constraints:

- pairing: both start location and end location of the same task should be visited.
- priority: a task must be picked up before it be delivered.
- time window: a task should only be picked up after its start time and be delivered before its end time.
- capacity: the load of all picked up tasks for a given rider can not greater than the capacity of that rider.
- no cycle: a path should be elementary, that is, contains no cycle in it.

Algorithm 2 task selection and scheduling for rider $n \in \mathcal{N}$

```
1: starting from $w_n$, and set $\mathcal{H}_p = \{w_n\}, T_p = U_p = L_p = 0$, $o_p = w_n$
2: for $i = 2 : |\mathcal{O}_n|$ do
3:     using the paths along with the tuples generated in $i-1$ iteration to construct paths and tuples which pass $i$ vertices and satisfy all five constraints (pairing, priority, time window, capacity, no cycle)
4:     eliminate paths according to the dominance and non post-reachable criteria (Proposition 1 to 3)
5:     for each constructed path $p$, save its tuple $(\mathcal{H}_p, o_p, T_p, U_p, L_p)$
6:     delete the tuples saved in $i-1$ iteration after all paths containing $i$ vertices have been generated
7:     while $L_p = 0$ do
8:         save that path $p$ into the complete path set $\mathcal{P}_n^{'}$ , also save its tuple.
9:     end while
10: end for
11: find the path $p_n^*$ with maximum utility in $\mathcal{P}_n^{'}$ along with the task set $\mathcal{K}_n^*$
```

Some elimination criteria are applied to further accelerate the computational speed. Those criteria are based on the notion of dominance and non post-reachable path. Different from the a-priori elimination, the elimination of a tuple here will cause the elimination of its associated path rather than just an edge.

Proposition 1. *For two tuples $(\mathcal{H}_p, o_p, T_p, U_p, L_p)$ and $(\mathcal{H}_{p'}, o_{p'}, T_{p'}, U_{p'}, L_{p'})$, if $H_p = H_{p'}$, $o_p = o_{p'}$, $T_p \leq T_{p'}$, and $U_p \geq U_{p'}$, then the tuple $(\mathcal{H}_{p'}, o_{p'}, T_{p'}, U_{p'}, L_{p'})$ can be eliminated.*

This proposition implies that for two different paths p , $p^{'}$ who arrived at the same vertex $o_p = o_{p'}$ and passed the same set of vertices, if the path p arrives at vertex o_p earlier than path $p^{'}$, that means any vertex extended from path $p^{'}$ can also be extended from p. What's more, if the utility U_p is larger than the utility of path $p^{'}$, then we can say that the path p dominates $p^{'}$, and thus the path $p^{'}$ can be eliminated.

Proposition 2. *For two tuples $(\mathcal{H}_p, o_p, T_p, U_p, L_p)$ and $(\mathcal{H}_{p'}, o_{p'}, T_{p'}, U_{p'}, L_{p'})$, if $H_p \subseteq H_{p'}$, $o_p = o_{p'}$, $T_p \leq T_{p'}$*

, $U_p \geq U_{p'}$, and $L_p = L_{p'} = 0$, then the tuple $(\mathcal{H}_{p'}, o_{p'}, T_{p'}, U_{p'}, L_{p'})$ can be eliminated.

This proposition is an extension of proposition 3, which does not require the past vertices of two paths to be the same but only requires the set of past vertices of p is a subset of the vertices passed by $p^{'}$. If in addition, the load of the rider at vertex o_p is zero for both of the two paths, path p is also dominates path $p^{'}$. Thus the path $p^{'}$ can be eliminated.

Proposition 3. *For a tuple $(\mathcal{H}_p, o_p, T_p, U_p, L_p)$, if $i^+ \in \mathcal{H}_p$ and $i^- \notin \mathcal{H}_p$, and o_p can not be extended to i^-, then the tuple $(\mathcal{H}_p, o_p, T_p, U_p, L_p)$ can be eliminated.*

This proposition implies that a path p can be eliminated if it has already passed the task start location however, can not be extended to its corresponding task end location due to time window or pairing constraints. We call this kind of path non post-reachable path.

IV. Performance Evaluation

In this section, we conduct extensive experiments to evaluate the performance of our algorithm compared with the centralized optimal algorithm and a naive mechanism. We study the total utility, average utility per rider, task completion ratio and fairness of these schemes under various parameters. The main mechanisms implemented in our experiment are as bellows.

- TSS: our game-based scheme for the task selection and scheduling problem in food delivery.
- CTA: a centralized task assignment mechanism that can achieve global optimal social welfare.
- naive mechanism: each rider ranks all the tasks he is able to do from high to low, according to the utility it achieves if he only do that task, then he selects all the tasks he can do following the rank until the task loads reach his capacity. He then reports his choice (what tasks to do) to the platform. After getting all riders' task reports, the server assigns each task to the rider who has the highest utility to do it. Finally, the riders who have been assigned tasks do the assigned tasks from the nearest to the farthest. A rider must first pick up the task then deliver it and a task is failed if the rider can not deliver the task on time.

The dataset we used in our experiment is NYC Citi Bike Trips from Google Cloud Platform [12], as there is no published real world food delivery dataset. The NYC Citi Bike Trips dataset is the largest bike sharing program in US, with 10000 bikes and 600 stations across Manhattan, Queens, Brooklyn and Jersey City. We use the start station locations and end station locations in the dataset to simulate the task start locations and task end locations respectively, and use the start times and stop times in the dataset to simulate the task start times and task end times separately.

The parameters are set as follows without further instructions. The riders' initial locations are randomly generated within the range of area in the dataset, which is $10km \times 10km$ large. The riders' speeds are uniformly distributed within the range from $10km/h$ to $20km/h$. The reward for each task r_k is uniformly distributed in range $[10, 20]$. The traveling cost for unit distance c_0^n is set as a random variable uniformly distributed within the range $[1, 2]$. The weight for each task d_k is set to be 1 for simplicity. The capacity of each rider z_n is uniformly distributed in range $[1, 10]$. The parameters used in our experiments are summarized in Table 1.

In the simulation, each round is set to be 3 minutes, which means that the TSS algorithm is run every 3 minutes. The total run time for each experiment is 1 hour. There are totally 400 tasks generated in each simulation, and the tasks are continuously generated, which is fit for the real life food delivery scenarios. If the tasks are not delivered successfully in three minutes, they can be delivered in the next round until the deadline is passed. In the beginning of each round, the task set and path of the rider can be re-calculated, which means that the riders can dynamically change their path during delivery.

TABLE I
SIMULATION PARAMETERS

parameters	values
riders' initial locations w_n	distributed in $10km \times 10km$ area
traveling speed v_n	$10km/h$ to $20km/h$
reward for each task r_k	10 to 20
cost index c_0^n	$[1, 2]$
capacity z_n	$[1, 10]$

A. Total Utility and Rider's Average Utility

Figure 2 and Figure 3 illustrate the total utility and average utility against the number of riders. The results show that the total utility increases with the number of riders increasing. The reason is that with the number of riders increasing, more tasks can be selected. However, the average utility decreases with the increasing number of riders because the competition between riders becomes severe with more riders in the system.

Due to the high computation complexity of the centralized algorithm, it can only run in a relatively small scale scenario. In the rest of the paper, for comparisons with centralized algorithm, we all run it in a small scale application where the total duration is half an hour with only 100 tasks generated. For simplicity, we set all traveling speed to be the same $v_n = v = 20km/h$, the capacity to be $z_n = z = 5$, the unit cost to be $c_0^n = c_0 = 1$, and the reward for each task to be $r_k = 15$. The results in Figure 4 and Figure 5 show that the total utility and the average utility of our mechanism is very closed to the centralized algorithm CTA.

B. Task Completion Ratio

We compare the task completion ratio for different algorithms. Figure 6 shows the results of task completion ratio against the number of riders in general scale, while Figure 7 describes the results in the small scale scenario. Results imply that the task completion ratio increases with the increasing number of riders, because tasks can be selected by more riders. Our mechanism achieves similar task completion ratio

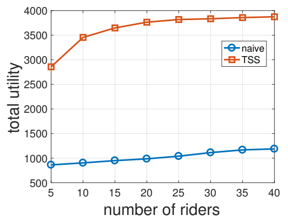
Fig. 2. The total utility vs. the number of riders in large scale.

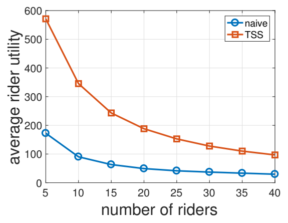
Fig. 3. The average rider utility vs. the number of riders in large scale.

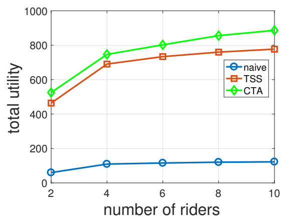
Fig. 4. The total utility vs. the number of riders in small scale.

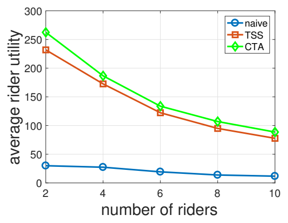
Fig. 5. The average rider utility vs. the number of riders in small scale.

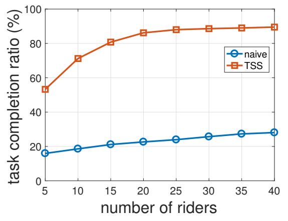
Fig. 6. The task completion ratio vs. the number of riders in large scale.

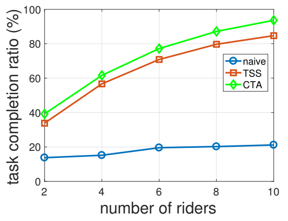
Fig. 7. The task completion ratio vs. the number of riders in small scale.

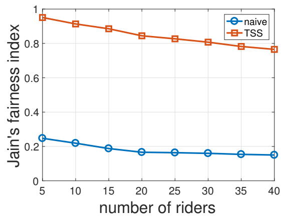
Fig. 8. The Jain's fairness index vs. the number of riders in large scale.

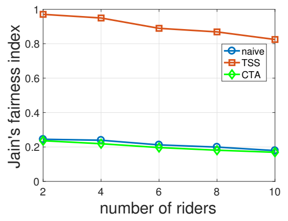
Fig. 9. The Jain's fairness index vs. the number of riders in small scale.

comparing with CTA mechanism. The task completion ratio of naive mechanism is low because not all tasks selected by riders can be finished.

C. *Fairness*

We use Jain's fairness index [9] $((\sum_{n\in\mathcal{N}} U_n)^2/(N\sum_{n\in\mathcal{N}} U_n^2))$ to examine the fairness of the schemes under various number of riders. The higher Jain's fairness index means the utilities are distributed more equally among riders, thus the system is more fair. Figure 8 and Figure 9 show that our mechanism achieves high Jain's fairness index, because our mechanism is based on game theory, each of the riders only responsible for maximizing its own utility without considering the total utility, thus more unwillingly to sacrifice its own revenue. With the number of riders increasing, the number of riders who can not select any task increases, which leads to the decrease of the Jain's fairness index.

V. Conclusions

In this paper, we study the task selection and scheduling problem in food delivery system in this paper. We formulate this problem as an online noncooperation game, and proposed a dynamic programming-based task selection and scheduling algorithm for riders to help them to decide the best response task set and task schedule. It is proved that our scheme can achieve the Nash Equilibrium in the task selection and scheduling game. Experimental results show that our scheme outperforms existing solutions in terms of the social welfare, average utility, task completion ratio, and fairness. Moreover, the social welfare attained by our scheme comes close to that of the centralized optimal solution.

Acknowledgement

This work was supported in part by the National Natural Science Foundation of China under Grant No. 61701216 and Shenzhen Science, Technology and Innovation Commission Basic Research Project under Grant No. JCYJ 20160531190935987 and Grant No. JCYJ 20160531191011045.

References

[1] Cormode, Graham, et al. "Differentially private spatial decompositions." Data engineering (ICDE), 2012 IEEE 28th international conference on. IEEE, 2012.
[2] Kazemi, Leyla, and Cyrus Shahabi. "Geocrowd: enabling query answering with spatial crowdsourcing." Proceedings of the 20th international conference on advances in geographic information systems. ACM, 2012.
[3] Deng, Dingxiong, Cyrus Shahabi, and Linhong Zhu. "Task matching and scheduling for multiple workers in spatial crowdsourcing." Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM, 2015.
[4] Cheung, Man Hon, et al. "Distributed time-sensitive task selection in mobile crowdsensing." Proceedings of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing. ACM, 2015.
[5] Chatzimilioudis, Georgios, et al. "Crowdsourcing with smartphones." IEEE Internet Computing 16.5 (2012): 36-44.
[6] Ganti, Raghu K., Fan Ye, and Hui Lei. "Mobile crowdsensing: current state and future challenges." IEEE Communications Magazine 49.11 (2011).
[7] Zhao, Yongjian, and Qi Han. "Spatial crowdsourcing: current state and future directions." IEEE communications magazine 54.7 (2016): 102-107.
[8] Lane, Nicholas D., et al. "A survey of mobile phone sensing." IEEE Communications magazine 48.9 (2010).
[9] Jain, Rajendra K., Dah-Ming W. Chiu, and William R. Hawe. "A Quantitative Measure of Fairness and Discrimination." Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA (1984).
[10] Deliveroo: https://deliveroo.ie
[11] Liu, Yan, et al. "Poster: FooDNet: Optimized On Demand Take-out Food Delivery using Spatial Crowdsourcing." Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking. ACM, 2017.
[12] NYC Citi Bike Trips: https://cloud.google.com/bigquery/public-data/nyc-citi-bike