accepted to appear in Proceedings of 23rd IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), Belfast, June, 2022.

Three-dimensional Stable Task Assignment in Semi-opportunistic Mobile Crowdsensing

Fatih Yucel and Eyuphan Bulut
Department of Computer Science, Virginia Commonwealth University
401 West Main St. Richmond, VA 23284, USA
{yucelf, ebulut}@vcu.edu

***Abstract*—In semi-opportunistic mobile crowdsensing (SO-MCS), workers are asked to provide the matching platform with multiple paths they find acceptable between their starting locations and destinations in order to alleviate the problem of poor coverage in opportunistic MCS without forcing them to take potentially much costly and hence undesirable paths as in participatory MCS. While these alternative paths open up new assignment possibilities between workers and tasks, they also make it more challenging to find a stable or preference-aware task assignment (TA), as they bring a new dimension to the TA problem (i.e., workers/paths/tasks instead of workers/tasks as in previous work), and introduce complex requirements to achieve stability by satisfying user preferences. In this paper, we formally define the stability conditions for three-dimensional task assignments in SO-MCS, and propose two polynomial-time TA algorithms: an exact algorithm for SO-MCS systems with uniform worker qualities, and a c-approximate algorithm for general SO-MCS systems, where c is the number of the acceptable paths of the worker with the largest set of acceptable paths. Through extensive simulations, we demonstrate that the proposed algorithms significantly outperform the state-of-the-art TA algorithms in terms of stability (or user happiness) in most scenarios.**

***Index Terms*—Semi-opportunistic mobile crowdsensing, task assignment, preference-awareness, stable matching.**

I. Introduction

Mobile crowdsensing (MCS) aims to leverage the portable devices of mobile users to cooperatively perform sensing tasks [1]. There are two main sensing modes in MCS: (i) participatory, and (ii) opportunistic. In the former, the MCS platform assigns a path and some set of tasks on this path to each worker in a way that maximizes the system utility. In the latter, on the other hand, workers move on the paths of their choosing, and can be assigned to perform a task only if the task region resides on their paths. The key issue in the participatory mode is that the paths assigned to workers are likely to disturb their daily schedules and introduce significant additional travel costs, whereas the opportunistic mode mainly suffers from the issue of poor coverage, as a task cannot be carried out if its region will not be visited in time by any worker in the system during their self-defined trips.

To address these issues and find a middle ground between the participatory and opportunistic modes, a new sensing mode, namely *semi-opportunistic*, has been proposed recently [2]. In this novel mode, workers provide the matching platform with alternative paths (e.g., dashed lines in Fig. 1) they would be willing to take within their comfort zones in addition to the path they would normally take. This yields a wider range of task assignment options for both workers and tasks, and hence not only improves the task coverage, but also expands the set of tasks that workers can carry out, allowing them to increase their profits by performing more tasks.

Fig. 1: Example paths of a user for different sensing modes.

Besides the sensing mode, the performance metric used to optimize the task assignments and the actual quality of the assignments according to this metric have a major impact on the success of the MCS campaign. Most of the existing studies in the MCS literature adopt a system-level performance metric (e.g., the number of completed tasks, the overall task quality). However, the individual preferences of the users (i.e., workers and task requesters) are generally not aligned with such system-level metrics, thus the resulting task assignments may deter their continuous participation in the MCS campaign. To resolve this problem, some recent studies [3], [4] adopt a user-centric approach, and seek to maximize the happiness of each user with their assignment according to their preferences. This is typically accomplished by producing a task assignment that ensures the absence of worker-task requester pairs that prefer to be matched with each other rather than their current partners.

This study is *the first to examine the user-centric task assignment problem in a semi-opportunistic mobile crowdsensing (SO-MCS) setting*. The key challenge in this problem is to satisfy the preferences of all users in a three-dimensional matching setting, where each worker is to be matched with one of his acceptable paths, and then with a set of tasks on this path. Thus, the path and task assignments are strongly interdependent, and must be compatible with each other. Besides, various factors such as task rewards, worker qualities and the number of tasks that workers can carry out on each of their paths (a worker may choose to perform fewer tasks on a longer path) need to be considered together to achieve a preference-aware task assignment. Our main contributions in this paper can be summarized as follows:

- We provide a formal definition of the preference-aware task assignment problem in an SO-MCS system.
- We show that a task assignment that satisfies all user preferences does not exist in some instances.
- We design two different task assignment algorithms, and prove their (near-)optimality for different settings.
- We carry out extensive simulations, and demonstrate the superiority of our algorithms over the existing solutions.

The rest of the paper is structured as follows. In Section II, we provide a summary of the related work. In Section III, we describe the system model and formally define the preference-aware task assignment problem. Then, we present our algorithms along with their theoretical analysis in Section IV, and their empirical analysis in Section V. Finally, in Section VI, we provide our conclusions.

II. Related Work

A. Task Assignment in MCS

In participatory MCS, since workers need to travel between the task regions to perform the assigned tasks, a key factor that shapes the task assignment process is the travel costs of the workers. Thus, in most of the studies minimization of the travel costs is aimed together with a second objective such as maximizing the number of completed tasks with minimal rewards [5], maximizing the total task quality [6], maintaining a maximum traveling distance for workers [7], preserving the privacy of users [8], and achieving an on-time arrival of the workers to their own destinations [9].

On the other hand, in opportunistic MCS, the main objectives are to maximize the coverage and to minimize the completion times of the tasks due to the uncontrolled mobility (i.e., a task can only be performed if its region resides on the route of a worker). In [10], the authors introduce the problem of maximizing the total task quality by recruiting workers with high QoS scores within the budget constraint of the platform. They prove the problem to be NP-hard, and propose an efficient algorithm with an approximation ratio of $1 - (1/e)$. In [11], the authors study the same problem, and present a heuristic algorithm that outperforms the approximation algorithm in [10]. In [12]–[14], not only the completion of tasks but also the delivery of sensed data is assumed to be carried out in an opportunistic manner, and the optimization of both has been studied considering various aspects. Uncertainty in worker trajectories due to environmental (e.g., road/traffic conditions) and personal factors (e.g., the trajectory of a taxi driver) have also been considered in some studies, and solutions that maximize task coverage [15] and minimize task completion times [16] have been proposed.

There are a few very recent studies [2], [17] that look at the task assignment problem in a hybrid system model to integrate the advantages of participatory and opportunistic MCS. In [17], the authors propose a two-phased task allocation process, where opportunistic task assignment is followed by participatory task assignment. The objective behind this design is to maximize the number of tasks that are performed in an opportunistic manner, which is much less costly compared to participatory MCS, and then to ensure that the tasks that cannot be completed by opportunistic workers are assigned to workers that are willing to perform tasks in a participatory manner to alleviate the coverage problem in opportunistic MCS. On the other hand, in [2], the workers carry out the sensing tasks only in opportunistic mode, but they provide the matching platform with multiple paths that they would take if requested, instead of a single path as in classic opportunistic MCS [10], [11]. This enables the platform to find a matching with a high task coverage as illustrated in Fig. 1. However, none of the studies mentioned thus far considers the preferences of the workers and task requesters in the assignment process, which may impair their long-term participation in the MCS campaign.

B. Preference-aware Matching

Since its introduction by Gale and Shapley [18], stable or preference-aware matching has been utilized in various real world applications such as residency matching [19], channel assignments in device-to-device [20] and V2X communications [21], and taxi dispatching [22]. A comprehensive list of stable matching problems can be found in [23].

Three dimensional version of stable matching was introduced by Knuth [24] by considering three sets of agents (e.g., woman, man, dogs) and their preferences on the others. Later, several variants that consider cyclic preference relations [25] as well as one-dimensional preference lists over all individuals from the other two sets [26] have also been studied. The three-dimensional stable matching has also been considered in several applications such as server-data source-user matching [27] in video streaming services under restricted preference settings.

Since workers and task requesters in a typical MCS system have preferences over each other to maximize their profits and to have their tasks completed with the highest quality possible, respectively, some recent studies have explored the issue of preference-awareness in MCS as well. [28] and [4] study the budget-constrained many-to-one and many-to-many stable task assignment problems, which are proven to be NP-hard, and propose efficient approximation algorithms. In [3], the authors study the former problem under a weaker stability criteria considering minimum task quality requirements. [29] studies the capacity-constrained many-to-many stable matching problem. [30] considers the stability and the profit of the platform simultaneously, and aims to find maximum size task assignments with minimum instability. Some other studies also consider online [31] and coverage-aware stability [32]. However, all of these studies focus on either participatory or opportunistic sensing, thus have the aforementioned disadvantages of that sensing mode. Moreover, they only consider a two-dimensional (bipartite) stable matching between workers and task requesters.

There are some recent studies that consider three-dimensional stability in spatial-crowdsourcing, however these studies have a limited understanding of user preferences and stability. For example, [33] only considers the preferences of users on the potential places (where the task will be completed) based on their proximity (i.e., workers and task requesters do

not have preferences over each other). On the other hand, there are also studies [34], [35] that consider trichromatic matching (i.e., matching of three items such as tasks, workers and workplaces/PoIs) with some stability definitions. However, these studies mainly focus on task scheduling within a deadline without considering the matching stability based on user preferences, and aim to maximize the number of matched items. Our problem is totally different from these studies as it is a many-to-one, capacity-constrained three-dimensional stable matching problem where only the nodes in two (i.e., workers, tasks) of the three sets have preferences over each other depending on the features of the nodes in the third set (i.e., acceptable paths of workers) as explained in the next section.

III. System Model

A. Assumptions

We assume a system model with a set of location-dependent sensing tasks $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ and a set of workers $\mathcal{W} = \{w_1, w_2, \ldots, w_m\}$ that accept to perform tasks in a semi-opportunistic setting. Each worker w_i provides the service provider *(SP)* with a set of paths $\mathcal{P}_i = \{p_{i,1}, p_{i,2}, \ldots, p_{i,a_i}\}$ that he finds acceptable from his current location to his destination. In each assignment period, it is the responsibility of SP to find a satisfactory assignment between workers and tasks by matching workers to one of their acceptable paths, and assigning a subset of tasks on their selected paths.

Each path $p_{i,j}$ has a capacity $c_{i,j}$ associated with it, which indicates the maximum number of tasks that worker w_i is willing to perform if he is assigned to path $p_{i,j}$. The ability to specify a capacity for each path enables workers to avoid any unacceptable delays in their daily schedule by controlling their sensing activity. Since acceptable paths of a worker may have different conditions (e.g., traffic, security) that can affect the comfort level of the worker for sensing, or may be of different lengths, it is crucial to allow workers to assign different capacities to their paths. For simplicity, we let the path set $\mathcal{P}_i$ of each worker w_i be in non-increasing order of path capacities. That is, we have $c_{i,j} \geq c_{i,j+1}$ for all j values between 1 and $a_i - 1$. Besides, if the region of task t_k resides on path $p_{i,j}$ (i.e., worker w_i can perform task t_k if he takes path $p_{i,j}$), we say t_k is on $p_{i,j}$ and let

$$\mathcal{T}_{i,j} = \{t_k : t_k \in \mathcal{T} \text{ and } t_k \text{ is on } p_{i,j}\}. \tag{1}$$

Our system model is also QoS-aware. That is, each worker w_i has a QoS score $q_{i,j}$ for each task t_j, which specifies the level of competence of worker w_i for task t_j, and can be determined based on various factors such as quality of the sensing equipment and trustworthiness or seniority of the worker. Moreover, we look at the task assignment problem in both *uniform* and *general* QoS settings [28]. In the uniform QoS setting, each worker has a universal QoS score that applies for all tasks, i.e., $q_{i,j} = q_{i,k}$ for all $1 \leq j, k \leq n$. On the other hand, in the general QoS setting, a worker may have different QoS scores for different tasks. For convenience, we simply call MCS instances with uniform and general QoS settings as uniform and general MCS instances, respectively.

Another important feature of our system model is that the task assignments are optimized with respect to the preferences of workers and tasks, which is called preference-awareness. Each task (requester) t_j would like to be matched with a worker with a high QoS score, thus prefers worker w_i to all workers with a QoS score smaller than $q_{i,j}$. Then, we can define the preference list L_j^t of task t_j for the general QoS setting as follows:

$$L_j^t = w_{\sigma_1}, w_{\sigma_2}, \ldots, w_{\sigma_k} \text{ where } q_{\sigma_i,j} \geq q_{\sigma_{i+1},j}. \tag{2}$$

Note that L_j^t may not contain all workers if t_j finds some workers unacceptable (e.g., workers with a QoS score smaller than a certain value). In the uniform QoS setting, assuming $\hat{L}_j^t$ is the preference list formed for task t_j according to (2) without leaving out any worker (i.e., $|\hat{L}_j^t| = m$), we can define a global preference list $L_{\mathcal{T}}$ for tasks as follows:

$$L_{\mathcal{T}} = \hat{L}_1^t = \hat{L}_2^t = \cdots = \hat{L}_n^t. \tag{3}$$

On the other hand, the requester of each task t_j offers a monetary reward of $r_{j,i}$ to each worker w_i to encourage worker participation. As rational individuals, the workers in our system aim to maximize their profits. Thus, the preference list L_i^w of worker w_i can be formed as:

$$L_i^w = t_{\sigma_1}, t_{\sigma_2}, \ldots, t_{\sigma_k} \text{ where } r_{\sigma_i,i} \geq r_{\sigma_{i+1},i}. \tag{4}$$

The preference list of a worker also does not need to contain all tasks in the system. Given a worker-task pair (w_i, t_j), if $w_i \notin L_j^t$ and $t_j \in L_i^w$, we remove t_j from L_i^w as worker w_i is not an acceptable partner for task t_j. Similarly, if $t_j \notin L_i^w$ and $w_i \in L_j^t$, we remove w_i from L_j^t.

We let $\mathcal{M}$ denote a feasible three-dimensional matching (task assignment) in our system model. For each worker w_i, $\mathcal{M}(w_i) = (A, p_{i,j})$ denotes the assignment of worker w_i in this matching, where $p_{i,j}$ is the path selected for worker w_i and A is the set of tasks that are assigned to worker w_i through path $p_{i,j}$. To be a feasible assignment, A and $p_{i,j}$ must satisfy the following conditions:

- *capacity constraint*: $|A| \leq c_{i,j}$,
- *acceptability constraint*: $A \subseteq L_i^w$,
- *regional constraint*: $A \subseteq \mathcal{T}_{i,j}$.

On the other hand, the assignment of each task t_k in this matching is denoted by $\mathcal{M}(t_k) = (w_i, p_{i,j})$, where w_i is the worker that is assigned to perform task t_k and $p_{i,j}$ is the path that is selected for worker w_i. The following conditions must be satisfied for feasibility:

- *acceptability constraint*: $w_i \in L_k^t$.
- *regional constraint*: $t_k \in \mathcal{T}_{i,j}$,

If a user (worker or task) v is left unmatched in $\mathcal{M}$, we let $\mathcal{M}(v) = (\varnothing, -)$. Also, given the assignment $\mathcal{M}(v) = (X, Y)$ of user v, we let $\mathcal{M}_u(v)$ and $\mathcal{M}_p(v)$ denote X and Y, respectively.

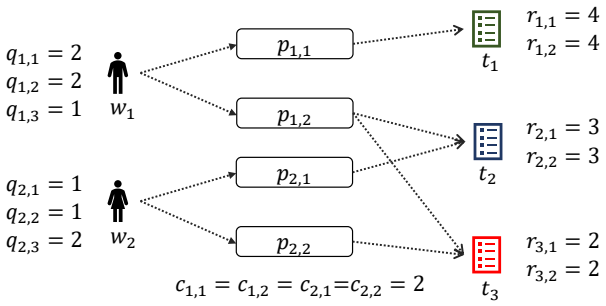

Fig. 2: An MCS instance for which no stable matching exists. There is an edge from a path $p_{i,j}$ to a task t_k if $t_k \in \mathcal{T}_{i,j}$.

B. Problem Statement

Our main objective in this study is to find a preference-aware, feasible matching according to our system model where the users are happy with their assignments according to their preferences. Below, we give the necessary definitions to formally evaluate the happiness of the users with a matching.

Definition 1 (Unhappy triad)**.** *Given a matching $\mathcal{M}$, worker w_i, path $p_{i,j}$ and a set S of tasks form an unhappy triad denoted by $\langle w_i, p_{i,j}, S\rangle$ if*

- *S is an acceptable assignment for w_i, i.e.,*

$$1 \leq |S| \leq c_{i,j}, S \subseteq L_i^w, \text{ and } S \subseteq \mathcal{T}_{i,j}, \tag{5}$$

- *w_i is an acceptable assignment for each $t_k \in S$, i.e.,*

$$w_i \in L_k^t \text{ and } t_k \in \mathcal{T}_{i,j}, \tag{6}$$

- *each task $t_k \in S$ either prefers worker w_i to their current assignment w_h in $\mathcal{M}$, i.e.,*

$$q_{i,k} > q_{h,k} \text{ where } q_{h,k} = 0 \text{ if } w_h = \varnothing, \tag{7}$$

  or is already assigned to worker w_i, i.e., $\mathcal{M}_u(t_k) = w_i$.
- *worker w_i prefers the task set S to his current assignment in $\mathcal{M}$, i.e.,*

$$\sum_{t_h \in S} r_{h,i} > \sum_{t_k \in \mathcal{M}_u(w_i)} r_{k,i}, \tag{8}$$

Thus, given an unhappy triad $\langle w_i, p_{i,j}, S\rangle$, we see from the first two conditions that it is possible to assign the tasks in the set S to worker w_i through path $p_{i,j}$ without violating any feasibility constraints, and see from the last two conditions that this would make at least one task in S and worker w_i strictly better off without making any task in S worse off.

Definition 2 (3D-Stable matching)**.** *A matching is said to be stable if it does not contain any unhappy triads.*

In order for a matching to be perfect in terms of preference-awareness, it should be stable. However, as we prove in the following theorem, it is not possible to construct a stable matching in all MCS instances.

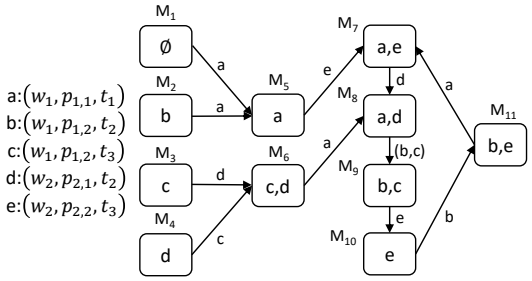

Fig. 3: Proof of Theorem 2. All possible matchings for the instance in Fig. 2 are shown with boxes. There is an edge k from matching (box) M_i to matching M_j, if k is an unhappy triad in M_i due to a more favorable assignment in M_j.

Theorem 1. *There exist MCS instances with a general QoS setting, in which all feasible matchings are unstable (i.e., contain at least one unhappy triad).*

Proof. We prove it by showing such an instance, which is illustrated in Fig. 2. There are 11 possible task assignments in this instance, and, as shown in Fig. 3, every one of them contains at least one unhappy triad. Thus, no stable matching exists for this instance, which completes our proof. □

On the other hand, a stable matching always exists in the uniform MCS instances, which we will prove in the following section by giving an algorithm that produces a stable matching for such instances.

Due to the nonexistence of stable matchings in general MCS systems, we formulate our objective function as:

$$\textit{maximize} \min_{x \in U(\mathcal{M})} \frac{1}{\delta_x}, \tag{9}$$

where $U(\mathcal{M})$ denotes the set of unhappy triads in the produced matching $\mathcal{M}$, and δ_x denotes the dissatisfaction ratio of a given unhappy triad $x = \langle w_i, p_{i,j}, S\rangle$, which is computed by

$$\delta_x = \frac{\sum_{t_h \in S} r_{h,i}}{\sum_{t_k \in \mathcal{M}_u(w_i)} r_{k,i}}, \tag{10}$$

(where $\delta_x = \infty$ if $\mathcal{M}_u(w_i) = \varnothing$). So, the dissatisfaction ratio of x quantifies the utility difference between the current matching and the matching, in which worker w and the unhappy tasks in S are matched with each other, and do not form an unhappy triad. Consequently, our goal is to optimize the worst-case performance by minimizing the maximum dissatisfaction ratio in the final matching.

Definition 3 (α-stable matching)**.** *A matching $\mathcal{M}$ is said to be α-stable if*

$$\max_{x \in U(\mathcal{M})} \delta_x \leq \alpha. \tag{11}$$

Note that a perfectly stable matching is 1-stable, and larger α values indicate worse task assignments in terms of user happiness. For the α-stability, we have the following lower bound, which may not be tight.

TABLE I: Key notations.

Notation	Description
$\mathcal{W}$, $\mathcal{T}$	Set of workers and tasks, respectively
m, n	Number of workers and tasks, respectively
$\mathcal{P}_i$	Set of acceptable paths of worker w_i
a_i	Number of acceptable paths of worker w_i
$c_{i,j}$	Capacity of path $p_{i,j}$
$\mathcal{T}_{i,j}$	Set of tasks that reside on path $p_{i,j}$
$q_{i,j}$	QoS of worker w_i for task t_j
$r_{j,i}$	Reward offered to worker w_i for task t_j
L_j^t	Preference list of task t_j
$L_{\mathcal{T}}$	Global preference list of tasks in uniform systems
L_i^w	Preference list of worker w_i
$\mathcal{M}$	A feasible matching (task assignment)
$\mathcal{M}(v)$	Assignment of worker/task v in $\mathcal{M}$
$\mathcal{M}_u(w_i)$	Set of tasks assigned to worker w_i in $\mathcal{M}$
$\mathcal{M}_p(w_i)$	Path selected for worker w_i in $\mathcal{M}$
$\mathcal{M}_u(t_j)$	Worker assigned to task t_j in $\mathcal{M}$
$\mathcal{M}_p(t_j)$	Path selected for the partner of task t_j in $\mathcal{M}$
δ_x	Dissatisfaction ratio of unhappy triad x
$U(\mathcal{M})$	Set of unhappy triads in $\mathcal{M}$

Theorem 2. *There exist MCS instances with a general QoS setting, which do not admit any α-stable matching for $\alpha < \hat{p}$, where $\hat{p} \approx 1.325$ is the plastic number [36].*

Proof. We prove this on a slightly modified version of the MCS instance given in Fig. 2, in which the task rewards are set as $r_{1,1} = r_{1,2} = \hat{p}^2$, $r_{2,1} = r_{2,2} = \hat{p}$, and $r_{3,1} = r_{3,2} = 1$. This instance has exactly the same set of possible matchings and unhappy triads as the original instance, which are illustrated in Fig. 3. Since the powers of the plastic number satisfy the equation $\hat{p}^{k+3} = \hat{p}^{k+1} + \hat{p}^k$, the ratio between the total reward of any feasible two task sets for both workers is guaranteed to be at least $\hat{p}$, which means if a worker is unhappy, his dissatisfaction ratio will be at least $\hat{p}$. In fact, the maximum dissatisfaction ratio in these matchings are as follows: ∞ for $\mathrm{M}_{[1..5]}, \mathrm{M}_9, \mathrm{M}_{10}$; $\hat{p}^3$ for M_6; and $\hat{p}$ for $\mathrm{M}_7, \mathrm{M}_8, \mathrm{M}_{11}$. This completes our proof, as the α-stability of all possible matchings in this instance is at least $\hat{p}$. □

A summary of the notations used throughout the paper is presented in Table I.

IV. Proposed Solution

In this section, we first present an algorithm that finds stable matchings in uniform MCS instances. Then, we consider general MCS instances where stable matchings may not exist, and propose an approximation algorithm that finds near-optimal matchings in terms of stability.

A. Stable Task Assignment in Uniform MCS Systems

In Algorithm 1, we describe our algorithm that finds stable matchings in uniform systems. In line 1, we initialize the matching $\mathcal{M}$. Then, we form the global preference list of tasks according to (3) in line 2. In the for loop starting at line 3, we iterate the workers in $L_{\mathcal{T}}$ from beginning to end, and find an assignment for the ith worker (w_h) in $L_{\mathcal{T}}$ in the ith iteration. To this end, we first form the preference list L_h^w of worker w_h in line 5. Then, in the for loop starting at line 7, we find the best feasible task set A' for each of his acceptable paths $p_{h,j}$ among the tasks that have not been matched yet. To find the best task set for $p_{h,j}$, we iterate the preference list of worker w_h in the for loop in lines 9-15, and add the tasks that are on path $p_{h,j}$ and currently unmatched (line 11) to A' until we reach the capacity limit $c_{h,j}$ of path $p_{h,j}$ (line 14). We keep the best task set found so far in A, the index of the corresponding path in r, and the sum of the rewards offered to worker w_h by the tasks in A in s (line 17). Finally, we match the tasks in A and worker w_h with each other (lines 18-20).

Algorithm 1: UniformSTA ($\mathcal{W}$, $\mathcal{T}$)

Input: $\mathcal{W}$: Set of workers, $\mathcal{T}$: Set of tasks

1 let $\mathcal{M}(u) = (\varnothing, -)$ for all $u \in \mathcal{W} \cup \mathcal{T}$
2 form $L_{\mathcal{T}}$ by (3)
3 **for** $i \leftarrow 1$ *to* m **do**
4 let w_h be the ith worker in $L_{\mathcal{T}}$
5 form L_h^w by (4)
6 $A \leftarrow \{\}$, $s \leftarrow 0$, $r \leftarrow 0$
7 **for** $j \leftarrow 1$ *to* a_h **do**
8 $A' \leftarrow \{\}$, $s' \leftarrow 0$
9 **for** $l \leftarrow 1$ *to* $|L_h^w|$ **do**
10 let t_k be the lth task in L_h^w
11 **if** $t_k \in \mathcal{T}_{h,j}$ *and* $\mathcal{M}_u(t_k) = \varnothing$ **then**
12 append t_k to A'
13 $s' \leftarrow s' + r_{k,h}$
14 **if** $|A'| = c_{h,j}$ **then**
15 **break**
16 **if** $s' > s$ **then**
17 $A \leftarrow A'$, $s \leftarrow s'$, $r \leftarrow j$
18 $\mathcal{M}(w_h) \leftarrow (A, p_{h,r})$
19 **foreach** $t \in A$ **do**
20 $\mathcal{M}(t) \leftarrow (w_h, p_{h,r})$
21 **return** $\mathcal{M}$

Theorem 3. *Algorithm 1 always produces a stable matching for uniform MCS instances.*

Proof. We prove this by contradiction. Assume the final matching contains an unhappy triad $\langle w_h, p_{h,j}, S \rangle$. Let T_i' denote the set of tasks that are unmatched in the beginning of the ith iteration of the for loop starting at line 3, so we have $T_1' = \mathcal{T}$. Also, let w_h be the kth worker in $L_{\mathcal{T}}$, i.e., the worker that is considered in the kth iteration. We first note that $\mathcal{T} \setminus T_k'$ is the set of tasks that have been matched before the kth iteration, and

$$S \cap (\mathcal{T} \setminus T_k') = \varnothing. \tag{12}$$

That is, S cannot contain any task that was matched before the kth iteration, because all tasks that were matched before the kth iteration were matched to a worker that precedes the worker w_h in $L_{\mathcal{T}}$. Therefore, the QoS scores of their partners must be equal to or greater than the QoS score of w_h due to

(2) and (3), which contradicts the unhappy triad definition due to (7). Then, by (12), we have $S \subseteq T'_k$, i.e., all tasks in S were unmatched in the beginning of the kth iteration. However, we match worker w_h with the best feasible task set in T'_k, thus we have

$$\sum_{t_x \in \mathcal{M}_u(w_h)} r_{x,h} \geq \sum_{t_y \in S} r_{y,h}. \tag{13}$$

This also contradicts the unhappy triad definition due to (8), hence we conclude that such an unhappy triad cannot exist in the matching produced by Algorithm 1. □

As a result of Theorem 3, we obtain the following corollary.

Corollary 1. *A stable matching always exists in all MCS instances with a uniform QoS setting.*

Running time. Forming the global preference list of tasks $L_{\mathcal{T}}$ in line 2 takes $O(m \log m)$ time. In each iteration of the for loop starting at line 3, we form the preference list of a worker (line 5) and iterate it once for each of his acceptable paths (lines 7-17), which respectively take $O(n \log n)$ and $O(na_{max})$ time, where $a_{max} = \max_{1 \leq i \leq m} a_i$. Thus, the overall time complexity of Algorithm 1 is $O(mna_{max} + mn \log n + m \log m)$.

B. Stable Task Assignment in General MCS Systems

In Algorithm 2, we present a pseudo-code description of our approximation algorithm for general MCS systems. In this algorithm, we attempt to match the tasks with their best preferences, but when we need to choose between the tasks that want to be matched with a worker due to the capacity or regional constraint (i.e., when we reach the capacity limit, or have tasks that are on different acceptable paths of the worker and hence cannot be matched to the worker at the same time), we choose a subset of these tasks that, though may not be optimal locally, have the best potential to yield the maximum total reward for the worker in the end based on the rewards they individually provide to the worker and the capacity of the corresponding path of the worker. Below, we first describe the steps of the algorithm, and then prove that it produces near-optimal matchings in terms of stability.

The algorithm begins by initializing the matching $\mathcal{M}$ in line 1, and three key variables x_i, σ_i and $index_k$ for each worker w_i and task t_k in lines 2-3. The variable σ_i keeps the value of the total reward to be obtained by worker w_i in the current matching, and x_i keeps the value of $r_{k,i} \times c_{i,j}$ for each worker w_i, where $r_{k,i}$ is the reward offered to worker w_i by the task (t_k) that has the maximum reward among the tasks that are currently matched to worker w_i, and $c_{i,j}$ is the capacity of the path $p_{i,j}$ currently selected for worker w_i. Thus, both x_i and σ_i are initialized to 0 in line 2. The variable $index_k$ keeps the index of the first worker in L_k^t that was not yet attempted to be matched to task t_k, so it is initially set to 1 for all tasks.

During execution of the algorithm, all tasks that are currently unmatched and are not yet attempted to be matched to all workers in their preference lists, i.e.,

$$\forall t_k \in \mathcal{T} : \mathcal{M}_u(t_k) = \varnothing \text{ and } index_k \leq |L_k^t|, \tag{14}$$

Algorithm 2: GeneralSTA ($\mathcal{W}$, $\mathcal{T}$)

Input: $\mathcal{W}$: Set of workers, $\mathcal{T}$: Set of tasks

1 let $\mathcal{M}(u) = (\varnothing, -)$ for all $u \in \mathcal{W} \cup \mathcal{T}$
2 let $x_i = \sigma_i = 0$ for all $1 \leq i \leq m$
3 let $index_k = 1$ for all $1 \leq k \leq n$
4 *Stack.push($\mathcal{T}$)*
5 **while** *Stack is not empty* **do**
6 $t_k \leftarrow$ *Stack.pop()*
7 **if** $index_k \leq |L_k^t|$ **then**
8 let w_i be the ($index_k$)th worker in L_k
9 $index_k \leftarrow index_k + 1$
10 $A \leftarrow \{\}$, $R \leftarrow \{\}$, $r \leftarrow 0$
11 **for** $j \leftarrow 1$ *to* a_i **do**
12 **if** $t_k \notin \mathcal{T}_{i,j}$ **then**
13 **continue**
14 $A' \leftarrow \{\}$, $R' \leftarrow \{\}$, $\sigma' \leftarrow 0$
15 **for** $l \leftarrow 1$ *to* $|\mathcal{M}_u(w_i)|$ **do**
16 let t_h be the lth task in $\mathcal{M}_u(w_i)$
17 **if** $t_h \in \mathcal{T}_{i,j}$ *and* $|A'| < c_{i,j}$ **then**
18 append t_h to A'
19 $\sigma' \leftarrow \sigma' + r_{h,i}$
20 **else**
21 append t_h to R'
22 insert t_k into A' by maintaining non-increasing order of task rewards
23 $\sigma' \leftarrow \sigma' + r_{k,i}$
24 **if** $|A'| > c_{i,j}$ **then**
25 let t_h be the last task in A
26 remove t_h from A'
27 append t_h to R'
28 $\sigma' \leftarrow \sigma' - r_{h,i}$
29 let t_h be the first task in A'
30 **if** $r_{h,i} \times c_{i,j} > x_i$ **then**
31 $x_i \leftarrow r_{h,i} \times c_{i,j}$
32 $A \leftarrow A'$, $R \leftarrow R'$, $\sigma_i \leftarrow \sigma'$, $r \leftarrow j$
33 **else if** $r_{h,i} \times c_{i,j} = x_i$ *and* $\sigma' > \sigma_i$ **then**
34 $A \leftarrow A'$, $R \leftarrow R'$, $\sigma_i \leftarrow \sigma'$, $r \leftarrow j$
35 **if** $|A| > 0$ **then**
36 $\mathcal{M}(w_i) \leftarrow (A, p_{i,r})$
37 **foreach** $t \in A$ **do**
38 $\mathcal{M}(t) \leftarrow (w_i, p_{i,r})$
39 **foreach** $t \in R$ **do**
40 $\mathcal{M}(t) \leftarrow (\varnothing, -)$
41 *Stack.push(t)*
42 **else**
43 *Stack.push(t_k)*
44 **return** $\mathcal{M}$

reside in a stack that is initialized in line 4. In the while loop starting in line 5, we attempt to match one (t_k) of the tasks in

the stack with the next worker (w_i) in its preference list (L_k^t) until there is no task left in the stack. When we attempt to match task t_k to worker w_i, we check each of the acceptable paths of worker w_i in non-increasing order of path capacities (i.e., $p_{i,1}, p_{i,2}, .., p_{i,a_i}$) in the for loop starting in line 11. During this process, we respectively maintain the task set and the path that we would like to assign to worker w_i after checking each path in the variables A and r, and maintain the set of tasks that are currently matched to worker w_i, but need to be removed from his assignment set for worker w_i to be able to match with A in the variable R. For each path $p_{i,j} : t_k \in \mathcal{T}_{i,j}$ (lines 12-13), we first find the best task set A' among the tasks in $\mathcal{M}_u(w_i) \cup \{t_k\}$ within the capacity constraint of $p_{i,j}$ (lines 14-28), and then choose the task set A' over the task set A (and update the variables A, R, σ_i and r accordingly) if one of the following two conditions is satisfied:

- (lines 30-32) x_i increases (regardless of the change in the total reward σ_i to be collected by worker w_i),
- (lines 33-34) x_i remains unchanged, but the value of σ_i increases.

Finally, in lines 35-43, if A is non-empty, we match worker w_i and the tasks in A with each other, set the tasks in R free, and push them back onto the stack. Otherwise, we only push task t_k onto the stack.

Theorem 4. *Algorithm 2 always produces a κ-stable matching for a general MCS instance, where κ is the maximum path capacity in the instance, i.e.,*

$$\kappa = \max_{\substack{1 \le i \le m \\ 1 \le j \le a_i}} c_{i,j}. \tag{15}$$

Proof. We prove this by contradiction as well. Assume that there is a unhappy triad $\langle w_i, p_{i,j}, S \rangle$ in the final matching $\mathcal{M}$ produced by the algorithm, which breaks the κ-stability of the matching. Thus, we must have

$$\sum_{t_x \in \mathcal{M}_u(w_i)} r_{x,i} \times \kappa < \sum_{t_y \in S} r_{y,i}. \tag{16}$$

We first note that all tasks in S must have been attempted to be matched to worker w_i at some point during the execution of the algorithm, because, by definition of unhappy triad (7), they must either currently be matched to worker w_i, or prefer worker w_i to their current assignments in $\mathcal{M}$. The latter case indicates that worker w_i precedes their current assignments in their preference lists (line 8), thus they have been attempted to matched worker w_i before they ended up getting matched with their current assignments.

We then note that every time a task that would increase the value of x_i (line 30) is being attempted to match to worker w_i, it will certainly be matched to worker w_i in that iteration, and increase x_i (line 31), which will have the maximum value possible in the end. Thus, we have

$$\forall t_e \in E : x_i \ge r_{e,i} \times \max_{1 \le h \le a_i} c_{i,h}, \tag{17}$$

where E is the set of tasks that were attempted to matched to worker w_i during the execution of the algorithm. Since $S \subseteq E$, we have

$$\begin{aligned} \forall t_s \in S : x_i &\ge r_{s,i} \times \max_{1 \le h \le a_i} c_{i,h}, \\ &\ge r_{s,i} \times c_{i,j}. \end{aligned} \tag{18}$$

Recall that $x_i = r_{m,i} \times c_{i,g}$, where (i) $r_{m,i}$ is the reward of task $t_m \in \mathcal{M}_u(w_i)$, which has the highest reward among the tasks in $\mathcal{M}_u(w_i)$, and $c_{i,g}$ is the capacity of $p_{i,g} = \mathcal{M}_p(w_i)$. Then, by (18), we get

$$\forall t_s \in S : r_{m,i} \times c_{i,g} \ge r_{s,i} \times c_{i,j}. \tag{19}$$

For the condition in (16) to hold, we must have

$$\sum_{t_x \in \mathcal{M}_u(w_i)} r_{x,i} \times \max\{c_{i,g}, c_{i,j}\} < \sum_{t_y \in S} r_{y,i}, \tag{20}$$

because $\max\{c_{i,g}, c_{i,j}\} \le \kappa$. If $c_{i,g} > c_{i,j}$, we would have

$$\sum_{t_x \in \mathcal{M}_u(w_i)} r_{x,i} \times c_{i,g} < \sum_{t_y \in S} r_{y,i} \tag{21a}$$

$$r_{m,i} \times c_{i,g} < \sum_{t_y \in S} r_{y,i} \text{ (by (i))} \tag{21b}$$

$$\forall t_s \in S : r_{s,i} \times c_{i,j} < \sum_{t_y \in S} r_{y,i} \text{ (by (19))}. \tag{21c}$$

For the task $t_{s'}$ with the highest reward in S, (21c) yields

$$r_{s',i} \times c_{i,j} \quad < \quad \sum_{t_y \in S} r_{y,i}, \tag{22}$$

which is a contradiction as S cannot contain more than $c_{i,j}$ tasks due to the capacity constraint of path $p_{i,j}$.

On the other hand, if (ii) $c_{i,g} \le c_{i,j}$, we would have

$$\sum_{t_x \in \mathcal{M}_u(w_i)} r_{x,i} \times c_{i,j} < \sum_{t_y \in S} r_{y,i} \tag{23a}$$

$$r_{m,i} \times c_{i,j} < \sum_{t_y \in S} r_{y,i} \text{ (by (i))} \tag{23b}$$

$$r_{m,i} \times c_{i,g} < \sum_{t_y \in S} r_{y,i} \text{ (by (ii))} \tag{23c}$$

which also leads to a contradiction as (23c) is identical to (21b). Therefore, we conclude that there cannot exist any unhappy triad that violates κ-stability in the matching produced by Algorithm 2, and it is always κ-stable. □

Running time. Algorithm 2 requires to form only the preference lists of the tasks, which takes $O(nm \log m)$ time. During the execution of the algorithm, each task t_k can be pushed on the stack at most $|L_k^t| \le m$ times, so the while loop starting in line 5 will iterate $O(mn)$ times. The for loop starting in line 11 will iterate at most $a_{max} = \max_{1 \le i \le m} a_i$ times, and the most expensive operation in it is the for loop starting in line 15, which can iterate at most $c_{max} = \max_{1 \le i \le m, 1 \le j \le a_i} c_{i,j}$ times, as the size of the assignment set of a worker cannot be larger than the maximum path capacity c_{max} in the instance. Thus, the worst-case running time of Algorithm 2 is $O(nm \log m + nmc_{max}a_{max})$.

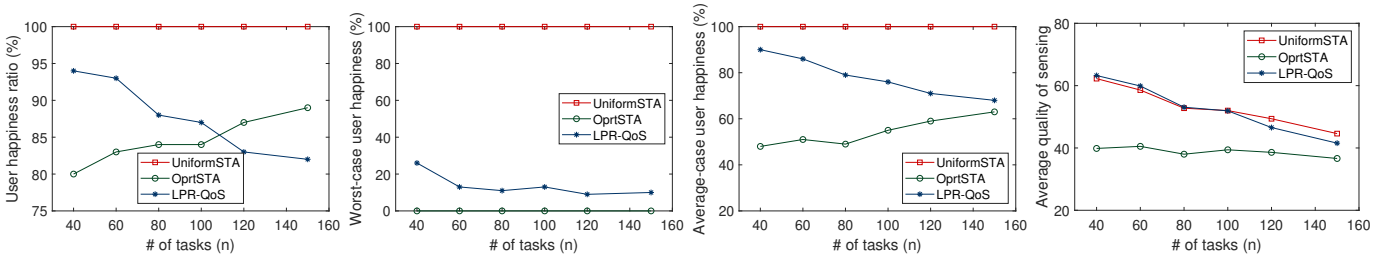

Fig. 4: Performance of algorithms with varying task counts in uniform MCS (m=30).

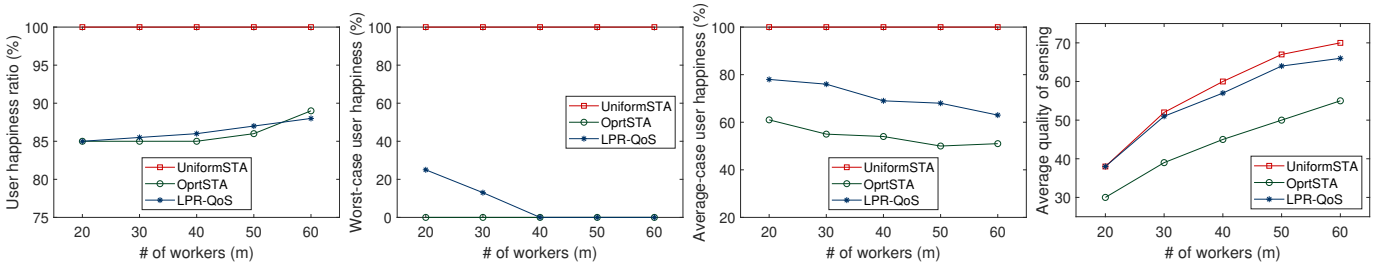

Fig. 5: Performance of algorithms with varying worker counts in uniform MCS (n=100).

V. Simulation Results

In this section, we present the empirical evaluation of the proposed algorithms.

A. Data set

For our simulations, we generate an SO-MCS instance in a real environment as follows. We randomly select n places of interest (PoI) in Lower Manhattan from the PoI list [37] provided by the City of New York, and create a task at each of these places. For each (w_i) of m workers in the instance, we randomly select two PoIs that are [2-4] kilometers away from each other from the same PoI set, and use these as their starting points and destinations. We then get $a_i \sim U\{4, 6\}$ different routes between these two PoIs using the Google's Directions API [38]. We obtain the best (shortest) path of w_i, which has the maximum capacity $c_{i,1} \sim U\{3, 5\}$, by requesting a direct route, and obtain the remaining paths by requesting a route with a waypoint at one of the PoIs located in the smallest circle that encloses the bird-eye route between the starting point and destination. The capacity of each $p_{i,j}$ of the latter paths is set as $c_{i,1} - \lfloor d/300 \rfloor$, where d is the route length difference (in meters) between $p_{i,1}$ and $p_{i,j}$. For each task-path pair $(t_k, p_{i,j})$, we add t_k to $\mathcal{T}_{i,j}$ if and only if t_k is within 50 meters of any point on $p_{i,j}$. Lastly, to create a uniform instance, we assign a global QoS score $q_i \sim U\{50, 100\}$ to each worker, and let $q_{i,j} = q_i$, $\forall t_j \in \mathcal{T}$. On the other hand, to create a general (non-uniform) instance, we simply let $q_{i,j} \sim U\{50, 100\}$ for all worker-task pairs (w_i, t_j). Task requesters are assumed to be offering rewards proportional to the QoS they will get from each worker, thus we let $r_{j,i} = q_{i,j} \times b_j$, where $b_j \sim U(0.2, 1)$ is the reward to QoS ratio of task t_j.

B. Benchmark algorithms

We compare the proposed algorithms (i.e., *UniformSTA* and *GeneralSTA*) with the following algorithms.

- *OprtSTA*: This algorithm finds the optimal solution in terms of stability in opportunistic MCS systems. We transform our semi-opportunistic instances to opportunistic ones by only considering the shortest path of each worker (which has the largest capacity). In the resulting instance, a stable matching can be found by the classic Gale-Shapley [18] algorithm in $O(mn)$ time.
- *LPR-QoS* [2]: This algorithm uses the linear programming relaxation (LPR) technique, and finds a task assignment for SO-MCS systems based on the solution of the relaxed version of the integer program that maximizes the total QoS of the workers assigned to the tasks. We use Google OR-Tools[1] to implement this algorithm.

C. Performance metrics

- *User happiness ratio:* The ratio of the number of triads that are not unhappy to the total number of triads that can be matched in any feasible matching.
- *Worst-case user happiness:* This is the value of the objective function defined in (9), i.e., the α-stability of the produced matching.
- *Average-case user happiness:* This is computed by $\sum_{w_i \in \mathcal{W}} (1/\delta^i_{max})/m$, where δ^i_{max} is the dissatisfaction ratio of the unhappy triad that causes the largest utility loss for worker w_i ($\delta^i_{max} = 1$ if w_i does not form any unhappy triads).

We also analyze the average QoS provided to task requesters and the running times of the algorithms.

[1]https://developers.google.com/optimization

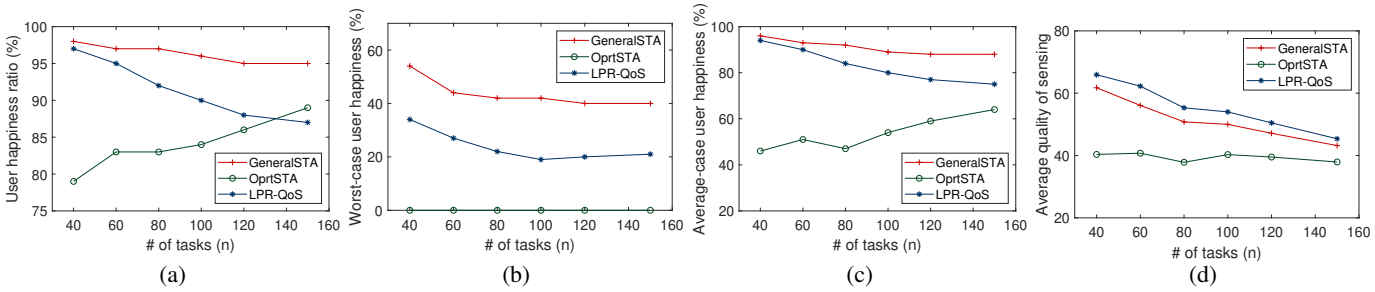

Fig. 6: Performance of algorithms with varying task counts in general MCS (m=30).

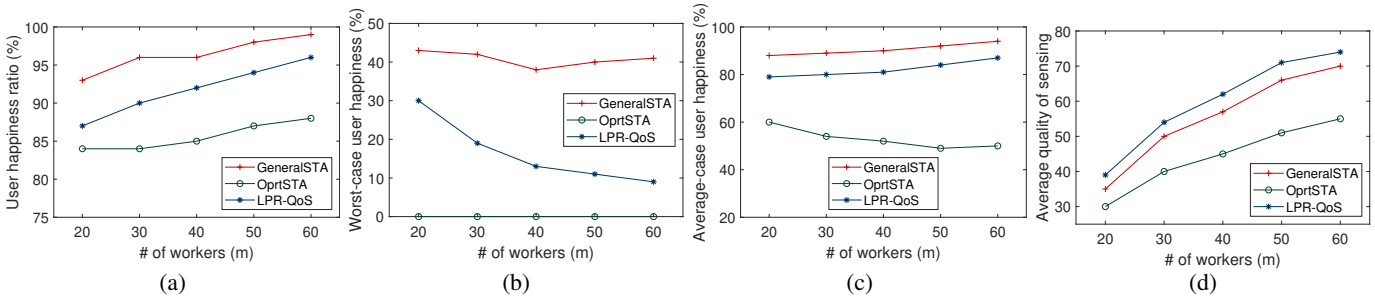

Fig. 7: Performance of algorithms with varying worker counts in general MCS (n=100).

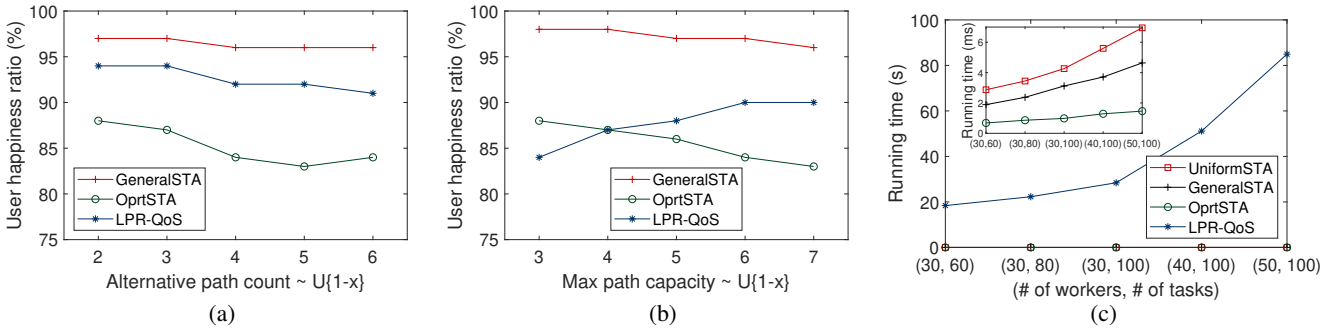

Fig. 8: User happiness (n=80, m=30) with varying path counts (a) and capacities (b), and running times of algorithms (c).

D. Results

We first look at the performance of the algorithms in the uniform instances with varying numbers of tasks (Fig. 4) and workers (Fig. 5). As expected (due to Theorem 3), our UniformSTA algorithm always achieves perfect user happiness scores, and greatly outperforms the benchmark algorithms. Moreover, it achieves to deliver a comparable average QoS score with the LPR-QoS algorithm. On the other hand, the OprtSTA algorithm mostly produces task assignments with the lowest user happiness scores, despite considering the user preferences during the matching process. This is because it disregards the alternative paths of workers along with the additional matching options they provide, and thus demonstrates the advantage of semi-opportunistic sensing over opportunistic sensing.

In Fig. 6 & 7, we look at the results on the general (non-uniform) MCS instances, which clearly show the superiority of our GeneralSTA algorithm over the other algorithms in terms of user happiness, particularly in terms of worst-case user happiness. On the other hand, in this setting, our algorithm provides slightly lower average quality of sensing than LPR-QoS algorithm. Also, in both uniform and general MCS instances, the QoS scores of all algorithms generally grow with increasing worker density, as tasks are more likely to get assigned to a worker when there is a larger number of workers in the instance.

Next, we analyze the performance of the algorithms with varying ranges of alternative path counts $(a_i - 1)$ and path capacities $(c_{i,1})$ in Fig. 8a and Fig. 8b, respectively. We observe that our GeneralSTA algorithm generally has a stable performance, and maintains its superiority in terms of user happiness regardless of the changes in these parameters. The performance of the OprtSTA algorithm is usually worse when workers have more alternative paths (and a higher task performing capacity on these paths), because, in these scenarios, the OprtSTA algorithm ends up failing to take advantage of a larger number of assignment possibilities created by alternative paths.

Finally, in Fig. 8c, we present the running times of the algorithms on uniform instances (this is to show the results for all four algorithms) with different worker-task counts. We note that the LPR-QoS algorithm has an excessive running time, which is a few orders of magnitude larger than that of the other algorithms. On the other hand, the OprtSTA algorithm has the shortest running time despite its poor performance in

terms of user happiness and average QoS in most settings. Lastly, our algorithms have a comparable running time, with the GeneralSTA algorithm being slightly faster.

VI. Conclusion

In this paper, we have introduced the preference-aware task assignment problem in a semi-opportunistic mobile crowdsensing setting. We have formally defined the requirements for preference-awareness (or user happiness) and shown that it is not possible to generate a perfectly preference-aware task assignment that satisfies all users in some instances. We have studied the problem in a system model with uniform worker qualities as well as in a non-restricted model, and presented an exact and an approximation task assignment algorithm, both with a polynomial-time complexity, for these models, respectively. Results of the simulations have shown that the proposed algorithms achieve to produce task assignments with significantly larger user happiness scores compared to the benchmark algorithms. In future work, we will investigate the user-centric task assignment problem in a system model that allows different workers to adopt different sensing modes.

References

[1] 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 & tutorials*, vol. 21, no. 3, pp. 2419–2465, 2019.

[2] W. Gong, B. Zhang, C. Li, and Z. Yao, "Task allocation in semi-opportunistic mobile crowdsensing: Paradigm and algorithms," *Mobile Networks and Applications*, pp. 1–11, 2019.

[3] X. Yin, Y. Chen, C. Xu, S. Yu, and B. Li, "Matchmaker: Stable task assignment with bounded constraints for crowdsourcing platforms," *IEEE Internet of Things Journal*, 2020.

[4] 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.

[5] Y. Liu, B. Guo, Y. Wang, W. Wu, Z. Yu, and D. Zhang, "TaskMe: Multi-task allocation in mobile crowd sensing," in *ACM international joint conference on pervasive and ubiquitous computing*, 2016, pp. 403–414.

[6] W. Gong, B. Zhang, and C. Li, "Location-based online task assignment and path planning for mobile crowdsensing," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1772–1783, 2018.

[7] X. Tao and W. Song, "Task allocation for mobile crowdsensing with deep reinforcement learning," in *IEEE Wireless Communications and Networking Conference (WCNC)*, 2020, pp. 1–7.

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

[9] Y. Zhao, K. Zheng, Y. Li, H. Su, J. Liu, and X. Zhou, "Destination-aware task assignment in spatial crowdsourcing: A worker decomposition approach," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 12, pp. 2336–2350, 2019.

[10] M. Zhang, P. Yang, C. Tian, S. Tang, X. Gao, B. Wang, and F. Xiao, "Quality-aware sensing coverage in budget-constrained mobile crowdsensing networks," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 9, pp. 7698–7707, 2015.

[11] J. Chen and J. Yang, "Maximizing coverage quality with budget constrained in mobile crowd-sensing network for environmental monitoring applications," *Sensors*, vol. 19, no. 10, p. 2399, 2019.

[12] M. Karaliopoulos, O. Telelis, and I. Koutsopoulos, "User recruitment for mobile crowdsensing over opportunistic networks," in *IEEE Conference on Computer Communications (INFOCOM)*, 2015, pp. 2254–2262.

[13] Y. Zhan, Y. Xia, Y. Liu, F. Li, and Y. Wang, "Incentive-aware time-sensitive data collection in mobile opportunistic crowdsensing," *IEEE Trans. on Vehicular Technology*, vol. 66, no. 9, pp. 7849–7861, 2017.

[14] F. Yucel and E. Bulut, "Location-dependent task assignment for opportunistic mobile crowdsensing," in *IEEE 17th Annual Consumer Communications & Networking Conference (CCNC)*, 2020, pp. 1–6.

[15] Z. He, J. Cao, and X. Liu, "High quality participant recruitment in vehicle-based crowdsourcing using predictable mobility," in *IEEE Conference on Computer Communications (INFOCOM)*, 2015, pp. 2542–2550.

[16] M. Xiao, J. Wu, L. Huang, R. Cheng, and Y. Wang, "Online task assignment for crowdsensing in predictable mobile social networks," *IEEE Trans. on Mobile Computing*, vol. 16, no. 8, pp. 2306–2320, 2016.

[17] 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, 2019.

[18] D. Gale and L. Shapley, "College admissions and stability of marriage. american mathematicas monthly, 69, 9-15," 1962.

[19] "National resident matching program," 2020. [Online]. Available: https://www.nrmp.org/matching-algorithm/

[20] S. Shamaei, S. Bayat, and A. M. A. Hemmatyar, "Interference management in D2D-enabled heterogeneous cellular networks using matching theory," *IEEE Trans. on Mobile Computing*, vol. 18, no. 9, pp. 2091–2102, 2018.

[21] F. Yucel, A. Bhuyan, and E. Bulut, "Secure, Resilient and Stable Resource Allocation for D2D-based V2X Communication," in *IEEE Resilience Week (RWS)*, 2020, pp. 71–77.

[22] H. Zheng and J. Wu, "Online to offline business: urban taxi dispatching with passenger-driver matching stability," in *IEEE 37th International Conf. on Distributed Computing Systems (ICDCS)*, 2017, pp. 816–825.

[23] D. Manlove, *Algorithmics of matching under preferences*. World Scientific, 2013, vol. 2.

[24] D. Knuth, "Mariages stables," *Les Presses de l'Universit de Montral*, 1976.

[25] C. Ng and D. S. Hirschberg, "Three-dimensional stabl matching problems," *SIAM Journal on Discrete Mathematics*, vol. 4, no. 2, pp. 245–252, 1991.

[26] J. Wu, "Stable matching beyond bipartite graphs," in *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2016, pp. 480–488.

[27] L. Cui and W. Jia, "Cyclic stable matching for three-sided networking services," *Computer Networks*, vol. 57, no. 1, pp. 351–363, 2013.

[28] F. Yucel, M. Yuksel, and E. Bulut, "QoS-based Budget Constrained Stable Task Assignment in Mobile Crowdsensing," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2020.

[29] M. Abououf, S. Singh, H. Otrok, R. Mizouni, and A. Ouali, "Gale-shapley matching game selection—a framework for user satisfaction," *IEEE Access*, vol. 7, pp. 3694–3703, 2018.

[30] F. Yucel and E. Bulut, "User satisfaction aware maximum utility task assignment in mobile crowdsensing," *Computer Networks*, vol. 172, p. 107156, 2020.

[31] F. Yucel and E. Bulut, "Online stable task assignment in opportunistic mobile crowdsensing with uncertain trajectories," *IEEE Internet of Things Journal*, 2021.

[32] F. Yucel, M. Yuksel, and E. Bulut, "Coverage-aware stable task assignment in opportunistic mobile crowdsensing," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 4, pp. 3831–3845, 2021.

[33] B. Li, Y. Cheng, Y. Yuan, G. Wang, and L. Chen, "Three-dimensional stable matching problem for spatial crowdsourcing platforms," in *25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 1643–1653.

[34] T. Song, Y. Tong, L. Wang, J. She, B. Yao, L. Chen, and K. Xu, "Trichromatic online matching in real-time spatial crowdsourcing," in *IEEE 33rd International Conf. on Data Engineering (ICDE)*, 2017, pp. 1009–1020.

[35] B. Zheng, C. Huang, C. S. Jensen, L. Chen, N. Q. V. Hung, G. Liu, G. Li, and K. Zheng, "Online trichromatic pickup and delivery scheduling in spatial crowdsourcing," in *IEEE 36th International Conference on Data Engineering (ICDE)*, 2020, pp. 973–984.

[36] Wikipedia contributors, "Plastic number - Wikipedia, the free encyclopedia," 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Plastic_number&oldid=1053611441

[37] "NYC Points of Interest," Department of Information Technology & Telecommunications (DOITT), 2021. [Online]. Available: https://data.cityofnewyork.us/City-Government/Points-Of-Interest/rxuy-2muj#revert

[38] "Google Directions API," 2021. [Online]. Available: https://developers.google.com/maps/documentation/directions/overview