Distributed Task Selection for Crowdsensing: a Game-Theoretical Approach

En Wang, Dongming Luan, Yuanbo Xu*, Yongjian Yang, and Jie Wu, *Fellow, IEEE*

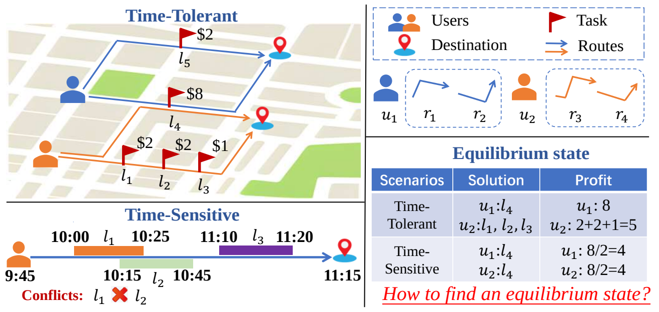
***Abstract*—中Mobile CrowdSensing (MCS) is a promising sensing paradigm that leverages users' mobile devices to collect and share data for various applications. A key challenge in MCS is task allocation, which aims to assign sensing tasks to suitable users efficiently and effectively. Existing task allocation approaches are mostly centralized, requiring users to disclose their private information and facing high computational complexity. Moreover, centralized approaches may not satisfy users' preferences or incentives. To address these issues, we propose a novel distributed task allocation scheme based on route navigation systems. We consider two scenarios: time-tolerant tasks and time-sensitive tasks, and formulate them as potential games. We design distributed algorithms to achieve Nash equilibria while considering users' individual preferences and the platform's task allocation objectives. We also analyze the convergence and performance of our algorithm theoretically. In the time-sensitive task scenario, the problem becomes even more intricate due to temporal conflicts among tasks. We prove the task selection problem is NP-hard and propose a distributed task selection algorithm. In contrast to existing distributed approaches that require users to deviate from their regular routes, our method ensures task completion while minimizing disruption to users. Trace-based simulation results validate that the proposed algorithm attains a Nash equilibrium and offers a total user profit performance closely aligned with that of the optimal solution.**

***Index Terms*—Mobile crowdSensing, route navigation, potential game, Nash equilibrium.**

I. Introduction

MOBILE CrowdSensing (MCS) [2], [3], [4] is a powerful sensing technique that has gained prominence with the proliferation of mobile devices. MCS approach leverages the capabilities of mobile devices to collect and share data for various sensing applications, including air quality monitoring [5], noise monitoring [6], and road traffic detection [7].

An important issue in MCS is how to efficiently allocate tasks to appropriate users, giving rise to the task allocation problem [8], [9]. A number of recent results adopt the centralized task allocation approach [10], [11], [12], [13], [14], [15], where the platform gathers user information in a centralized manner and subsequently makes task allocation decisions. We contend that this information collection process exposes users to potential privacy breaches, and finding the optimal task-user matches with spatial-temporal constraints is intricate.



Scenarios	Solution	Profit
Time-Tolerant	u_1:l_4 u_2:l_1, l_2, l_3	u_1: 8 u_2: 2+2+1=5
Time-Sensitive	u_1:l_4 u_2:l_4	u_1: 8/2=4 u_2: 8/2=4

Fig. 1: An illustrative example for distributed task selection.

Furthermore, centralized task allocation may not always result in user satisfaction. For instance, a user may be unwilling to deviate from their original route to complete a task assigned at a remote location, even if the task offers an attractive reward. Conversely, when users independently perform sensing tasks according to their own plans, certain tasks with low rewards or distant locations may remain incomplete.

Some existing works [16], [17] have proposed using distributed algorithms to address the multi-user task selection problem. However, these approaches do not consider users' original routes. Consequently, users still need to *deviate from their daily routes* to complete tasks. This approach brings additional movement costs to users and is intrusive to users. Furthermore, these works do not consider *users' individual preferences*, such as task rewards and detour distances.

Given the widespread use of map navigation systems (e.g., Google Maps [18]), we are inspired to explore distributed task allocation with the assistance of route navigation. The route navigation system recommends several routes to users after they input their initial locations and destinations in applications. Each route may involve various MCS tasks. When a user selects a route, the user can complete the tasks along that route and receive the corresponding task rewards. This approach allows users to choose routes for task completion in a distributed manner, rather than uploading their information to a centralized platform. Moreover, users complete tasks on the selected route without deviating from their daily routes.

We consider the following two scenarios: time-tolerant tasks and time-sensitive tasks [19], [20]. Time-tolerant tasks have flexible temporal requirements and can tolerate delays in data collection, such as noise monitoring [6] and air pollution monitoring [5]. Since time-tolerant tasks do not have strict temporal requirements, a user can complete all the tasks on a route one by one when passing by each task. Time-sensitive tasks, on the other hand, have strict temporal requirements, necessitating completion within specific time constraints. For

A conference version of the paper has appeared in Proceedings of ICPP 2021 [1].

En Wang, Dongming Luan, Yuanbo Xu, and Yongjian Yang are with the Department of Computer Science and Technology, Jilin University, Changchun, Jilin 130012, China. (E-mail: wangen@jlu.edu.cn; luandm20@mails.jlu.edu.cn; yuanbox@jlu.edu.cn; yyj@jlu.edu.cn)
Jie Wu is with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, USA. (E-mail:jiewu@temple.edu)
*The corresponding author is Yuanbo Xu.

example, checking the queue situation at a restaurant at 11:00 am [21] or assessing the on-shelf availability and price of Coca-Cola in a supermarket at 9:00 am [22]. In this case, users are required to collect data at precise times and locations, leading to temporal conflicts among tasks. As a result, users may be unable to complete all the tasks on their selected routes. The user must simultaneously select a route and choose the tasks to be performed along that route.

We illustrate an example in Fig. 1. In this scenario, there are two users, and the route navigation system recommends two routes for each of them. The service provider offers rewards to incentivize user participation [23], [24]. Due to the limited reward, task rewards are shared among users who complete them, resulting in *coupled* user task selections. Therefore, our objective is to find a stable equilibrium state in which no user has the incentive to change the decision to increase their profit unilaterally. In the time-tolerant task scenario depicted in the top-left part of Fig. 1, the final equilibrium state sees user u_1 selecting route r_2 and user u_2 selecting route r_4. Both users complete all tasks on their respective routes and receive a reward of \$8 and \$5 each. While in the time-sensitive task scenario in the bottom-left part of Fig. 1, user u_2 begins its journey at 9:45 and must reach the destination by 11:15. The user is required to start task l_1 at 10:00, with an estimated task duration of 25 minutes. Since l_2 should be started at 10:15, the temporal conflict arises between l_1 and l_2. As a result, different from the time-tolerant task scenario, user u_2 selects to perform l_4 and share the reward with u_1.

Since users need to compete for a limited reward, a user's task selection decision may affect other users' profits, resulting in coupled task selections. The first challenge lies in proving *the existence of an equilibrium state* and constructing a distributed model to *achieve an equilibrium while guaranteeing a lower performance bound*. Moreover, during the task selection process, each user has individual preferences. Similarly, the platform also has specific task allocation goals (e.g., maximizing task completion). Hence, the second challenge is how to design *a unified distributed algorithm* that considers the requirements of both the platform and users. Furthermore, in the time-sensitive task scenario, the user must simultaneously select a route and choose the tasks to be performed along that route. This complexity arises from temporal conflicts among tasks, making the task selection problem more intricate. Therefore, the third challenge is how to *find a solution to tackle these temporal conflicts* and reach an equilibrium state in the time-sensitive task scenario.

To address the aforementioned challenges, we first formulate the task selection problem as a multi-user route navigation game for the time-tolerant task scenario, where each user makes a decision to maximize their own profit function independently. Then, we prove that the formulated game is a potential game by constructing a global potential function. The change in the profit functions of all the users can be uniformly mapped into the change in the global potential function. By continuously approaching the maximum value of the global potential function, we achieve an equilibrium state where each user's profit function reaches a local maximum. Furthermore, we design a distributed game-theoretical algorithm to achieve the Nash equilibrium. For the profit function, the weighting parameters can be adjusted by the users based on their individual preferences, as well as by the platform according to the task allocation objectives. Finally, for the more intricate time-sensitive task scenario, we further design a branch and bound task selection algorithm and incorporate it into the potential game framework.

In summary, the contributions are listed as follows:

- *Distributed Task Selection Scheme:* Traditional distributed task allocation schemes are participatory crowdsensing, which require users to move to task locations, often deviating from their regular routes. Motivated by route navigation systems, we propose a new distributed crowdsensing task selection scheme that ensures task completion without necessitating users to deviate from their regular routes.
- *Multi-User Route Navigation Game:* We formulate the time-tolerant task selection problem as a potential game. To reach an equilibrium state, we design a distributed route navigation algorithm, where users can adjust the parameters of the profit function to align with their individual preferences, and the platform can do the same to achieve its task allocation purpose. Furthermore, we analyze its convergence and performance theoretically.
- *Distributed Task Selection Algorithm:* Temporal conflicts make the problem more complex for time-sensitive task selection. We have proven that both the centralized route navigation problem and the task selection problem are NP-hard. We formulate the problem as a multi-user task selection game and design a distributed task selection algorithm to reach a Nash equilibrium.
- *Extensive Trace-based Simulations:* We perform extensive trace-based simulations based on three data sets. The results verify that our proposed algorithm can achieve a Nash equilibrium, while achieving a total user profit close to that of the optimal solution.

The remainder of the paper is organized as follows: After reviewing the related works in Section II, we introduce the system model, the proof of NP-hardness, and the potential game formulation in Section III. Then, we propose the distributed route navigation algorithm for time-tolerant selection and analyze its performance theoretically in Section IV. Furthermore, we propose the task-sensitive selection algorithm in Section V. Finally, we conduct extensive simulations to evaluate the algorithm in Section VI and conclude the paper in Section VII.

II. RELATED WORKS

A. Crowdsensing task allocation

Crowdsensing tasks can be classified into two categories based on their time constraints:time-tolerant tasks and time-sensitive tasks [19], [20]. Time-tolerant tasks have flexible temporal requirements and can tolerate delays in data collection, such as noise monitoring and air pollution monitoring. For example, Yang *et al.* [21] design a PoI-based mobility prediction model to calculate the task completion probabilities and propose a greedy offline user selection algorithm. Dai *et al.* [25] construct a distributed matching model and design the budget-constrained task allocation strategies for MCS based on stable matching theory. Nguyen *et al.* [5] utilize the air quality sensors mounted on buses that move along their routes to broaden the sensing areas. Yin *et al.* [26] propose a task allocation scheme to maximize total rationality, where the rationality is measured by route distance, task similarity, and task priority.

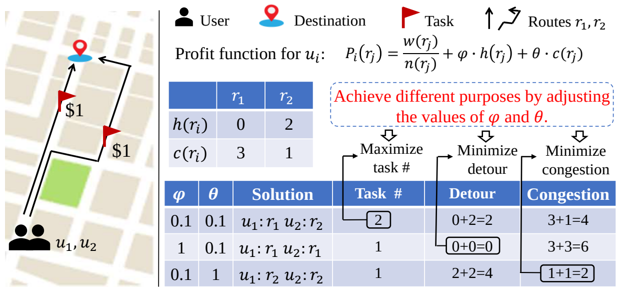
On the other hand, time-sensitive tasks come with stringent temporal requirements, necessitating the completion of tasks within specific time constraints. This is particularly relevant in scenarios such as real-time traffic monitoring and emergency response. For instance, Wang *et al.* [27] propose a time-sensitive task allocation approach, enabling the planning of task execution paths for participants. Cai *et al.* [28] delve into the complex problem of sensing task assignment and scheduling with multi-dimensional task diversity. They put forward a distributed auction scheme in which each task owner can autonomously process the auction procedure. Lai *et al.* [29] devise a duration-sensitive task allocation scheme aimed at maximizing the task completion ratio while adhering to task duration constraints. Dai *et al.* [30] design a decentralized deep reinforcement learning model tailored for delay-sensitive and energy-efficient UAV crowdsensing.

In contrast to the previous research, we introduce a distributed task selection approach that takes into account both time-tolerant and time-sensitive task scenarios. This approach guarantees task completion without requiring users to deviate from their daily routes.

B. Potential Game Applications

Recently, a lot of studies have utilized the potential game theory to derive distributed game-theoretical decisions and achieve the Nash equilibria. Chen *et al.* [31] propose a social recommendation-aided DSA framework, which exploits the temporal and spatial correlations for spectrum utilization and potential game theory to achieve a Nash equilibrium. Fabiani *et al.* [32] formulate the multi-vehicle driving coordination problem as a mixed-integer potential game and find an equilibrium solution. Hong *et al.* [33] formulate the computation offloading problem as a potential game in which the mobile devices make the offloading decisions in a distributed manner. He *et al.* [34] formulate the edge user allocation problem as a potential game and propose a decentralized algorithm to serve the maximum number of users with minimum overall system cost. Cheung *et al.* [16] investigate a distributed task selection problem in mobile crowdsensing and propose a distributed potential game algorithm that helps the users determine their task selections and mobility plans. Liu *et al.* [35] develop a potential game-based decision-making framework for autonomous driving, which provides theoretical guarantees for the existence of Nash equilibria. Varga *et al.* [36] propose a potential game approach for the design of a limited information-shared control system. The method is applied to the control of a large vehicle manipulator system.

However, in the above research, the corresponding potential game does not take the diverse requirements of both the platform and mobile users into consideration. In this paper, we propose a distributed game-theoretical approach, where both the platform and mobile users can achieve different purposes by adjusting the parameters of the profit function.



Profit function for u_i: $P_i(r_j) = \frac{w(r_j)}{n(r_j)} + \varphi \cdot h(r_j) + \theta \cdot c(r_j)$

	r_1	r_2
$h(r_i)$	0	2
$c(r_i)$	3	1

φ	θ	Solution	Task #	Detour	Congestion
0.1	0.1	$u_1{:}\,r_1\ u_2{:}\,r_2$	$\boxed{2}$	0+2=2	3+1=4
1	0.1	$u_1{:}\,r_1\ u_2{:}\,r_1$	1	$\boxed{0+0=0}$	3+3=6
0.1	1	$u_1{:}\,r_2\ u_2{:}\,r_2$	1	2+2=4	$\boxed{1+1=2}$

Fig. 2: An illustrative example of the influence of ϕ and θ.

III. System model and Problem formulation

A. Route Navigation for Time-Tolerant Tasks

We first introduce the route navigation model for time-tolerant tasks. We assume that there are a sufficient number of users to cover most of the tasks. When a user reports the initial location and destination to the platform, the platform recommends available routes to the user. Additionally, these routes potentially cover various tasks. Users can review the recommended routes and see which tasks these routes cover using their smartphones.

Let $\mathcal{U} = \{1, 2, \cdots, M\}$ be the set of users and $\mathcal{L} = \{1, 2, \cdots, N\}$ be the set of tasks covered by the routes of users. Each user i will receive a set of available routes from the platform, denoted as R_i, and each route $r \in R_i$ may cover a set of MCS tasks, denoted as $\mathcal{L}_r$. Each task $k \in \mathcal{L}$ is associated with a reward w_k, which is defined as follows:

$$w_k(x) = a_k + \mu_k \cdot \ln x, \tag{1}$$

where x is the number of users performing task k, a_k signifies an initial reward of task k, and μ_k acts as a weight parameter quantifying the increased degree in reward with the growing number of users. This parameter is constrained within the range of [0,1]. The task requester of task k could set initial reward a_k and weight parameter μ_k according to its own demand. Given that task completion quality is enhanced by accumulating results from multiple users [37], we design the reward function as follows: As more users engage in the sensing task, there is a slight increment in the task reward. This aligns with real-world practical scenarios.

In the context of strategy profile for all users $\mathbf{s} = (s_i, s_{-i})$, where s_i represents the decision of user i and s_{-i} represents the decisions of all users except user i. More specifically, the route selection decision s_i of user i in the time-tolerant task scenario is to select a route r from the recommended route set R_i, as a user can perform all the tasks on the selected route. The profit of user i under this strategy profile $\mathbf{s}$ is expressed as follows:

$$P_i(\mathbf{s}){=}\alpha_i{\cdot}\sum\nolimits_{k\in\mathcal{L}_{s_i}} w_k(n_k(\mathbf{s}))/n_k(\mathbf{s}){-}\beta_i\cdot d(s_i){-}\gamma_i{\cdot}b(s_i), \tag{2}$$

where α_i, β_i, γ_i represent user weight parameters that gauge a user's preferences. Users have the flexibility to adjust these

TABLE I: Main notations

Symbol	Meaning
$\mathcal{U}, \mathcal{L}$	the sets of mobile users, MCS tasks.
$\mathcal{L}_r$	the set of tasks that is covered by route r
$\mathbf{s}, s_i, s_{-i}$	the strategy profile for route selection, the route selection decision of user i, and the strategies of users except user i.
$\mathbf{L}, L_i, L_{-i}$	the task selection strategy profile, the task selection decision of user i, and the strategies of users except user i.
R_i	the recommended route set of user i.
$P_i(\mathbf{s})$	the profit of user i under the strategy profile $\mathbf{s}$.
$n_k(\mathbf{s})$	the number of users performing the task k under strategy profile $\mathbf{s}$.
$w_k(x)$	the reward of task k when the number of users performing the task is x.
$d(s_i), b(s_i)$	the cost incurred by detour distance, and the congestion level under strategy s_i.
$h(s_i), c(s_i)$	the detour distance, and the congestion level under strategy s_i.
$\alpha_i, \beta_i, \gamma_i$	the weight parameters controlled by user i.
φ, θ	the weight parameters controlled by the platform.
$\phi(\mathbf{s})$	the potential function.

parameters to align with their preferences. $e_{min} < \alpha_i$, β_i, $\gamma_i < e_{max}$, where $e_{min} > 0$. For example, if user i prefers a high reward, it can increase the value of α_i. We use $n_k(\mathbf{s})$ to represent the number of users performing task k under strategy profile $\mathbf{s}$. $d(s_i)$ is the cost incurred when traveling the detour distance of s_i, which is defined as follows:

$$d(s_i) = \varphi \cdot h(s_i), \tag{3}$$

where $h(s_i)$ is the detour distance of the selected route s_i compared to the shortest route between the initial location and the destination. φ is a weight parameter adjusted by the platform, where $0 < \varphi < 1$. $b(s_i)$ in Eq. (2) is the cost incurred by the congestion on the route s_i, which is defined as follows:

$$b(s_i) = \theta \cdot c(s_i), \tag{4}$$

where $c(s_i)$ is used to measure the congestion level of the selected route s_i. θ is a weight parameter controlled by the platform and $0 < \theta < 1$. The number of mobile users participating in crowdsensing on a certain route is usually far less than the considerable traffic flow on the road. Changes in user route selection strategies have a negligible impact on the congestion level of a road and can be ignored. Hence, we assume that the congestion level of route s_i selected by user i is irrelevant to other users' route selection decisions. In other words, we assume the congestion level of a route is the same under different route strategy profiles.

By adjusting the values of φ and θ, the platform can achieve different task allocation purposes. As shown in Fig. 2, we consider a simple case to demonstrate the influence of the weight parameters φ and θ, where each route only contains one task and the two users are at the same initial location. We observe the number of covered tasks, the detour distance and the congestion level with the change of φ and θ. The platform can decrease the values of both φ and θ to maximize the number of tasks covered by users. Moreover, by increasing the values of φ and θ, the platform can guide users to select the routes with short detour distance and low congestion level, respectively. Similarly, user i can also achieve individual preference by adjusting the values of α_i, β_i, γ_i in Eq. (2). In other words, when the platform has set the system parameters (φ and θ), user i can adjust its user weight parameters (α_i, β_i and γ_i) under these settings of the platform to achieve the individual preference.

B. NP-hardness of The Centralized Route Navigation Problem

We first consider the centralized optimization problem of finding a solution to maximize the total profit of all users. Mathematically, given strategy profile $\mathbf{s} = (s_i, s_{-i})$, the problem can be formulated as follows:

$$\max_{\mathbf{s}} \sum\nolimits_{i\in\mathcal{U}} P_i(\mathbf{s}), \quad \text{subject to } s_i \in R_i, \forall i \in \mathcal{U}. \tag{5}$$

Then, we try to prove that finding the optimal solution to the formulated centralized optimization problem is quite difficult, as shown in Theorem 1.

Theorem 1. *The centralized route navigation problem is NP-hard.*

Proof. The main idea is to reduce the maximum set cover problem, which is NP-complete [38], to a special case of our centralized route navigation problem.

First, we provide a definition of the maximum set cover problem. Given a universal set E of n elements, a collection of subsets of E, and an integer h, select h subsets so as to maximize the number of covered elements.

Then, we construct a special case of our centralized profit maximization problem with the following restrictions. Let $\mu_k = 0, a_k = a, \forall k \in \mathcal{L}$; that is, the reward of all tasks is a fixed value and exactly the same. Moreover, all users have the same recommended route set. Finally, set φ, θ to zero, and set $\alpha_i = 1, \forall i \in \mathcal{U}$. Correspondingly, $P_i(\mathbf{s}) = \sum_{k\in\mathcal{L}_{s_i}} \frac{a}{n_k(\mathbf{s})}$.

In the constructed special case, each user selects only one route. Consequently, the total number of selected routes is equivalent to the number of users. Since all tasks offer the same reward, we can maximize the total profit by selecting routes that cover the maximum number of tasks. These tasks can be viewed as elements in a maximum set cover problem. Each recommended route covers a subset of tasks, which directly corresponds to a subset of elements in the maximum set cover problem. As a result, the maximum set cover problem reduces to a special case of the centralized maximization problem, and this problem is NP-hard. □

According to the proof of Theorem 1, finding an optimal solution to our problem in a centralized manner is extremely difficult. Therefore, we turn to consider a distributed mechanism with low computational complexity. The idea that comes to mind is the application of game theory in distributed scenarios, which can lead to an equilibrium state. This inspiration motivates us to formulate the route navigation problem as a multi-user route navigation game.

C. Potential Game Formulation

To investigate the existence of Nash equilibrium, we try to formulate the multi-user route navigation game as a weighted potential game. Before the description about the potential game formulation, we first introduce some definitions.

Definition 1. ***(Better and best response update)*** *For a strategy profile* $\mathbf{s} = (s_i, s_{-i})$*, in the better response update, user* i *changes the strategy from* $s_i \in R_i$ *to* $s_i' \in R_i$*, which can lead to an increase of its profit, i.e.,* $P_i(s_i', s_{-i}) > P_i(s_i, s_{-i})$*. The best response update is a special type of the better response update. Each user* $i \in \mathcal{U}$ *will select a new strategy* s_i'*, which maximizes the profit among all better response updates.*

Definition 2. ***(Nash equilibrium)*** *In the multi-user route navigation game, a strategy profile* $\hat{\mathbf{s}} = (\hat{s}_1, \cdots, \hat{s}_M)$ *is a Nash equilibrium if and only if*

$$P_i(\hat{s}_i, \hat{s}_{-i}) = \max_{s_i \in R_i} P_i(s_i, \hat{s}_{-i}) \quad \forall i \in \mathcal{U}. \tag{6}$$

It is obvious that no user can improve the profit by altering the strategy unilaterally in a Nash equilibrium.

Definition 3. ***(Weighted potential game)*** *A game is defined as a weighted potential game if and only if there exists a potential function* $\phi(\mathbf{s})$ *such that* $\forall i \in \mathcal{U}, \forall s_i, \forall s_i' \in R_i$, $\forall s_{-i} \in \prod_{j \neq i} R_j$:

$$P_i(s_i, s_{-i}) - P_i(s_i', s_{-i}) = w_i(\phi(s_i, s_{-i}) - \phi(s_i', s_{-i})), \tag{7}$$

where $(w_i)_{i \in \mathcal{U}}$ *constitutes a vector of positive numbers.*

The potential game has two significant properties: (1) *Nash equilibrium existence:* there always exists at least one Nash equilibrium in the potential game. (2) *Finite improvement property:* the potential game always converges to a Nash equilibrium in a finite number of decision steps when taking better/best response updates, irrespective of the initial strategy profile or the users' updating order.

Then, Theorem 2 shows that the multi-user route navigation game is a weighted potential game.

Theorem 2. *The multi-user route navigation game is a weighted potential game and has a Nash equilibrium and the finite improvement property.*

Proof. We first construct the potential function as follows:

$$\phi(\mathbf{s}) = \sum_{k \in \mathcal{L}} \sum_{q=1}^{n_k(\mathbf{s})} \frac{w_k(q)}{q} - \sum_{i \in \mathcal{U}} \frac{\beta_i}{\alpha_i} d(s_i) - \sum_{i \in \mathcal{U}} \frac{\gamma_i}{\alpha_i} b(s_i). \tag{8}$$

We define the original strategy profile as $\mathbf{s} = (s_i, s_{-i})$ and a new strategy profile $\mathbf{s}' = (s_i', s_{-i})$ after user i changes the decision from s_i to s_i'. We can divide the set of tasks $\mathcal{L}$ into four non-overlapping portions: $\mathcal{L}_1 = \{k : k \in \mathcal{L}_{s_i}, k \in \mathcal{L}_{s_i'}\}$, $\mathcal{L}_2 = \{k : k \in \mathcal{L}_{s_i}, k \notin \mathcal{L}_{s_i'}\}$, $\mathcal{L}_3 = \{k : k \notin \mathcal{L}_{s_i}, k \in \mathcal{L}_{s_i'}\}$, and $\mathcal{L}_4 = \{k : k \notin \mathcal{L}_{s_i}, k \notin \mathcal{L}_{s_i'}\}$. Obviously, $\mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2 \cup \mathcal{L}_3 \cup \mathcal{L}_4$.

Then, we let F be defined as follows:

$$F = \sum_{i \in \mathcal{U}} \frac{\beta_i}{\alpha_i} d(s_i') + \sum_{i \in \mathcal{U}} \frac{\gamma_i}{\alpha_i} b(s_i') - \sum_{i \in \mathcal{U}} \frac{\beta_i}{\alpha_i} d(s_i) - \sum_{i \in \mathcal{U}} \frac{\gamma_i}{\alpha_i} b(s_i). \tag{9}$$

Furthermore, we have

$$\begin{aligned}
&\phi(\mathbf{s}) - \phi(\mathbf{s}') \\
&= \sum_{k \in \mathcal{L}} \sum_{q=1}^{n_k(\mathbf{s})} \frac{w_k(q)}{q} - \sum_{k \in \mathcal{L}} \sum_{q=1}^{n_k(\mathbf{s}')} \frac{w_k(q)}{q} + F \\
&= \sum_{k \in \mathcal{L}_1} \Big(\sum_{q=1}^{n_k(\mathbf{s})} \frac{w_k(q)}{q} - \sum_{q=1}^{n_k(\mathbf{s}')} \frac{w_k(q)}{q}\Big) \\
&\quad + \sum_{k \in \mathcal{L}_2} \Big(\sum_{q=1}^{n_k(\mathbf{s})} \frac{w_k(q)}{q} - \sum_{q=1}^{n_k(\mathbf{s}')} \frac{w_k(q)}{q}\Big) \\
&\quad + \sum_{k \in \mathcal{L}_3} \Big(\sum_{q=1}^{n_k(\mathbf{s})} \frac{w_k(q)}{q} - \sum_{q=1}^{n_k(\mathbf{s}')} \frac{w_k(q)}{q}\Big) \\
&\quad + \sum_{k \in \mathcal{L}_4} \Big(\sum_{q=1}^{n_k(\mathbf{s})} \frac{w_k(q)}{q} - \sum_{q=1}^{n_k(\mathbf{s}')} \frac{w_k(q)}{q}\Big) + F
\end{aligned} \tag{10}$$

Since user i switches from route s_i to route s_i', we have the following situations:

$$n_k(\mathbf{s}) - n_k(\mathbf{s}') = \begin{cases} 1, & k \in \mathcal{L}_2; \\ -1, & k \in \mathcal{L}_3; \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

Hence, according to Eq. (11), Eq. (10) can be derived as follows

$$\begin{aligned}
&\phi(\mathbf{s}) - \phi(\mathbf{s}') \\
&= \sum_{k \in \mathcal{L}_1} \Big(\frac{w_k(n_k(\mathbf{s}))}{n_k(\mathbf{s})} - \frac{w_k(n_k(\mathbf{s}'))}{n_k(\mathbf{s}')}\Big) \\
&\quad + \sum_{k \in \mathcal{L}_2} \frac{w_k(n_k(\mathbf{s}))}{n_k(\mathbf{s})} - \sum_{k \in \mathcal{L}_3} \frac{w_k(n_k(\mathbf{s}'))}{n_k(\mathbf{s}')} + F \\
&= \sum_{k \in \mathcal{L}_1 \cup \mathcal{L}_2} \frac{w_k(n_k(\mathbf{s}))}{n_k(\mathbf{s})} - \sum_{k \in \mathcal{L}_1 \cup \mathcal{L}_3} \frac{w_k(n_k(\mathbf{s}'))}{n_k(\mathbf{s}')} + F.
\end{aligned} \tag{12}$$

Since $\mathcal{L}_{s_i} = \mathcal{L}_1 \cup \mathcal{L}_2$ and $\mathcal{L}_{s_i'} = \mathcal{L}_1 \cup \mathcal{L}_3$ we have

$$\begin{aligned}
&\phi(\mathbf{s}) - \phi(\mathbf{s}') \\
&= \sum_{k \in \mathcal{L}_{s_i}} \frac{w_k(n_k(\mathbf{s}))}{n_k(\mathbf{s})} - \sum_{k \in \mathcal{L}_{s_i'}} \frac{w_k(n_k(\mathbf{s}'))}{n_k(\mathbf{s}')} + F.
\end{aligned} \tag{13}$$

Finally, according to Eq. (13), the following equation holds.

$$P_i(\mathbf{s}) - P_i(\mathbf{s}') = \alpha_i(\phi(\mathbf{s}) - \phi(\mathbf{s}')). \tag{14}$$

Hence, Theorem 2 is proven. □

According to Theorem 2, the multi-user route navigation game is a weighted potential game, in which the change in the profit of each user can be mapped to the potential function. Each user's decision update in each decision slot results in an increase in their profit function, consequently raising the potential function value. When the potential function value reaches its maximum, a Nash equilibrium is achieved.

D. Time-Sensitive Task Selection Model

We introduce time-sensitive task selection in this subsection. Time-sensitive tasks come with strict temporal requirements, mandating that the task be completed within specific time constraints. We denote t_k as the start time and g_k as the task duration of task k, respectively. Therefore, the estimated end time $e_k = t_k + g_k$. A user i starts from an origin at time t_i and should reach the destination by time e_i. We denote t_i^r as the time spent by user i traveling to the destination via route r without performing any tasks. Therefore, the spare time, denoted as z_i^r, for user i on route r is calculated as

$z_i^r = e_i - t_i - t_i^r$. In other words, z_i^r represents the spare time available to user i for task execution when selecting route r. Due to the temporal conflicts among tasks with different time constraints, a user may be unable to perform all the tasks on the selected route. Consequently, different from the time-tolerant task scenario, user i's decision includes simultaneously selecting a route $s_i \in R_i$ and choosing the tasks to be performed along that route, denoted as (s_i, L_i), where L_i represents the set of selected tasks on route s_i.

We represent the strategy profile for all users as $(\mathbf{s}, \mathbf{L})$, where $\mathbf{s} = (s_i, s_{-i})$ and $\mathbf{L} = (L_i, L_{-i})$ denote route decisions and task decisions for all users, respectively. The profit of user i under strategy profile $(\mathbf{s}, \mathbf{L})$ is expressed as follows:

$$P_i(\mathbf{s},\mathbf{L})=\alpha_i \cdot \sum_{k \in L_i} \frac{w_k(n_k(\mathbf{s},\mathbf{L}))}{n_k(\mathbf{s},\mathbf{L})} - \beta_i \cdot d(s_i) - \gamma_i \cdot b(s_i). \quad (15)$$

Furthermore, we demonstrate that the multi-user task selection game is also a weighted potential game, as demonstrated in Theorem 3.

Theorem 3. *The multi-user task selection game is a weighted potential game and has a Nash equilibrium and the finite improvement property.*

Proof. The proof of Theorem 3 can be constructed using a method similar to that used in proving Theorem 2. Therefore, we will not provide it here. □

Unlike time-tolerant task selection, where users only need to select routes, users in this context are faced with the dual challenge of not only choosing a route but also selecting the tasks along that route. Even after the route selection, they must further determine the set of tasks on the chosen route to maximize profit. We have demonstrated that the centralized route navigation problem is NP-hard in Theorem 1. Then, we demonstrate that the time-sensitive task selection on a selected route is also NP-hard, as demonstrated in Theorem 4.

Theorem 4. *The time-sensitive task selection problem on a route is NP-hard.*

Proof. The central idea is to reduce the Knapsack Problem with Conflicts (KPC), known to be NP-complete [38], to a specific case of our task selection problem.

To begin with, we provide a definition of the KPC problem. In the KPC problem, there is a knapsack with a fixed capacity C and a set of items $V = \{1, \cdots, n\}$, where each item k has a profit p_k and a weight w_k. Additionally, there is a conflict graph $G = (V, E)$, where V represents the set of n items, and an edge $i, j \in E$ signifies that items i and j cannot be placed into the knapsack simultaneously. The objective in the KPC problem is to select a subset of non-conflicting items, denoted as $S \subseteq V$, in such a way that the total profit of the selected items is maximized, while ensuring that the total weight does not exceed the given knapsack capacity C.

Next, we construct a specialized instance of our task selection problem, subject to certain constraints. For a user and a set of tasks on a route, each task corresponds to an item, with its task duration reflecting the weight of the item. The spare time for the user corresponds to the knapsack capacity. We create a conflict graph based on temporal conflicts among tasks. Our objective is to select a subset of non-conflicting tasks, maximizing the total task profits while ensuring that the total task duration does not exceed the user's allotted spare time duration.

By doing so, we effectively reduce the KPC problem to our task selection problem on a route, establishing the NP-hardness of this problem. □

Algorithm 1 Distributed Game-Theoretical Route Navigation Algorithm for user $i \in \mathcal{U}$.

```
1: Input $\alpha_i$, $\beta_i$, $\lambda_i$, the initial location and the destination.
2: Receive the recommended routes $R_i$.
3: Initialize $s_i(0)=r$ by randomly selecting a route $r \in R_i$.
4: Report $s_i(0)$ to the platform.
5: Receive $n_k$ for each task $k$ that is covered by $s_i(0)$.
6: Calculate the profit $P_i$.
7: Receive $d(r)$ and $b(r)$ for each route $r$ in $R_i$.
8: repeat for each decision slot $t$
9:     Obtain $n_k$ for each task $k$ that is covered by $R_i$.
10:    Compute the best route set $\triangle_i(t)$.
11:    if $\triangle_i(t) \neq \varnothing$ then
12:        Send the request to contend the opportunity for updating decision.
13:        if Win the opportunity then
14:            Update the route selection decision $s_i(t)$ by selecting a route $r \in \triangle_i(t)$.
15:            Report $s_i(t)$ to the platform.
16:        else
17:            Choose the original decision $s_i(t) = s_i(t-1)$.
18: until The termination message is received.
```

According to Theorem 4, the presence of temporal conflicts among tasks significantly amplifies the problem's complexity. To address this challenge, it is imperative that we develop a distributed game theoretical task selection algorithm for time-sensitive task selection.

IV. Time-tolerant task selection

To reach a Nash equilibrium, we introduce the following three algorithms: the distributed route navigation algorithm (Algorithm 1), the information update algorithm (Algorithm 2), and the parallel user update algorithm (Algorithm 3). The distributed route navigation algorithm runs on each user's smartphone, helping each user autonomously select a route from the recommended route set to maximize their own profits. More specifically, the algorithm iteratively reaches a Nash equilibrium by selecting a user to update their route selection decision in each decision slot. The information update algorithm runs on the platform, updating and exchanging information, and selecting users to update their decisions in each decision slot. To accelerate the convergence of the algorithm, the parallel user update algorithm is proposed, allowing multiple users to update their decisions in each decision slot.

Algorithm 2 Information Update Algorithm for the platform.

```
1:  Send the recommended route set $R_i$ to the user $i \in \mathcal{U}$.
2:  Receive $s_i(0)$ from each user $i \in \mathcal{U}$.
3:  Calculate $n_k$ for each task $k \in \mathcal{L}$.
4:  Send $n_k$, $d(r)$ and $b(r)$ to the corresponding user.
5:  repeat for each decision slot $t$
6:      Receive the request from the users and let $\mathcal{U}'$ denote the set of users that send the request.
7:      if $\mathcal{U}' \neq \varnothing$ then
8:          Select a set of users $\mu$ by SUU or PUU algorithm.
9:          Inform the users in $\mu$ to update the decisions.
10:         Receive $s_i(t)$ from user $i \in \mu$ and update $n_k$ for each task $k \in \mathcal{L}$.
11: until No request is received from the user.
12: Send the termination message to all users.
```

A. Distributed Route Navigation Algorithm

Theorem 2 guarantees that the multi-user route navigation game converges to a Nash equilibrium within a finite number of decision slots. The main idea of the distributed route navigation algorithm is to utilize the finite improvement property and select a set of mobile users to improve their profits by updating their route selection decisions in each decision slot.

In the initialization phase (lines 1-7) of Algorithm 1, the mobile user first inputs the weight parameters of preferences α_i, β_i, λ_i, as well as the initial location and the destination (line 1). Then, the recommended routes will be sent to the mobile application (line 2). We use $s_i(t)$ to represent the route decision of user i in decision slot t. The algorithm initializes the route selection decision by randomly selecting a route from the recommended route set and calculates the profit (lines 3-6). Finally, the mobile application receives the detour distance $d(r)$ and congestion level $b(r)$ of each route r in the recommended route set (line 7).

In the calculation phase (lines 8-18), each user obtains the information on the number of users performing each task covered by R_i (line 9). The algorithm then calculates the best route set $\triangle_i(t)$ (line 10). The best route set is defined as the set of route decisions that maximize the profit of the user and can improve the profit compared to the previous decision slot.

If the best route set $\triangle_i(t) \neq \varnothing$, which means that the user can improve the profit by altering the route selection decision, the user sends a request to the platform for updating the decision. If the user wins the opportunity to update the decision, it selects a route from the best route set to update the current decision. Otherwise, the user will maintain the decision consistent with the previous decision slot (lines 11-17). The calculation process repeats until the termination message is received from the platform (line 18). It is worth noting that when all the users receive the termination message, the algorithm converges to a Nash equilibrium, and the mobile users achieve mutually satisfactory decisions.

B. Information Update Algorithm

The information update algorithm updates the number of users that perform each task and interacts with the mobile users, such as exchanging the information and selecting a set of users to update the decision in each decision slot.

Algorithm 3 Parallel User Update Algorithm.

```
Input: $\mathcal{U}'$, $\boldsymbol{\tau}$, $\mathbf{B}$.
1: Initialize $\mu = \varnothing$, $\sigma = \varnothing$.
2: Calculate $\delta_i = \frac{\tau_i}{|B_i|}, \forall i \in \mathcal{U}'$.
3: Sort $i \in \mathcal{U}'$ in a non-ascending order of $\delta_i$.
4: for all $i \in \mathcal{U}'$ do
5:     if $\sigma \cap B_i = \varnothing$ then $\mu \leftarrow \mu \cup i$, $\sigma \leftarrow \sigma \cup B_i$.
   return $\mu$.
```

As shown in Algorithm 2, in the initialization phase (lines 1-4), the information update algorithm first sends the recommended routes to each user (line 1). After receiving the initial decisions from all the users (line 2), the algorithm calculates the number of users (n_k) that perform each task $k \in \mathcal{L}$ (line 3). Finally, the information that is required to calculate the profit is sent to the user (line 4).

Next, in each decision slot, the platform receives the requests from the users (line 6). Then, the platform utilizes Single User Update algorithm (SUU) or Parallel User Update algorithm (PUU) to determine the set of users to update their decisions and informs the selected users to update the decision (lines 8-9). When receiving the updated decision from the user, the algorithm updates the number of participants in each task $k \in \mathcal{L}$ (line 10). When no request is received from the users in a decision slot, the algorithm sends the termination messages to all users and the information update algorithm terminates (lines 11-12).

Furthermore, we introduce the above two user update algorithms, SUU and PUU. SUU algorithm randomly selects only one user from the set of users that send the requests to the platform and allows the user to update the decision in each decision slot. To decrease the convergence time, we further propose PUU algorithm, which is inspired by the idea that some users whose selected routes cover no overlapping tasks could concurrently update their route selections in the same decision slot. This leads to a larger increase in the potential function value in each decision slot and reduces the convergence time.

The detailed description is as follows. As shown in Algorithm 3, the inputs are $\mathcal{U}'$, $\boldsymbol{\tau}$ and $\mathbf{B}$. Specifically, $\mathcal{U}'$ is the set of users sending the requests to update the decisions, $\boldsymbol{\tau} = \{\tau_i, \forall i \in \mathcal{U}'\}$ and $\mathbf{B} = \{B_i, \forall i \in \mathcal{U}'\}$. Let s_i denote the original strategy of user i and s_i' denote a new strategy that maximizes the profit in the best route set. We use B_i to denote the union set of tasks covered by s_i and s_i', and $\tau_i = (P_i(s_i', s_{-i}) - P_i(s_i, s_{-i}))/\alpha_i$. For each user $i \in \mathcal{U}'$, it sends B_i and τ_i to the platform. PUU algorithm first calculates δ_i for each user $i \in \mathcal{U}'$ (line 2) and sorts the users in $\mathcal{U}'$ in a non-ascending order of δ_i (line 3). Then, the algorithm greedily chooses the set of users μ, which maximizes the sum of τ_i and satisfies the constraint that the covered task set B_i of $i \in \mu$ does not intersect with each other (lines 4-5). After inspecting all the users in $\mathcal{U}'$, the algorithm returns the user set μ and informs the users in μ to update decisions concurrently.

$\sum_{i \in \mu} \tau_i$ corresponds to the increase of the potential function value. PUU algorithm greedily selects a set of users with the objective of maximizing the value of $\sum_{i\in\mu}\tau_i$ in each decision slot. We then analyze the performance of PUU algorithm theoretically. Let $\hat{\mu}$ denote the set of selected users to update the route decisions of the optimal solution, which maximizes the value of $\sum_{i\in\mu}\tau_i$. Let $\tau=\sum_{i\in\mu}\tau_i$ and $\hat{\tau}=\sum_{i\in\hat{\mu}}\tau_i$. Theorem 5 analyzes the performance bound of PUU algorithm.

Theorem 5. *The performance bound between the PUU algorithm and the optimal solution maximizing the value of $\sum_{i\in\mu}\tau_i$ satisfies the following relation:*

$$\tau/\hat{\tau}\geq |B_{i'}|/(|\hat{\mu}|\cdot B_{max}), \tag{16}$$

where $i'=\arg\max_{i\in\mu}\delta_i$, $B_{max}=\max_{i\in\hat{\mu}}|B_i|$, and $|\hat{\mu}|$ represents the number of selected users needed to update the decisions in the optimal solution.

Proof. The PUU algorithm first selects the user that has the maximum value of δ_i in $\mathcal{U}'$, $\delta_{i'}\geq\tau_i/|B_i|\ \forall i\in\hat{\mu}$. Hence, the following equation holds.

$$|\hat{\mu}|\tau_{i'}/|B_{i'}|\geq\sum\nolimits_{i\in\hat{\mu}}\tau_i/|B_i|. \tag{17}$$

Since $B_{max}=\max_{i\in\hat{\mu}}|B_i|$, we get Eq. (18).

$$\sum\nolimits_{i\in\hat{\mu}}\tau_i/|B_i|\geq\hat{\tau}/B_{max}. \tag{18}$$

There exists $\tau\geq\tau_{i'}$. According to Eq. (17) and Eq. (18), we get the following equation by rearranging the items.

$$\tau/\hat{\tau}\geq|B_{i'}|/(|\hat{\mu}|\cdot B_{max}). \tag{19}$$

Hence, Theorem 5 is proved. □

Theorem 5 demonstrates a theoretical guarantee of the performance of PUU algorithm. It is obvious that the increase of the potential function value achieved by PUU algorithm is not lower than $\hat{\tau}\cdot|B_{i'}|/(|\hat{\mu}|\cdot B_{max})$, even in the worst case. As shown in Theorem 2, the change in the profit of each user can be mapped into the potential function. The faster the potential function value grows, the faster the algorithm converges. Therefore, PUU algorithm speeds up the convergence for a Nash equilibrium. As shown in Theorem 2, the change in the profit of each user can be mapped to the potential function. The faster the potential function value grows, the faster the algorithm converges. Therefore, Algorithm 3 speeds up the convergence for a Nash equilibrium.

The above algorithms proceed in an iterative way. In each decision slot, Algorithm 1 receives the information from the user and the platform. Then, Algorithm 1 calculates if the user could update the decision to increase the profit and send the updating decision request to the platform. Algorithm 2 on the platform updates the information and selects a user (SUU algorithm) or a set of users (PUU algorithm, i.e., Algorithm 3) to update the decision. The algorithms will finally converge to a Nash equilibrium.

The proposed time-tolerant task selection algorithm is a distributed algorithm from the computation perspective. Most of the computation load is distributed to users. Each user calculates the profit and makes the decision locally to maximize a local profit. The centralized platform cannot control users. It merely selects a random user to update the decision and transmits the parameters among users.

C. Convergence Analysis

According to Theorem 2, the proposed distributed route navigation algorithm will converge to a Nash equilibrium within a finite number of update iterations. We then analyze the upper bound of the iteration number for convergence. Let $\mathcal{S}$ denote the strategy space of all the users. For $\forall k\in\mathcal{L},\forall\mathbf{s}\in\mathcal{S}, 1\leq q\leq n_k(\mathbf{s})$, let $g_{min}=\min\{w_k(q)/q\}$, $g_{max}=\max\{w_k(q)/q\}$. There exists $0\leq d(s_i)\leq d_{max},\forall i\in\mathcal{U}$ and $0\leq b(s_i)\leq b_{max},\forall i\in\mathcal{U}$. We denote the minimum change value of the users' profit when the user updates the decision as ΔP_{min}. For the number of decision slots for convergence, the following theorem holds.

Theorem 6. *The number of decision slots C for convergence of the distributed route navigation algorithm satisfies the following equation.*

$$C<\frac{e_{max}}{\Delta P_{min}}|\mathcal{U}|(|\mathcal{L}|(g_{max}-g_{min})+\frac{e_{max}}{e_{min}}d_{max}+\frac{e_{max}}{e_{min}}b_{max}). \tag{20}$$

Proof. During a decision slot, consider the worst case where there is only a user $i\in\mathcal{U}$ who alters the current strategy s_i to s_i', which leads to an increase in its profit function, i.e., $P(s_i,s_{-i})<P(s_i',s_{-i})$. According to Eq. (8), we have:

$$\phi(\mathbf{s})>|\mathcal{L}||\mathcal{U}|g_{min}-|\mathcal{U}|e_{max}d_{max}/e_{min}-|\mathcal{U}|e_{max}b_{max}/e_{min}. \tag{21}$$

$$\phi(\mathbf{s})<|\mathcal{L}||\mathcal{U}|g_{max}. \tag{22}$$

Furthermore, we have:

$$\phi(\mathbf{s}')-\phi(\mathbf{s})<|\mathcal{U}|(|\mathcal{L}|(g_{max}-g_{min})+\frac{e_{max}}{e_{min}}d_{max}+\frac{e_{max}}{e_{min}}b_{max}). \tag{23}$$

According to Eq. (14), the change of the value of the potential function is equivalent to the change of a user's profit divided by α_i. Therefore, for the number of decision slots C for convergence, we have the following results:

$$C<\frac{e_{max}}{\Delta P_{min}}|\mathcal{U}|(|\mathcal{L}|(g_{max}-g_{min})+\frac{e_{max}}{e_{min}}d_{max}+\frac{e_{max}}{e_{min}}b_{max}). \tag{24}$$

Hence, Theorem 6 is proved. □

From Theorem 6, to accelerate the convergence time, we can select a set of users with no overlapping tasks to concurrently update their route decisions in each decision slot as shown in Algorithm 3. Since each selected user can increase the profit by updating the decision in parallel, which leads to an increase in the value of the potential function, the potential function reaches the maximum value quickly. Hence, the convergence time is decreased.

D. Theoretical Analysis

We then analyze the performance of the proposed distributed route navigation algorithm by analyzing Price of Anarchy (PoA) [34]. PoA is a metric measured by the ratio of the total profit of all users in the worst case of Nash equilibrium to the maximum total profit of the optimal strategy, as defined in Eq. (25). By analyzing PoA, we quantify the efficiency of the worst-case Nash equilibrium over the centralized optimal solution in terms of the total profit. Let S' be the set of strategy profiles that can achieve Nash equilibrium and $\mathbf{s}^*$ denote the centralized optimal strategy.

$$PoA = \min_{\mathbf{s}\in S'} \sum\nolimits_{i\in\mathcal{U}} P_i(\mathbf{s}) / \sum\nolimits_{i\in\mathcal{U}} P_i(\mathbf{s}^*). \quad (25)$$

We consider a special case of the multi-user route navigation game with the following restrictions. First, each route only covers one task, and a task may be covered by several routes. Second, the recommended route set R_i for each user $i \in \mathcal{U}$ contains two parts: r_i', and R, where the task on route r_i' is only covered by r_i' and R is a set of routes that covers the same set of tasks for all users. The common task set covered by R is denoted as $\mathcal{L}'$. In other words, the task on r_i' is only covered by user i and the tasks in $\mathcal{L}'$ can be covered by all users. Third, the reward of a task $k \in \mathcal{L}'$ is defined as $w_k = a + \ln x$, $a > 0$, and we do not make any restrictions for the reward of other task $k \notin \mathcal{L}'$. Let k_i denote the task covered by the route decision s_i for user i. Since a route only covers one task, the profit for each user i is calculated as $P_i(\mathbf{s}) = w_{k_i}(n_{k_i}(\mathbf{s}))/n_{k_i}(\mathbf{s})$. Specifically, the profit that user i achieves by selecting route r_i' is denoted as P_i.

Theorem 7. *For the multi-user route navigation game, the PoA metric of the overall profits satisfies that*

$$\frac{\sum_{i\in\mathcal{U}} \max\{P_i, P_i^{min}\}}{\sum_{i\in\mathcal{U}} \max\{P_i, P_i^{max}\}} \le PoA \le 1, \quad (26)$$

where $P_i^{min} = \frac{a+\ln p}{p}$, $p = \frac{|\mathcal{U}|+|\mathcal{L}'|-1}{|\mathcal{L}'|}$, $P_i^{max} = a$.

Proof. There exists $n_{k_i}(\mathbf{s}) = 1 + \sum_{j\in\mathcal{U}\setminus\{i\}} I\{k_j = k_i\}$. $I\{E\}$ is an indicator function, where $I\{E\} = 1$ if the event E is true and $I\{E\} = 0$ otherwise. In the formulated special case, since a route only covers one task, the profit of a route decreases with the growth of the number of users performing the task on that route. Since no user can increase the profit by changing the decision unilaterally in Nash equilibrium, considering a strategy profile $\mathbf{s}$ that achieves a Nash equilibrium and user i selects a route in R, the following equation holds:

$$\sum\nolimits_{j\in\mathcal{U}\setminus\{i\}} I\{k_j = k_i\} \le \sum\nolimits_{j\in\mathcal{U}\setminus\{i\}} I\{k_j = k\} \ \forall k \in \mathcal{L}'. \quad (27)$$

Furthermore, we have:

$$|\mathcal{L}'|(\sum\nolimits_{j\in\mathcal{U}\setminus\{i\}} I\{k_j{=}k_i\}) \le \sum\nolimits_{k\in\mathcal{L}'} \sum\nolimits_{j\in\mathcal{U}\setminus\{i\}} I\{k_j{=}k\}. \quad (28)$$

According to Eq. (28), we substitute and rearrange the corresponding terms. The following equation holds:

$$\sum\nolimits_{j\in\mathcal{U}\setminus\{i\}} I\{k_j = k_i\} \le (|\mathcal{U}| - 1)/|\mathcal{L}'|. \quad (29)$$

Based on Eq. (29), there exists $n_{k_i}(\mathbf{s}) \le (|\mathcal{U}|+|\mathcal{L}'|-1)/|\mathcal{L}'|$. Furthermore, the following equation holds.

$$P_i(\mathbf{s}) \ge \frac{a + \ln((|\mathcal{U}| + |\mathcal{L}'| - 1)/|\mathcal{L}'|)}{(|\mathcal{U}| + |\mathcal{L}'| - 1)/|\mathcal{L}'|)}. \quad (30)$$

We use p to denote $\frac{|\mathcal{U}|+|\mathcal{L}'|-1}{|\mathcal{L}'|}$ and P_i^{min} to denote $\frac{a+\ln p}{p}$. As mentioned above, if user i selects r_i', the profit is P_i. Hence, $P_i(\mathbf{s}) \ge \max\{P_i, P_i^{min}\}$.

On the other hand, due to the fact that $n_{k_i}(\mathbf{s}) \ge 1$, there exists $P_i(\mathbf{s}) \le a$. Let $P_i^{max} = a$. Furthermore, we can conclude that $P_i(\mathbf{s}^*) \le \max\{P_i, P_i^{max}\}$.

In conclusion, according to the above description, the following equation holds:

Algorithm 4 FindTask Algorithm.

```
Input: $\rho$, $\xi$, $L_s$, $F$, $v_l$, $B$
Output: $B$
1: if $\rho > v_l$ then
2:     $v_l \leftarrow \rho$;
3:     $B \leftarrow L_s$;
4: $v_u \leftarrow \rho$;
5: $c \leftarrow \xi$;
6: $k \leftarrow \Omega_0(F)$;
7: while $c < z_i^r$ and $k \le \Omega_{-1}(F)$ do
8:     if $c + \xi_k \le z_i^r$ then
9:         $v_u \leftarrow \rho_k + v_u$;
10:        $c \leftarrow \xi_k + c$;
11:        $k \leftarrow \Omega_k(F)$;
12:    else
13:        $v_u \leftarrow v_u + (z_i^r - c)\cdot(\rho_k/\xi_k)$;
14:        $c \leftarrow z_i^r$;
15: if $v_u \le v_l$ then return $B$;
16: $k \leftarrow \Omega_0(F)$;
17: while $\rho + (z_i^r - \xi)\cdot(\rho_k/\xi_k) > v_l$ and $k \le \Omega_{-1}(F)$ do
18:     $j \leftarrow \Omega_k(F)$;
19:     $F \leftarrow F \setminus \{k\}$;
20:     if $\xi + \xi_k \le z_i^r$ then
21:         $L_s' \leftarrow L_s \cup \{k\}$;
22:         $\hat{F} \leftarrow F \setminus C_k$;
23:         $B \leftarrow FindTask(\rho + \rho_k, \xi, L_s', \hat{F}, v_l, B)$;
24:     $k \leftarrow j$;
25: return $B$.
```

$$\frac{\sum_{i\in\mathcal{U}} \max\{P_i, P_i^{min}\}}{\sum_{i\in\mathcal{U}} \max\{P_i, P_i^{max}\}} \le PoA \le 1. \quad (31)$$

□

V. TIME-SENSITIVE TASK SELECTION

In this section, we introduce the distributed task selection algorithm for the time-sensitive task scenario. In the time-tolerant task scenario, users update their decisions by selecting a route in each decision slot. When it comes to calculating the profit for a chosen route, users simply add up the shared rewards of all tasks along that route. While in the time-sensitive task scenario, the strict temporal constraints introduce temporal conflicts among tasks on a route. Here, users face the challenge of simultaneously selecting a route and choosing a set of tasks that are conflict-free along that chosen route. The objective remains consistent: maximize the profit while updating the decision. When calculating the profit for a route, users must carefully select a set of tasks that do not conflict with one another in order to maximize the profit on the route.

For the tasks included in a route, we must first perform a task filtering operation to remove tasks that do not meet one of the following two conditions: (1) the task has a start time earlier than the user's start time; (2) the task has an end time later than the user's deadline. The user will be unable to complete a task that satisfies these conditions.

Algorithm 5 Distributed Game-Theoretical Task Selection Algorithm for user $i \in \mathcal{U}$.

```
1: Input $\alpha_i$, $\beta_i$, $\lambda_i$, the initial location and the destination.
2: Receive the recommended routes $R_i$ and task set $L_i$.
3: Initialize $s_i(0)=r$ by randomly selecting a route $r \in R_i$.
4: Calculate the optimal task set $L_i(0)$ on route $s_i(0)$ using Algorithm 4.
5: Report $(s_i(0), L_i(0))$ to the platform.
6: Receive $n_k$ for each task $k$ that is in $L_i(0)$.
7: Calculate the profit $P_i$.
8: Receive $d(r)$ and $b(r)$ for each route $r$ in $R_i$.
9: repeat for each decision slot $t$
10:     Obtain $n_k$ for each task $k$ that is covered by $R_i$.
11:     Calculate the optimal task set and the corresponding profit for each route $r$ in $R_i$ using Algorithm 4.
12:     Obtain the best route set $\triangle_i(t)$.
13:     if $\triangle_i(t) \neq \varnothing$ then
14:         Send the request to contest the opportunity to update the decision.
15:         if Win the opportunity then
16:             Update the current task selection decision $(s_i(t), L_i(t)) = (r, L_r)$ by selecting a route $r \in \triangle_i(t)$.
17:             Report $(s_i(t), L_i(t))$ to the platform.
18:         else
19:             Choose the original decision $(s_i(t), L_i(t)) = (s_i(t-1), L_i(t-1))$.
20: until The termination message is received.
```

In order to select a set of tasks on a route without temporal conflicts and maximize the profit when updating the decisions, we then devise FindTask Algorithm to help users make the task selection decision. The algorithm is demonstrated in Algorithm 4. The potential solution space can be constructed as a tree structure. The primary goal is to systematically explore the search space by traversing the tree structure in a depth-first manner. Each leaf node and its parent nodes in the tree correspond to a feasible solution. At each node in the search tree, the algorithm calculates a lower bound and an upper bound for the profit function value. As it explores the tree, it uses bounds to prune branches of the tree that are guaranteed not to lead to optimal solutions, making it more efficient in finding the best solution within the search space.

For each task $k \in \mathcal{L}_r$ on route r, we define a set of tasks C_k that are in conflict with task k. At any node of the search tree, we denote L_s as the set of selected tasks in the current partial solution. The set F represents candidate tasks that are not currently inspected and do not conflict with tasks in L_s. We denote ρ_k as the profit that the user could receive if it performs task k and ξ_k as the time duration required to perform task k. We sort the tasks in non-decreasing order of the ratio ρ_k/ξ_k. $\Omega_k(F)$ represents the task following task k in the sorted list of F. More specifically, $\Omega_0(F)$ represents the first task in F and $\Omega_{-1}(F)$ represents the last task in F.

Algorithm 4 uses a branch and bound way to select a set of tasks from task set $\mathcal{L}_r$ on route r. The inputs of Algorithm 4 include the following variables: the total profit in the current partial solution ρ, the total time duration of the current partial solution ξ, the selected tasks in the current partial solution L_s, the candidate task set F, the current lower bound v_l, and the current optimal solution B.

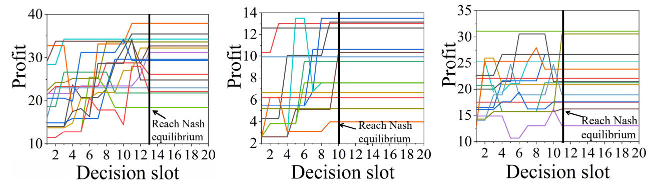
Algorithm 4 proceeds in a recursive way and initializes by calling $FindTask(0, 0, \varnothing, \mathcal{L}_r, 0, \varnothing)$. If the current total profit ρ is greater than the current lower bound v_l (i.e., the current optimal profit), the algorithm updates the current lower bound and the current optimal solution (lines 1-3). Then, the algorithm updates the upper bound v_u based on the current node (lines 4-14). More specifically, the upper bound v_u is calculated at each node of the tree by solving the continuous relaxation of the task selection problem ignoring temporal conflicts:

$$\begin{aligned} v_u = \max \{&\textstyle\sum_{k\in F} \rho_k x_k + \sum_{k\in L_s} \rho_k\}, \\ \text{subject to } &\textstyle\sum_{k\in F} \xi_k x_k \le \xi - \sum_{k\in L_s} \xi_k \quad k \in F, \\ &0 \le x_k \le 1 \quad k \in F. \end{aligned} \tag{32}$$

Problem (32) can be solved in $O(n)$ time using a greedy algorithm, as the tasks in F are sorted in a non-decreasing order of the ratio ρ_k/ξ_k [39]. If the current upper bound v_u is less than or equal to the current lower bound v_l, the algorithm prunes the node, ending the recursive function (line 15). Otherwise, the algorithm branches the node in a depth-first search manner. More specifically, it first extends the child node k, which is sorted first in F, and removes it from F. After exploring all branches of this child node, the algorithm continues to extend the node following k in the sorted list of F using a depth-first search approach (lines 16-24). The algorithm terminates when all branches have been explored and returns the optimal selected task set B (line 25).

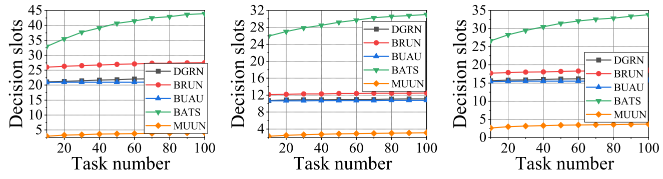
Finally, we introduce the distributed time-sensitive task selection algorithm (i.e., Algorithm 5) that integrates Algorithm 4. Similar to Algorithm 1, the algorithm first initializes the parameters and the solution (lines 1-8). Each user calculates their best decision and competes for the opportunity to update the decision in each decision slot (lines 9-20). Different from Algorithm 1, user i calculates the optimal task set for each route in the recommended route set R_i using Algorithm 4 and obtains the best route set $\triangle_i(t)$ (lines 11-12). The calculation process repeats until the termination message is received from the platform (line 20), indicating that the algorithm has converged to a Nash equilibrium. The information update algorithm is similar to Algorithm 2, and we omit the corresponding description for the sake of brevity.

The main idea of the time-sensitive task selection algorithm is similar to that in the time-tolerant scenario. The main difference between the time-tolerant task selection and the time-sensitive task selection exists in the process of calculating the best decision. Algorithm 4 selects a set of tasks without temporal conflicts that maximizes the profit for a route. Algorithm 5 utilizes Algorithm 4 to calculate the optimal task set and the corresponding profit for each route in the recommended route set. Conversely, in the time-tolerant scenario, Algorithm 1 simply adds up the shared rewards of all tasks on a route and selects the route with the maximum profit.



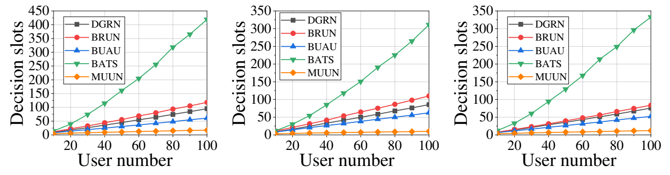
(a) Shanghai (b) Roma (c) Epfl

Fig. 3: User profit vs. decision slot.



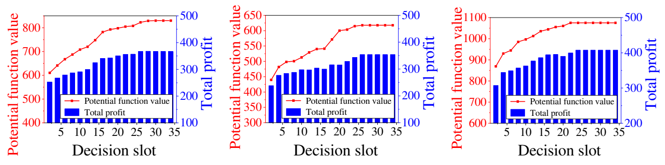
(a) Shanghai (b) Roma (c) Epfl

Fig. 5: Decision slot vs. task number.



(a) Shanghai (b) Roma (c) Epfl

Fig. 4: Decision slot vs. user number.



(a) Shanghai (b) Roma (c) Epfl

Fig. 6: Potential function and total profit vs. decision slot.

TABLE II: Simulation Parameters

Parameters	Value
Route number recommended to a user	1∼5
Original reward of a task a_k	10∼20
Parameter measuring reward increment μ_k	0∼1
User weight parameters α_i, β_i and γ_i	0.1∼0.9
System weight parameters φ, θ	0.1∼0.8
Number of repeated simulations	500

TABLE III: The selected user number vs. overlap ratio.

Total task #	50	60	70	80	90
Overlap ratio	0.526	0.531	0.532	0.536	0.538
Selected user #	2.013	1.978	1.842	1.751	1.701

VI. PERFORMANCE EVALUATION

A. Trace-Based Sets

We use three widely-used real-world data sets to evaluate the proposed algorithm. *Shanghai* [40] contains GPS trace data of taxis collected from August 2006 to October 2006 in Shanghai, China. We select 200 traces from a single day. *Roma* [41] contains GPS information from 320 taxis collected over 30 days in Rome, Italy. We select 150 traces in the city center. *Epfl* [42] consists of mobility data from 500 taxi cabs collected over 30 days in the San Francisco Bay Area, USA. From this dataset, we select 200 traces, each collected during a one-day period.

We extract the origin and destination from the traces and utilize the Google Maps API to generate a recommended route set for each origin-destination pair. The tasks are randomly generated with rewards and each recommended route may cover some tasks. The detour distance for each route is calculated as the additional distance compared to the shortest route, and the congestion level is calculated by the velocity of the vehicles on the route. The simulation parameters are shown in Table II.

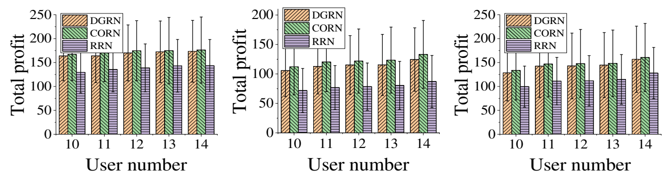
B. Comparison Algorithms

We use the following algorithms in the simulations.

- **Distributed Game-theoretical Route Navigation(DGRN):** The proposed route navigation algorithm for the time-tolerant task scenario uses the SUU algorithm to randomly select a user for decision updates.
- **Multi-User Update Navigation (MUUN):** The proposed algorithm uses the PUU algorithm to select a set of users from those who send update requests, enabling the selected users to concurrently update their route decisions by choosing the routes that maximize their profits.
- **Better Response Update Navigation (BRUN):** BRUN randomly selects a user from the set of users who send requests and allows them to choose a route better than the current one in each decision slot.
- **Best Update of All Users (BUAU):** BUAU inspects all users and selects the user that maximizes the potential function's value to update the decision.
- **Bayesian Asynchronous Task Selection (BATS) [16]:** We adapt the task selection approach from the existing research [16] to our scenario.
- **Centralized Optimal Route Navigation (CORN):** The centralized optimal approach to maximize the total profit of all users.
- **Random Route Navigation (RRN):** Each user randomly selects a route from the recommended route set.
- **Distributed Game-theoretical Task Selection (DGTS):** The proposed task selection algorithm for the time-sensitive task scenario utilizes the PUU algorithm, enabling multiple users to update their decisions in parallel.
- **Greedy Task Selection (GTS) [26]:** We adapt the greedy task allocation approach from [26] to the time-sensitive task scenario. More specifically, each user selects a set of tasks without temporal conflicts in a non-decreasing order of the profit-to-time-duration ratio on a route.

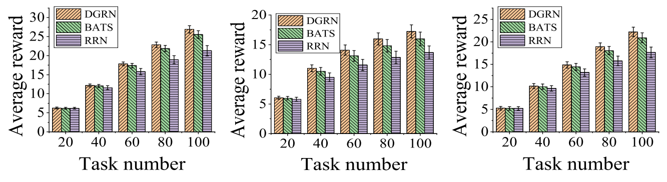
C. Numerical Results of Time-Tolerant Task Selection

1) Convergence for Nash equilibrium: We initially validate the convergence of the proposed distributed algorithm, as illustrated in Fig. 3. Specifically, we randomly select 15 users



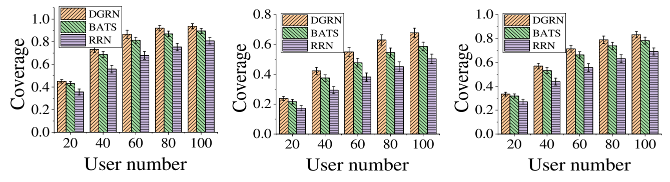
(a) Shanghai (b) Roma (c) Epfl

Fig. 7: Total profit vs. user number.



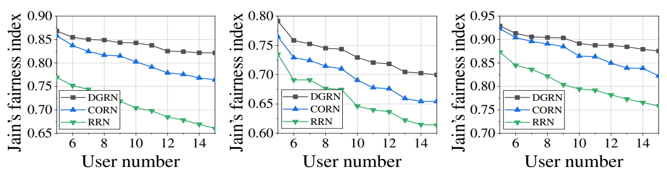
(a) Shanghai (b) Roma (c) Epfl

Fig. 9: Average reward vs. task number.



(a) Shanghai (b) Roma (c) Epfl

Fig. 8: Coverage vs. user number.



(a) Shanghai (b) Roma (c) Epfl

Fig. 10: Jain's fairness index vs. user number.

from each dataset and observe the dynamics of profits over 20 decision slots. It becomes evident that the profit of each user fluctuates with the decision updates in the initial phases but eventually converges to a stable point, which corresponds to the Nash equilibrium of the multi-user game. It's important to note that in this process, some users' profits may decrease due to the decision updates made by other users. Since the reward for a task is equally shared among the participants, when a user selects a particular route, the profits of all users involved in performing the task on that route decrease.

In Fig. 4, we investigate the number of decision slots required for convergence as the number of users changes. The simulation results reveal the following ranking of decision slots: MUUN<BUAU<DGRN<BRUN<BATS. The reason behind this order is that MUUN selects multiple users who update their decisions in parallel, while BUAU selects only one user who maximizes the potential function in each decision slot. Consequently, MUUN reaches the maximum value as quickly as possible. DGRN and BRUN randomly select a user to update the decision using the best and better response update manner, respectively. Therefore, DGRN converges to equilibrium slightly faster than BRUN. In the case of BATS, users update their decisions sequentially to maximize profit in each decision slot. In some decision slots, some users cannot increase their profits but still update their decisions, which results in an increased number of decision slots required for convergence.

In Fig. 5, we investigate the number of decision slots required for convergence as the number of tasks changes. The simulation results reveal the following ranking of decision slots: MUUN<BUAU<DGRN<BRUN<BATS. The reason for this ranking is the same as the description in Fig. 4. With the increase in the number of tasks, the number of decision slots shows a slight increase. This is because, as the number of tasks grows, users are more likely to overlap in covering the same tasks. Consequently, the route decisions of users become more intertwined, and it takes more decision slots to reach a Nash equilibrium state.

In Fig. 6, we observe the dynamics of the potential function's value and the total profit of all users as the decision slots change. We can observe that the potential function's value initially increases and eventually converges to a stable state, which aligns with the theoretical analysis. The total profit generally increases as the number of decision slots grows, with some fluctuations. This occurs because in the multi-user game, each user seeks to maximize their own profit rather than the total profit, and a user's decision update may sometimes reduce the profits of other users. Hence, there may be occasional decreases in the total profit.

In Table. III, we analyze the selected number of users for decision updates in MUUN as the overlap ratio changes. The overlap ratio is defined as the ratio between the number of tasks with multiple participants and the total number of tasks. Specifically, we vary the total number of tasks from 50 to 90 to manipulate the overlap ratio and then observe the average number of selected users across all decision slots. We conduct the simulation on the $Shanghai$ dataset and repeat the simulation 500 times. As the number of tasks increases, more routes intersect with each other at certain task locations. Since MUUN selects users whose chosen routes do not intersect with others at these task locations, the average number of selected users decreases as the overlap ratio increases.

2) Profit, coverage, fairness and reward: As shown in Fig. 7, we examine the trend of total profit as the number of users increases, conducting the simulations 500 times. Total profit is calculated as the sum of the profit function values for all users. The ranking of total profit is as follows: RRN<DGRN<CORN. DGRN, which maximizes each user's profit in an equilibrium state but doesn't prioritize total profit, yields slightly lower total profit than CORN.

As shown in Fig. 8, we investigate task coverage as the number of users grows. The simulation is repeated 500 times. Task coverage is calculated as the ratio between the number of covered tasks and the total number of tasks. The coverage ranks as follows: RRN<BATS<DGRN, as DGRN can adjust the settings to increase task coverage.

As shown in Fig. 9, we investigate the trend of the average reward as the number of tasks increases. The average reward

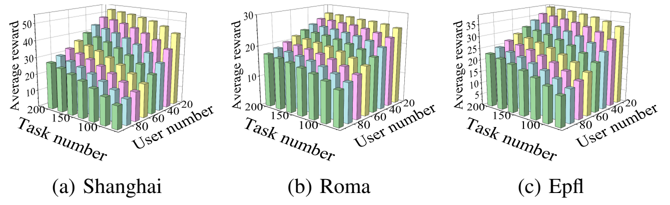

(a) Shanghai (b) Roma (c) Epfl

Fig. 11: Average reward vs. task number and user number.

TABLE IV: Comparison between DGRN and CORN.

User #	DGRN	CORN	Ratio	Bound
9	65	65	1	0.717
10	78	81	0.963	0.688
11	87	89	0.977	0.753
12	94	97	0.969	0.809
13	109	110	0.990	0.785
14	115	116	0.991	0.876

is defined as the total reward of all users divided by the number of users. We repeat the simulations 500 times, and the simulation results rank as follows: RRN<BATS<DGRN. It is evident that the average reward increases with the growth of the number of tasks, as users tend to perform more tasks when the number of tasks increases. The error bars ensure the accuracy of the simulation results.

Fig. 10 shows the dynamics of Jain's fairness index as the number of users increases. We repeat the simulations 500 times. Jain's fairness index [43] is used to measure the fairness of the user's profit, which is defined as $\frac{(\sum_{i\in\mathcal{U}} P_i(\mathbf{s}))^2}{|\mathcal{U}|\sum_{i\in\mathcal{U}} P_i(\mathbf{s})^2}$. Jain's fairness index depends on how evenly distributed the profit is among users. The simulation results reveal that the proposed DGRN achieves the highest Jain's fairness index compared to CORN and RRN. This is attributed to DGRN's ability to reach a Nash equilibrium in a multi-user game.

Fig. 11 displays the average reward of the proposed algorithm as the number of tasks and users change. We repeat the simulation 500 times. From the figure, we observe that the average reward increases as the number of tasks grows but decreases as the number of users increases. This is because as the number of tasks increases, users tend to perform more tasks, thus increasing their rewards. However, when the number of users increases, the reward for a task is shared among more participants, leading to a decrease in average reward. These simulation results align with our theoretical analysis.

In Table. IV, we examine the gap between DGRN and CORN. As the number of users changes, the ratio between the total profit of DGRN and CORN consistently exceeds the lower bound of Price of Anarchy (PoA), in accordance with our theoretical analysis.

3) The influence of algorithm parameters and the real-world example: In Fig. 12, we evaluate the influence of the system parameters φ and θ on the $Shanghai$ dataset. We conducted 500 simulation runs. It is noteworthy that the average reward increases as both φ and θ decrease. This decrease in φ and θ indicates that the platform places a greater emphasis on enabling users to receive higher rewards. Correspondingly,

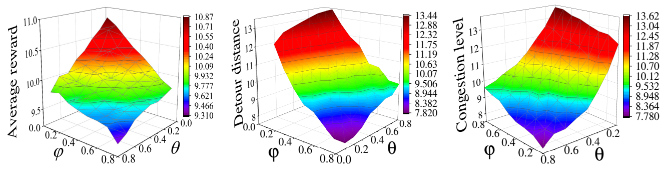

(a) Average reward (b) Detour distance (c) Congestion level

Fig. 12: The influence of system parameters.

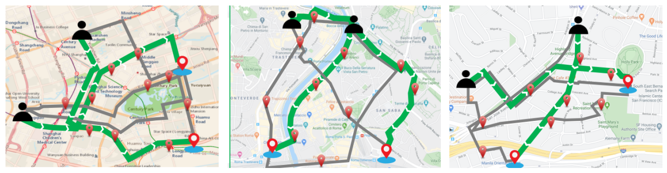

(a) Shanghai (b) Roma (c) Epfl

Fig. 13: The presentation on real data sets.

TABLE V: The influence of the user parameters.

α_i	reward	β_i	detour	γ_i	congestion
0.1	7.74	0.1	12.24	0.1	12.03
0.2	7.85	0.2	10.97	0.2	10.48
0.3	7.94	0.3	9.88	0.3	9.52
0.4	7.96	0.4	9.38	0.4	8.75
0.5	7.98	0.5	8.84	0.5	8.48
0.6	8.08	0.6	8.38	0.6	8.20
0.7	8.10	0.7	8.07	0.7	8.05
0.8	8.16	0.8	7.99	0.8	7.97

the detour distance decreases with the growth of φ and the congestion level decreases with the growth of θ, respectively.

In Table. V, we randomly select a user from the user set and vary the parameters α_i, β_i and γ_i from 0.1 to 0.8, respectively. We repeat the simulation 500 times. When changing α_i, we observe the value of the reward obtained by user i. It is easy to see that the reward increases with the growth of α_i, because α_i is the weight parameter concerning how much the user emphasizes receiving the task reward. As α_i increases, user i prefers to select the route with a high reward to increase the profit. Similarly, the detour distance and the congestion level decrease with the growth of β_i and γ_i, respectively. Hence, users can adjust the values of α_i, β_i and γ_i to achieve different individual preferences.

To better illustrate our schemes, we present three examples using data from three different datasets with the assistance of Google Maps. In Figure 13 (a), sensing tasks are distributed throughout the city. We consider two users and utilize the Google Maps API to generate recommended routes between their starting and destination points. The platform suggests 2 or 3 routes. The user selects one route (highlighted in green) and completes the tasks along that route. As the scenarios in Figures 13 (b) and (c) are similar to that in Figure 13 (a), no additional description is provided.

D. Numerical Results of Time-Sensitive Task Selection

1) Convergence for Nash equilibrium: We initially validate the convergence of the proposed distributed algorithm, as illustrated in Fig. 14. We randomly select 15 users and observe

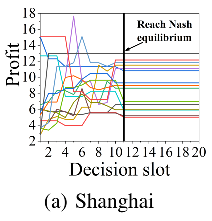

(a) Shanghai
(b) Roma
(c) Epfl

Fig. 14: User profit vs. decision slot.

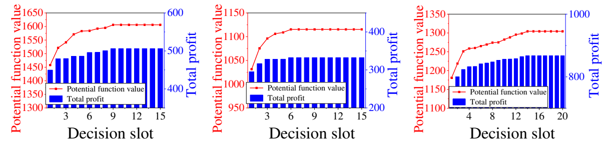

(a) Shanghai (b) Roma (c) Epfl

Fig. 17: Potential function and total profit vs. decision slot.

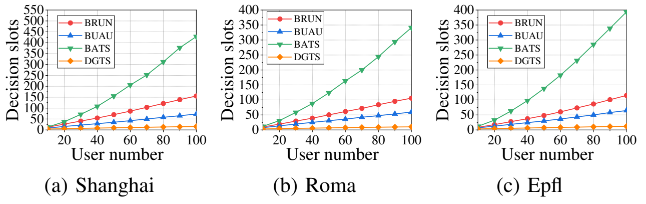

(a) Shanghai (b) Roma (c) Epfl

Fig. 15: Decision slot vs. user number.

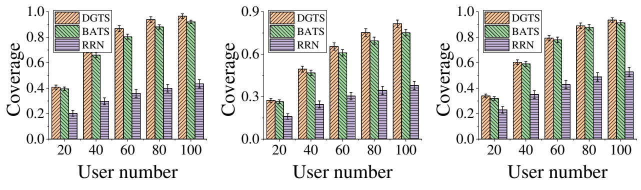

(a) Shanghai (b) Roma (c) Epfl

Fig. 18: Coverage vs. user number.

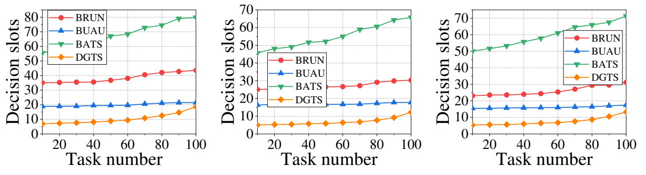

(a) Shanghai (b) Roma (c) Epfl

Fig. 16: Decision slot vs. task number.

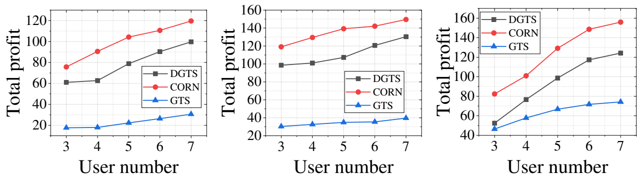

(a) Shanghai (b) Roma (c) Epfl

Fig. 19: Total profit vs. user number.

the dynamics of profits over 20 decision slots. Similar to the previous results, the profit of each user first fluctuates with the decision updates and eventually converges to a stable state, which is the Nash equilibrium. Since the task's reward is equally shared by the participants, a user's decision update may affect other users' profits, resulting in fluctuations in the initial phase.

Then, we investigate the number of decision slots required for convergence as the number of users and tasks change, respectively. The simulation results are demonstrated in Figs. 15-16. The simulation results reveal the following ranking of decision slots: DGTS<BUAU<BRUN<BATS. The reason behind this order is that DGTS uses the PUU algorithm to select multiple users for parallel decision updates. BUAU selects a single user to maximize the potential function value, and BRUN updates the decision in a better response update manner. These different decision update manners result in different convergence rates.

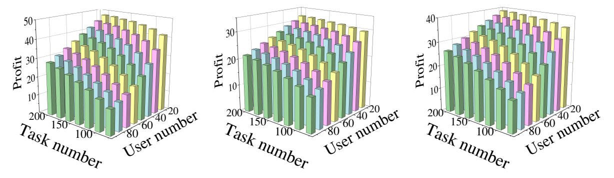
Furthermore, we investigate the dynamics of the potential function's value and the total profit of all users as the number of decision slots changes. The simulation results are demonstrated in Fig. 17. It is evident that the potential function's value initially increases and eventually converges to a stable point, which aligns with the theoretical analysis. The total profit of all users follows a similar trend as the number of decision slots changes.

2) Convergence and profit: We first investigate the coverage of tasks as the number of users grows in Fig. 18. The simulation is repeated 500 times. As mentioned above, task coverage is calculated as the ratio between the number of the tasks covered by users and the total number of tasks. The simulation results reveal the following ranking of coverage: DGTS>BATS>RRN. Compared to BATS, DGTS can adjust the parameters of the settings to increase task coverage.

Then, we investigate the trend of total profit as the number of users changes in Fig. 19. Total profit is calculated as the sum of the profit obtained by all users. The simulation results reveal the following ranking: GTS<DGTS<CORN. DGTS maximizes each user's profit in an equilibrium state but does not prioritize total profit, resulting in a lower total profit than that of CORN.

In Fig. 20, we further investigate the total profit achieved by DGTS as the number of users and tasks changes, conducting the simulation 500 times. It is evident that the total profit increases as the number of tasks grows but decreases as the number of users increases. As the number of tasks increases, users tend to perform more tasks, thus increasing their profits. However, when the number of users increases, the reward for a task is shared among more participants, leading to a decrease in their profits.

Finally, we conduct a simulation to investigate the trend of total profit as the conflict ratio changes in Table. VI. The conflict ratio is defined as the probability that there exists a temporal conflict between two tasks on a route. We generate tasks with different conflict ratios varying from 0.1 to 0.8. The simulation results show that the total profit decreases as the conflict ratio increases. The reason is that the number of candidate tasks on a route decreases as the conflict ratio increases.



(a) Shanghai (b) Roma (c) Epfl

Fig. 20: Total profit vs. task number and user number.

TABLE VI: Total profit vs. Conflict ratio.

Conflict ratio	Total Profit	Conflict ratio	Total Profit
0.1	931.8970	0.5	805.4229
0.2	887.9729	0.6	750.6201
0.3	850.7179	0.7	724.2337
0.4	822.0991	0.8	709.4291

VII. Conclusion

In this paper, drawing inspiration from widely-used map navigation systems, we introduce the utilization of route navigation systems for distributed crowdsensing task allocation. Our approach covers two scenarios: time-tolerant tasks and time-sensitive tasks. In the time-tolerant task scenario, we commence by demonstrating the NP-hardness of the centralized optimization problem and then represent it as a multi-user potential game. We subsequently present a distributed route navigation algorithm, which allows users to fine-tune the parameters of the profit function according to their individual preferences, while the platform can do the same to achieve various task allocation objectives. Additionally, we provide a theoretical analysis of this algorithm. In the time-sensitive task scenario, the problem complexity escalates due to temporal conflicts among tasks. In this scenario, users must simultaneously make route selections and choose the tasks to be executed along those routes. We prove the NP-hardness of the task selection problem and put forward a distributed task selection algorithm. Finally, we conduct extensive simulations based on three real-world datasets. The simulation results demonstrate that our proposed approach not only attains a Nash equilibrium but also achieves a total user profit closely approximating that of the optimal solution.

References

[1] E. Wang, D. Luan, Y. Yang, Z. Wang, P. Dong, D. Li, W. Liu, and J. Wu, "Distributed game-theoretical route navigation for vehicular crowdsensing," in *Proceedings of the 50th International Conference on Parallel Processing*, 2021, pp. 1–11.

[2] T. Liu, Y. Zhu, and L. Huang, "Tgba: A two-phase group buying based auction mechanism for recruiting workers in mobile crowd sensing," *Computer networks*, vol. 149, no. FEB.11, pp. 56–75, 2019.

[3] E. Wang, Y. Yang, J. Wu, W. Liu, and X. Wang, "An efficient prediction-based user recruitment for mobile crowdsensing," *IEEE Transactions on Mobile Computing*, vol. 17, no. 1, pp. 16–28, 2018.

[4] Y. Yang, W. Liu, E. Wang, and J. Wu, "A prediction-based user selection framework for heterogeneous mobile crowdsensing," *IEEE Transactions on Mobile Computing*, vol. 18, no. 11, pp. 2460–2473, 2019.

[5] V. D. Nguyen, P. L. Nguyen, K. Nguyen, and P. T. Do, "Constant approximation for opportunistic sensing in mobile air quality monitoring system," *Computer Networks*, vol. 202, 1 2022.

[6] M. Marjanović, S. Grubeša, and I. P. Žarko, "Air and noise pollution monitoring in the city of zagreb by using mobile crowdsensing," in *2017 25th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, 2017, pp. 1–5.

[7] X. Wang, J. Zhang, X. Tian, X. Gan, Y. Guan, and X. Wang, "Crowdsensing-based consensus incident report for road traffic acquisition," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 8, pp. 2536–2547, 2018.

[8] J. Wang, L. Wang, Y. Wang, D. Zhang, and L. Kong, "Task allocation in mobile crowd sensing: State-of-the-art and future opportunities," *IEEE Internet of Things Journal.*, vol. 5, no. 5, pp. 3747–3757, 2018.

[9] S. He, D. Shin, J. Zhang, and J. Chen, "Toward optimal allocation of location dependent tasks in crowdsensing," in *IEEE INFOCOM 2014*, 2014, pp. 745–753.

[10] B. Guo, Y. Liu, W. Wu, Z. Yu, and Q. Han, "Activecrowd: A framework for optimized multi-task allocation in mobile crowdsensing systems," *IEEE Transactions on Human-Machine Systems*, 08 2016.

[11] B. Zhao, S. Tang, X. Liu, X. Zhang, and W. Chen, "itam: Bilateral privacy-preserving task assignment for mobile crowdsensing," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2020.

[12] J. Wang, F. Wang, Y. Wang, L. Wang, Z. Qiu, D. Zhang, B. Guo, and Q. Lv, "Hytasker: Hybrid task allocation in mobile crowd sensing," *IEEE Transactions on Mobile Computing*, vol. 19, no. 3, pp. 598–611, 2020.

[13] H. Jin, H. Guo, L. Su, K. Nahrstedt, and X. Wang, "Dynamic task pricing in multi-requester mobile crowd sensing with markov correlated equilibrium," in *IEEE INFOCOM 2019*, 2019, pp. 1063–1071.

[14] M. Xiao, J. Wu, L. Huang, R. Cheng, and Y. Wang, "Online task assignment for crowdsensing in predictable mobile social networks," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2017.

[15] W. Liu, Y. Yang, E. Wang, and J. Wu, "Dynamic user recruitment with truthful pricing for mobile crowdsensing," in *IEEE INFOCOM 2020*, April 2020.

[16] M. H. Cheung, F. Hou, J. Huang, and R. Southwell, "Distributed time-sensitive task selection in mobile crowdsensing," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2020.

[17] M. H. Cheung, R. Southwell, F. Hou, and J. Huang, "Distributed time-sensitive task selection in mobile crowdsensing," in *Proceedings of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc 2015*. ACM, pp. 157–166.

[18] H. Li and L. Zhijian, "The study and implementation of mobile gps navigation system based on google maps," in *2010 International Conference on Computer and Information Application*, 2010, pp. 87–90.

[19] J. Wang, L. Wang, Y. Wang, D. Zhang, and L. Kong, "Task allocation in mobile crowd sensing: State-of-the-art and future opportunities," *IEEE Internet of Things journal*, vol. 5, no. 5, pp. 3747–3757, 2018.

[20] A. Capponi, C. Fiandrino, B. Kantarci, L. Foschini, D. Kliazovich, and P. Bouvry, "A survey on mobile crowdsensing systems: Challenges, solutions, and opportunities," *IEEE Communications Surveys and Tutorials*, vol. 21, no. 3, pp. 2419–2465, 2019.

[21] Y. Yang, W. Liu, E. Wang, and J. Wu, "A prediction-based user selection framework for heterogeneous mobile crowdsensing," *IEEE Transactions on Mobile Computing*, vol. 18, pp. 2460–2473, 11 2019.

[22] M. H. Cheung, F. Hou, J. Huang, and R. Southwell, "Distributed time-sensitive task selection in mobile crowdsensing," *IEEE Transactions on Mobile Computing*, vol. 20, pp. 2172–2185, 2021.

[23] Z. Wang, J. Li, J. Hu, J. Ren, Q. Wang, Z. Li, and Y. Li, "Towards privacy-driven truthful incentives for mobile crowdsensing under untrusted platform," *IEEE Transactions on Mobile Computing*, vol. 22, no. 2, pp. 1198–1212, 2023.

[24] X. Cai, L. Zhou, F. Li, Y. Fu, P. Zhao, C. Li, and F. R. Yu, "An incentive mechanism for vehicular crowdsensing with security protection and data quality assurance," *IEEE Transactions on Vehicular Technology*, vol. 72, no. 8, pp. 9984–9998, 2023.

[25] C. Dai, X. Wang, K. Liu, D. Qi, W. Lin, and P. Zhou, "Stable task assignment for mobile crowdsensing with budget constraint," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2020.

[26] B. Yin, J. Li, and X. Wei, "Rational task assignment and path planning based on location and task characteristics in mobile crowdsensing," *IEEE Transactions on Computational Social Systems*, vol. 9, no. 3, pp. 781–793, 2022.

[27] H. Wang, D. Zhao, H. Ma, and L. Ding, "Mb-gvns: Memetic based bidirectional general variable neighborhood search for time-sensitive task allocation in mobile crowd sensing," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 2, pp. 2219–2229, 2020.

[28] Z. Cai, Z. Duan, and W. Li, "Exploiting multi-dimensional task diversity in distributed auctions for mobile crowdsensing," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2020.

[29] C. Lai and X. Zhang, "Duration-sensitive task allocation for mobile crowd sensing," *IEEE Systems Journal*, vol. 14, no. 3, pp. 4430–4441, 2020.

[30] Z. Dai, C. H. Liu, R. Han, G. Wang, K. K. Leung, and J. Tang, "Delay-sensitive energy-efficient uav crowdsensing by deep reinforcement learning," *IEEE Transactions on Mobile Computing*, vol. 22, no. 4, pp. 2038–2052, 2023.

[31] X. Chen, X. Gong, L. Yang, and J. Zhang, "Amazon in the white space:social recommendation aided distributed spectrum access," *IEEE/ACM Transactions on Networking*, vol. 25, no. 1, 2017.

[32] F. Fabiani and S. Grammatico, "Multi-vehicle automated driving as a generalized mixed-integer potential game," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 1064–1073, 2020.

[33] Z. Hong, W. Chen, H. Huang, S. Guo, and Z. Zheng, "Multi-hop cooperative computation offloading for industrial iot-edge-cloud computing environments," *IEEE Transactions on Parallel and Distributed Systems*, no. 99, pp. 1–1, 2019.

[34] Q. He, G. Cui, X. Zhang, F. Chen, S. Deng, H. Jin, Y. Li, and Y. Yang, "A game-theoretical approach for user allocation in edge computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 3, pp. 515–529, 2020.

[35] M. Liu, I. Kolmanovsky, H. E. Tseng, S. Huang, D. Filev, and A. Girard, "Potential game-based decision-making for autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 8, pp. 8014–8027, 2023.

[36] B. Varga, J. Inga, and S. Hohmann, "Limited information shared control: A potential game approach," *IEEE Transactions on Human-Machine Systems*, vol. 53, no. 2, pp. 282–292, 2023.

[37] M. Chessa, J. Grossklags, and P. Loiseau, "A game-theoretic study on non-monetary incentives in data analytics projects with privacy implications," in *2015 IEEE 28th Computer Security Foundations Symposium*, 2015, pp. 90–104.

[38] M. Cardei, M. T. Thai, Y. Li, and W. Wu, "Energy-efficient target coverage in wireless sensor networks," in *IEEE INFOCOM 2005*.

[39] R. Sadykov and F. Vanderbeck, "Bin packing with conflicts: A generic branch-and-price algorithm," *INFORMS Journal on Computing*, vol. 25, pp. 244–255, 3 2013.

[40] H. Zhu, Y. Zhu, M. Li, and L. M. Ni, "Hero: Online real-time vehicle tracking in shanghai," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 5, pp. 740–752, 2009.

[41] L. Bracciale, M. Bonola, P. Loreti, G. Bianchi, R. Amici, and A. Rabuffi, "CRAWDAD dataset roma/taxi (v. 2014-07-17)," Downloaded from https://crawdad.org/roma/taxi/20140717, Jul. 2014.

[42] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser, "CRAWDAD dataset epfl/mobility (v. 2009-02-24)," Downloaded from https://crawdad.org/epfl/mobility/20090224, Feb. 2009.

[43] R. Jain, D. M. Chiu, and H. WR, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," *ACM Transactions on Computer Systems*, 1984.

En Wang received his B.E. degree in software engineering from Jilin University, Changchun in 2011, his M.E. degree in computer science and technology from Jilin University, Changchun in 2013, and his Ph.D. in computer science and technology from Jilin University, Changchun in 2016. He is currently a professor in the Department of Computer Science and Technology at Jilin University, Changchun. He is also a visiting scholar in the Department of Computer and Information Sciences at Temple University in Philadelphia. His current research focuses on the efficient utilization of network resources, energy-efficient communication between human-carried devices, and mobile crowdsensing.

Dongming Luan received his B.E.degree in Software Engineering and M.E.degree in computer science and technology from Jilin University, Changchun, Jilin, China, in 2017 and 2020. Now, he is a Ph.D. candidate in College of Computer Science and Technology in Jilin University, Changchun, Jilin, China. His current research interest is Mobile Crowdsensing.

Yuanbo Xu received his B.E., M.E., and Ph.D. in computer science and technology from Jilin University, Changchun, in 2012, 2015, and 2019. He is an associate professor and Ph.D. supervisor in the College of Computer Science and Technology at Jilin University, Changchun. He is also a visiting scholar in the Management Science and Information Systems Department at Rutgers, the State University of New Jersey. His research interests include data mining applications, recommender systems, and mobile computing. He has published 40+ research results in journals such as IEEE TKDE, IEEE TMC, IEEE TMM, IEEE TNNLS, ACM TKDD, and conferences such as ICDE, INFOCOM, CIKM, and ICDM.

Yongjian Yang received his B.E. degree in automatization from Jilin University of Technology, Changchun, Jilin, China in 1983; his M.E. degree in computer communication from Beijing University of Post and Telecommunications, Beijing, China in 1991; and his Ph.D. in software and theory of computer from Jilin University, Changchun, Jilin, China in 2005. He is currently a professor and a PhD supervisor at Jilin University, the Vice Dean of the Software College of Jilin University, Director of Key lab under the Ministry of Information Industry, Standing Director of the Communication Academy, and a member of the Computer Science Academy of Jilin Province. His research interests include: network intelligence management, wireless mobile communication and services, and wireless mobile communication.

Jie Wu is the Associate Vice Provost for International Affairs at Temple University. He also serves as Director of the Center for Networked Computing and Laura H. Carnell professor in the Department of Computer and Information Sciences. Prior to joining Temple University, he was a program director at the National Science Foundation and was a distinguished professor at Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. Dr. Wu regularly publishes in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE Transactions on Service Computing and the Journal of Parallel and Distributed Computing. Dr. Wu was general cochair/chair for IEEE MASS 2006, IEEE IPDPS 2008, IEEE ICDCS 2013, and ACM MobiHoc 2014, as well as program co-chair for IEEE INFOCOM 2011 and CCF CNCC 2013. He was an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). Dr. Wu is a CCF Distinguished Speaker and a Fellow of the IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.