2021

PREFERENCE-AWARE TASK ASSIGNMENT IN MOBILE CROWDSENSING

Fatih Yucel
Virginia Commonwealth University

PREFERENCE-AWARE TASK ASSIGNMENT IN MOBILE CROWDSENSING

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

by

FATIH YUCEL

Doctorate in Computer Engineering with specialization in Computer Science - 2017-2021

Director: Dr. Eyuphan Bulut,

Associate Professor, Department of Computer Science

Virginia Commonwealth University

Richmond, Virginia

November, 2021

Acknowledgements

I would like to thank my advisor, Dr. Eyuphan Bulut, for his endless support from the very beginning of my doctoral studies. I am truly grateful for his invaluable guidance, expertise, encouragement, and boundless patience.

I would also like to extend my thanks to Dr. Carol Fung, Dr. Preetam Ghosh, Dr. Burak Kantarci, and Dr. Murat Yuksel for their kind approval to serve on my dissertation committee amid their many commitments.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract

PREFERENCE-AWARE TASK ASSIGNMENT IN MOBILE CROWDSENSING

By Fatih Yucel

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

Virginia Commonwealth University, 2021.

Director: Dr. Eyuphan Bulut,
Associate Professor, Department of Computer Science

Mobile crowdsensing (MCS) is an emerging form of crowdsourcing, which facilitates the sensing data collection with the help of mobile participants (workers). A central problem in MCS is the assignment of sensing tasks to workers. Existing work in the field mostly seek a system-level optimization of task assignments (e.g., maximize the number of completed tasks, minimize the total distance traveled by workers) without considering individual preferences of task requesters and workers. However, users may be reluctant to participate in MCS campaigns that disregard their preferences. In this dissertation, we argue that user preferences should be a primary concern in the task assignment process for an MCS campaign to be effective, and we develop preference-aware task assignment (PTA) mechanisms for five different MCS settings. We first look at the PTA problem in a setting, where the objective of task requesters is to maximize their additive utility functions based on worker qualities by recruiting multiple high-quality workers within their budgets, and that of workers is to maximize their profits. Following that, we consider a more general

model that allows for non-additive, coverage-based utility functions to describe task requester preferences. We then study the PTA problem in a setting with uncertain and uncontrollable worker trajectories, where the assignments have to be made in an online manner based on the visits of workers to task regions unfolding in real time. Next, we consider a semi-controlled mobility setting, where workers are asked to provide multiple paths they find acceptable between their starting locations and destinations. Finally, we investigate the problem of finding a maximum size task assignment that satisfies user preferences as much as possible. Since the PTA problem is computationally hard in most of these settings, we present efficient approximation and heuristic algorithms. Extensive simulations performed on synthetic and real data sets validate our theoretical results, and demonstrate that the proposed algorithms produce near-optimal solutions in terms of preference-awareness, outperforming the state-of-the-art assignment algorithms by a wide margin in most cases.

CHAPTER 1

INTRODUCTION

Mobile crowdsensing (MCS) is an emerging form of crowdsourcing that aims to accomplish spatiotemporal sensing tasks with the help of mobile participants (workers). It connects the entities that need to collect a certain type of sensing data from a set of regions of interest with the individuals who are willing to perform the desired sensing tasks for them in return for monetary rewards or voluntarily to support the cause of the sensing campaign (e.g., environmental protection [1]). Notable recent applications of MCS can be found in various fields such as urban planning [2], public safety [3], and indoor localization [4]. A recent study [5] also discusses the timely applications of MCS in Spain to address the COVID-19 outbreak. Typical sensing tasks in MCS include recording noise pollution for a urban planning system [2] and taking pictures of buildings for a place naming system [6].

According to the involvement level of workers, MCS applications can be classified into two types [7]: *participatory* and *opportunistic*. In the former, workers are expected to travel to the regions of the assigned sensing tasks by interrupting their own schedules for a period of time, while in the latter they are asked to perform a sensing task if they are already in or close to the task region. Thus, participatory MCS campaigns usually have shorter task completion times as workers are immediately dispatched to the task regions. However, they require workers to devote their resources and time to carry out the assigned tasks, which introduces significant extra costs for workers who then need to be compensated for these costs by task requesters.

Opportunistic MCS campaigns, on the other hand, only deal with the costs of

sensing and delivery of the sensed data, which tend to be much smaller compared to the time and travel costs incurred in participatory MCS campaigns. However, for opportunistic MCS campaigns to function effectively, the assignment of sensing tasks to workers should be made very efficiently especially under capacity or budget constraints [8], because the assignment opportunities hinge upon the presence of workers in the task regions, and they emerge and vanish as workers move.

Accordingly, a pivotal problem in both participatory and opportunistic MCS campaigns is the assignment of sensing tasks to workers. In fact, the utility of an MCS system is generally quantified by the quality of the assignments made by the adopted task assignment algorithm. However, the quality of assignments has been measured differently in the literature. For example, some studies [36, 11] favor the assignments that minimize the travel distance of workers, while others [39, 12] prefer those that maximize the total quality of service (QoS) received by task requesters. Despite the variety of existing task assignment algorithms [40], the ultimate goal of the assignment is mostly defined as the maximization of a system utility without considering the individual user needs and preferences. However, such solutions may result in dissatisfied users and impair their future participation, as users in practice may not want to sacrifice their individual convenience for the system utility.

Preference-awareness has been extensively studied in general bipartite matching problems especially in the economics literature [41] under the name of *stable matching problem*. However, these studies do not tackle the issues peculiar to MCS such as budget constraints of task requesters, uncertain matching opportunities due to unknown worker trajectories, and time constraints of tasks. In this dissertation, we study the preference-aware (or stable) task assignment problem in MCS in a variety of settings. Below, we first discuss the benefits of considering the concept of preference-awareness (or stability) in the task assignment process of an MCS campaign, and

then summarize our contributions.

Stability is an important concept in matching problems with selfish and rational individuals. In broad terms, it defines the satisfaction of users with their assignments. A stable matching promotes long-term user participation by making certain that users are not upset by being forced into less favorable assignments whilst there are better options available. Thus, considering stability in the matching process is not only beneficial for task requesters and workers, but also for the platform.

The goal with the stability is to ensure that no user u (i.e., a worker or a requester) can lay claim to have deserved a better assignment. That is, all the possible assignments that user u prefers[1] more than his/her current assignment in the stable matching are matched with someone they prefer to user u; thus, they would not like to break up with their current assignments to get matched with user u. This ensures that workers will be satisfied with their assignments and will be motivated to carry out their tasks, which will in turn improve their performance [43] and the quality of results. Moreover, this also ensures that task requesters will obtain the most benefit while respecting the workers' preferences.

A stable matching solution based on individual user preferences also allows incorporating various metrics that are appraised uniquely by each user. For example, the priority of a worker might be the proximity to the task location, so he would form his preference list in a way that the closer tasks precede the others. On the other hand, another worker who does not mind traveling long distances can form his preference list solely based on the rewards he will be paid. Moreover, they can even reflect their own personal interests in their preference lists such as location of tasks being close to

[1]When we use the active voice for a task (e.g., prefers, pays a reward), we mean the person/entity that corresponds to it, i.e., the task requester/owner. Also, we use male and female subject pronouns for workers and tasks respectively as a reference to the original stable marriage problem [42].

their home or work, and pleasure of performing the tasks (e.g., preference for taking pictures of a scene of interest over recording noise pollution).

Despite these benefits attached to stability, user preferences have been mostly ignored in the existing literature on the task assignment problem in MCS. In this dissertation, we study the preference-aware task assignment problem in various types of MCS systems, define the stability conditions peculiar to MCS systems, develop efficient task assignment algorithms, and conduct extensive experiments using real and synthetic data sets to show their performances in comparison with the benchmark task assignment algorithms. Our major contributions are as follows:

- In Chapter 3, we study the many-to-one stable task assignment problem in a system model where task requesters aim to maximize the quality of sensing (or service) they get by hiring multiple high-quality workers within their budget. We prove that a stable matching may not exist in some MCS instances in this model, and that it is NP-hard to find a stable matching (if exists). We propose two different pseudo-polynomial time approximation algorithms, which produce 2-approximate and pairwise (i.e., a relaxed stability requirement) stable task assignments under certain assumptions about worker qualities and task rewards. We also develop a pseudo-polynomial time heuristic algorithm that guarantees the perfect happiness of at least one task requester. Through extensive simulations, we show that the proposed algorithms significantly outperform the benchmark stable matching algorithms.

  Related publication [44]: Fatih Yucel, Murat Yuksel, and Eyuphan Bulut."QoS-based budget constrained stable task assignment in mobile crowdsensing." IEEE Transactions on Mobile Computing (2020).

- In Chapter 4, we investigate the problem of finding a stable matching between

workers that perform assigned tasks only in an opportunistic manner and complex sensing tasks with coverage requirements. In presence of coverage requirements, the utility of workers for sensing tasks become non-additive (different from the problem studied in Chapter 3, which deals with additive utility functions based on worker qualities) due to overlaps in the regions covered by workers. We begin by showing that the nonexistence and hardness results mentioned above hold for this problem as well. We then present a polynomial-time task assignment algorithm, and derive its (non-constant) approximation ratio. However, we prove that a variant of this algorithm has a constant approximation ratio under the assumption that task rewards are proportional to coverage-based utilities of workers. In simulations, we compare the performance of our algorithms with the state-of-the-art task assignment algorithms that also consider coverage requirements of task requesters, and show that our algorithms not only make users happier with their assignments, but also achieve better system utility scores in most of the scenarios examined.

Related publication [45]: Fatih Yucel, Murat Yuksel, and Eyuphan Bulut. "Coverage-aware stable task assignment in opportunistic mobile crowdsensing." IEEE Transactions on Vehicular Technology (2021).

- In Chapter 5, we study the problem in an online setting, where workers perform sensing tasks only opportunistically, but their trajectories are uncertain or unknown to the matching platform in advance. This introduces novel issues such as uncertain matching opportunities between worker-task pairs, and makes the existing stability conditions invalid. Thus, we first describe how to measure stability in this online setting, and then present a method to compute the average utility of users in all possible (exponentially many) stable matchings in

a given matching instance in polynomial-time, which enables us to make optimal decisions in the online setting. We propose efficient online task assignment algorithms for MCS systems with and without capacity constraints, and prove their optimality. Extensive simulations performed on both synthetic and real data sets validate our theoretical results, and demonstrate that the proposed algorithms significantly outperform the existing solutions.

Related publication [46]: Fatih Yucel and Eyuphan Bulut. "Online stable task assignment in opportunistic mobile crowdsensing with uncertain trajectories." IEEE Internet of Things Journal (2021).

- In Chapter 6, we consider a semi-opportunistic sensing mode, where 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 costlier and hence undesirable paths as in participatory MCS. We formally define the stability conditions for this three-dimensional task assignment problem (i.e., workers/paths/tasks), and propose two polynomial-time task assignment algorithms: an exact algorithm for systems with uniform worker qualities, and a c-approximate algorithm for general systems (i.e., with/without uniform worker qualities), where c is the number of the acceptable paths of the worker with the largest set of acceptable paths. We also provide an empirical analysis of the proposed algorithms.

  Related publication: Fatih Yucel and Eyuphan Bulut. "Three-dimensional stable task assignment in semi-opportunistic mobile crowdsensing." (Submitted for publication).

- In Chapter 7, we consider the happiness of the matching platform as well as the

happiness of the workers and task requesters. We assume that the happiness of the matching platform is proportional to the number of matched users, and study the problem of finding maximum size task assignments with as few unhappy worker-task pairs as possible, which turns out to be NP-hard. We first present an Integer Linear Programming model to solve the problem optimally. Then, we propose two novel methods to adjust the number of matched users and happiness of users in a given task assignment by re-matching a carefully-selected user set. Based on these methods, we develop two polynomial-time heuristic algorithms, and show via extensive simulations that they produce near-optimal task assignments.

Related publication [47]: Fatih Yucel and Eyuphan Bulut. "User satisfaction aware maximum utility task assignment in mobile crowdsensing." Computer Networks 172 (2020): 107156.

Next, we present a summary of the related work in Chapter 2. In the following five chapters, we study the stable task assignment problem in different settings as outlined above. Finally, we provide our concluding remarks in Chapter 8.

CHAPTER 2

LITERATURE REVIEW

In this chapter, we review the recent studies on mobile crowdsensing and matching under preferences. A comprehensive survey on mobile crowdsensing and a summary of various matching problems with user preferences can be found in [48] and [49], respectively.

2.1 Mobile Crowdsensing

Mobile crowdsensing has attracted a lot of attention recently, and a number of studies have explored the issues that need to be addressed to realize its true potential. One of the key problems investigated in these studies is the task assignment (or worker recruitment) problem since the overall performance of an MCS system and the satisfaction of its users are highly dependent on the efficiency of the assignments. These studies have considered various objectives in the task assignment process such as maximizing the number of completed tasks [25], minimizing the completion times of tasks [13], minimizing the incentives provided to the users [14, 15, 29], assuring the task or sensing quality [12] under some constraints on traveling distance [11], energy consumption [39], and expenses of task requesters [26]. Beyond these works, the issues of security [16], privacy [17, 18], and truthfulness [22, 19] have also been considered in the worker recruitment process.

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. In [27], the authors investigate the problem of minimizing

the total travel costs of the workers while maximizing the number of completed tasks and keeping the rewards to be paid to the workers as low as possible. In [11], the authors study the task assignment problem in an online setting, and aim to maximize the total task quality while ensuring that the travel costs of the workers do not exceed their individual travel budgets. In [20], the authors adopt a system model in which each worker has a maximum travelling distance that needs to be considered in the assignment process, and the objective is to maximize the profit of the platform. The authors propose a deep reinforcement learning-based scheme that significantly outperforms the other heuristic algorithms. In [21], the goal is also to minimize the travel distance of workers, however, differently from the aforementioned studies, the authors consider the issue of user privacy, and present a mechanism that finds the task assignments without exposing any private information about workers or task requesters. Lastly, in [28], the authors study the destination-aware task assignment problem in participatory crowdsourcing systems. The unique aspect of this problem is the constraint that each worker should be able to arrive at his destination by the deadline he specifies, thus he cannot be assigned to too many tasks or tasks that are far away from his destination.

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 trajectory of a worker). In [8], the authors study the maximum coverage task assignment problem in opportunistic MCS with worker trajectories that are known beforehand. It is assumed that each task needs to be performed at a certain point of interest and has a weight that indicates how important its completion is to the platform, which has a fixed budget and can hence recruit only so many workers. The objective of the platform is to select a set of workers within the budget constraints, which maximizes the

weighted coverage over the set of tasks according to the given trajectories of workers. The authors develop a $(1 - 1/e)$-approximate algorithm with a time complexity of $\mathcal{O}(n^5)$, where n is the number of workers in the system. [9] studies the same problem, and proposes a greedy algorithm that, despite not having a theoretical guarantee, outperforms the algorithm proposed in [8] in terms of achieved coverage in certain settings, and runs in $\mathcal{O}(n^2)$ time.

The problem studied in [10] differs from those in [8] and [9] as it also considers the delivery of the sensed data in an opportunistic manner. That is, after carrying out a task, a worker should either deliver the sensed data to the server through one of the collection points (i.e., WiFi APs) on his trajectory, or transmit it to another user who will deliver it for him. Thus, here, not only does the platform need to estimate whether and when workers would visit task regions and collection points, but it is also crucial to obtain and utilize the encounter frequencies of workers to improve the delivery probability of the sensed data. The authors present different approximation algorithms for the systems with deterministic and uncertain worker trajectories, and evaluate their performance on real data sets. The data delivery aspect of the problem in [10] has also been studied in [23] and [24]. They both utilize Nash Bargaining Theory to decide on whether or not selfish data collectors and mobile (relay) users who only take part in delivery of sensed data would like to cooperate with each other according to their utility in either scenario. However, in [24], the authors consider a more complete mobile social network model and present an enhanced data collection mechanism.

In [31], the problem of maximizing spatio-temporal coverage in vehicular mobile crowdsensing with uncertain but predictable vehicle (i.e., worker) trajectories is investigated. The authors first prove that the problem is NP-hard when there is a budget constraint, and then propose a greedy approximation algorithm and a genetic

algorithm. In [30], the authors also assume predictable worker trajectories. However, they focus on the task assignment problem in a mobile social network where task assignments and delivery of sensed data are realized in an online manner when task requesters and workers encounter with each other. They aim to minimize the task completion times, and propose different approximation algorithms to optimize both worst-case and average-case performance. For predictions of worker trajectories, [31] uses spatio-temporal trajectory matrices, while [30] assumes that user inter-meeting times follow an exponential distribution, which is used widely in mobile social networks [50, 51, 52] literature.

There are a few recent studies [37, 38] that look at the task assignment problem in a hybrid system model to integrate the advantages of participatory and opportunistic MCS. In [37], 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 [38], 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. This enables the platform to find a matching with a high task coverage.

2.2 Matching under Preferences

Stable Matching (SM) problem is introduced in the seminal paper of Gale and Shapley [42] and can simply be defined as the problem of finding a matching between

two groups of objects such that no pair of objects favor each other over their partners in the matching. Gale and Shapley have also introduced what is called the deferred acceptance procedure that finds stable matchings in both one-to-one matching scenarios (i.e., stable marriages) and many-to-one matching scenarios with capacity constraints (i.e., stable college admissions) in $\mathcal{O}(mn)$ time, where m and n are the size of the sets being matched (e.g., men/women, colleges/students, workers/tasks). Since its introduction in [42], the concept of stability has been utilized in different problems including hospital resident assignment [53], resource allocation in device-to-device communications [54], SDN controller assignment in data center networks [55], and supplier and demander matching in electric vehicle charging [56].

Since there can be multiple stable matchings in a matching instance, finding the best SM in terms of another performance metric has received a lot of attention. First, [57] proves that the set of matched nodes is identical in all stable matchings, therefore all stable matchings are of the same size. [58] studies the problem of finding sex-equal stable matchings where the difference between the quality of the matching for two sides (e.g., men/women) of the matching is minimum. The authors prove that the problem is NP-hard and propose a polynomial time approximation algorithm. [59] focuses on the maximum weighted stable matching problem, which turns out to be solvable in $\mathcal{O}(N^4 \log N)$ time ($N = \max\{m, n\}$) and remains as one of the few significant optimal stable matching problems that are solvable in polynomial time. Note that an explicit method to solve these problems and all optimal stable matching problems is simply to enumerate all stable matchings and find the best according to the utility metric considered. In fact, [60] proposes an algorithm that iterates all stable matchings in a matching instance of size N in $\mathcal{O}(N^2 + N|S|)$ time, where S is the set of all stable matchings in the instance. However, since the number of stable matchings ($|S|$) can be massive even for small problem sizes (e.g., the maximum

number of stable matching for a matching instance of size $N = 32$ is larger than 10^{11}) and grows exponentially with increasing problem size [61], this method (i.e., enumerating stable matchings to find the optimal one) would be a feasible solution only in a very limited set of scenarios.

In a given matching instance with preferences, the number of the matched users in a stable matching may be significantly less than that in a maximum size matching if the preferences of some users are incomplete (i.e., they prefer being unmatched to being matched with some users on the other side). When this is the case, it may be desirable to find a maximum size matching that satisfies users based on their preferences as much as possible. This issue is firstly addressed in [62] from a theoretical perspective, and it has been shown that given a matching instance with incomplete preference lists, the problem of finding a matching of maximum size with as few blocking/unhappy pairs as possible is NP-hard and is not approximable within a constant factor. In other words, there cannot exist a polynomial time c-approximation algorithm that would produce matchings with at most $c \times \beta$ unhappy pairs unless P=NP, where c is a constant (≥ 1) and β is the number of blocking pairs in the optimal matching. In the SM literature, the idea of relaxing the stability in order to achieve a better matching in terms of another utility has also been studied under the concept of *almost stable matchings*, but these studies [63, 64] have mostly focused on reducing the running time by sacrificing from the stability, for which they propose truncated versions of the deferred acceptance procedure.

Some matching problems allow or require nodes in one or both sides to be matched with multiple nodes (i.e., many-to-one and many-to-many matching problems). A few studies investigate the issue of stability in such matching problems. For instance, [65] and [66] study the many-to-one stable matching of students-colleges and doctors-hospitals, respectively. In [65], all colleges define a utility and a wage

value for students, and aim to hire the best set of students (i.e., with the highest total utility) within their budget constraints. Each student also forms a preference list over colleges. The authors prove that there may not exist a stable matching in this setting and even checking the existence is NP-hard. However, they provide a polynomial time algorithm that finds pairwise stable matchings in the so called typed weighted model where students are categorized into groups (e.g., Master's and PhD students) and colleges are restricted to define a set of possible wages for each group (i.e., they cannot define a particular wage for each student). [66] studies the same problem and proposes two different fully polynomial-time approximation algorithms with some performance guarantee in terms of coalitional stability for general and proportional (i.e., the wage of doctors are proportional to their utility for hospitals) settings. However, the study does not provide an experimental analysis of the algorithms or discuss their actual/expected performance in these settings. Moreover, the proposed solutions can only be applied to a limited set of scenarios.

There are some studies that look at the stable matching problem in settings with incomplete information on user preferences or dynamic user arrivals/departures. [67] and [68] both study the dynamic stable taxi dispatching problem considering passenger and taxi preferences. However, the objective adopted in [68] is to find locally optimal stable assignments for a given time-point, whereas that in [67] is to minimize the number of unhappy taxi-passenger pairs globally. [69] investigates the stable matching problem in the presence of uncertainty in user preferences. [70] looks at the problem of minimizing the number of partner changes that need to be made in a stable matching to maintain stability when preference profiles of some users change in time. Lastly, [71] studies an interesting combination of famous stable marriage and secretary (hiring) problems.

The concept of stability is studied in multi-dimensional matching problems as

well. In [72], the authors introduce the three-dimensional stable matching problem. In this problem, there are three sets of different types, each individual from a set has a preference list over all pairs from the other two sets, and the goal is to form stable/satisfactory families of three, where each individual in a family is a member of a different set. Wu [73] investigates a different version of this problem, where each individual has a one-dimensional preference list over the individuals from the other two sets instead of over all pairs of individuals as in [72]. In [74], the authors extend the stable roommates problem [75] to a three-dimensional setting, where a set of individuals are assigned into groups of three instead of two based on their preferences. Lastly, in [76], the authors study the problem of matching data sources, servers, and users in a stable manner in video streaming services under restricted preference settings.

In a typical MCS system, the objectives of workers and task requesters can be defined as to maximize their profits and to have their tasks completed with the highest quality possible, respectively. Thus, they are likely to have preferences over possible assignments they can get, and the task assignment in MCS can be consequently characterised as a matching problem under preferences. However, apart from the work in this dissertation, there are only a few studies that consider user preferences in mobile crowdsensing (or in mobile crowdsourcing). In [32], the authors study the budget-constrained many-to-many stable task assignment problem, which they prove to be NP-hard, and propose efficient approximation algorithms. In [33], the authors study the same problem, but in a system model with capacity constraints. On the other hand, [34] considers a many-to-one matching setting and introduce additional constraints (e.g., minimum task quality requirements) that are taken into account in the matching process, along with user preferences. Lastly, in [35], the authors consider a budget-constrained MCS system where the quality of a worker is identical for all

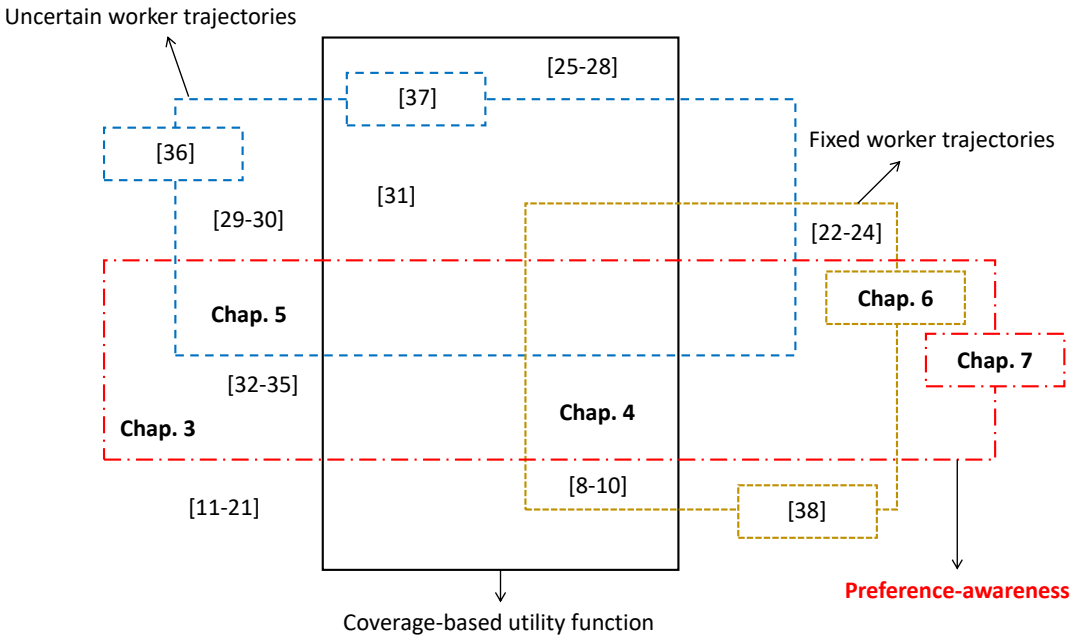


Fig. 1. A summary of related work. If a study considers a quality-based additive utility function, or adopts a system-level objective disregarding user preferences, then it is, respectively, placed outside of the 'Coverage-based utility function' and 'Preference-awareness' boxes. If it assumes a participatory sensing mode with controllable worker trajectories, then it is placed outside of both 'Uncertain worker trajectories' and 'Fixed worker trajectories' boxes, which contain the related work that assume an opportunistic sensing mode. [36], [37], [38], and Chap. 6 are on the borders of these two boxes, as [36] studies the task assignment problem in participatory and opportunistic (with uncertain worker trajectories) MCS separately, [37] considers a hybrid model, where an opportunistic sensing process is followed by a participatory one to reduce costs, and [38] and Chap. 6 assume a semi-opportunistic sensing mode with partly controllable worker trajectories. Lastly, our work in Chapter 7, which considers both overall system utility and user preferences in the task assignment process, is accordingly on the border of the 'Preference-awareness' box.

tasks, and presents an exponential-time algorithm to find weakly-stable many-to-one matchings. A summary of the related work discussed in this chapter is presented in Fig. 1.

CHAPTER 3

QOS-ORIENTED PREFERENCE-AWARE TASK ASSIGNMENT WITH BUDGET CONSTRAINTS

3.1 Introduction

Most of the existing work in the MCS literature aim to maximize a predefined system utility (e.g., quality of service or sensing) in the task assignment process, however users (i.e., task requesters and workers) may value different parameters and hence find an assignment unsatisfying if it is produced by disregarding these parameters that define their preferences. While several studies utilize incentive mechanisms to motivate user participation in different ways, they do not take individual user preferences into account either. Besides, the existing approaches to find preference-aware matchings between two groups of objects (e.g., men/women, students/colleges) do not work in MCS systems that contain task requesters with budget constraints, and that require many-to-one or many-to-many assignments between tasks and workers.

In this chapter, we focus on these issues, and study the problem of finding many-to-one, preference-aware task assignments in MCS systems with task requesters that aim to maximize the quality of sensing (QoS) they get by recruiting high-quality workers within a given budget. We begin by defining the criteria for stability as the existing ones are not applicable to our setting due to its unique requirements (i.e., multiple assignments and budget constraints). We then provide a classification of MCS systems based on their reward scheme and QoS setting, and present three different algorithms that find optimal or near-optimal task assignments in terms of stability in different types of MCS systems.

Our key contributions in this chapter can be summarized as follows:

- We present QoS-based stability conditions under budget constraints in many-to-one MCS systems, and discuss existence and hardness results for stable matchings in different types of MCS systems.

- We propose a polynomial time algorithm to find pairwise stable matchings in MCS systems in which the preference profiles of tasks are identical (i.e., *uniform MCS system*), which is a significant improvement from the exponential time algorithm proposed in [35]. Also, our algorithm does not require the rewards that workers will be paid to be proportional with the corresponding QoS they provide (i.e., *proportional MCS system*), while [35] does.

- Despite the nonexistence results for pairwise stable matchings in general settings, we prove that there always exists a pairwise stable matching in a proportional MCS system by providing a pseudo-polynomial time algorithm that always finds pairwise stable matchings in these systems. Furthermore, this algorithm outperforms all the benchmark algorithms in terms of pairwise stability regardless of the type of the MCS system.

- We propose a heuristic algorithm that also runs in pseudo-polynomial time and produces considerably higher quality assignments in terms of overall stability and user happiness compared to the benchmark algorithms especially in proportional MCS systems.

- In addition to the theoretical analysis of the proposed algorithms, we provide extensive simulation results where we compare the performance of our algorithms with three benchmark algorithms in different types of MCS systems.

3.2 System Model

3.2.1 Assumptions

We assume a system model with a set of workers $\mathcal{W} = \{w_1, w_2, \ldots, w_n\}$ and a set of sensing tasks $\mathcal{T} = \{t_1, t_2, \ldots, t_m\}$. Let $c_t(w)$ denote the cost of performing task t for worker w, which may be calculated by taking into account various factors such as the time and cost required to travel to the task location and perform the task, energy consumption on the worker's device due to sensing, and privacy risks to the worker. Also, let $r_t(w)$ denote the reward that worker w is offered to carry out task t. We assume $c_t(w)$ and $r_t(w)$ to have a financial value so that $r_t(w) - c_t(w)$ is simply the profit worker w would make by performing task t. Additional cost functions can be used to determine the eligibility of the workers for the tasks based on worker resources and task requirements (e.g., to check whether a worker has sufficient residual battery power in his sensing device to carry out a given task).

Since a rational worker will aim to maximize his profit and will not accept to perform the tasks that cost higher than the corresponding rewards he will be paid, we can define the preference list of worker w as

$$P_w = t_{i_1}, t_{i_2}, \ldots, t_{i_k} \tag{3.1}$$

where $P_w \subseteq \mathcal{T}$, $\forall t \in P_w$, $r_t(w) > c_t(w)$, and $\forall t' = t_{i_j}, t'' = t_{i_{j+1}}$, $r_{t'}(w) - c_{t'}(w) > r_{t''}(w) - c_{t''}(w)$. We denote the jth task (t_{i_j}) in P_w by $P_w(j)$ and utilize $t' \succ_w t''$ notation to express that t' precedes t'' in P_w. For each worker w, $\succ_w$ is a strict relation, so even if two tasks provide the same gain to worker w, we assume that he prefers one over another (i.e., no ties are allowed). Note that in our system model, we only need the preference profiles of workers, so a worker can form his preference list himself by estimating his profit from each task using the announced rewards for the

tasks on the platform[2] and then submit only the list to the platform. Alternatively, he can submit an estimated cost value for each task to the platform, which can then form his preference list based on this information.

On the other hand, a rational task requester will try to maximize the total quality of service/sensing (QoS) she gets from the workers she hires within her budget constraint. Let $q_t(w)$ denote the QoS that worker w can provide for task t, which may be based on various factors such as the quality of the sensing device and the average satisfaction of the task requesters that have hired worker w in the past. Also, let b_t denote the budget of task t. Then, we can define the preference list of task t as

$$P_t = S_1, S_2, \ldots, S_k \tag{3.2}$$

where $\forall S \in P_t,\ S \subseteq \mathcal{W}$ and $\sum_{w \in S} r_t(w) \leq b_t$, and $\forall S_i, S_{i+1} \in P_t,\ \sum_{w \in S_i} q_t(w) \geq \sum_{w \in S_{i+1}} q_t(w)$. Note that we allow ties in preference lists of tasks as it is very likely for tasks to have multiple sets with an equal total quality value in their preference list. Thus, a task is said to prefer a set S' of workers to another set S'' of workers only if S' has a greater total quality value than S''. For ease of reading, we let $r_t(S) = \sum_{w \in S} r_t(w)$, $q_t(S) = \sum_{w \in S} q_t(w)$ and use $S' \succ_t S''$ notation to indicate $q_t(S') > q_t(S'')$.

Given a worker-task pair (w, t), we assume that $\nexists S \in P_t : w \in S$ if worker w finds task t unacceptable (i.e., $r_t(w) <= c_t(w)$). Also, in MCS systems where the rewards are determined by the server instead of the task requesters, we might have $r_t(w) > b_t$ for a worker-task pair (w, t). In this case, if $t \in P_w$, we remove task t from P_w as the

[2]Throughout this dissertation, we assume that workers and task requesters have a perfect communication medium through a central server (matching platform), and that the delivery of sensed data from a worker to a task requester is either realized using the central server as an intermediary, or via a direct communication channel between the worker and the task requester, which is established by the central server.

Notation	Description
$\mathcal{W}$, $\mathcal{T}$	Set of workers and tasks, respectively
n, m	Number of workers and tasks, respectively
$\mathcal{M}$	Many-to-one task assignment
$c_t(w)$	Cost of performing task t for worker w
$q_t(w)$	QoS that worker w can provide for task t
$q_t(S)$	Total QoS that $S \subseteq \mathcal{W}$ can provide for task t
$Q_t(S)$	List of QoS that workers in S can provide for task t
$r_t(w)$	Reward offered to worker w by task t
$r_t(S)$	Total reward offered to $S \subseteq \mathcal{W}$ by task t
$R_t(S)$	List of rewards offered to workers in S by task t
b_t	Budget of task t
β	$\max_{t \in \mathcal{T}} b_t$
$b_t^{\mathcal{M}}$	Remaining budget of task t in $\mathcal{M}$
P_u	Preference list of user (worker/task) u
$P_w(x)$	xth task in P_w

Table 1. Notations used in Chapter 3.

budget of task t is insufficient to recruit worker w. Lastly, we assume that for each task t, b_t and $r_t(w)$ values for all $w \in \mathcal{W}$ are either defined as integers or scaled into integers with the smallest scaling factor (different tasks might have different scaling factors).

To make a formal development and evaluation of our matching algorithms, we make the following definitions and observations:

Fig. 2. A uniform and proportional MCS instance with 3 workers $(1, 2, 3)$ and 2 tasks (x, y). $[crq_t(w) = c_t(w), r_t(w), q_t(w)]$

Definition 1 (Feasible matching)**.** *A mapping*

$$\mathcal{M} : (\mathcal{W} \mapsto \mathcal{T} \cup \{\emptyset\}) \cup (\mathcal{T} \mapsto 2^{\mathcal{W}}) \tag{3.3}$$

is a feasible many-to-one matching if it satisfies the following:

- $\forall (w, t) \in \mathcal{W} \times \mathcal{T}$, $\mathcal{M}(w) = t$ *iff* $w \in \mathcal{M}(t)$,
- $\forall w \in \mathcal{W}$, $t \in P_w$ *if* $\mathcal{M}(w) = t$,
- $r_t(\mathcal{M}(t)) \leq b_t$.

Here, given a matching $\mathcal{M}$ and $w \in \mathcal{W}, t \in \mathcal{T}$, the partner[3] of worker w is denoted by $\mathcal{M}(w)$ and the partner set of task t is denoted by $\mathcal{M}(t)$. If $\mathcal{M}(u) = \emptyset$ for user $u \in \mathcal{W} \cup \mathcal{T}$, it means user u is unmatched in $\mathcal{M}$. Note that the last set in the preference list of each task t is $\emptyset$, so we have $S \succ_t \emptyset$, $\forall S \in (P_t \setminus \emptyset)$. Also, even though the preference lists of workers do not include $\emptyset$, since we assume that the workers in

[3]The partner of a worker refers to the task that the worker is assigned to perform, while the partner set of a task refers to the set of all workers assigned with the task.

User	Preference list
x	$\{2,3\},\{1\},\{2\},\{3\},\emptyset$
y	$\{2\},\{3\},\emptyset$
1	x
2	x,y
3	y,x

Table 2. Preference lists of the users in Fig. 2.

our system are rational, we have $t \succ_w \emptyset$, for all $w \in \mathcal{W}$ and $t \in P_w$. We denote the remaining budget of task t in $\mathcal{M}$ by $b_t^{\mathcal{M}} = b_t - r_t(\mathcal{M}(t))$.

Definition 2 (Unhappy pair)**.** *Given a matching $\mathcal{M}$, a worker w and a task t form an unhappy (blocking) pair $\langle w,t\rangle$ if $t \succ_w \mathcal{M}(w)$ and there is a subset $S \subseteq \mathcal{M}(t)$ such that $\{w\} \succ_t S$ and $r_t(w) \leq b_t^{\mathcal{M}} + r_t(S)$.*

Definition 3 (Pairwise stable matching)**.** *A matching $\mathcal{M}$ is said to be pairwise stable if it does not admit any unhappy pairs.*

Definition 4 (Unhappy coalition)**.** *Given a matching $\mathcal{M}$, a subset of workers $S \subseteq \mathcal{W}$ and a task t form an unhappy (blocking) coalition $\langle S,t\rangle$ if $\forall w \in S$, $t \succ_w \mathcal{M}(w)$ and there is a subset $S' \subseteq \mathcal{M}(t)$ such that $S \succ_t S'$ and $r_t(S) \leq b_t^{\mathcal{M}} + r_t(S')$.*

The reason such a coalition $\langle S,t\rangle$ is said to *block* the stability of the matching is that the users in the coalition can communicate with each other and decide to jointly update their partners to have a better assignment.

Definition 5 (Coalitionally stable matching)**.** *A matching $\mathcal{M}$ is said to be coalitionally stable if it does not admit any unhappy coalitions.*

Note that the coalitional stability is a stronger requirement compared with the pairwise stability. In fact, since every unhappy pair $\langle w, t\rangle$ corresponds to an unhappy coalition $\langle\{w\}, t\rangle$, coalitionally stable matchings are also pairwise stable, but not the other way around. For example, on the MCS instance shown in Fig 2 (with preference lists given in Table 2), the matching in which task x is matched with the worker 1 and task y is matched with the worker 2 is a pairwise stable matching, yet it has an unhappy coalition ($\langle\{2, 3\}, x\rangle$), and hence is not coalitionally stable. This is because worker set $\{2, 3\}$ provides a higher QoS (i.e., 7) to x than what her current assignment, worker 1, provides (i.e., 5) and both worker 2 and worker 3 prefer task x to their currently assigned tasks as their net income (i.e., reward - cost) with task x are larger.

Definition 6 (Coalitionally unhappy pair)**.** *Given a matching $\mathcal{M}$, a worker w and a task t form a coalitionally unhappy pair if there is an unhappy coalition $\langle S, t\rangle$ such that $w \in S$.*

When the objective is to achieve pairwise stability, the number of unhappy pairs can be used as the degree of instability of the resulting matching. On the other hand, it is not feasible to use the number of unhappy coalitions to measure the coalitional stability for two reasons. First, since the number of unhappy coalitions in a matching can be as large as $m(2^n - 1)$, even just enumerating them would take exponential time. Second, given a worker-task pair (w, t), knowing all the unhappy coalitions $\langle S, t\rangle : w \in S$ does not provide any useful information to either party, whereas knowing that there is at least one makes them aware that there is a matching in which they are matched to each other and are both better off. For these reasons, we propose to use the number of coalitionally unhappy pairs to measure the coalitional instability of a matching. Note that we can check whether a certain worker-task pair (w, t) is a

coalitionally unhappy pair as described in Algorithm 1. This algorithm uses a sub-procedure named *solve01Knapsack*(c, W, V) in line 9, which denotes the dynamic-programming based algorithm [77] to find the optimal solution for an instance of 0-1 knapsack problem with a knapsack capacity of c, and k items ($k = |W| = |V|$) whose weights and values are given in order in W and V, respectively. It returns the item set that has the largest total value among the sets that have a total weight less than c. Since solving the 0-1 knapsack problem is the most costly operation in Algorithm 1 and has a time complexity of $O(nb_t)$, we can find and count the unhappy and coalitionally unhappy pairs in a matching in pseudo-polynomial time $O(mn^2\beta)$, where $\beta = \max_{t \in \mathcal{T}} b_t$. Lastly, it should be noted that every unhappy pair is also a coalitionally unhappy pair.

We classify MCS systems according to the variability in the QoS provided by the workers for different tasks (uniform/non-uniform), and the relationship between the QoS provided by the workers and the rewards they are offered (proportional/non-proportional). Note that these classifications are exclusive; thus, it is possible to have four different MCS systems, namely, (i) proportional and non-uniform, (ii) non-proportional and non-uniform, (iii) proportional and uniform, and (iv) non-proportional and uniform.

Definition 7 (Uniform MCS system)**.** *An MCS system is called uniform if the QoS provided by each worker is the same for all tasks.*

That is, for all $(w, t, t') \in \mathcal{W} \times \mathcal{T}^2$, $q_t(w) = q_{t'}(w)$. This indicates that all tasks have the same preference ordering for all $S, S' \subseteq \mathcal{W}$ since we have $q_t(S) = q_{t'}(S)$ and $q_t(S') = q_{t'}(S')$, $\forall t, t' \in \mathcal{T}$. However, they may not have the same preference list because of the difference in their budgets (i.e., a task will not include a certain worker set in her preference list if the total reward to be paid to that worker set exceeds her

Algorithm 1: Check Pair $(w, t, \mathcal{M}, \mathit{CheckIf})$

Input: (w, t): Worker, task pair to check

$\mathcal{M}$: The current many-to-one matching

$\mathit{CheckIf}$ = CoalitionallyUnhappyPair or UnhappyPair

```
1  if M(w) = t or M(w) ≻_w t then
2      return false
3  S ← M(t)
4  if CheckIf = CoalitionallyUnhappyPair then
5      foreach w' ∈ W do
6          if t ∈ P_w and t ≻_{w'} M(w') then
7              S ← S ∪ {w'}
8      S ← S \ {w}
9  S_max ← solve01Knapsack(b_t − r_t(w), R_t(S), Q_t(S))
10 if q_t(S_max) + q_t(w) > q_t(M(t)) then
11     return true
12 else
13     return false
```

budget) and being unacceptable to different workers. Despite their simplicity, uniform MCS systems are actually quite common. For example, all MCS systems in which the QoS of workers are determined solely based on trustworthiness or seniority scores of workers (e.g., Waze [78] in which users are ranked according to what is called Waze points that they collect by performing different tasks such as editing the map), or that only contain very basic tasks (e.g., taking a picture of a scene, measuring noise pollution) that do not demand any expertise and can be performed as effectively by

all workers can be viewed as uniform MCS systems. An MCS system that is not uniform is called *a non-uniform MCS system.*

Definition 8 (Proportional MCS system)**.** *An MCS system is called proportional if, for each task, the rewards that are offered to the workers are proportional to the QoS they provide.*

That is, $\frac{r_t(w)}{q_t(w)} = \theta_t$ for all $(w, t) \in \mathcal{W} \times \mathcal{T}$, where θ_t is a constant defined by task t. Thus, different tasks might have a different reward per QoS ratio. Note that in proportional MCS systems, the objective of tasks can be expressed as maximizing the total reward paid to workers within the budget constraints as it also maximizes the total QoS they get. Hence, we will use $r_t(w)$ in place of $q_t(w)$ in the relevant sections. Also, if an MCS system is not proportional, we simply call it *a non-proportional MCS system.*

3.2.1.1 Existence of Stable Matchings

In the following theorems, we give the existence results for pairwise and coalitionally stable matchings in different types of MCS systems.

Theorem 1. *There exists a non-proportional MCS instance, in which none of the feasible matchings is pairwise stable.*

Proof. We prove by giving a counterexample. Let $q_x(3) = 6$ in the instance given in Fig. 2. This adjustment results in $\frac{r_x(2)}{q_x(2)} \neq \frac{r_x(3)}{q_x(3)}$, hence makes the instance non-proportional. It also changes the preference list of task x, which becomes $P_x = \{2, 3\}, \{3\}, \{1\}, \{2\}, \emptyset$. The preference list of the other users remain the same as given in Table 2. Let us analyze all possible task assignments in this modified instance.

- Assume $\mathcal{M}(y) \neq \{3\}$. Then, for $(3, y)$ to not be an unhappy pair, we must have

$\mathcal{M}(y) = \{2\}$. In this case, if $\mathcal{M}(x) \neq \{3\}$, then $(3, x)$ is an unhappy pair. If $\mathcal{M}(x) = \{3\}$, then $(2, x)$ is an unhappy pair.

- Assume $\mathcal{M}(y) = \{3\}$. If $\mathcal{M}(x) \neq \{1\}$, then $(1, x)$ is an unhappy pair, and if $\mathcal{M}(x) = \{1\}$, then $(2, y)$ is an unhappy pair.

Therefore, we conclude that no pairwise stable matching exists in the given non-proportional MCS instance. □

Since every unhappy pair is also an unhappy coalition, the following corollary is an immediate result of Theorem 1.

Corollary 1.1. *There exists a non-proportional MCS instance, in which none of the feasible matchings is coalitionally stable.*

Theorem 2. *There exists a uniform and/or proportional MCS instance, in which none of the feasible matchings is coalitionally stable.*

Proof. We prove by giving a counterexample. Note that the MCS instance given in Fig. 2 is both uniform and proportional. Then, it suffices to show that all feasible matchings that can be defined on this instance have at least one unhappy coalition.

- Assume $\mathcal{M}(x) \neq \{1\}$. Then, the worker set $\{1\}$ and task x do not form an unhappy coalition only if $\mathcal{M}(x) = \{2, 3\}$. However, if $\mathcal{M}(x) = \{2, 3\}$, then the worker set $\{3\}$ and task y form an unhappy coalition.
- Assume $\mathcal{M}(x) = \{1\}$. If $\mathcal{M}(y) \neq \{2\}$, then $(\{2\}, y)$ is an unhappy coalition. If $\mathcal{M}(y) = \{2\}$, then $(\{2, 3\}, x)$ is an unhappy coalition.

□

We will show in the next section that there always exists a pairwise stable matching in uniform and proportional MCS systems.

3.2.1.2 Hardness of Finding Stable Matchings

Consider an MCS instance with n workers $(w_1, w_2, \ldots, w_n)$ and a single task (t). Note that, by definition, finding a coalitionally stable matching in this instance is exactly the same problem with finding an optimal solution for a 0-1 knapsack instance with a knapsack that has a weight capacity of b_t and n items such that the weight and value of ith item are, respectively, the reward $(r_t(w_i))$ and the QoS $(q_t(w_i))$ of ith worker (w_i) for task t. Since the 0-1 knapsack problem is NP-hard, we can conclude that the problem of finding a coalitionally stable matching (even for an MCS system that has only one task) is NP-hard as well.

In fact, as proved in [65], given a many-to-one matching instance with budget constraints, both checking the existence of a coalitionally stable matching and finding one if exists are NP-hard. Also, since even checking whether a particular worker-task pair form an unhappy pair in a given matching is NP-hard (as it requires to solve the corresponding 0-1 knapsack problem shown in Algorithm 1), it is highly likely that the same hardness results apply to the pairwise stable matchings as well.

3.2.2 Problem Formulation

Given the definitions as well as the nonexistence and hardness results provided above, we can formally define our objective function as

$$\text{minimize} \sum_i \sum_j u_{ij} \tag{3.4}$$

such that

$$\sum_j x_{ij} \leq 1 \qquad \forall i \tag{3.5}$$

$$\sum_i x_{ij} \times r_{t_j}(w_i) \leq b_{t_j} \qquad \forall j \tag{3.6}$$

$$x_{ij} \leq e_{ij} \qquad \forall i, j \tag{3.7}$$

where

$$u_{ij} = \begin{cases} 1, & \text{if } \langle w_i, t_j \rangle \text{ is a (coalitionally) unhappy pair} \\ 0, & \text{otherwise} \end{cases} \tag{3.8}$$

$$x_{ij} = \begin{cases} 1, & \text{if } w_i \text{ is assigned to } t_j \\ 0, & \text{otherwise} \end{cases} \tag{3.9}$$

$$e_{ij} = \begin{cases} 1, & \text{if } t_j \in P_{w_i} \ (\textit{eligibility}) \\ 0, & \text{otherwise.} \end{cases} \tag{3.10}$$

That is, we would like to produce feasible matchings with as few (coalitionally) unhappy pairs as possible. When the goal is to minimize the number of *coalitionally* unhappy pairs (i.e., $u_{ij} = 1$ for coalitionally unhappy pairs), the optimization objective in (3.4) attains the strongest stability conditions, but becomes intractable in all types of MCS systems. On the other hand, minimizing the number of *unhappy pairs* is a more practical objective and generally adequate to virtually satisfy the users for two reasons. First, it is much harder for a pair of users to find out that they are a coalitionally unhappy pair than that they are simply an unhappy pair, since the former requires them to know the preferences and the current partners of all workers in the platform. Second, unlike unhappy pairs, modifying the matching to make a coalitionally unhappy pair happy necessitates that all the workers in the corresponding unhappy coalition (see Definition 6) cooperate and break up with their current partners simultaneously, which might be hard to attain.

It is also desirable to minimize the degree of user unhappiness in general rather than the number of unhappy users. In this case, an alternative objective would be

to minimize the highest dissatisfaction ratio in the matching from the perspective of tasks, since they, unlike workers who are simply either happy or not with their assignments, have a degree of unhappiness based on the total QoS service they receive (i.e., *non-binary utility*) in the many-to-one matching scenario described earlier. Formally, let

$$\mathcal{S}_t = \{S : \langle S, t\rangle \text{ is an unhappy coalition}\}, \tag{3.11}$$

and $\forall S \in \mathcal{S}_t$, let $S^R \subseteq \mathcal{M}(t)$ be the set with the lowest total quality such that its removal from the partner set of task t suffices to accept S (i.e., $r_t(S) \leq b_t^{\mathcal{M}} + r_t(S^R)$). Then, the dissatisfaction ratio of task t can be computed by:

$$\delta_t = \begin{cases} 1, & \text{if } \mathcal{S}_t = \emptyset \\ \infty, & \text{if } \mathcal{S}_t \neq \emptyset \text{ and } \mathcal{M}(t) = \emptyset \\ \max\limits_{S \in \mathcal{S}_t} \frac{q_t(S) + q_t(\mathcal{M}(t) \setminus S^R)}{q_t(\mathcal{M}(t))}, & \text{otherwise.} \end{cases} \tag{3.12}$$

Thus, for a task t the optimal (minimum) value of δ_t is 1. Finally, the objective function can formally be defined as:

$$\text{minimize} \quad \max_{t \in \mathcal{T}} \delta_t \tag{3.13}$$

We will address this version of the problem using the following definition.

Definition 9 (Coalitionally α-stable matching)**.** *A matching $\mathcal{M}$ is said to be coalitionally α-stable if $\forall t \in \mathcal{T}$*

$$\delta_t \leq \alpha. \tag{3.14}$$

3.3 Proposed Solution

In this section, we provide the details of the proposed task assignment algorithms.

Algorithm 2: Uniform Task Assignment ($\mathcal{W}$, $\mathcal{T}$, $P_{\mathcal{T}}$)

Input: $\mathcal{W}$: The set of workers

$\mathcal{T}$: The set of tasks

$P_{\mathcal{T}}$: The common preference profile of tasks

```
for i ← 1 to n do
    w ← ith worker in P_T
    for j ← 1 to |P_w| do
        t ← P_w(j)
        if b_t^M ≥ r_t(w) then
            M(w) ← t
            M(t) ← M(t) ∪ {w}
            b_t^M ← b_t^M − r_t(w)
            break
return M
```

3.3.1 Uniform Task Assignment (UTA) Algorithm

The stable task assignment problem in uniform MCS systems has recently been investigated in [35], and an ILP-based algorithm with a $O(nm2^n)$ time complexity was proposed to find pairwise stable matchings in MCS systems that are both uniform and proportional. Here, we propose *UTA algorithm* that finds pairwise stable matchings in all uniform MCS systems (i.e., proportional/non-proportional) in only $O(n \log n+nm)$ time.

A pseudo-code description of UTA algorithm is given in Algorithm 2. First, apart from the set of workers and tasks, UTA algorithm takes the common preference profile of tasks ($P_{\mathcal{T}}$) as input, which is simply a sorted version of the worker set $\mathcal{W}$, in

which workers with higher QoS values precede the others. Formally, it can be defined as

$$P_{\mathcal{T}} = w_{i_1}, w_{i_2}, \ldots, w_{i_n} \tag{3.15}$$

where $\forall w' = w_{i_j}, w'' = w_{i_{j+1}}$, $q_t(w') \geq q_t(w'')$, $\forall t \in \mathcal{T}$. Since we assume that the system is uniform, hence the QoS value of a worker is same for all tasks, it is in fact possible to create such a list. Then, the algorithm begins to seek the best available assignment for the workers (w) in order of their appearance in $P_{\mathcal{T}}$ (i.e., in decreasing order of their QoS). To this end, it iterates through the tasks in their preference lists (P_w) in order to find the first task in their preference lists (i.e., the most preferred) that has sufficient amount of remaining budget to hire them. If it finds such a task t for worker w, it updates the matching and the remaining budget of task t, otherwise it leaves worker w unassigned and continues the assignment process with the next worker in $P_{\mathcal{T}}$. We now show that the resulting matching will always be pairwise stable.

Theorem 3. *In uniform MCS systems, UTA algorithm always produces a pairwise stable matching.*

Proof. We will prove it by contradiction. Assume that the matching $\mathcal{M}$ returned by UTA algorithm contains at least one unhappy pair, say $\langle w, t \rangle$. Since worker w and task t form an unhappy pair, we know that they are not matched to each other and worker w prefers task t to his current partner in $\mathcal{M}$. This means when the algorithm was iterating the preference list of worker w in line 3, it has attempted to match him with task t, but could not do it due to the limited budget of task t, so that it either matched worker w with a task that come after t in P_w or left him unmatched. Let A be the partner set of task t and ρ be her remaining budget when the algorithm tried,

but failed to assign worker w to her. Then, (i) $\rho < r_t(w)$. Since the algorithm matches the workers in order of their appearance in the common preference list of the tasks, we have $\{\overline{w}\} \succ_t \{w\}$, $\forall \overline{w} \in A$. Also, note that once the algorithm matches a worker and a task, it never unmatches them again. Thus, we have $A \subseteq \mathcal{M}(t)$. However, for (w, t) to be an unhappy pair, there should exist a subset $S' \subseteq \mathcal{M}(t)$ such that (ii) $r_t(w) \leq r_t(S') + b_t^{\mathcal{M}}$ and (iii) $\{w\} \succ_t S'$. From (iii), we have $\{w\} \succ_t \{\overline{w}\}$, $\forall \overline{w} \in S'$, which means all workers in S' got matched with task t after the algorithm failed to match worker w and task t, hence we have

$$\rho \geq r_t(S') + b_t^{\mathcal{M}} \tag{3.16}$$

$$\rho \geq r_t(w) \qquad \text{(by (ii))} \tag{3.17}$$

$$\rho > \rho \qquad \text{(by (i))} \tag{3.18}$$

which is a contradiction. □

The following corollary is a direct result of Theorem 3.

Corollary 3.1. *In uniform MCS systems, there always exists a pairwise stable matching.*

Note that even if an MCS system is not uniform (i.e., $\exists (w, t, t') \in \mathcal{W} \times \mathcal{T}^2$, $q_t(w) \neq q_{t'}(w)$), UTA algorithm can still produce pairwise stable matchings if it is possible to create a common preference list ($P_{\mathcal{T}}$) for tasks. In other words, if $\nexists (w, w', t, t') \in \mathcal{W}^2 \times \mathcal{T}^2$, $q_t(w) > q_t(w')$, $q_{t'}(w) < q_{t'}(w')$, UTA algorithm can still be used to find pairwise stable matchings.

Example. The instance given in Fig. 2 is a uniform MCS system, so UTA algorithm can be used to find a pairwise stable matching in this instance as follows. First, we create the common preference list of tasks as $P_{\mathcal{T}} = 1, 2, 3$ since $q_{x,y}(1) > q_{x,y}(2) > q_{x,y}(3)$. The first worker in $P_{\mathcal{T}}$ is worker 1, so the algorithm starts

the matching process with him. Since the first and only task in his preference list, task x, has enough budget to hire him (the preference lists of users are given in Table 2), it assigns worker 1 to task x, and updates the remaining budget of task x as $7 - 5 = 2$. The next worker in $P_{\mathcal{T}}$ is worker 2, who also prefers task x to task y, but task x does not have enough budget to hire him ($2 < 4$), so the algorithm tries to match him with task y. Task y has enough budget to hire worker 2, so they get matched and the remaining budget of task y becomes $5 - 4 = 1$. Worker 3 is the last worker in $P_{\mathcal{T}}$. However, neither task x nor task y has sufficient remaining budget to hire him, so he will be left unmatched. Thus, the final matching will be ($x \leftrightarrow 1$, $y \leftrightarrow 2$), and it can easily be checked that it does not contain any unhappy pairs and hence it is pairwise stable.

Running time. Forming the common preference profile of tasks ($P_{\mathcal{T}}$) requires to sort the workers according to their QoS values and thus takes $O(n \log n)$. Then, since the first for loop will iterate n times and the second will iterate at most m times (i.e., $|P_w| \leq m, \forall w \in \mathcal{W}$), it is straightforward to see that the worst-case running time of UTA algorithm is $O(n \log n + nm)$.

3.3.2 Pairwise Stable Task Assignment (PSTA) Algorithm

PSTA algorithm is a pseudo-polynomial time algorithm that, unlike UTA algorithm, can be run in any type of MCS system, and aims to produce matchings with as little pairwise instability as possible (which is why it is named as *Pairwise*-STA). In fact, in Theorem 4, we will show that it always produces pairwise stable matchings in proportional MCS systems. Besides, in non-proportional MCS systems where a pairwise stable matching may not exist, it manages to produce matchings with almost optimal pairwise stability as it will be shown in Section 3.4.

The details of PSTA algorithm are given in Algorithm 3. It follows the classic

deferred acceptance mechanism [42] but updates the set of workers assigned to a task optimally from the set of workers currently assigned to the task and the worker under consideration. It keeps a stack of unmatched workers that still have tasks to propose to[4], pops them one by one (line 3) and lets them (worker w) propose to the next task (task t) in their preference list (line 5). If task t has enough remaining budget, the algorithm directly matches them (lines 8-10). Otherwise, it finds the most favorable worker set (S_{max}) for task t among the workers in her current partner set and worker w within her budget constraint (lines 12-13), assigns that worker set as the new partner set of task t (lines 14-17) and pushes the remaining workers back onto the stack after setting them free (lines 18-20). If the partner set of task t has not changed, worker w will be the only worker to be pushed onto the stack, in which case we say that task t rejected the proposal of worker w. This continues until there is no unmatched worker that still has a task that he has not yet proposed to in his preference list. In the following theorem, we prove that the resulting matching is guaranteed to be pairwise stable if the MCS system is proportional.

Theorem 4. *In proportional MCS systems, PSTA algorithm always produces a pairwise stable matching.*

Proof. We will prove it by contradiction. Assume that in a proportional MCS system (without loss of generality, let $q_t(w) = r_t(w)$, $\forall w, t$), PSTA algorithm produces a matching $\mathcal{M}$ that contains at least one unhappy pair, say $\langle w, t \rangle$, which either means that task t rejected worker w's proposal and they never got matched, or that task t accepted worker w's proposal, but then discarded him (possibly along with a set of workers) from her partner set to accept the proposal of another worker with a higher

[4]We use the terminology of the stable marriage problem (i.e., propose, accept, reject) to indicate that this algorithm can also be run in a distributed manner using the procedures proposed in [79] to find stable marriages in a distributed manner.

Algorithm 3: PairwiseStableTaskAssignment($\mathcal{W}$,$\mathcal{T}$)

Input: $\mathcal{W}$: The set of workers

$\mathcal{T}$: The set of tasks

Stack.push($\mathcal{W}$)
while *Stack is not empty* **do**
  $w \leftarrow$ *Stack.pop()*
  if P_w *is not empty* **then**
    $t \leftarrow P_w(1)$; ▷ w `proposes to` t
    $P_w \leftarrow P_w \setminus \{t\}$
    if $b_t^{\mathcal{M}} \geq r_t(w)$ **then**
      $\mathcal{M}(w) \leftarrow t$
      $\mathcal{M}(t) \leftarrow \mathcal{M}(t) \cup \{w\}$
      $b_t^{\mathcal{M}} \leftarrow b_t^{\mathcal{M}} - r_t(w)$
    else
      $S \leftarrow \mathcal{M}(t) \cup w$
      $S_{max} \leftarrow$ *solve01Knapsack*$(b_t, R_t(S), Q_t(S))$
      $\mathcal{M}(t) \leftarrow S_{max}$, $b_t^{\mathcal{M}} \leftarrow b_t - r_t(S_{max})$
      if $w \in S_{max}$ **then**
        $\mathcal{M}(w) \leftarrow t$
      foreach $w' \in S \setminus S_{max}$ **do**
        $\mathcal{M}(w') \leftarrow \emptyset$, *Stack.push($w'$)*
return $\mathcal{M}$

QoS. Let A and ρ denote task t's partner set and remaining budget at the time task t and worker w broke up by one of these two cases (i.e., rejected or discarded),

respectively. Then, we have

$$\nexists S \subseteq A : r_t(S) < r_t(w),\ r_t(w) \leq r_t(S) + \rho \tag{3.19}$$

because if there were such a subset S, the 0-1 knapsack solution would include worker w instead of S, and w would not get rejected/discarded.

We define *a node* as a tuple (x, y), where x and y are the label and length of the node, respectively. Let $A_0 = \{(w', r_t(w')) : w' \in A\}$, so each node in A_0 corresponds to a worker in A. Also, let $l(S)$ be the total length of the nodes in S. Then, from (3.19), we have

$$\nexists S \subseteq A_0 : l(S) < r_t(w),\ r_t(w) - l(S) \leq \rho \tag{3.20}$$

Note that task t will discard a group G of workers from her partner set only when she is proposed by a worker w' who has a higher total reward than G and will not violate her budget constraint when replaced by G. After the break up of worker w and task t, whenever such a change (say ith change) occurs in the partner set of task t, we create A_i from A_{i-1} as follows:

1. $A_i \leftarrow A_{i-1}$.
2. Let K be the set of nodes in A_i that have the same label with any worker in G.
3. Change the labels of all nodes in K as w'.
4. Create a new node $(w', r_t(w') - r_t(G))$ (note that $r_t(G) = l(K)$) and add it to A_i.

In other words, we add the new worker to A_i by dividing it into several nodes so that the node lengths in A_{i-1} are preserved. An example is provided in Fig. 3 to illustrate this process.

Let c be the number of changes occurred in the partner set of task t since her

break up with worker w until the end. Then, the last node set created, A_c, can be partitioned into two sets A'_0 and B, where A'_0 contains $|A_0|$ nodes that have exactly the same lengths with the nodes in A_0, but possibly have different labels, and B is the set of newly created nodes such that (i) $l(B) + b_t^{\mathcal{M}} = \rho$. Since we assumed that (w, t) form an unhappy pair in the final matching $\mathcal{M}$, there should exist a subset $S' \subseteq \mathcal{M}(t)$ such that $r_t(w) > r_t(S')$ and $r_t(w) \leq r_t(S') + b_t^{\mathcal{M}}$. As A_c has a distinct set P of nodes that jointly correspond to each worker w' in $\mathcal{M}(t)$ (i.e., $l(P) = r_t(w')$), there should also exist a subset $\overline{S} \subseteq A'_0 \cup B$ such that (ii) $r_t(w) > l(\overline{S})$ and (iii) $r_t(w) \leq l(\overline{S}) + b_t^{\mathcal{M}}$. Then, we have

$$r_t(w) - l(\overline{S} \cap A'_0) \leq l(\overline{S} \cap B) + b_t^{\mathcal{M}} \qquad \text{(by (iii))} \tag{3.21}$$

$$\rho < l(\overline{S} \cap B) + b_t^{\mathcal{M}} \qquad \text{(by (3.20) and (ii))} \tag{3.22}$$

$$\rho < \rho \qquad \text{(by (i))} \tag{3.23}$$

which is a contradiction. □

A significant result of Theorem 4 is the following corollary.

Corollary 4.1. *In proportional MCS systems, there always exists a pairwise stable matching.*

In the following theorem, we show that pairwise stability ensures a certain degree of coalitional stability in proportional MCS systems.

Theorem 5. *In proportional MCS systems, a pairwise stable matching is coalitionally 2-stable.*

Proof. We prove by contradiction. Let $\mathcal{M}$ be a pairwise stable matching in a proportional MCS system and $\langle S, t \rangle$ be an unhappy coalition in $\mathcal{M}$ such that

$$r_t(S \cup (\mathcal{M}(t) \setminus S')) > 2r_t(\mathcal{M}(t)) \tag{3.24}$$

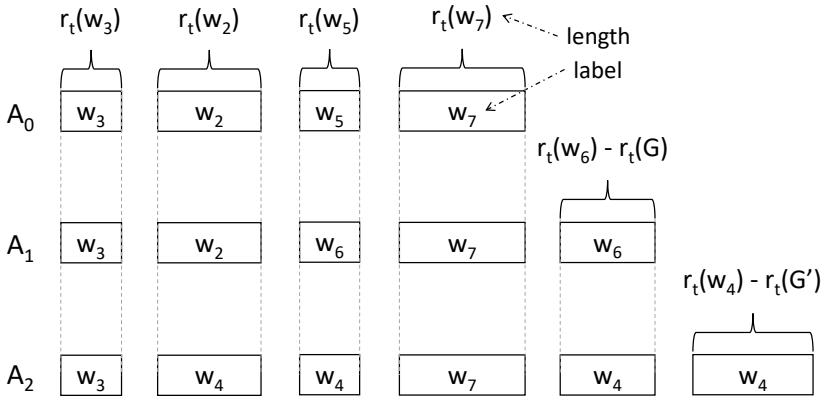


Fig. 3. An example illustrating the process used in the proof of Theorem 4. A_0 is created right after task t and worker w broke up, so the partner set of task t is $\{w_3, w_2, w_5, w_7\}$ at that time. A_1 is created after the first change in the partner set of task t, which is the substitution of $G = \{w_5\}$ with w_6, and A_2 is created after the second change in the partner set of task t, which is the substitution of $G' = \{w_2, w_6\}$ with w_4. Note that the length of the nodes in A_0 are always preserved throughout the process.

where $S' \subseteq \mathcal{M}(t)$ satisfies $r_t(S) > r_t(S')$ and (i) $r_t(S) \leq b_t^{\mathcal{M}} + r_t(S')$. Thus, $\langle S, t\rangle$ breaks the coalitional 2-stability of $\mathcal{M}$ according to (3.14). First, note that if (ii) $r_t(\mathcal{M}(t)) \geq b_t/2$, we would have

$$r_t(S) + r_t(\mathcal{M}(t)) - r_t(S') > 2r_t(\mathcal{M}(t)) \qquad \text{(by (3.24))} \tag{3.25}$$

$$r_t(\mathcal{M}(t)) + b_t^{\mathcal{M}} > 2r_t(\mathcal{M}(t)) \qquad \text{(by (i))} \tag{3.26}$$

$$r_t(\mathcal{M}(t)) + b_t^{\mathcal{M}} > b_t \qquad \text{(by (ii))} \tag{3.27}$$

$$b_t > b_t \tag{3.28}$$

which is false. So, we have $r_t(\mathcal{M}(t)) < b_t/2$. Let w be any worker in S. If $r_t(w) \leq r_t(\mathcal{M}(t))$, then $\mathcal{M}$ is not pairwise stable because task t can add worker w to her partner list without removing anyone as $r_t(w) + r_t(\mathcal{M}(t)) < b_t$. Thus, we have $r_t(w) > r_t(\mathcal{M}(t))$, which also implies that $\mathcal{M}$ is not a pairwise stable matching as

task t can simply replace $\mathcal{M}(t)$ with worker w to obtain a better partner set within her budget constraint (i.e., $r_t(w) \leq r_t(S) \leq b_t^{\mathcal{M}} + r_t(S') \leq b_t$). □

From Theorem 3, Theorem 4 and Theorem 5, we obtain the following corollaries.

Corollary 5.1. *In MCS systems that are both proportional and uniform, UTA algorithm always returns coalitionally 2-stable matchings.*

Corollary 5.2. *In proportional MCS systems, PSTA algorithm always returns coalitionally 2-stable matchings.*

Example. We run PSTA algorithm on the same MCS instance illustrated in Fig. 2. To make things slightly different, we assume that the workers are pushed onto the stack in line 1 in increasing order of their identifiers so that the first worker that is popped in line 3 is worker 3. As shown in Table 2, the first task in the preference list (P_3) of worker 3 is task y, so he first proposes to her (task y gets removed from P_3). Task y has enough budget, so worker 3 and task y get matched to each other and the remaining budget of task y becomes 2 (line 8-10). The next worker popped from the stack is worker 2, whose first preference is task x. Thus, worker 2 proposes to task x (task x gets removed from P_2). Since task x also has enough budget, she gets matched with worker 2, which reduces her remaining budget to 3. Next, worker 1 gets popped and proposes to the only task in his preference list: task x (task x gets removed from P_1, so $P_1 = \emptyset$). Task x does not have enough remaining budget to hire worker 1, so the algorithm finds the best set of workers among the workers in $\mathcal{M}(x) \cup \{1\} = \{1, 2\}$ that does not exceed the budget limit of task x by solving the corresponding 0-1 knapsack problem (line 13). The subset $\{1\}$ provides the highest QoS without violating the budget constraints, so worker 1 and task x will get matched to each other, and worker 2 will be set free and pushed onto the stack. In the next

step, worker 2 will be popped and propose to task y (task y gets removed from P_2, so $P_2 = \emptyset$). Task y does not have sufficient remaining budget, but the algorithm, after solving the knapsack problem, will replace her current partner, worker 3, with worker 2 as this will increase the total QoS task y gets. Consequently, it will set worker 3 free and push him onto the stack. Then, it will pop him from the stack and let him propose to task x (task x gets removed from P_3, so $P_3 = \emptyset$). The budget of task x is not adequate to hire worker 3, and replacing his current partner is also not beneficial, hence worker 3 will be pushed onto the stack, again. When popped next time, since there does not remain any other task for worker 3 to propose to in his preference list, he will not be pushed onto the stack again. This will leave the stack empty, so the matching ($x \leftrightarrow 1$, $y \leftrightarrow 2$) will be returned by the algorithm, which is the same pairwise stable matching found by UTA algorithm.

Running time. Note that since the preference list of a worker (w) shrinks in size by 1 every time he is popped from the stack (line 6) until his preference list becomes empty (after which he will not be pushed onto the stack ever again), he can be pushed onto the stack at most $O(m)$ times as $|P_w| \leq m$. Thus, the while loop in line 2 will iterate at most $O(nm)$ times. Since the most costly operation in each iteration is solving the 0-1 knapsack problem, which takes $O(n\beta)$ where $\beta = \max_{t \in \mathcal{T}} b_t$, the time complexity of PSTA algorithm is $O(n^2 m \beta)$.

3.3.3 Heuristic Algorithm

Heuristic algorithm is a task-oriented, pseudo-polynomial time algorithm that can also be run in any type of MCS system and is designed in a way that the tasks in the system take turns at modifying the matching according to their preferences. That is, each task t, in her turn, changes her partner set to the best feasible set of workers among all the workers that are already in her partner set, or that prefer

herself to their current assignments. Thus, Heuristic algorithm ensures that there is no unhappy coalition $\langle S, t \rangle$ for any $S \subseteq \mathcal{W}$ immediately after the turns of task t. It can, hence, be expected that as we increase the number of iterations/turns, there will be fewer unhappy coalitions, which will improve the coalitional stability of the matching.

The outline of Heuristic algorithm is provided in Algorithm 4. In each of the k iterations, the algorithm goes through all tasks (t) in the system (line 2) and first finds all workers that are either already matched with task t or would be better off with task t compared to their current partners (lines 3-6). Then, among these workers, it identifies the set S_{max} of workers that require a total reward of less than b_t and provide as large total QoS as possible for task t by solving the corresponding knapsack problem (line 7). Finally, it sets the workers that are presently matched with task t, but are not in S_{max} free (lines 8-9), and matches task t and the workers in S_{max} with each other after removing these workers from the partner sets of the tasks with whom they were previously matched (lines 10-14).

The fact that a task is perfectly happy (i.e., has a dissatisfaction ratio of 1) right after her turns indicates that the task that is considered the latest in the for loop in line 2 will be perfectly happy in the end as well. This nice property of Heuristic algorithm can be used to make a different task requester happy at each (e.g., hourly, daily) assignment cycle, and to ensure that all task requesters become perfectly happy with their assignments periodically. Another useful property of Heuristic algorithm is that it allows to explore different feasible matchings that are shaped by the preference profile of a different task, which will become clearer in the toy example provided below.

Example. We once again utilize the MCS instance given in Fig. 2 to show how Heuristic algorithm functions. Assume that the for loop in line 2 iterates through the tasks in order of task x and task y. In the first iteration, since all workers are

Algorithm 4: Heuristic Approach($\mathcal{W}$, $\mathcal{T}$)

Input: $\mathcal{W}$: The set of workers

$\mathcal{T}$: The set of tasks

k: The number of iterations

```
1  for i ← 1 to k do
2      foreach t ∈ 𝒯 do
3          S ← ℳ(t)
4          foreach w ∈ 𝒲 do
5              if t ∈ P_w and t ≻_w ℳ(w) then
6                  S ← S ∪ {w}
7          S_max ← solve01Knapsack(b_t, R_t(S), Q_t(S))
8          foreach w' ∈ ℳ(t) \ S_max do
9              ℳ(w') ← ∅
10         foreach w' ∈ S_max do
11             let t' denote ℳ(w')
12             ℳ(t') ← ℳ(t') \ w'
13             ℳ(w') ← t
14         ℳ(t) ← S_max, b_t^ℳ ← b_t − r_t(S_max)
15 return ℳ
```

unmatched, task x will be assigned to the best (i.e., with the highest total QoS) subset of workers in $\{1, 2, 3\}$ with a total reward of less than 7, which is $\{2, 3\}$:

$$\mathcal{M} : x \leftrightarrow \{2, 3\}, y \leftrightarrow \emptyset$$

In this matching. worker 3 is the only one that prefers task y to task x, so when it is

task y's turn, she will directly be matched with worker 3:

$$\mathcal{M} : x \leftrightarrow \{2\}, y \leftrightarrow \{3\}$$

In the next iteration, task x will be assigned to the best worker set from $\{1, 2\}$, which are the workers that are either matched with task x (i.e., worker 2) or prefer task x to their current partners (i.e., worker 1). Since her budget does not allow to hire both, worker 2 will be replaced by worker 1 who provides a higher QoS:

$$\mathcal{M} : x \leftrightarrow \{1\}, y \leftrightarrow \{3\}$$

At this point, since worker 2 prefers task y to being unassigned, task y will be assigned to the best feasible worker set from $\{2, 3\}$ in her turn, which is $\{2\}$:

$$\mathcal{M} : x \leftrightarrow \{1\}, y \leftrightarrow \{2\}$$

In this matching, both worker 2 and worker 3 prefers task x to their current partners, so the algorithm will assign task x the best feasible worker set from $\{1, 2, 3\}$, which is $\{2,3\}$:

$$\mathcal{M} : x \leftrightarrow \{2, 3\}, y \leftrightarrow \emptyset$$

Note that this is exactly the same as the first matching we obtained above. In fact, after this point, the algorithm will repeatedly generate the same matchings, and return the matching ($x \leftrightarrow \{2\}$, $y \leftrightarrow \{3\}$) if k is odd, and the matching ($x \leftrightarrow \{1\}$, $y \leftrightarrow \{2\}$), otherwise. The former is the best possible matching in terms of coalitional stability, as there does not exist any coalitionally stable matching in this instance (Theorem 2) and this matching contains only one coalitionally unhappy pair (worker 1 and task x). On the other hand, the latter is the same matching as the one found by UTA and PSTA algorithms and is the optimal matching in terms of pairwise stability.

Running time. The two outermost loops in line 1 and 2 will iterate k and m times, respectively. Similar to PSTA algorithm, solving the 0-1 knapsack instance in line 7 takes $O(n\beta)$ where $\beta = \max_{t \in \mathcal{T}} b_t$, and is the most costly operation within the for loop in line 2. This makes the total running time of Heuristic algorithm $O(knm\beta)$.

3.4 Evaluation

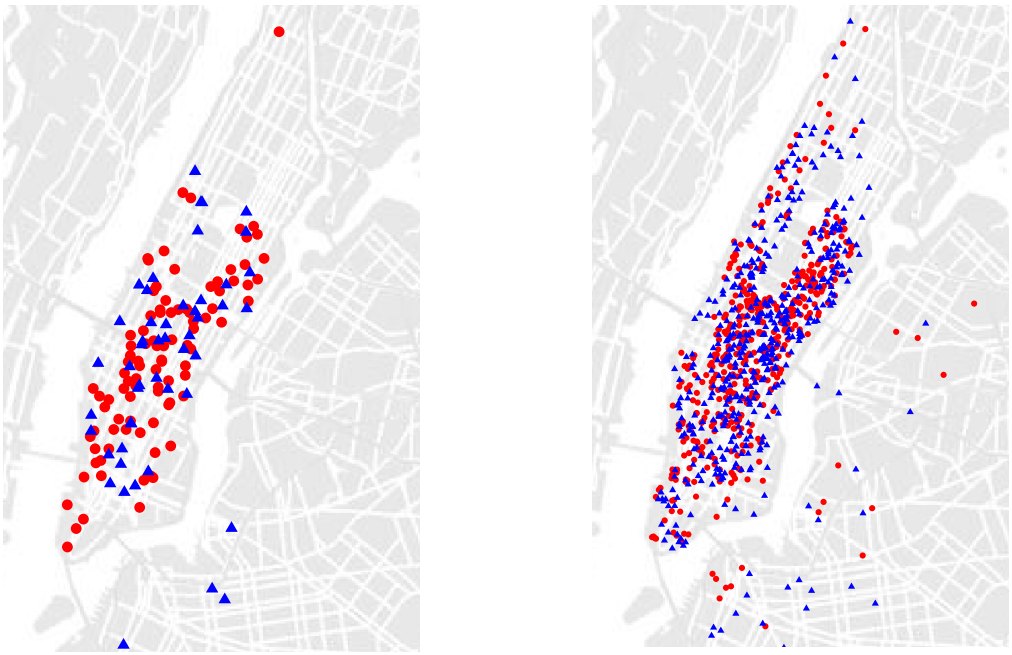
In this section, we evaluate the performance of the proposed algorithms in different types of MCS systems.

3.4.1 Simulation Settings

Similar to previous work [29, 80], we utilize a taxi trip dataset [81] in a city (i.e., New York City (NYC)) to have a realistic geographic distribution of workers and tasks. Specifically, we randomly select a day in 2015 and then create a worker at the most recent drop-off location of each taxi that has become available between 1-2 pm on the selected day, and a task at the pick-up location of each passenger that has demanded a taxi in the next hour of the same day. Then, we use random-sampling to obtain the worker and task sets of certain size based on the experiment requirements.

Note that each of the four types of MCS systems (i.e., proportional (P.) and non-uniform (N.U.), non-proportional (N.P.) and non-uniform, proportional and uniform (U.), non-proportional and uniform) necessitates different QoS and reward settings. Thus, we generate a unique scenario for each MCS system by integrating this information on top of the geographical information.

As the default setup for all scenarios, we sample $n = 100$ workers and $m = 50$ tasks (an instance is illustrated in Fig. 4), and randomly assign a budget for each task between $B_{min} = 100$ and $B_{max} = 1000$. Given the distance d between a worker w and a task t, we let $c_t(w) = d \times C$, where $C = 20$ denotes the cost per kilometer.



(a) (b)

Fig. 4. Distribution of workers (circles) and tasks (triangles) on the NYC map with different sampling ratios: (a) 100 workers, 50 tasks; (b) 500 workers, 500 tasks.

Below, we describe the typical settings for each scenario.

- **Proportional (P.) and uniform (U.)**: We assign a unique QoS value v to each worker w randomly from $[1, 200]$ and let $q_t(w) = v$, $\forall t \in \mathcal{T}$. Then, the rewards are set as

$$r_t(w) = \begin{cases} \theta_t \times q_t(w) & \text{if } \theta_t \times q_t(w) \leq b_t \\ 0 & \text{otherwise} \end{cases} \tag{3.29}$$

  where θ_t is randomly selected from $[1, 5]$ for each task t.

- **Proportional (P.) and non-uniform (N.U.)**: Given a worker-task pair (w, t), we randomly assign the reward $r_t(w)$ from $[1, b_t]$ (we also examine the cases where reward values are assigned from $[1, b_t/2]$ and $[b_t/2, b_t]$), and let

$q_t(w) = r_t(w)/\theta_t$, where θ_t is set as in the previous scenario, yet its value is actually arbitrary for all the algorithms considered in this scenario, unlike the previous scenario where it is unarbitrary for UTA algorithm.

- **Non-proportional (N.P.) and uniform (U.)**: For this scenario, the QoS information is produced exactly as it is in the proportional and uniform scenario. The only difference is that for each worker-task pair (w, t), the reward $r_t(w)$ is assigned randomly from $[1, b_t]$.

- **Non-proportional (N.P.) and non-uniform (N.U.)**: Given a worker-task pair (w, t), we randomly assign the reward $r_t(w)$ from $[1, b_t]$ and $q_t(w)$ from $[1, 200]$.

Given the settings described above, the preference profiles of workers and tasks can be determined by (3.1) and (3.2), respectively. (Yet it should be noted that in practice none of the algorithms requires tasks to form their preference lists, which would take $O(n2^n)$ time and space.)

Lastly, we run the simulations 100 times with a different user set in each run and present the averaged results.

3.4.1.1 Benchmark Algorithms

We compare the performance of our algorithms with the following algorithms proposed in [35] and [66] (see Table 3 for a comparison of the time complexities of all algorithms).

- *Stable Job Assignment (SJA)*: This ILP-based algorithm [35] produces pairwise stable matchings solely in proportional and uniform MCS systems.

- *ϕ-Stable Task Assignment (ϕ-STA)*: Proposed in [66], this approximation algorithm

Algorithm	Time complexity
UTA*	$O(n \log n + nm)$
PSTA*	$O(n^2 m\beta)$
Heuristic*	$O(knm\beta)$
SJA	$O(nm2^n)$
ϕ-STA	$O(mn \log(mn))$
θ-STA	$O(mn \log(mn))$

Table 3. Time complexities of all algorithms considered in the simulations (* indicates the algorithms proposed in this study).

produces matchings that are guaranteed to be coalitionally ϕ-stable in proportional matching markets, where ϕ (≈ 1.618) denotes the golden ratio. In this algorithm, tasks (t) simply run the ϕ-approximation algorithm for the single bin removable online knapsack problem proposed in [82] to decide whether to accept (and discard some other workers if needed) or reject the proposing workers (w) using the rewards $r_t(w)$ as the weights of items and b_t as the size of the knapsack. The running time of this algorithm is $O(mn \log(mn))$.

- *θ-Stable Task Assignment (θ-STA)*: This approximation algorithm is also proposed in [66] and generates coalitionally θ-stable matchings in general matching markets, where

$$\theta = \frac{1}{1 - \max_{w \in \mathcal{W}, t \in \mathcal{T}} \frac{r_t(w)}{b_t}} \tag{3.30}$$

It follows the classic deferred acceptance mechanism: workers make the proposals, tasks (t) that have available budget accept the incoming proposals and those that do not have available budget temporarily add the proposing worker to their partner

MCS Type	UTA	PSTA	Heuristic	SJA	ϕ-STA	θ-STA
P. & U.	✓	✓	✓	✓	✓	*
P. & N.U.		✓	✓		✓	*
N.P. & U.	✓	✓	✓			✓
N.P. & N.U.		✓	✓			✓

Table 4. Mobile crowdsensing scenarios and corresponding applicable algorithms ($*$ indicates that the algorithm is applicable but has a very poor performance since it is not specifically designed for that scenario).

set and then discard the workers (w') with the lowest $\frac{q_t(w')}{r_t(w')}$ ratio until the sum of rewards to be paid to the remaining workers is less than or equal to b_t. As it is shown in Table 4, although it can be run in all types of MCS systems, we do not provide results for this algorithm in proportional MCS systems due to its unpredictable and mostly poor performance in these systems. This is because it randomly selects the workers to be discarded in proportional settings as all workers (w') have the same $\frac{q_t(w')}{r_t(w')}$ ratio for each task t. The time complexity of this algorithm is also $O(mn \log(mn))$.

Note that neither ϕ-STA nor θ-STA has a performance guarantee in terms of pairwise stability.

3.4.1.2 Performance Metrics

We utilize the following performance metrics in the evaluations.

- *Overall user happiness*: This is calculated as

$$100 \times \left(1 - \frac{\text{\# of coalitionally unhappy pairs}}{\text{\# of all matchable worker-task pairs}}\right) \tag{3.31}$$

and expresses the overall user happiness based on the instability of the matching. Thus, it is the main metric that defines the performance of the algorithms.

- *Outward user happiness*: This is calculated as

$$100 \times \left(1 - \frac{\text{\# of unhappy pairs}}{\text{\# of all matchable worker-task pairs}}\right) \tag{3.32}$$

and quantifies the outward user happiness based on the one-dimensional instability of the matching. The reason it is called *outward* is that compared to coalitionally unhappy pairs, unhappy pairs are easier to notice for users, and the users forming an unhappy pair have a stronger incentive to deviate from (a subset of) their partners to each other as they do not need a collective agreement that involves other users (unlike coalitionally unhappy pairs).

- *Maximum dissatisfaction ratio*: This is the dissatisfaction of the task in the unhappy coalition with the largest incentive to deviate from the current matching, which is formally defined as

$$\delta_{max} = \max_{t \in \mathcal{T}} \delta_t. \tag{3.33}$$

Given the maximum dissatisfaction ratio δ_{max} of a matching $\mathcal{M}$, we can say that $\mathcal{M}$ is coalitionally δ_{max}-stable and is not coalitionally $(\delta_{max} - \epsilon)$-stable for any positive real ϵ. If a matching does not have any unhappy coalition, then $\delta_{max} = 1$ by definition.

- *Running time*: We also compare the algorithms with respect to their running time, which might be critical for MCS systems with strict time constraints.

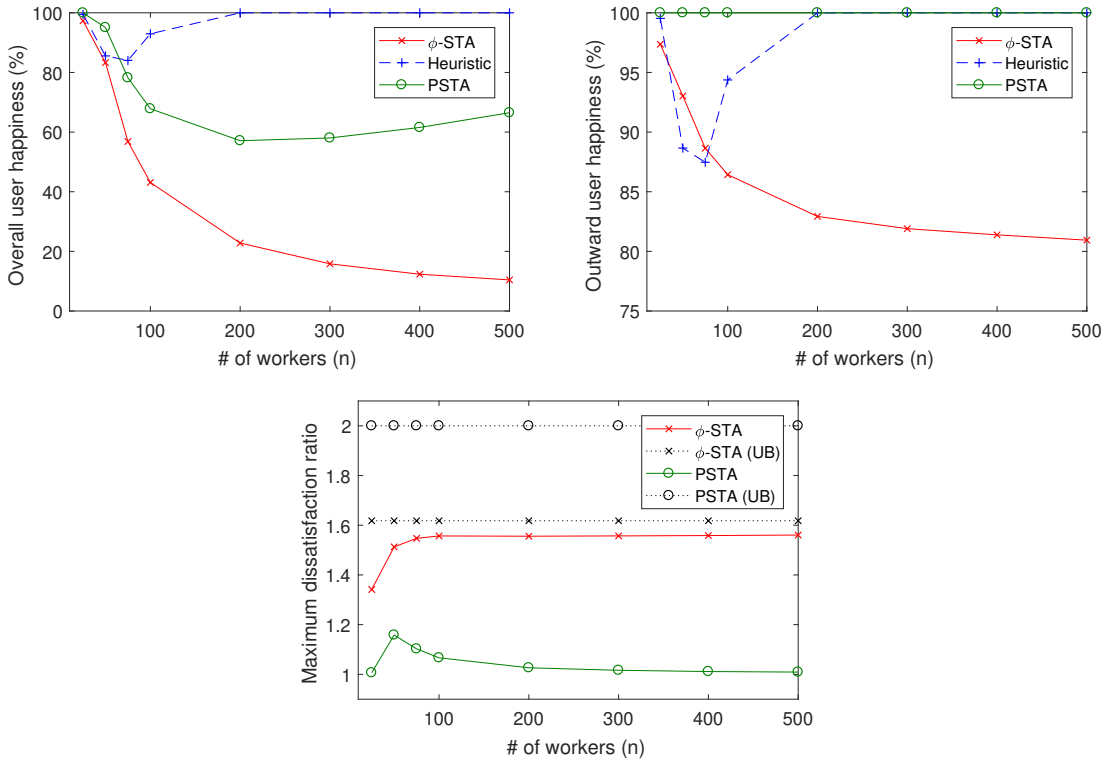


Fig. 5. Performance comparison of algorithms in proportional and non-uniform scenario with varying number of workers ($m = 50$ tasks).

3.4.2 Results

We first look at the performance of the proposed algorithms in proportional and non-uniform scenario. Fig. 5 shows the performance of algorithms with different number of workers. First, note that Heuristic algorithm (which is run with $k = 3$ as default) usually performs the best and produces optimal assignments in terms of both overall and outward user happiness when the number of workers is larger than 200. It is interesting that it achieves very similar overall and outward user happiness scores, which indicates that it yields matchings in which most of the coalitionally unhappy pairs are also unhappy pairs. Second, we see that despite having a better upper bound (UB) in terms of maximum dissatisfaction ratio (i.e., the upper bounds for ϕ-STA and

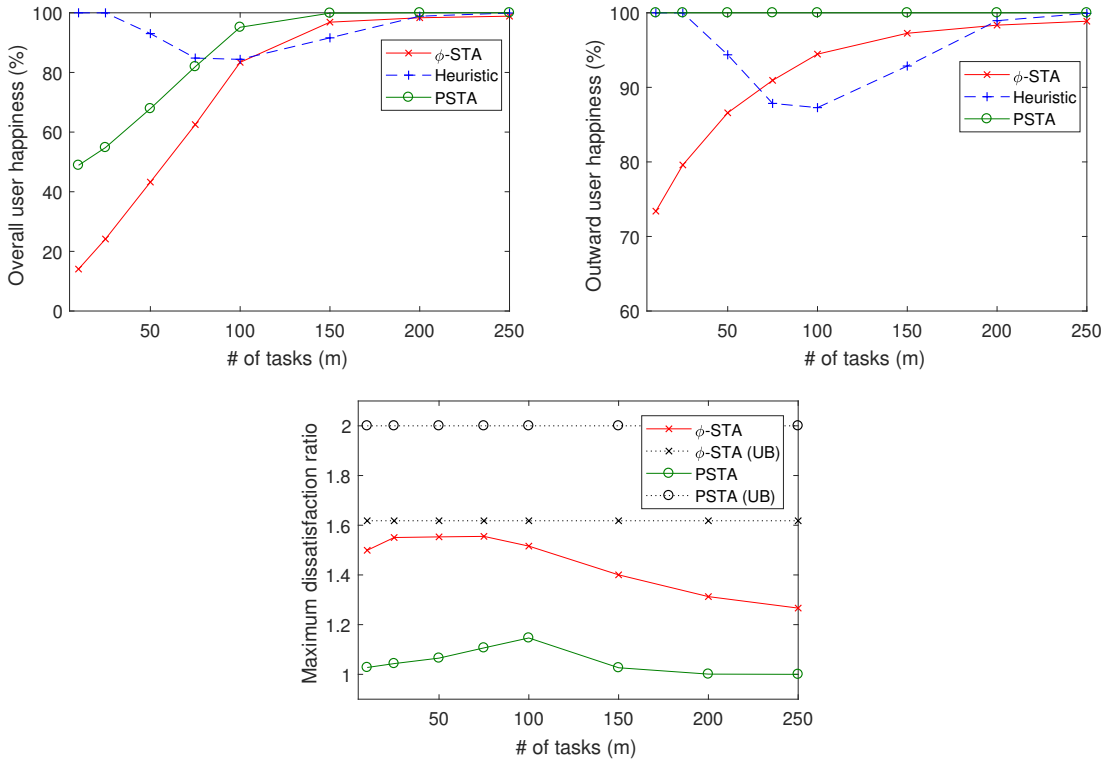


Fig. 6. Performance comparison of algorithms in proportional and non-uniform scenario with varying number of tasks ($n = 100$ workers)

PSTA are, respectively, ϕ (≈ 1.618) and 2, while Heuristic algorithm is unbounded), ϕ-STA mostly performs much worse than our algorithms. Besides, its performance gets worse as the number of workers increases and it even produces matchings with as low as 10% overall user happiness. Since the system is proportional, PSTA algorithm always achieves 100% outward user happiness (Theorem 4), but we still include it in all figures for completeness. Also, it outperforms Heuristic algorithm when the number of workers is small and generally achieves a maximum dissatisfaction ratio that is much smaller than its upper bound and very close to the optimal value (i.e., 1).

Fig. 6 shows the performance of algorithms with varying number of tasks. As for the performance of Heuristic algorithm, we see a trend that is similar to what we

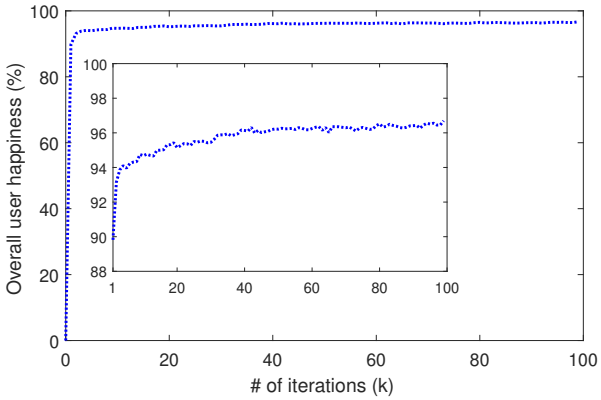


Fig. 7. Performance of Heuristic algorithm with increasing number of iterations in proportional and non-uniform scenario with $m = 50$ tasks and $n = 100$ workers.

have seen in Fig. 5. It performs worse when the number of workers and tasks are close to each other. On the other hand, ϕ-STA and PSTA algorithms always have a better performance with increased number of tasks. This is because they are worker-oriented algorithms where the proposals are made by workers, so the increase in the number of tasks reduces the competition among workers and results in improved user happiness. Here, PSTA always performs better than ϕ-STA, and it also outperforms Heuristic algorithm when the number of tasks get larger than that of workers.

In Fig. 7, we show the performance of Heuristic algorithm when it is run with different k values. We see that even with a few number of iterations, it achieves about 95% overall user happiness, and that increasing the number of iterations continues to improve the performance even up until 100 iterations, but it may not be worth the increase to be seen in the runtime, which is linear to the number of iterations.

In Fig. 8, we look at the impact of C (cost per kilometer) on the performance of the algorithms. Note that an increased C value can be interpreted as having very selective workers who do not even find most of the tasks acceptable. Thus, it deescalates the competition among workers and results in an improvement in the performance of

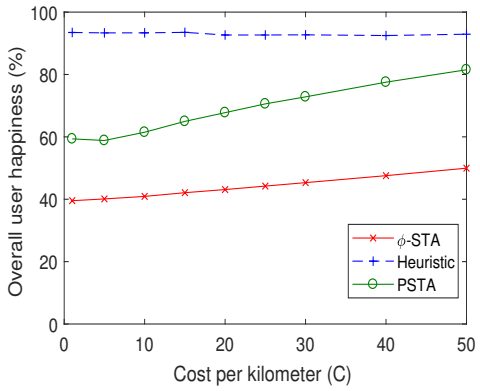


Fig. 8. Performance comparison of algorithms with varying cost per kilometer values in proportional and non-uniform scenario with $m = 50$ tasks and $n = 100$ workers.

worker-oriented algorithms (ϕ-STA and PSTA). Conversely, it escalates the competition among tasks as there will be less number of eligible workers for each task, so the performance of task-oriented Heuristic algorithm gets slightly worse. The similar outcomes are also seen in Fig. 9 where we look at the impact of reward ranges. When we lower the reward range from $[1, b_t]$ to $[1, b_t/2]$, there will be fewer workers eligible for each task, so the performance of ϕ-STA and PSTA algorithms gets better, while that of Heuristic algorithm gets worse due to the same reasons pointed out above. However, when the reward range is made $[b_t/2, b_t]$, the performance of all algorithms get better (PSTA and Heuristic algorithms even achieve optimal overall user happiness) because tasks can now hire at most 2 workers, making it an almost competition-free setting for both workers and tasks.

Next, we analyze the performance of algorithms in non-proportional and non-uniform scenario in Fig. 10. Note that since the system is non-proportional, PSTA algorithm fails to achieve perfect outward user happiness most of the times, but still manages to outperform the others in terms of outward user happiness. The performance of Heuristic algorithm seems similar to its performance in proportional

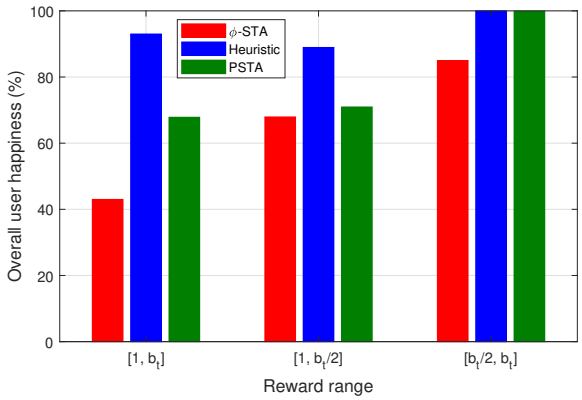


Fig. 9. Performance comparison of algorithms with varying reward ranges in proportional and non-uniform scenario with $m = 50$ tasks and $n = 100$ workers.

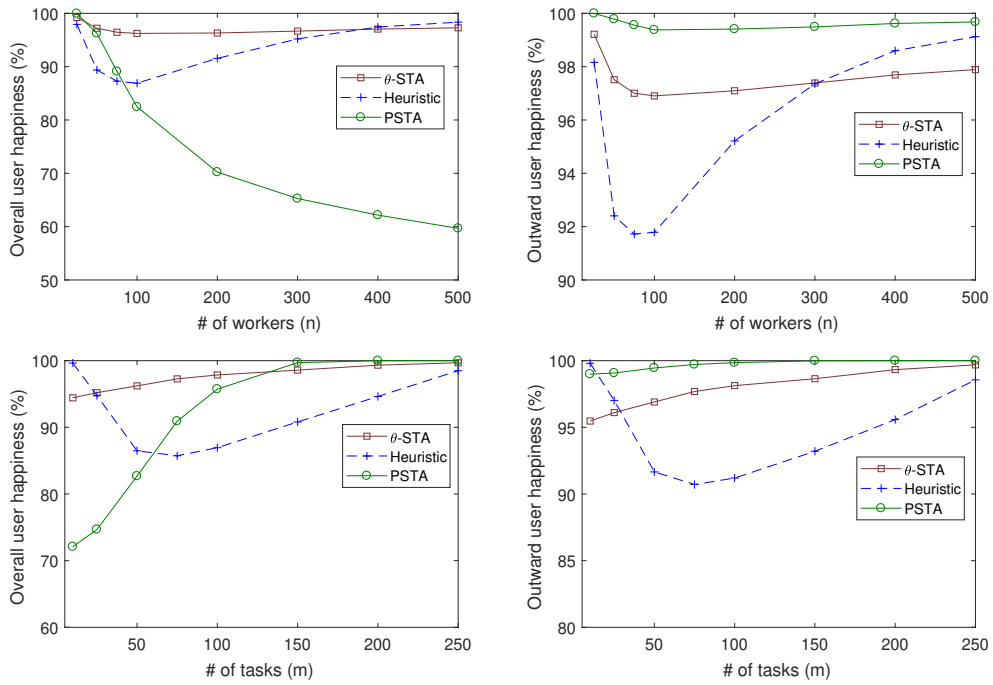


Fig. 10. Performance comparison of algorithms in non-proportional and non-uniform scenario with varying number of workers and tasks.

and non-uniform scenario: it achieves higher than 85% overall/outward user happiness regardless of the number of workers/tasks and performs the worst when the number of workers and tasks are similar. However, in this scenario, our algorithms are usually

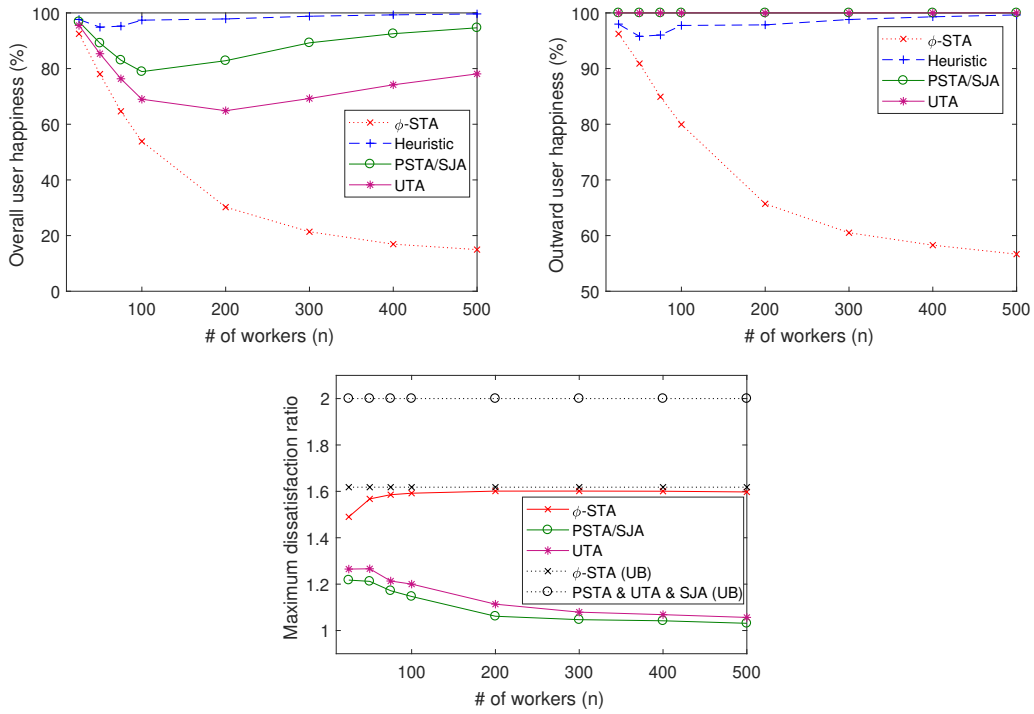


Fig. 11. Performance comparison of algorithms in proportional and uniform scenario with varying number of workers ($m = 50$ tasks).

outperformed by θ-STA algorithm in terms of overall user happiness. In fact, θ-STA algorithm has a quite reliable performance in this scenario as its overall user happiness score never drops below 95%.

Fig. 11 and 12 show the performance comparison of algorithms in proportional and uniform scenario for varying number of workers and tasks, respectively. Here, Heuristic algorithm always outperforms all other algorithms in terms of overall user happiness by steadily achieving very close to optimal scores ($\geq$ 97%) regardless of worker and task counts. On the other hand, ϕ-STA algorithm always has the poorest performance in terms of all metrics considered here. Since the system is both proportional and uniform, UTA and PSTA algorithms are guaranteed to achieve perfect outward user happiness due to Theorems 3 and 4, respectively. It is remarkable that

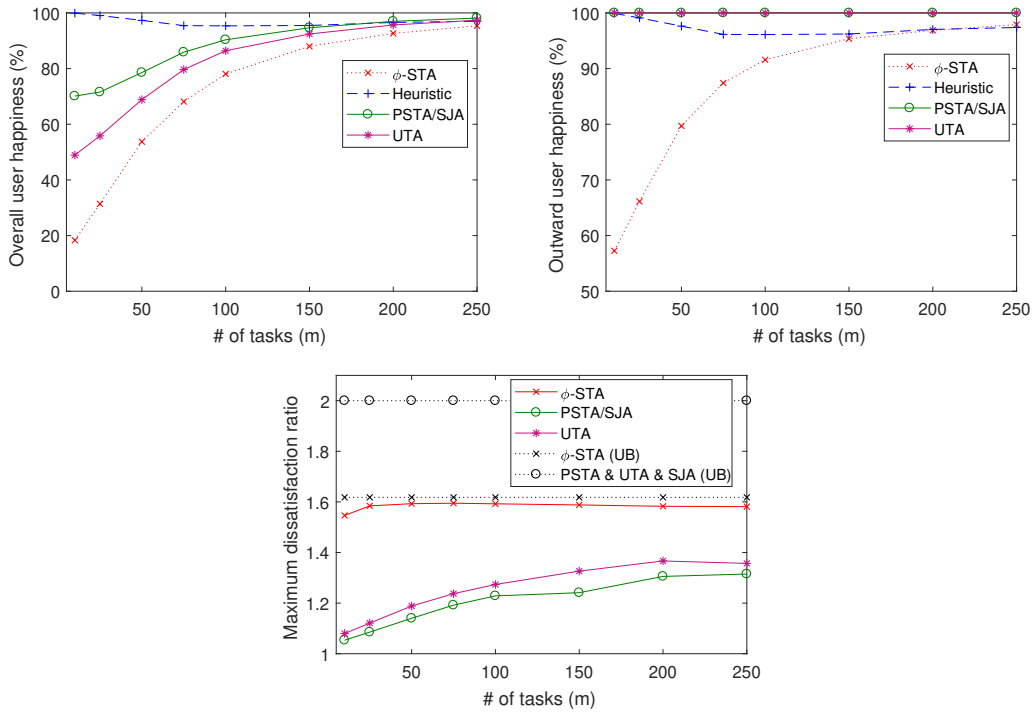


Fig. 12. Performance comparison of algorithms in proportional and uniform scenario with varying number of tasks ($n = 100$ workers).

the increase in the number of tasks improves the performance of UTA and PSTA algorithms in terms of overall user happiness, while it has a detrimental effect on them in terms of maximum dissatisfaction ratio.

In Fig. 13, we look at the performance of algorithms in non-proportional and uniform scenario. We first observe that UTA algorithm generally has the worst performance in terms of overall user happiness, yet it is the only algorithm that ensures a perfect outward user happiness. Second, the performance of Heuristic algorithm is significantly worse in this scenario compared to that in the others. In fact, this is the only scenario where it achieves less than 85% overall user happiness. Similar to the non-proportional and non-uniform scenario, θ-STA algorithm usually manages to deliver a higher overall user happiness score than the others when there are more

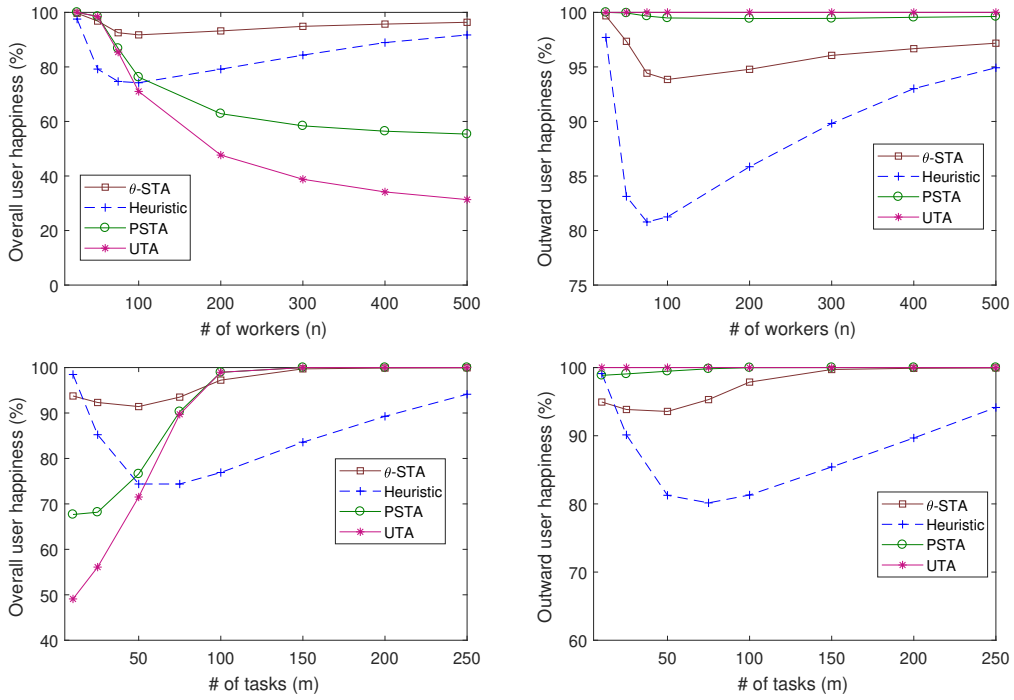


Fig. 13. Performance comparison of algorithms in non-proportional and uniform scenario with varying number of workers and tasks.

workers than tasks, but its performance is also worse in this scenario. Besides, it is significantly outperformed by PSTA and UTA algorithms in terms of outward user happiness for the most part.

Lastly, in Fig. 14a-b, we compare the running time of all algorithms. We only provide the results for the proportional and uniform scenario as all of the algorithms can be run in this scenario and those that are also run in different scenarios have an almost indistinguishable running time in all scenarios they have been used with. First, note that the objective of SJA algorithm is to obtain assignments with perfect outward user happiness in proportional and uniform systems. Our UTA algorithm achieves the same in not only proportional and uniform systems, but also in non-proportional and uniform systems in an extremely shorter time (by a few orders of magnitude). Furthermore, it has the lowest running time among all, which is consistent with its

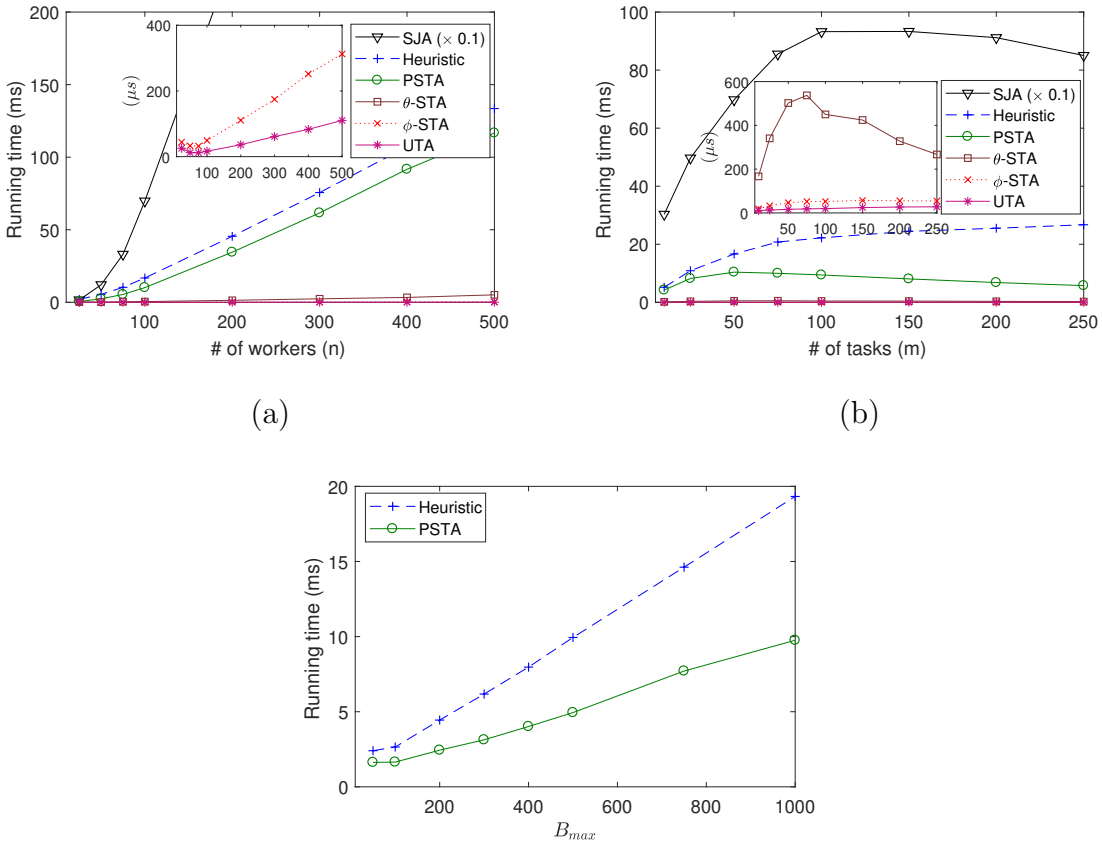


Fig. 14. Comparison of algorithms in terms of running time (a-b); Impact of B_{max} on the running time of Heuristic and PSTA algorithms (c).

superior time complexity ($O(n \log n + mn)$). Our other algorithms (Heuristic and PSTA) are also much more time-efficient than SJA algorithm, though they have notably larger running time compared to the rest of the algorithms. Nonetheless, it should be noted that we have assumed a pre-scaling will have to be made in case budgets and rewards are not defined as integers. Therefore, we used relatively large budget (up to $B_{max} = 1000$) and hence reward values in all experiments. If pre-scaling can be avoided (or the task requesters in the system have a rather limited budget), the running time of Heuristic and PSTA algorithms will decrease linearly

with reduced budget values as shown in Fig. 14c. Finally, we observe that increasing the number of tasks after when there are equal number of workers and tasks in the system ($n = m = 100$) either reduces the running time or the increase in the running time of all algorithms. This is because when there are more tasks than workers, workers will be able to find their stable partners in a shorter time as they are more likely to be matched with a task that is at the top of their preference lists.

3.5 Conclusion

In this chapter, we studied the stable task assignment problem in MCS systems. Different from the classic stable matching problem, the task requesters in an MCS system may recruit multiple workers within their budget ranges to reinforce the quality of the sensed data. This makes the generic stability definitions obsolete and the existing approaches to find stable matchings inapplicable in MCS systems. To address this problem, we first defined the stability conditions peculiar to MCS systems, and provided the existence and hardness results for stable task assignments in different types of MCS systems. Then, we introduced three different stable task assignment algorithms, namely UTA, PSTA, and Heuristic. We proved that UTA and PSTA algorithms always produce pairwise stable task assignments in uniform and proportional MCS systems, respectively. Finally, we evaluated the performance of the proposed algorithms in terms of user happiness through extensive simulations. The results have shown that our algorithms significantly outperform the state-of-the-art stable task assignment algorithms in most scenarios. Specifically, PSTA and Heuristic algorithms usually achieve the highest outward and overall user happiness, respectively.

CHAPTER 4

PREFERENCE-AWARE TASK ASSIGNMENT WITH COVERAGE REQUIREMENTS

4.1 Introduction

In opportunistic mobile crowdsensing, the objective of service requesters is to have as many of their sensing tasks completed as possible within their budget constraints, whereas that of participants (workers) is to collect the highest monetary reward possible on their trajectories. However, these objectives can conflict and may result in unhappy service requesters or workers if the matching between them is not handled carefully. In this chapter, we study the problem of finding task assignments that fulfill both coverage-aware preferences of service requesters and profit-based preferences of workers in a budget-constrained, opportunistic mobile crowdsensing system. In the problem studied in the previous chapter, task requesters were assumed to have additive utility functions based on worker qualities. However, in this study, task requesters have non-additive utility functions due to their coverage requirements, which makes the solutions proposed in the previous chapter inapplicable to the setting considered here. The key issues that need to be taken into account in this problem and the studies that partly address them can be summarized as follows:

- *Task requester preferences [8, 9, 26]:* Each task requester desires to have a matching that maximizes the coverage over the PoIs that her task needs.
- *Budget feasibility [8, 9, 26, 44, 35]:* Each task requester has a budget constraint which should not be violated.

- *Worker preferences [47, 44, 35, 33]:* Each worker desires to maximize his net profit from the system in each task assignment period.

- *Stability [47, 44, 35, 33]:* Since the objectives above are likely to be in conflict with each other, they should be achieved in a fair way that results in as few unhappy users as possible.

In this study, we address all of these issues together and make the following contributions:

- We formally define the stability conditions for task assignments in coverage-aware, opportunistic MCS systems with budget constraints.

- We prove that a fully stable task assignment may not exist in some MCS instances, and it is NP-hard even to check whether one exists in a given instance.

- We present a polynomial-time approximation algorithm for the stable task assignment problem, and prove that it always produces $\frac{4}{1-\rho}-$stable matchings, where ρ is the largest reward to budget ratio (normalized between 0 and 1) in the system.

- We show that a variant of our algorithm has an approximation ratio of 5 in MCS systems with proportional reward schemes, where the rewards offered to the workers are proportional to the utility they provide for the tasks.

- We compare the performance of our algorithms with two benchmark algorithms proposed in [8] and [9] via real-data based, extensive simulations, and show that our algorithms produce significantly better task assignments in terms of both user happiness (up to 25%) and achieved coverage (up to 18%), and run up to four orders of magnitude faster compared to the benchmark algorithms in most settings.

4.2 System Model

4.2.1 Assumptions

We assume a system model with a matching platform that receives sensing task requests $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ over a set of PoIs $\mathcal{P} = \{p_1, p_2, \ldots, p_k\}$, and determines the assignments between these tasks and workers (data contributors). Each task t needs a type of sensed data from a certain subset of PoIs, denoted by $P(t) \subseteq \mathcal{P}$. For a task, some of the PoIs might be more important than the others due to their spatial features (e.g., being close to a production plant for an air pollution sensing task), thus we also let each task t assign a weight $v_t(p)$ to each PoI p in $P(t)$.

Let $\mathcal{W} = \{w_1, w_2, \ldots, w_m\}$ be the set of registered workers in the system. We assume that workers are not willing to interrupt their daily schedule, but they accept to perform the tasks on their trajectories, i.e., *opportunistic sensing*. According to the frequency of task assignments and the nature and time sensitivity of tasks in the system, a different portion and timescale of their future trajectories (e.g., daily, hourly) can be considered in the task assignment process. Let X_w be the set of all locations that will be visited by worker w during the considered time frame. Similar to the previous work [8], we assume that a PoI is covered by worker w if it falls in the sensing range d_w of the worker. Then, the set of PoIs that are covered by worker w is given by

$$C(w) = \{p \in \mathcal{P} : d(p, x) \leq d_w, \exists x \in X_w\}, \tag{4.1}$$

where $d(p, x)$ is the Euclidean distance between the PoI p and $x \in X_w$. The coverage set of worker w for task t can then be defined as:

$$C_t^w = C(w) \cap P(t) \tag{4.2}$$

If the requester of task t needs to have some of the PoIs sensed by a deadline that is earlier than the end of the current assignment cycle, and worker w will not arrive at the corresponding locations in time according to his trajectory, then we can simply remove these PoIs from C_t^w. The total utility of a set S of workers for task t is equal to their total weighted coverage over $P(t)$, which can be calculated by

$$U_t(S) = \sum_{p \in C_t^S} v_t(p), \text{ where } C_t^S = \cup_{w \in S} C_t^w \tag{4.3}$$

Consider the instance illustrated in Fig. 15, and let t be a task in this instance with the following properties:

$$P(t) = \{p_1, p_2, p_3, p_4\},$$
$$v_t(p_i) = v_t(p_j) = 1, \forall i, j \in \{1, 2, 3, 4\}.$$

Then, the individual utilities of workers w_1 and w_2 for task t would be $U_t(w_1) = 3$ and $U_t(w_2) = 2$, since they cover 3 and 2 of the PoIs requested in task t, respectively. Their joint utility for task t would be $U_t(\{w_1, w_2\}) = 4$, which is evidently less than the sum of their individual utilities. This demonstrates the non-additiveness of the utility function given in Eq. 4.3.

Moreover, we assume a budget-constrained system model with monetary incentives, where the requester of task t has a budget b_t that limits the amount of monetary incentives to be spent for the completion of task t, and offers each eligible worker w a reward $r_t(w)$ ($\leq b_t$) to cover the PoIs in C_t^w. Let $r_t(S)$ be the total rewards offered to the worker set S (i.e., $r_t(S) = \sum_{w \in S} r_t(w)$). Besides, for each worker w, there is a cost $c_t(w)$ associated with each task t, which worker w can estimate considering the factors such as cost of delivering the sensed data to the task requester via cellular networks, energy consumption due to sensing, and privacy risks.

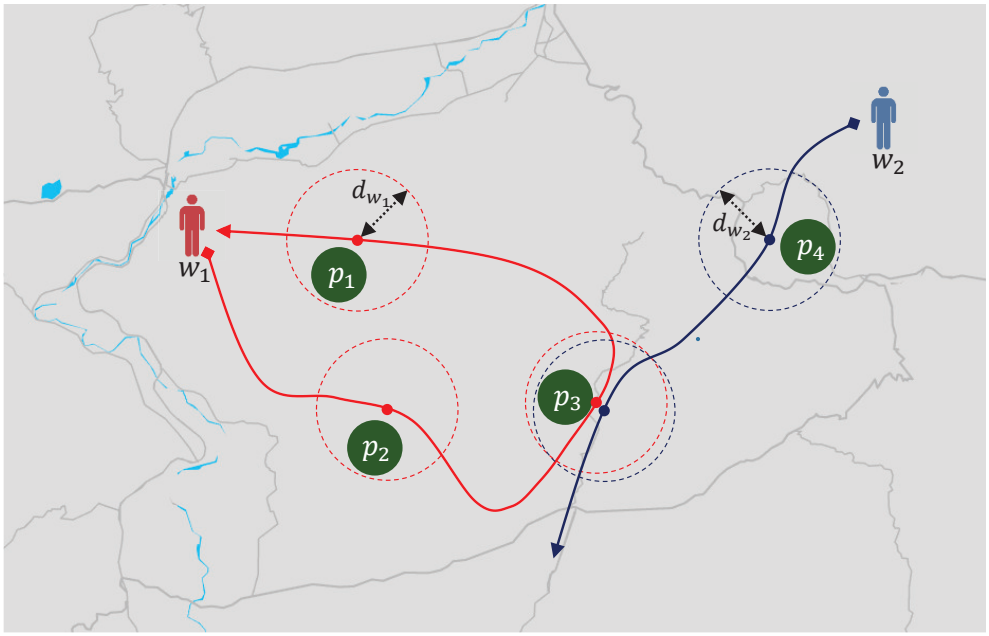


Fig. 15. An MCS instance with 2 workers (w_1, w_2) and 4 PoIs (p_1, p_2, p_3, p_4). The trajectories of workers are shown with solid lines.

Let $\mathcal{M}$ be a matching between the tasks and the workers in the system. Also, let $\mathcal{M}(u)$ denote the assigned task (worker set) to worker (task) u in $\mathcal{M}$. In order for $\mathcal{M}$ to be a feasible and individually rational matching, it should satisfy the following conditions:

- a worker w is either unmatched or matched with a task, i.e.,

$$\mathcal{M}(w) = \emptyset \ or \ \mathcal{M}(w) \in \mathcal{T}, \tag{4.4}$$

- a task is matched with a subset of workers (which may be $\emptyset$), i.e.,

$$\mathcal{M}(t) \subseteq \mathcal{W}, \tag{4.5}$$

- if worker w is matched with task t, the worker set of task t also includes worker

w, and vice versa, i.e.,

$$\mathcal{M}(w) = t \ \textit{iff}\ w \in \mathcal{M}(t), \tag{4.6}$$

- no worker w is matched with a task that is not economically beneficial for him, i.e.,

$$r_t(w) > c_t(w) \ \textit{if}\ \mathcal{M}(w) = t, \tag{4.7}$$

- and, no task t is matched with a set of workers that she cannot afford, i.e.,

$$\sum_{w \in \mathcal{M}(t)} r_t(w) \leq b_t. \tag{4.8}$$

The worker-task pair (w, t) is said to be a *qualified* pair if there exists a feasible and individually rational matching, in which worker w is matched with task t.

Reward schemes. In this study, we consider two different reward schemes: *general* and *proportional*. In the general scheme, there is not any assumed relation between the rewards a task requester offers to workers and the utility they provide for the relevant task. However, based on the common practice seen in most of the real-world applications (e.g., Amazon MTurk [83]), it is natural to see a correlation between the two. Hence, in the proportional scheme, we assume that for each task t, the amount of rewards offered to the workers are proportional to their utility. That is

$$r_t(w) = \theta_t \times U_t(w), \tag{4.9}$$

where θ_t denotes the reward per utility value for task t. It should be noted that a different task t' may have a different reward per utility value (i.e., $\theta_t \neq \theta_{t'}$).

A summary of the key notations used in this chapter is presented in Table 5.

Notation	Description
$\mathcal{W}$, $\mathcal{T}$, $\mathcal{P}$	Set of workers, tasks and PoIs, respectively
m, n, k	Number of workers, tasks and PoIs, respectively
$\mathcal{M}$	A feasible & individually rational task assignment
$\mathcal{M}(u)$	Assigned task (worker set) to worker (task) u in $\mathcal{M}$
$c_t(w)$	Cost of performing task t for worker w
$r_t(w)$	Reward offered to worker w to perform task t
$g_t(w)$	Profit of worker w from task t $(r_t(w) - c_t(w))$
$r_t(S)$	$\sum_{w \in S} r_t(w)$
$P(t)$	PoI set of task t
$v_t(p)$	Weight of PoI $p \in P(t)$ for task t
$C(w)$	Set of PoIs covered by worker w
C_t^w	$C(w) \cap P(t)$
$U_t(S)$	Utility of $S \subseteq \mathcal{W}$ for task t
b_t	Budget of task t
$b_t^{\mathcal{M}}$	Remaining budget of task t in $\mathcal{M}$
δ_t	Dissatisfaction ratio of task t
L_w	Preference list of worker w
d_w	Sensing range of worker w
ρ	$\max_{w \in \mathcal{W}, t \in \mathcal{T}} r_t(w)/b_t$

Table 5. Notations used in Chapter 4.

4.2.2 Problem Formulation

Below, we formally define the stability conditions for our settings.

Definition 10 (Unhappy coalition)**.** *Given a matching $\mathcal{M}$, a task t and a subset S of workers form an unhappy coalition (denoted by $\langle S, t\rangle$) if the following conditions hold for a subset S' of the workers assigned to task t in $\mathcal{M}$:*

- *task t would be better off with S than with S', i.e.,*

$$U_t(S \cup (\mathcal{M}(t) \setminus S')) > U_t(\mathcal{M}(t)), \tag{4.10}$$

- *task t can replace S' with S without violating her budget constraint, i.e.,*

$$r_t(S) - r_t(S') \leq b_t^{\mathcal{M}}, \tag{4.11}$$

  where $b_t^{\mathcal{M}}$ is the remaining budget of task t in $\mathcal{M}$ (i.e., $b_t^{\mathcal{M}} = b_t - \sum_{w \in \mathcal{M}(t)} r_t(w)$),

- *every worker w in S prefers task t to task t' to whom he is currently assigned in $\mathcal{M}$, i.e.,*

$$\forall w \in S,\ g_t(w) > g_{t'}(w), \tag{4.12}$$

  where $g_t(w) = r_t(w) - c_t(w)$ is the net profit of performing task t for worker w, and $g_{t'}(w) = 0$ if worker w is currently unmatched (i.e., $\mathcal{M}(w) = t' = \emptyset$).

Given a worker-task pair (w, t), if there exists an unhappy coalition $\langle S, t\rangle$ such that $w \in S$, we call this pair a *coalitionally unhappy pair*.

Definition 11 (Stable matching)**.** *A matching $\mathcal{M}$ is (coalitionally) stable if it does not contain any unhappy coalitions.*

Note that this is the strongest stability definition in many-to-one matching problems (see [65] for weaker stability definitions). Therefore, if a matching is stable, no one in the matching has even a small incentive to deviate from their current assignment. However, even with additive utilities where the total utility of a set of workers

for a task is simply the sum of their individual utilities, a stable matching may not exist, and checking the existence (and finding one if exists) is NP-hard [65]. Since non-additive utilities are a more generalized form of additive utilities (i.e., a problem instance with additive utilities can easily be converted to one with non-additive utilities, but not vice versa), we conclude that the same existence and hardness results also apply to the problem of finding stable matchings in our settings where the utilities of workers for tasks are non-additive as we have

$$U_t(\{w_i, w_j\}) < U_t(\{w_i\}) + U_t(\{w_j\}). \tag{4.13}$$

when $C_t(w_i) \cap C_t(w_j) \neq \emptyset$. The following theorem formally proves nonexistence of optimal solutions in some MCS instances.

Theorem 6. *There exist MCS instances that do not allow for a stable matching.*

Proof. We prove by giving an example, which is described in Table 6. All of the feasible and individually rational matchings that can be defined on this instance is also provided in Table 7. Since each matching contains at least one unhappy coalition, we conclude that no stable matching exists in this instance. □

Due to the nonexistence and hardness results for stable matchings, we consider the following relaxation. First, let $\mathcal{S}_t$ be the set of all worker sets that form an unhappy coalition with task t in a given matching $\mathcal{M}$. Formally,

$$\mathcal{S}_t = \{G \subseteq \mathcal{W} : \langle G, t\rangle \text{ is an unhappy coalition}\}. \tag{4.14}$$

Also, $\forall S \in \mathcal{S}_t$, let

$$\mathcal{E}_S = \{E \subseteq \mathcal{M}(t) : r_t(S) \leq b_t^{\mathcal{M}} + r_t(E)\}, \tag{4.15}$$

		Rewards	
$\mathcal{W}$	$C(w)$	t_1	t_2
w_1	p_1, p_2, p_3, p_4	4	0
w_2	$p_5, p_6, p_7, p_{10}, p_{11}$	3	2
w_3	p_8, p_9, p_{12}	2	3

$\mathcal{T}$	$P(t)$	b_t
t_1	p_{1-9}	5
t_2	p_{10-12}	3

Table 6. An MCS instance where no fully optimal or stable task assignment exists. The weights of the PoIs are identical for both tasks, and $c_w(t) = 0$ for all (w, t) pairs.

$\mathcal{M}$	Unhappy coalition	δ_{t_1}	δ_{t_2}
$t_1 \to \emptyset$; $t_2 \to \emptyset$	$\langle \{w_1\}, t_1 \rangle$	∞	∞
$t_1 \to \emptyset$; $t_2 \to w_2$	$\langle \{w_1\}, t_1 \rangle$	∞	1
$t_1 \to \emptyset$; $t_2 \to w_3$	$\langle \{w_1\}, t_1 \rangle$	∞	2
$t_1 \to w_1$; $t_2 \to \emptyset$	$\langle \{w_2\}, t_2 \rangle$	5/4	∞
$t_1 \to w_2$; $t_2 \to \emptyset$	$\langle \{w_3\}, t_2 \rangle$	5/3	∞
$t_1 \to w_3$; $t_2 \to \emptyset$	$\langle \{w_2\}, t_2 \rangle$	5/2	∞
$t_1 \to w_2, w_3$; $t_2 \to \emptyset$	$\langle \{w_3\}, t_2 \rangle$	1	∞
$t_1 \to w_1$, $t_2 \to w_2$	$\langle \{w_2, w_3\}, t_1 \rangle$	5/4	1
$t_1 \to w_1$, $t_2 \to w_3$	$\langle \{w_2\}, t_2 \rangle$	1	2
$t_1 \to w_2$, $t_2 \to w_3$	$\langle \{w_1\}, t_1 \rangle$	4/3	1
$t_1 \to w_3$, $t_2 \to w_2$	$\langle \{w_2\}, t_1 \rangle$	5/2	1

Table 7. All feasible matchings that can be defined on the instance given in Table 6 along with one of the unhappy coalitions they contain and the dissatisfaction ratios of tasks.

and

$$S^R = \arg\min_{E \in \mathcal{E}_S} U_t(E \setminus (S \cup \mathcal{M}(t))). \tag{4.16}$$

That is, $S^R \subseteq \mathcal{M}(t)$ is the minimum loss worker set that can be replaced by S within the budget constraint of task t. Then, we can calculate the dissatisfaction ratio of task t in this matching by:

$$\delta_t = \begin{cases} 1, & \text{if } \mathcal{S}_t = \emptyset \\ \infty, & \text{if } \mathcal{S}_t \neq \emptyset,\ \mathcal{M}(t) = \emptyset \\ \max\limits_{S \in \mathcal{S}_t} \left\{ \frac{U_t(S \cup (\mathcal{M}(t) \setminus S^R))}{U_t(\mathcal{M}(t))} \right\}, & \text{otherwise.} \end{cases} \tag{4.17}$$

Note that the minimum value that δ_t can have is 1, which indicates that task t is perfectly happy in the matching. Finally, we can formally define our objective function as:

$$\text{maximize } \min_{t \in \mathcal{T}} \frac{1}{\delta_t}. \tag{4.18}$$

Consider the instance given in Table 6. Based on the dissatisfaction ratios of tasks in different feasible matchings provided in Table 7, the optimal matching with respect to (4.18) is $t_1 \to w_1$, $t_2 \to w_2$ where the value of (4.18) is 0.8. The following definition will be used hereafter to signify how optimal a matching is in terms of stability.

Definition 12 (α-stable matching)**.** *A matching $\mathcal{M}$ is α-stable ($\alpha \geq 1$) if*

$$\max_{t \in \mathcal{T}} \delta_t \leq \alpha. \tag{4.19}$$

Algorithm 5: Initialize $(\mathcal{W}, \mathcal{T})$, $\mathcal{M}$

Input: $\mathcal{W}$: Set of workers

$\mathcal{T}$: Set of tasks

$\mathcal{M}$: Matching between $\mathcal{W}$ and $\mathcal{T}$

```
1  foreach w ∈ W do
2      L_w ← order T by non-increasing value of g_t(w)
3      L_w ← L_w \ {t ∈ L_w | g_t(w) ≤ 0}
4      index_w = 1
5      M(w) = ∅
6  foreach t ∈ T do
7      foreach w ∈ W do
8          x_t(w) = 0, η_t(w) = 0
9          foreach p ∈ P(t) do
10             z_t(p, w) = 0
11     H_t = ∅
12     M(t) = ∅
13     A_t = false
```

4.3 Proposed Solution

The outline of our polynomial-time approximation algorithm is presented in Algorithm 6. The main idea behind it is to check the potential assignments between the qualified pairs following the order in the preference lists of the workers, and make matching decisions by converting the worker selection problem to an online optimization problem. It begins by calling Algorithm 5, which forms the preference list L_w

Algorithm 6: StableTaskAssignment ($\mathcal{W}$, $\mathcal{T}$, σ)

Input: $\mathcal{W}$: Set of workers

$\mathcal{T}$: Set of tasks

σ: Reward scheme (general or proportional)

Let $\mathcal{M}$ be a matching between $\mathcal{W}$ and $\mathcal{R}$

Initialize$(\mathcal{W}, \mathcal{T}, \mathcal{M})$

Stack.push$(\mathcal{W})$

while *Stack is not empty* **do**

  $w \leftarrow$ *Stack.pop()*

  if $index_w \leq |L_w|$ **then**

    $t \leftarrow (index_w)$th task in L_w

    $index_w = index_w + 1$

    $\mathcal{M}(w) = t$, $\mathcal{M}(t) = \mathcal{M}(t) \cup \{w\}$

    if σ *is general* **then**

      $\mathcal{R} \leftarrow$ *WorkerSelection*$(t, w, \mathcal{M})$

    else

      $\mathcal{R} \leftarrow$ *WorkerSelectionProportional*$(t, w, \mathcal{M})$

    foreach $w' \in \mathcal{R}$ **do**

      $\mathcal{M}(t) \leftarrow \mathcal{M}(t) \setminus w'$, $\mathcal{M}(w') = \emptyset$

      Stack.push(w')

return $\mathcal{M}$

of each worker w[5] (i.e., tasks in non-increasing order of profits they will provide to

[5]A worker can also form his preference list himself and submit only this list to the platform, if he does not like to disclose his cost (or profit) for each task.

worker w), and initializes the matching and the other required variables. Throughout its execution, our algorithm maintains a stack that consists of the workers that are unmatched and whose preference lists have not been entirely traversed by the algorithm (i.e., $index_w \leq |L_w|$). In each iteration of the while loop, it pops one (w) of these workers from the stack and attempts to assign him to the next task (t) in his preference list. Although the matching is temporarily updated by assigning worker w to task t (line 9), the actual decision of acceptance is made by calling Algorithm 7 or Algorithm 8 (which are described later in this section) according to the adopted reward scheme. These algorithms return the workers ($\mathcal{R}$) that are removed from the current assignment set of task t. Then, the algorithm sets these workers free again and pushes them back onto the stack (lines 14-16).

Below, we present the performance guarantees of the algorithm with both general and proportional reward schemes by leveraging the analogy between the worker selection step of the algorithm (lines 11 and 13) and the online budgeted maximum coverage *(OBMC)* problem. To this end, we first give a brief description of this problem.

OBMC problem: Assume that a predefined budget B and a universal set $\mathcal{U} = \{u_1, u_2, \ldots, u_{\hat{p}}\}$ with associated weights $\{\hat{v}_i : i = 1, \ldots, \hat{p}\}$ are given. In each iteration i, a set $S_i \subseteq \mathcal{U}$ with a cost of c_i is introduced in an online manner, and the objective is to maximize the weighted coverage over $\mathcal{U}$ within the budget constraint by keeping a certain subset of the introduced sets $\mathcal{S} = \{S_1, S_2, \ldots, S_{\hat{n}}\}$. However, the budget limit cannot be exceeded at any time (i.e., the total cost of the retained sets should always be less than B), and a set that has been rejected/preempted at some point cannot be included in the solution later.

Theorem 7. *(Rawitz and Rosen [84]) There is a* $\frac{4}{1-r}$*-competitive online deterministic*

algorithm for the OBMC problem, where $r = \max_{S_i \in \mathcal{S}} \frac{c_i}{B}$.

4.3.1 General Reward Scheme

We first describe the task selection mechanism for the general reward scheme by giving a pseudo-code description in Algorithm 7. This algorithm is adapted from the OBMC algorithm mentioned in Theorem 7 to our setting (also optimized for running time, which was not a primary concern in [84])). It accepts a new set if the ratio of the additional utility (i.e., weighted coverage) the set will provide to its cost is larger than 2 times the ratio of the total utility to the total cost in the current solution (line 6). If accepting the set violates the budget constraint, the sets in the current solution are discarded one by one in non-decreasing order of efficiency (utility provided per cost) until the budget constraint is satisfied. Another unique aspect of this algorithm is that when it calculates the total utility in the current solution, it also accounts for the distinct utility that was provided by the set that was discarded the latest as if it had been partially kept in the solution. For each task t, the workers in this imaginary solution are stored in H_t, the fraction of C_t^w used in the imaginary solution is stored in $x_t(w)$, the efficiency of a worker w in H_t is stored in $\eta_t(w)$, and the fraction of a PoI p covered by C_t^w is stored in $z_t(p, w)$. Interested readers are referred to [84] for more detailed descriptions of these variables.

Theorem 8. *Algorithm 6 always produces* $\frac{4}{1-\rho}$*-stable matchings in an MCS system with a general reward scheme, where*

$$\rho = \max_{w \in \mathcal{W}, t \in \mathcal{T}} (\frac{r_t(w)}{b_t}). \tag{4.20}$$

Proof. We prove this by contradiction. Assume that there is an unhappy coalition $\langle \hat{S}, \hat{t} \rangle$ that prevents the final matching produced by the algorithm from being a $\frac{4}{1-\rho}$-stable matching. Thus, based on Definition 12, there must be a set $S' \subseteq \mathcal{M}(\hat{t})$ such

Algorithm 7: WorkerSelection (t, w, $\mathcal{M}$)

Input: t, w, $\mathcal{M}$: task evaluating worker w, candidate worker, current matching

$x_t(w) = 1$

foreach $p \in C_t^w$ **do**

  $z_t(p, w) = 1 - \sum_{w' \in H_t} z_t(p, w')$

  $\eta_t(w) = \eta_t(w) + z_t(p, w) v_t(p)$

$\eta_t(w) = \eta_t(w) \times b_t / r_t(w)$

if $\eta_t(w) \leq 2 \times \sum_{p \in P(t)} \sum_{w' \in H_t} z_t(p, w') v_t(p)$ **then return** $\{w\}$

insert w to H_t by maintaining the order, i.e., $H_t = \{\hat{w}_1, \hat{w}_2, .., \hat{w}_q\}$ s.t. $\eta_t(\hat{w}_i) \geq \eta_t(\hat{w}_{i+1}), \forall i < q$

$k \leftarrow \max\{k' \leq |H_t| : \gamma = \sum_{i=1}^{k-1} r_t(w) < b_t\}$

$b_t^{\mathcal{M}} = b_t - \gamma$

$w' \leftarrow k$th worker in H_t

$\beta = \min\{b_t^{\mathcal{M}} / r_t(w'), 1\}$

foreach $p \in C_t^{w'}$ **do** $z_t(p, w') = \frac{\beta}{x_t(w')} z_t(p, w')$

$\mathcal{R} = \emptyset$

if $x_t(w') = 1$ *and* $\beta < 1$ **then** $\mathcal{R} \leftarrow \{w'\}$

$x_t(w') = \beta$

for $i \leftarrow |H_t|$ *down to* $k + 1$ **do**

  $\hat{w} \leftarrow i$th worker in H_t

  H_t.remove($\hat{w}$)

  if $x_t(\hat{w}) = 1$ **then** $\mathcal{R} \leftarrow \mathcal{R} \cup \hat{w}$

return $\mathcal{R}$

that

$$U_{\hat{t}}(\hat{S} \cup (\mathcal{M}(\hat{t}) \setminus S')) > \frac{4 \times U_{\hat{t}}(\mathcal{M}(\hat{t}))}{1-\rho} \quad \text{and } r_{\hat{t}}(\hat{S}) \leq b_{\hat{t}}^{\mathcal{M}} + r_{\hat{t}}(S'). \tag{4.21}$$

Let $\overline{\mathcal{W}} = \{\overline{w}_1, \overline{w}_2, \ldots, \overline{w}_l\}$ be the set of workers that have been matched to task $\hat{t}$ at some point during the course of the execution of Algorithm 6. The corresponding coverage sets of these workers are $\overline{\mathcal{C}}_{\hat{t}} = \{C_{\hat{t}}^{\overline{w}_i} : 1 \leq i \leq l\}$, and the rewards offered to these workers by task $\hat{t}$ are given as $\overline{\mathcal{R}}_{\hat{t}} = \{r_{\hat{t}}(\overline{w}_i) : 1 \leq i \leq l\}$.

Note also that the algorithm attempts to match a single worker-task pair at a time, and if a worker w is first matched with a task t (line 9) and then removed from $\mathcal{M}(t)$ at some point (lines 14-16), the algorithm will never attempt to match him with task t afterwards, instead it will try to match him with other tasks which come after task t in his preference list L_w. Thus, from a task's perspective, say task $\hat{t}$, this is exactly the same problem with the OBMC problem, as we have a collection $\overline{\mathcal{C}}_{\hat{t}}$ of sets with associated costs $\overline{\mathcal{R}}_{\hat{t}}$ that arrive one at a time, and that cannot be later included to the solution $\mathcal{M}(\hat{t})$ after they are discarded, and the goal of task $\hat{t}$ is also to maximize the weighted coverage within the budget. In fact, we can map the two problems to each other as follows:

$$\begin{aligned} \mathcal{U} &\longleftrightarrow P(\hat{t}), \\ \mathcal{S} &\longleftrightarrow \overline{\mathcal{C}}_{\hat{t}}, \\ S_i &\longleftrightarrow C_{\hat{t}}^{\overline{w}_i}, \\ c_i &\longleftrightarrow r_{\hat{t}}(\overline{w}_i), \\ B &\longleftrightarrow b_{\hat{t}}. \end{aligned} \tag{4.22}$$

For this reason, Algorithm 6 runs the adapted version of the OBMC algorithm (i.e.,

Algorithm 7) as a subroutine (line 11) to decide which workers to keep in the worker set of a task after the algorithm attempts to assign another worker to her. Then, by Theorem 7, we have that

$$U_{\hat{t}}(\mathcal{S}_{best}) < \frac{4 \times U_{\hat{t}}(\mathcal{M}(\hat{t}))}{1 - \rho_{\hat{t}}} \tag{4.23}$$

where $\mathcal{S}_{best} \subseteq \overline{W}$ is the best set that could be assigned to task $\hat{t}$ providing the highest total weighted coverage within the budget constraint (i.e., an optimal solution of the corresponding online budgeted maximum coverage problem), and $\rho_{\hat{t}} = \max_{w \in \overline{\mathcal{W}}} \frac{r_{\hat{t}}(w)}{b_{\hat{t}}}$.

Note that $\hat{S}$ cannot include a worker that is not in $\overline{\mathcal{W}}$, thus we have

$$\left(\hat{S} \cup (\mathcal{M}(\hat{t}) \setminus S')\right) \subseteq \overline{\mathcal{W}}. \tag{4.24}$$

That is because, by Definition 10, all workers in $\hat{S}$ must be preferring task $\hat{t}$ to the tasks they are currently assigned. However, if a worker w is currently matched with task t', the algorithm should have attempted to assign him all the other tasks that precede task t' in his preference list. Then, if worker w prefers task $\hat{t}$ to task t', which means that task $\hat{t}$ also precedes task t' in his preference list, we must have $w \in \overline{\mathcal{W}}$.

Due to (4.24) and the fact that $\mathcal{S}_{best}$ is the best feasible set in $\overline{\mathcal{W}}$ for task $\hat{t}$, we have

$$U_{\hat{t}}(\mathcal{S}_{best}) \geq U_{\hat{t}}(\hat{S} \cup (\mathcal{M}(\hat{t}) \setminus S')). \tag{4.25}$$

Then, combining the inequalities (4.21), (4.23) and (4.25), we get

$$\frac{4 \times U_{\hat{t}}(\mathcal{M}(\hat{t}))}{1 - \rho_{\hat{t}}} > \frac{4 \times U_{\hat{t}}(\mathcal{M}(\hat{t}))}{1 - \rho} \tag{4.26}$$

which is a contradiction as $\rho \geq \rho_{\hat{t}}$. □

Algorithm 8: WorkerSelectionProportional (t,w,$\mathcal{M}$)

Input: t: Task evaluating worker w

w: Candidate worker

$\mathcal{M}$: Current matching

1 **if** $\mathcal{A}_t$ *is false and* $r_t(w) \geq 0.2 \times b_t$ **then**

2 $\quad \mathcal{A}_t \leftarrow true$

3 $\quad \mu \leftarrow \mathcal{M}(t)$

4 $\quad \mathcal{M}(t) \leftarrow \{w\}$

5 $\quad b_t = b_t - r_t(w)$

6 $\quad$ **foreach** $w' \in \mathcal{W} \setminus \{w\}$ **do**

7 $\quad\quad C_t^{w'} \leftarrow C_t^{w'} \setminus C_t^{w}$

8 $\quad H_t \leftarrow \emptyset$

9 $\quad \mathcal{R} \leftarrow \emptyset$

10 $\quad$ **foreach** $w' \in \mu$ **do**

11 $\quad\quad \mathcal{M}(t) = \mathcal{M}(t) \cup \{w'\}$

12 $\quad\quad \eta_t(w') = 0$

13 $\quad\quad R \leftarrow R \cup \text{WorkerSelection}(t, w', \mathcal{M})$

14 $\quad$ **return** $\mathcal{R}$

15 **else**

16 $\quad$ **return** *WorkerSelection*$(t, w, \mathcal{M})$

4.3.2 Proportional Reward Scheme

We propose Algorithm 8 for the proportional reward scheme. It directly runs Algorithm 7 (line 16) for task t until the main algorithm attempts to assign her a worker w that satisfies $r_t(w) \geq 0.2 \times b_t$. When this happens, the algorithm finalizes the

assignment of worker w to task t (i.e., in the end, they will be matched to each other), and updates the budget of task t (line 5) and coverage sets of workers (lines 6-7) to reflect the fact that a certain proportion of task t's budget is not available anymore, and that the utility of the other workers should be computed considering only the PoIs that are not covered by worker w. It also attempts to re-assign the previous worker set of task t to her considering the modified budget and coverage sets (lines 10-14). In the subsequent iterations in which the main algorithm attempts to assign another worker to task t, since $\mathcal{A}_t$ is previously set to *true* (line 2), the algorithm continues to run Algorithm 7 with the modified budget of task t and coverage sets of workers (line 16).

Theorem 9. *Algorithm 6 always produces* 5*-stable matchings in the presence of a proportional reward scheme.*

Proof. We prove it by contradiction. Let $\mathcal{M}$ be the returned matching by the algorithm, and $\langle \hat{S}, \hat{t} \rangle$ be an unhappy coalition in $\mathcal{M}$ that, for some $S' \subseteq \mathcal{M}(\hat{t})$, satisfies

$$U_{\hat{t}}(\hat{S} \cup (\mathcal{M}(\hat{t}) \setminus S')) > 5 \times U_{\hat{t}}(\mathcal{M}(\hat{t})) \tag{4.27}$$
$$\text{and } r_{\hat{t}}(\hat{S}) \leq b_{\hat{t}}^{\mathcal{M}} + r_{\hat{t}}(S').$$

Thus, $\langle \hat{S}, \hat{t} \rangle$ prevents $\mathcal{M}$ from being a 5-stable matching according to Definition 12.

If $\mathcal{A}_{\hat{t}}$ is true in the end, then $\mathcal{M}(\hat{t})$ includes a worker w such that

$$r_{\hat{t}}(w) \geq 0.2 \times b_{\hat{t}}. \tag{4.28}$$

Since the utility $U_{\hat{t}}(w)$ of worker w is proportional to his reward $r_{\hat{t}}(w)$, we have

$$U_{\hat{t}}(\mathcal{M}(\hat{t})) \geq U_{\hat{t}}(w) = \theta_t \times r_{\hat{t}}(w) \tag{4.29}$$

Also, given the budget limit of task $\hat{t}$, the total weighted coverage that the best

feasible worker set can provide for task $\hat{t}$ is at most

$$U_{max} = \theta_t \times b_{\hat{t}}. \tag{4.30}$$

Then, by (4.28), (4.29) and (4.30), we get

$$5 \times U_{\hat{t}}(\mathcal{M}(\hat{t})) \geq U_{max} \tag{4.31}$$

$$\geq U_{\hat{t}}(\hat{S} \cup (\mathcal{M}(\hat{t}) \setminus S')) \tag{4.32}$$

which contradicts (4.27).

On the other hand, if $\mathcal{A}_{\hat{t}}$ is false in the end, then only Algorithm 7 has been run for task t, similar to the case with the general reward scheme, thus the inequality (4.23) must hold here as well. Since the inside of the if block in Algorithm 8 is never executed, we have $r_{\hat{t}}(w) < 0.2 \times b_{\hat{t}}$, for all $w \in \overline{\mathcal{W}}$. Then, (4.23) becomes

$$U_{\hat{t}}(\mathcal{S}_{best}) < 5 \times U_{\hat{t}}(\mathcal{M}(\hat{t})) \tag{4.33}$$

Following the same steps ((4.24) and (4.25)) in the proof of Theorem 8, we obtain

$$5 \times U_{\hat{t}}(\mathcal{M}(\hat{t})) > 5 \times U_{\hat{t}}(\mathcal{M}(\hat{t})) \tag{4.34}$$

which is also false and completes the proof. □

4.3.3 Feasibility, Rationality and Efficiency Analysis

We lastly show that the proposed algorithms always produce individually rational and feasible matchings, and analyze their asymptotic running times.

Theorem 10. *Algorithm 6 always produces individually rational and feasible matchings.*

Proof. Note that a worker can only get matched with a task in his preference list

(line 7), and matching with any of the tasks in his preference list is profitable for him since those that are not so are removed from his preference list in Algorithm 5 (line 3). Thus, we conclude that the produced matchings are individually rational. It is clear from lines 9 & 15 of Algorithm 6 that the produced matchings are feasible in terms of mutual partnership. As for the budget feasibility, when the reward scheme is general, Algorithm 7 returns (line 8) the set of the least efficient workers that need to be removed from the worker set of task t to stay within the budget constraint b_t, which are then actually removed from the worker set of task t in lines 14-16 of Algorithm 6. When the reward scheme is proportional, Algorithm 8 either only runs Algorithm 7 (line 16), or executes the inside of the if block beginning in line 1 at most once for each task t, where the budget of task t is decreased (line 5) by the reward amount that will be paid to the accepted worker w. After that, it always runs Algorithm 7 for task t. Therefore, the produced matchings are also feasible. □

We note that similar to the PSTA algorithm of Chapter 3, Algorithm 6 can also be run in a distributed manner [79], because in each iteration of the main while loop, the matching is updated according to the preference relation between a single worker-task pair (w, t), so the changes in the matching can be communicated by this worker-task pair to the other users that get affected by these changes (i.e., workers in the set $\mathcal{R}$ computed in line 11 or 13 based on the reward scheme).

Running time. The initialization (i.e., running Algorithm 5) takes $O(knm + mn\log(n))$, where the latter term is due to sorting $\mathcal{T}$ for each worker. In Algorithm 6, each worker w is pushed onto the stack at most $|L_w| \leq \mathcal{T}$ times, thus the while loop iterates at most nm times. The costliest operations in the while loop for the *general* and *proportional* reward schemes are running Algorithm 7 (line 11) and Algorithm 8 (line 13), respectively. Algorithm 7 runs in $O(km)$ time. Since the inside of the if

block in Algorithm 8 will be run at most once and hence Algorithm 7 will be called at most 2 times for each worker-task pair (from lines 13 and 16 in Algorithm 8), the amortized cost of Algorithm 8 also becomes $O(km)$. Therefore, the worst-case running time of our approximation algorithm is $O(knm^2 + mn\log(n))$ regardless of the reward scheme.

4.4 Evaluation

In this section, we present an extensive evaluation of the proposed algorithms in MCS systems with both general and proportional reward schemes.

4.4.1 Simulation Settings

Similar to [8], we utilize two real data sets [85, 86] that consist of the trajectories of 39 and 92 participants from New York City (NYC) and the campus of the Korea Advanced Institute of Science and Technology (KAIST), respectively. We create 300 PoIs at random locations that are on the trajectory (i.e., within 50 meters) of at least one participant. Fig. 16 and 17 show the trajectories in the data sets and an example of PoI distribution. Note that the way we determine PoI locations ensures that PoIs are, as it can be seen in both figures, mostly in hot spots that are visited frequently by participants as we would expect to see in a real-world scenario.

We let the trajectories in the data sets to be the trajectories of the workers in our system. To look at the impact of the number of workers, we use random-sampling to obtain a worker set of certain size. According to the experiment requirements, we create n tasks whose budgets are assigned randomly from [10, 100]. In order to determine the PoI list of a task t, we first select a random number s from [1, 25]. Then, we randomly insert s of the all PoIs to $P(t)$ after assigning a random weight value from (0, 1]. Since we utilize deterministic trajectories, the lists of PoIs to be visited

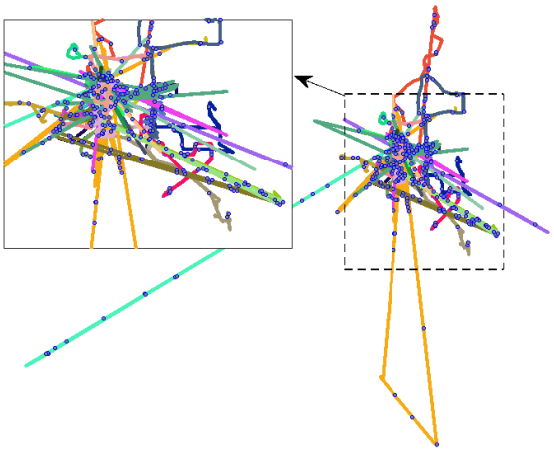


Fig. 16. Trajectories of the workers in the KAIST data set (circles denote the PoIs).

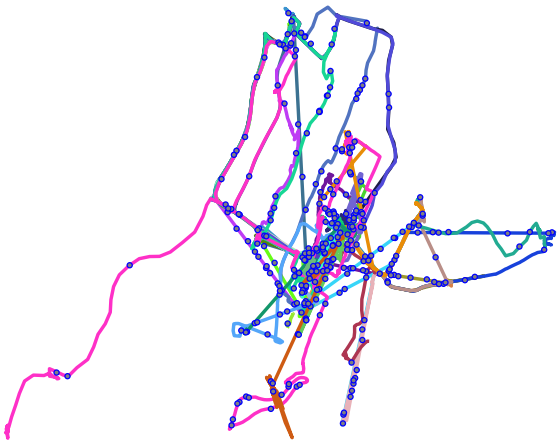


Fig. 17. Trajectories of the workers in the NYC data set (circles denote the PoIs).

by workers and the tasks they can complete are known in advance. The assignment of the rewards for different reward schemes is made as follows:

- **General reward scheme:** For each worker-task pair (w, t), we assign the reward $r_t(w)$ randomly from the range $[0.05 \times b_t,\ 0.95 \times b_t]$. If $C_t^w = \emptyset$, we set

$r_t(w) = 0$.

- **Proportional reward scheme:** For each worker-task pair (w, t), the reward is set as

$$r_t(w) = b_t \times \frac{\sum_{p \in C_t^w} v_t(p)}{\sum_{p \in P(t)} v_t(p)}. \tag{4.35}$$

Since the rewards are already determined based on the randomly assigned budget values, we let $c_t(w) = 0$ for all worker-task pairs (w, t)[6].

4.4.1.1 Benchmark Algorithms

In the simulations, we let CSTA$_G$ and CSTA$_P$ denote the execution of the proposed *C*overage-aware *S*table *T*ask *A*ssignment algorithm with general and proportional reward schemes, respectively. We compare the performance of these algorithms with that of Maximum Coverage Quality Assignment (or *MCQA*) algorithm proposed in [8] and *Greedy* algorithm proposed in [9] for the problem of finding the worker set with the maximum total weighted coverage over a given set of PoIs. They both are originally proposed for the MCS systems with only a single task requester, m workers, and k PoIs. The MCQA algorithm has an approximation ratio of $(1 - 1/e)$ for the aforementioned optimization problem and a time complexity of $O(kn^5)$. On the other hand, the Greedy algorithm does not have a theoretical performance guarantee and runs in $O(knm^2)$. We adapt them to our settings with multiple task requesters as follows. For each task t in the system, we first find the set S of workers that, among all the tasks in the system, prefer task t the most. Then, we separately run the MCQA/Greedy algorithm for each such (t, S) pair with $P(t)$ being the set of

[6]Introducing extra, random cost values naturally reduces the number of qualified pairs and consequently the average coverage quality, but it does not have a notable effect on the relative performance of the algorithms.

PoIs, and finalize the assignments made in each run. Lastly, for each worker w that is still unmatched, we traverse his preference list L_w from the beginning, and match him with the first task that benefits from hiring him (i.e., worker w increases the coverage quality of task t) and has sufficient budget to do so. These adapted versions are denoted by *MCQA** and *Greedy** in the simulations.

4.4.1.2 Performance Metrics

Here, we introduce the performance metrics that will be used in the evaluation of the results.

- *Stability success ratio (%)*: This metric shows how often an algorithm achieves the best known upper-bound in terms of stability in different settings. Specifically, let $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_{100}$ be the matchings produced by an algorithm $\mathcal{A}$ in 100 consecutive runs on different MCS instances. Also let

$$s(\mathcal{M}_i, \alpha) = \begin{cases} 1, & \text{if } \mathcal{M}_i \text{ is an } \alpha\text{-stable matching,} \\ 0, & \text{otherwise.} \end{cases} \tag{4.36}$$

  Then, the stability success ratio of $\mathcal{A}$ is calculated by

$$\sum_{i=1}^{100} s(\mathcal{M}_i, \frac{4}{1-\rho}) \tag{4.37}$$

  for the MCS systems with a general reward scheme, and by

$$\sum_{i=1}^{100} s(\mathcal{M}_i, 5) \tag{4.38}$$

  for the MCS systems with a proportional reward scheme.

- *User happiness (%)*: This quantifies the user happiness based on the stability of

the matching as follows:

$$100 \times \left(1 - \frac{\text{\# of coalitionally unhappy pairs}}{\text{\# of qualified pairs}}\right) \tag{4.39}$$

- *Stability (σ)*: This is the value of the objective function defined in (4.18), and indicates that the produced matching is $(1/\sigma)$-stable. If there is not any unhappy coalition in the matching, then $\sigma = 1$ (by definition of δ_t).

- *Average coverage quality (%)*: This is the average weighted coverage that the produced matching, $\mathcal{M}$, provides for the tasks, or formally:

$$\frac{100}{n} \times \sum_{t \in \mathcal{T}} \frac{U_t(\mathcal{M}(t))}{\sum_{p \in P(t)} v_t(p)}. \tag{4.40}$$

- *Running time*: In order to show the scalability of the algorithms, we also present their running times with increasing number of workers/tasks/PoIs on an Intel core i7 processor with 16 GB memory and 2.5 GHz speed.

Lastly, we note that all results provided in this section are the average of the results obtained in 100 runs (each with a different MCS instance).

4.4.2 Results

We first analyze the results for the KAIST data set. Fig. 18 & 19 show the impact of the number of tasks n on the performance of the algorithms with general and proportional reward schemes, respectively. First, note that the performances of the MCQA* and Greedy* algorithms in terms of stability (Fig. 18b and Fig. 19b) deteriorate significantly as n increases. This is due to the fact that these algorithms do not consider the system as a whole, and aim to maximize the coverage for each task separately. However, since they optimize the assignments for individual tasks

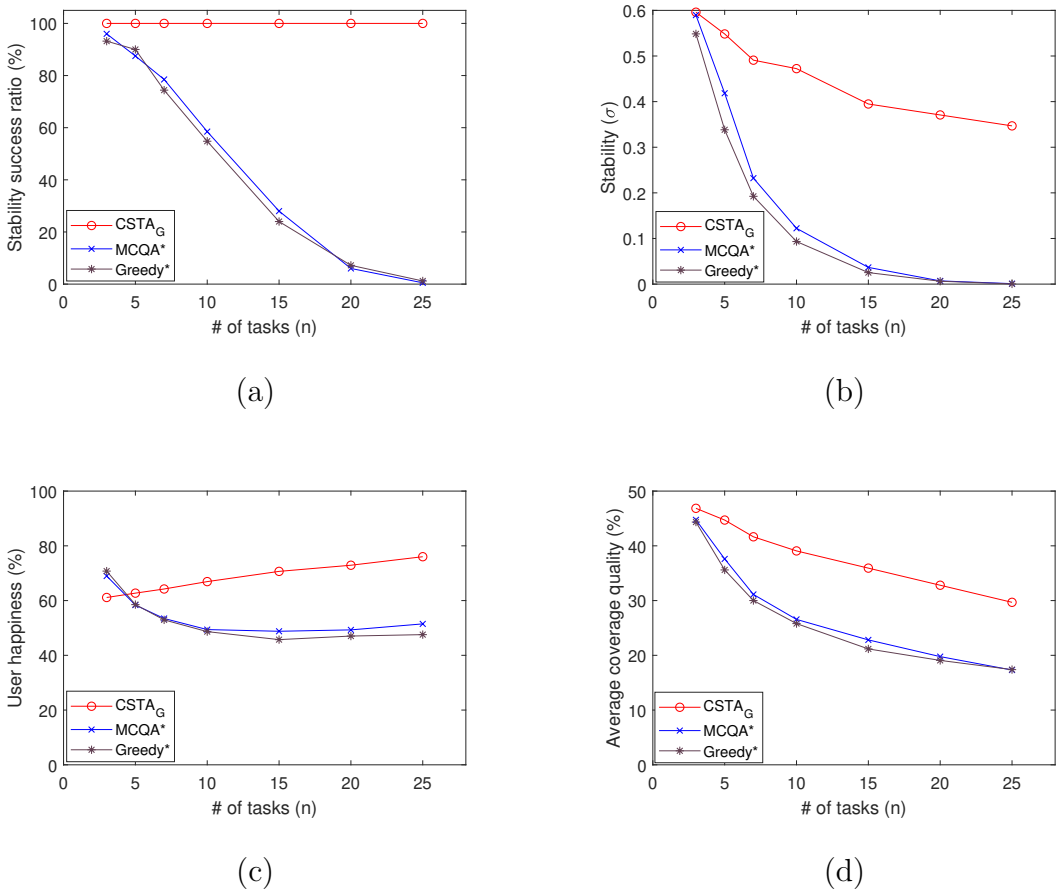


Fig. 18. General setting: performance comparison against varying number of tasks in the KAIST data set ($m = 92$).

extensively (which, in turn, increases their running time significantly as can be seen in Fig. 26), they outperform the other algorithms when n is small in terms of user happiness (Fig. 18c and Fig. 19c) and average coverage quality (Fig. 19d).

In terms of stability success ratio, the CSTA$_G$ and CSTA$_P$ algorithms produce perfect task assignments in the general and proportional settings, respectively, as expected (due to Theorem 8 & 9), and vastly outperform the other algorithms. We see that the CSTA$_G$ algorithm occasionally fails to produce perfect assignments in terms of stability success ratio in the proportional setting, which indicates that assigning a

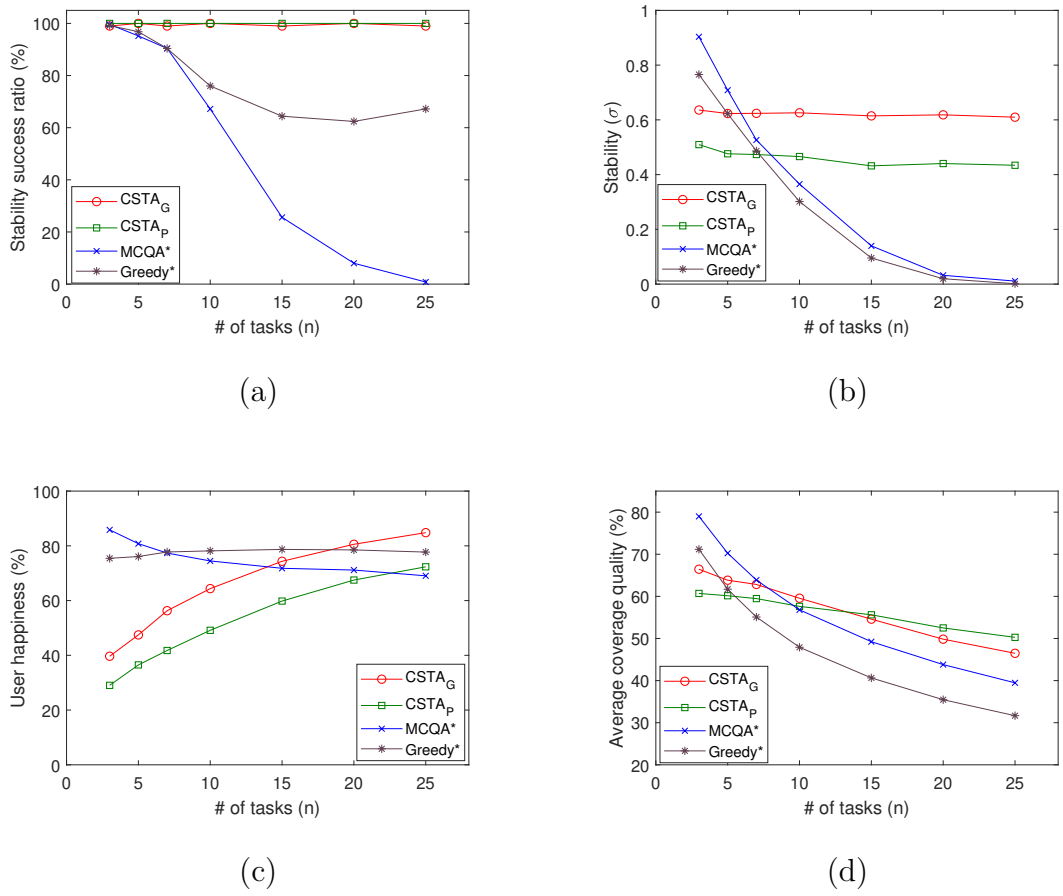


Fig. 19. Proportional setting: performance comparison against varying number of tasks in the KAIST data set ($m = 92$).

task t with the first worker w such that $r_t(w) \geq 0.2 \times b_t$ (lines 1-14 in the CSTA$_P$ algorithm) is required to achieve 5-stable matchings. Yet, this comes with a trade-off as the CSTA$_P$ algorithm yields significantly lower stability scores (σ) compared to the CSTA$_G$ algorithm as seen in Fig. 19b. Since the Greedy* algorithm selects workers according to the ratio of how much utility they will bring for the tasks to the reward they will be paid, its performance is much better than the MCQA* algorithm in the proportional setting where the value of the proposed reward per utility is constant for all workers.

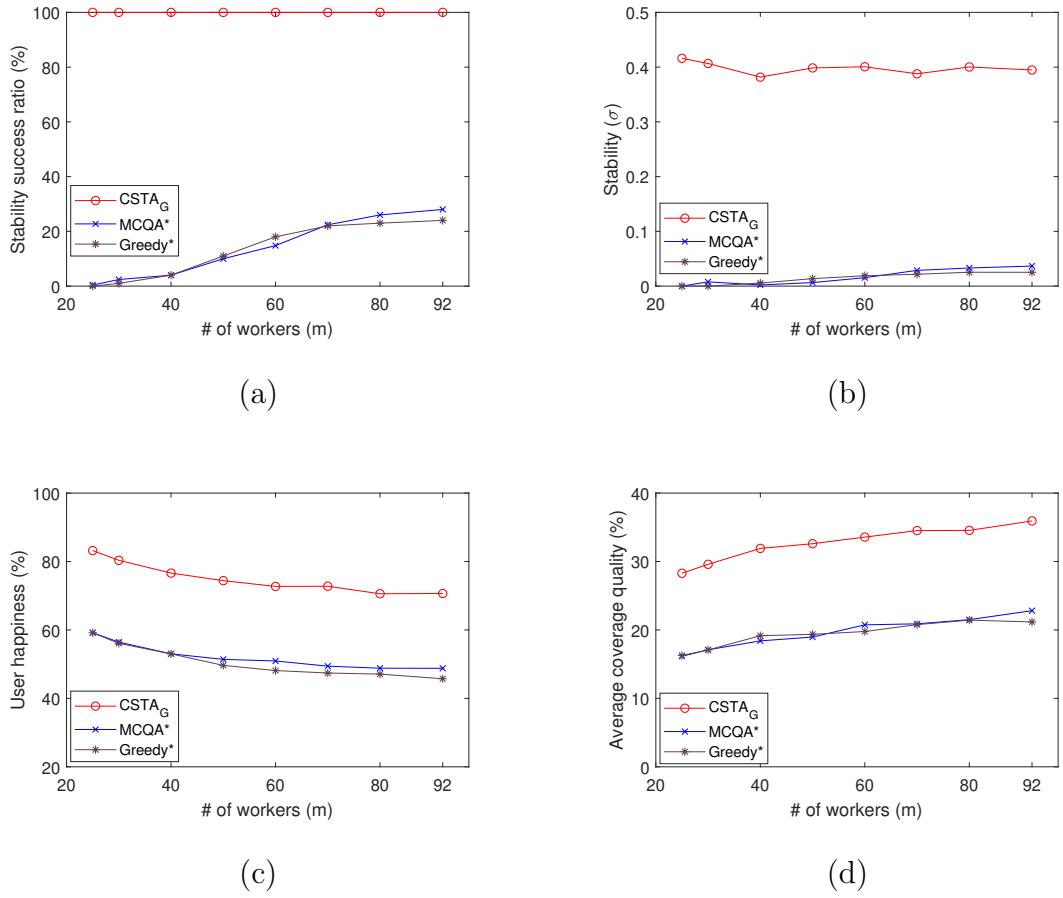


Fig. 20. General setting: performance comparison against varying number of workers in the KAIST data set ($n = 15$).

In the average coverage quality graphs (Fig. 18d and 19d), we see that the average coverage decreases for all algorithms with increasing n as there will be fewer workers assigned to each task. We also see that the coverage scores in the proportional setting are remarkably larger than those in the general setting mainly because of the discrepancy between reward and utility values in the latter setting (i.e., a high reward does not indicate a high utility for tasks, unlike the proportional setting). It is also noteworthy that in terms of coverage, the proposed algorithms mostly outperform the MCQA* and Greedy* algorithms, whose sole objective is to maximize the coverage.

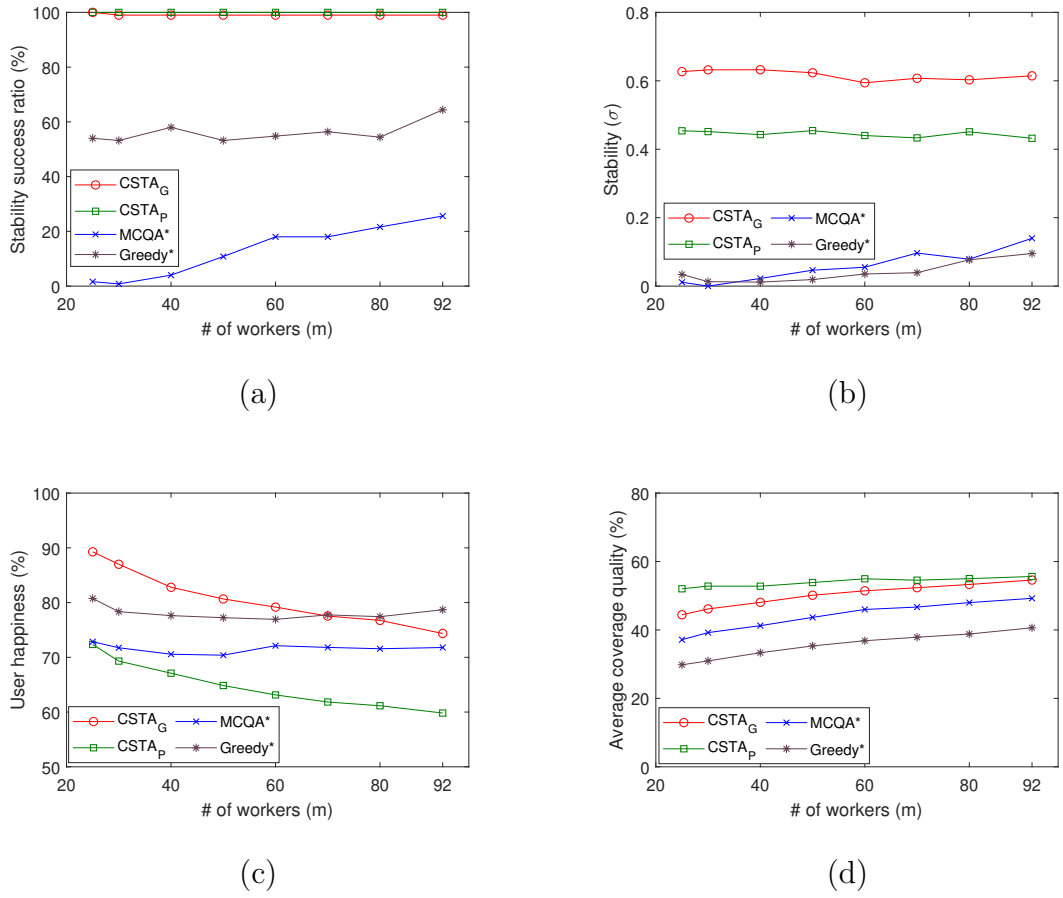


Fig. 21. Proportional setting: performance comparison against varying number of workers in the KAIST data set ($n = 15$).

This demonstrates that taking user preferences into account does not necessarily yield less efficient assignments in terms of system-level utility metrics such as coverage.

Next, we look at the performance of the algorithms with varying number of workers in Fig. 20 & 21. Except for the user happiness results, we observe that *increasing* the number of workers m has a similar impact on the performance of the MCQA* algorithm with *decreasing* the number of tasks n. This is because both changes result in a smaller ratio of n to m (i.e., task scarcity), which alleviates the deficiency of the MCQA* algorithm in handling multi-task assignments. This is also

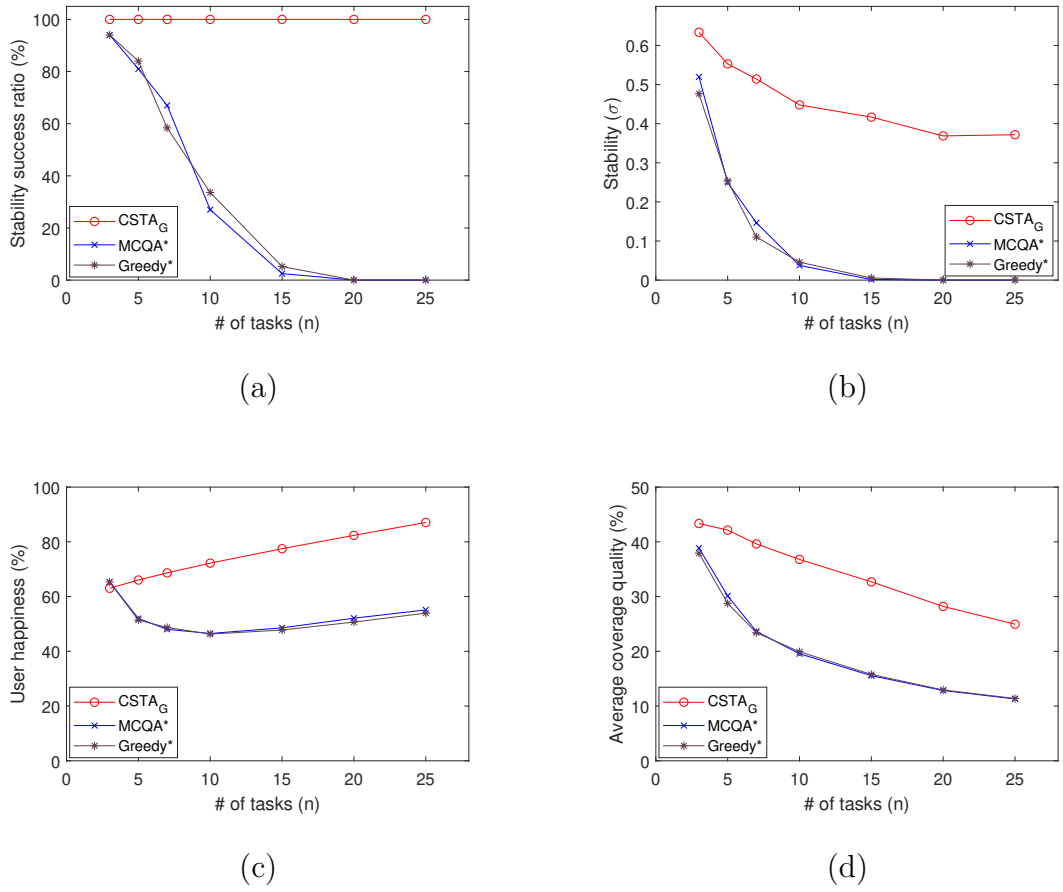


Fig. 22. General setting: performance comparison against varying number of tasks in the NYC data set ($m = 39$).

mostly true for the Greedy* algorithm, however its performance in terms of stability success ratio in the proportional setting is more stable. We note that the changes in the number of tasks or workers do not have a significant impact on the stability and stability success ratio scores of the proposed algorithms in the proportional setting as seen in Fig. 19a-b and 21a-b. Another remarkable point is that the MCQA* and Greedy* algorithms have almost identical performance in the general setting with varying n and m values, yet their performance in the proportional setting is quite different. Specifically, in terms of stability success ratio and user happiness,

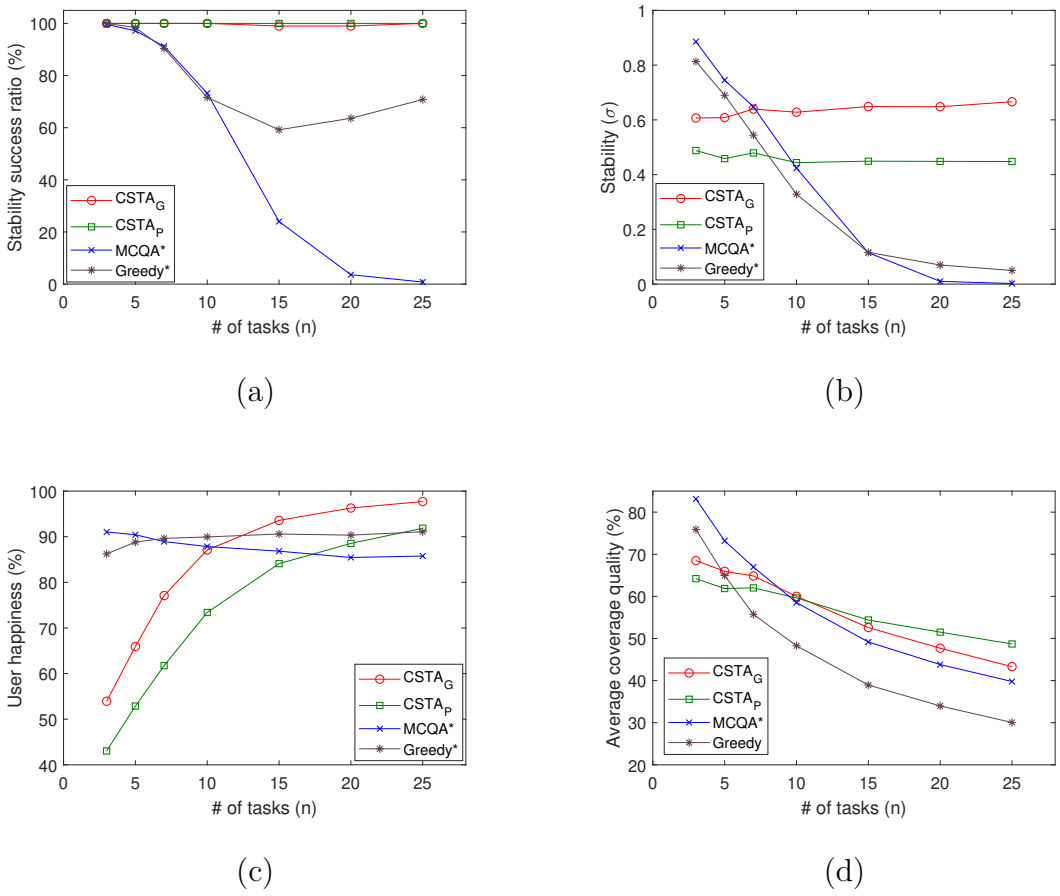


Fig. 23. Proportional setting: performance comparison against varying number of tasks in the NYC data set ($m = 39$).

the Greedy* algorithm mostly outperforms the MCQA* algorithm, while it is the opposite in terms of stability and average coverage quality.

Fig. 20 & 21 show that the CSTA$_G$ algorithm always outperforms the MCQA* algorithm in terms of user happiness (by up to 25%) regardless of the number of workers, but it is slightly outperformed by the Greedy* algorithm when m is larger than 70 in the proportional setting, and that the performance of the proposed algorithms mostly degrades as m increases. We observe that all algorithms achieve higher coverage scores with increasing m values, which is naturally the opposite of what we

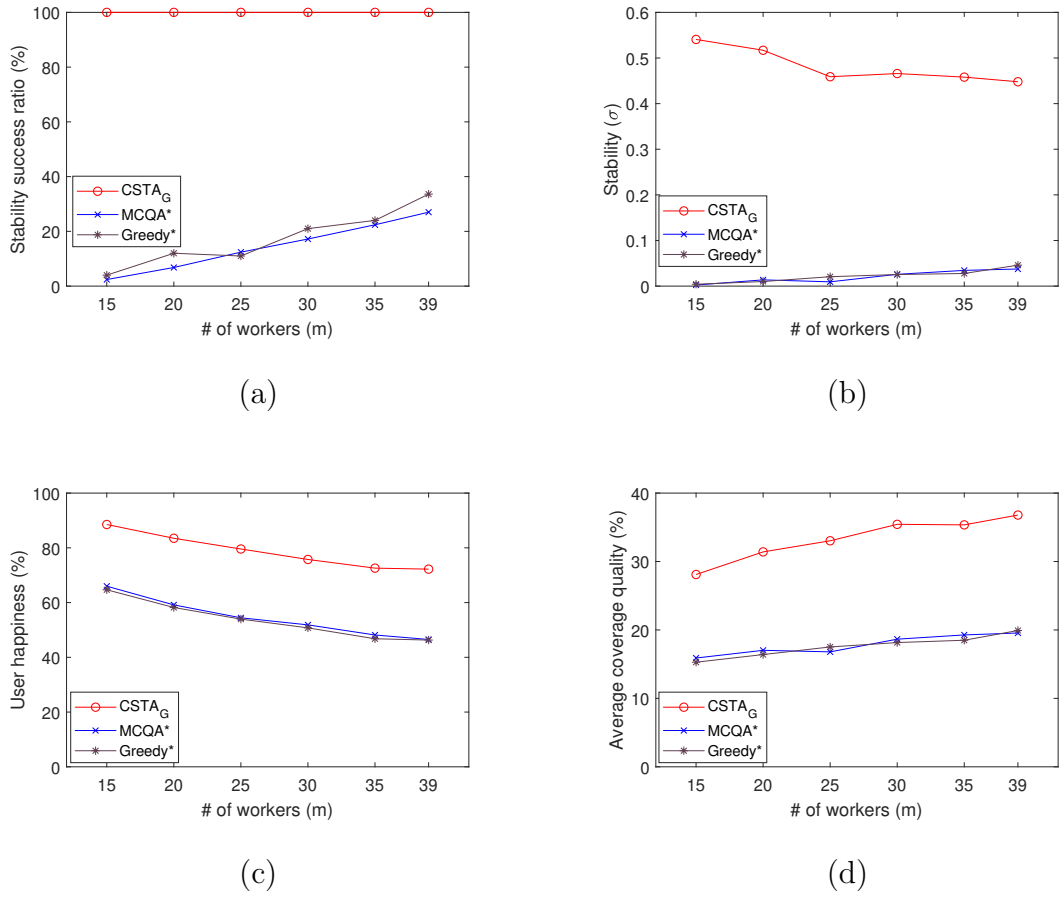


Fig. 24. General setting: performance comparison against varying number of workers in the NYC data set ($n = 10$).

see with increasing n values. This is because if there are more workers per task in the system, the competition between tasks will be less severe, and each task will be assigned to a higher number of workers, on average. However, the budget constraints of the tasks limit the number of workers that can be assigned to them, hence we start to see a smaller or no increase in coverage after some point, especially in the proportional setting. We also note that the stability of the matchings produced by the proposed algorithms is significantly higher than the theoretical upper-bound (0.2) in the proportional settings (Fig. 19b & 21b). Besides, Fig. 20b & 21b demonstrate

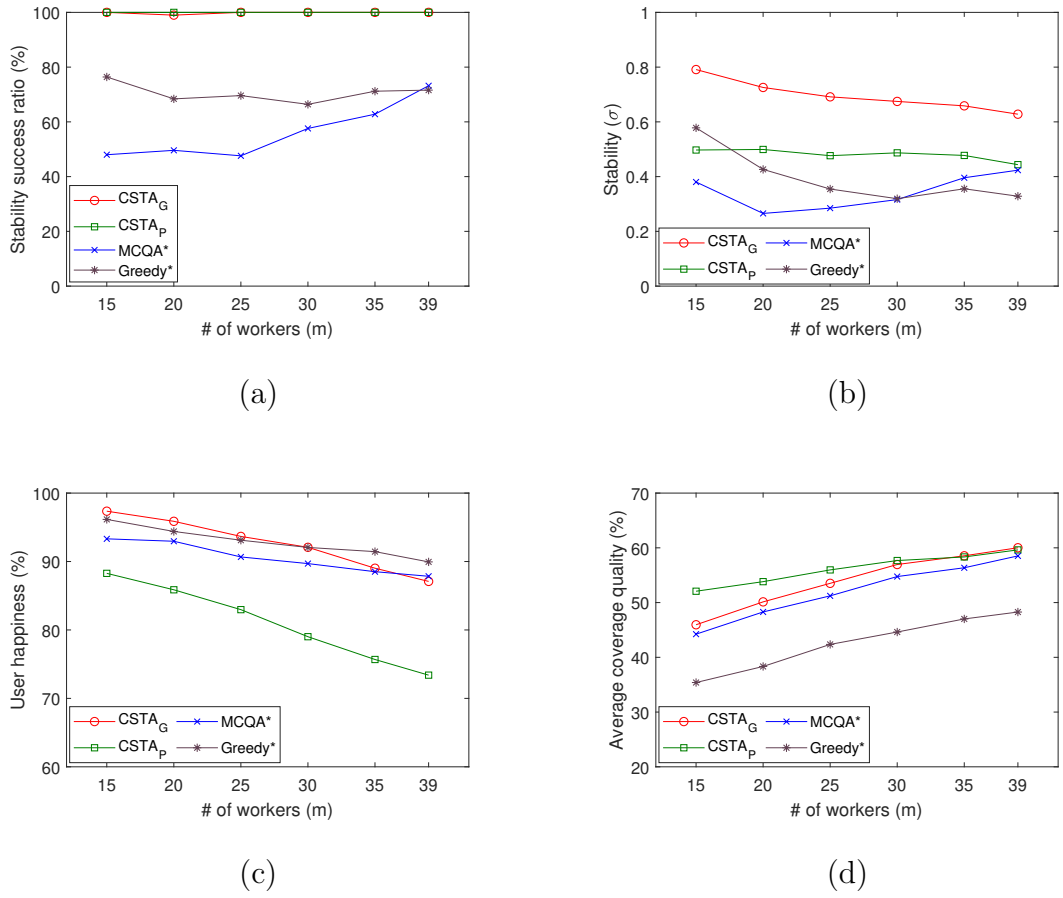


Fig. 25. Proportional setting: performance comparison against varying number of workers in the NYC data set ($n = 10$).

that our algorithms always significantly outperform the MCQA* and Greedy* algorithm in terms of σ. The difference in σ is especially big (up to 0.6) when the ratio of n to m is larger.

In order to demonstrate that the results provided above are not specific to a data set, we also examine the performance of the algorithms in the NYC data set in Fig. 22, 23, 24 & 25. The proposed algorithms in general perform better than the benchmark algorithm as in the KAIST data set. The results in both data sets are similar, thus the majority of our comments above for the KAIST data set also apply

to the results for the NYC data set. However, there are some differences in results we see in the NYC data set. The first significant difference can be seen between Fig. 21c and Fig. 25c. Here, we see that increasing the number of workers continues to improve the achieved coverage in the NYC data set, while there is mostly little to no improvement in the KAIST data set. This is primarily because of the limited number of workers (39) available in the NYC data set. That is, since the tasks still have budget for more workers, adding new workers to the system simply expands the coverage. This can also be partially observed in Fig. 21 up until $m = 60$. Another noteworthy difference is in the user happiness results in proportional setting. All algorithms accomplish better user happiness scores (up to 20%) in the NYC data set compared to those in KAIST data set (i.e., Fig. 19b vs. Fig. 23b and Fig. 21b vs. Fig. 25b). This might be because the trajectories of the workers in the NYC data set are more dispersed than those in the KAIST data set (see Fig. 16 & 17), which, in turn, reduces the overall competition between the tasks as such a difference in the trajectories implies that the PoIs covered by the workers differ more in the NYC data set, and the workers are hence favored by different tasks.

Lastly, we compare the running times of the algorithms in Fig. 26. In order to show the scalability of the algorithms for large numbers of tasks, workers and PoIs, we generated a synthetic data set in a 3,000 m $\times$ 3,000 m area with k randomly located PoIs, m workers whose trajectories are created using the random-walk mobility model (as in [8]) for 2,000 meters (with a direction change at every 200 meters), and n tasks whose PoI sets and budgets are determined exactly as in the real data sets. Since the proportional setting allows us to compare the running times of all algorithms, the rewards are assigned using the proportional reward mechanism (the running times of the MCQA*, Greedy* and CSTA$_G$ algorithms in proportional setting are similar to their running times in the general setting).

Recall that in the MCQA* algorithm, the original MCQA algorithm is run separately for each task t and the set Q_t of workers that prefer task t the most, and the time complexity of each such run is $O(\hat{k}\hat{m}^5)$, where $\hat{k} = |P(t)|$ and $\hat{m} = |Q_t|$. Since fewer tasks in the system means that for each task t, there will be more workers that prefer t the most (i.e., a larger $|Q_t|$), the running time of the MCQA* algorithm increases when n decreases (Fig. 26a) or m increases (Fig. 26b). The time complexity of the original Greedy algorithm is $O(knm^2)$, and its running time decreases with increasing n values up until $n = 25$ due to the same reason (i.e., fewer workers per task). After this point, the second phase of the Greedy* algorithm, which also has a time complexity of $O(knm^2)$ and is where the algorithm tries to match the workers that could not get matched with any tasks during the first phase as proposed in our adaptation, starts to dominate the running time and we begin to see a linear growth. In these figures, we also see that the running times of the CSTA$_G$ and CSTA$_P$ algorithms are mostly a few orders of magnitude smaller than that of the MCQA* algorithm. This is simply because of the superior time complexity of these algorithms: $O(knm^2 + mn\log(n))$. They also run significantly faster than the Greedy* algorithm. Note that while the complexity of proposed algorithms will be the same as the Greedy* algorithm, because $\log(n) \ll km$ for most values used in practice, their actual running times are less than that of the Greedy* algorithm. This is because the preference list of each worker in the proposed algorithms contains only a limited number of tasks as a rational worker will accept only the tasks that request data from some of the PoIs on his trajectory (line 3 of Algorithm 1). So, the number of times workers are pushed onto the stack is generally much smaller than $n \times m$, making $O(knm^2)$ not tight. Finally, we note that the running times of all algorithms increase linearly with the increasing number of PoIs k as seen in Fig. 26c, which is in accordance with the influence of k in their asymptotic running times.

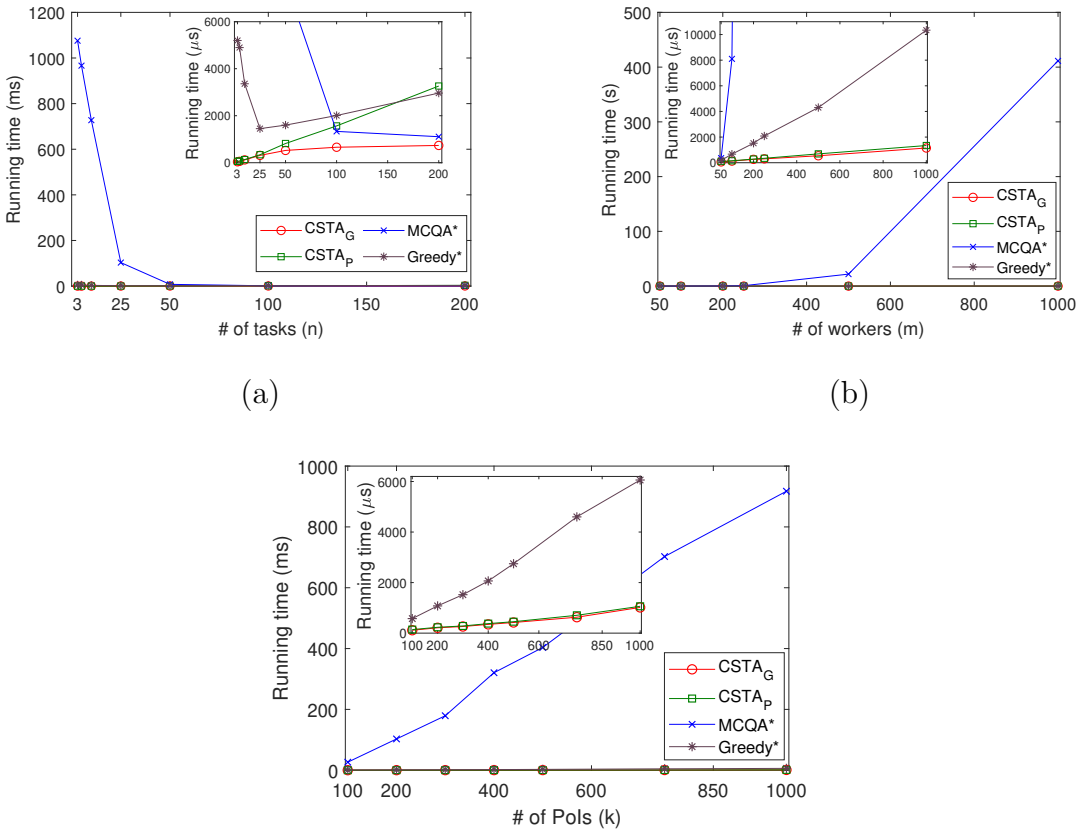


Fig. 26. Running times of the algorithms with varying number of (a) tasks ($m = 200$, $k = 300$); (b) workers ($n = 20$, $k = 300$); and (c) PoIs ($n = 20$, $m = 200$) with the proportional reward scheme in the synthetic data set.

4.5 Conclusion

In this chapter, we studied the problem of finding stable multi-task assignments with weighted coverage-based utility functions in a budget-constrained and opportunistic mobile crowdsensing scenario. We first defined the stability (or user happiness) conditions within this scenario, and pointed out the hardness of the problem and nonexistence of optimal solutions in some cases. We then presented two approximation algorithms and derived their approximation ratios in different settings.

Finally, we provided an extensive evaluation of the proposed algorithms, and showed that they largely outperform the considered benchmark algorithms in terms of both user happiness and coverage quality while having significantly smaller running times (up to 4 orders of magnitude).

CHAPTER 5

ONLINE PREFERENCE-AWARE TASK ASSIGNMENT WITH UNCERTAIN WORKER TRAJECTORIES

5.1 Introduction

In this chapter, we focus on the online task assignment problem in opportunistic MCS. Here, different from the previous two chapters, we consider uncertain and uncontrollable worker trajectories, and simple tasks, which do not have coverage requirements and can be efficiently completed by a single worker. The three key issues that need to be addressed in this problem are (i) preference-awareness, (ii) uncertainty in worker trajectories, and (iii) capacity constraints. Below, we explore each of these issues along with the challenges they present, how they have been so far addressed in the MCS literature, and the key contributions of this study on each of these issues.

(i) *Preference-awareness:* A task assignment strategy is said to be preference-aware if its primary objective is to make all workers and task requesters in the system happy with their assignments based on their individual preferences. A preference-aware mechanism should ensure that the produced assignments are fair and do not sacrifice the utility of a group of workers and task requesters in favor of some others.

(ii) *Uncertainty in worker trajectories:* This issue arises in MCS systems, where workers prefer not to disclose their exact trajectories due to privacy concerns, or their trajectories change dynamically due to traffic/road conditions or individual factors (e.g., a taxi driver's trajectory depends on pick-up and drop-off locations of passen-

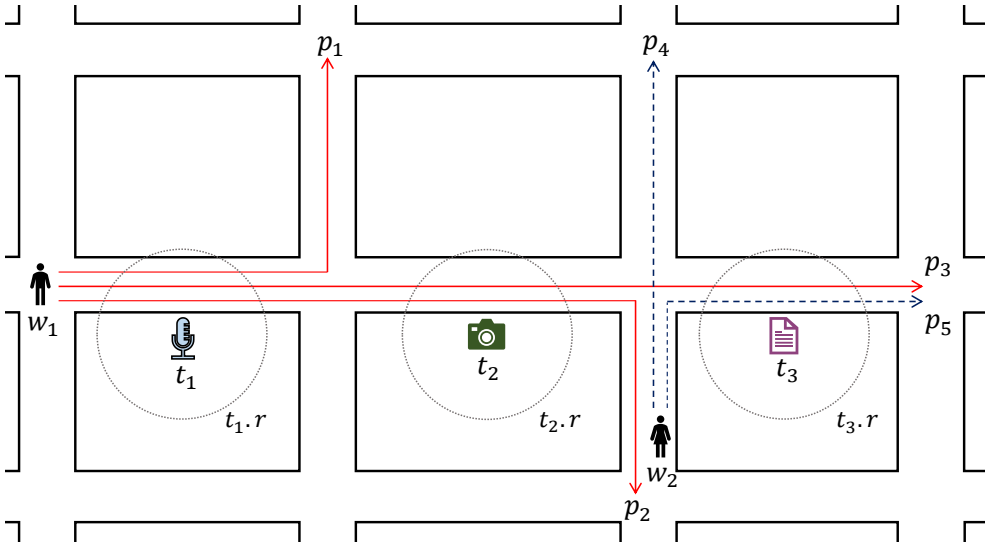


Fig. 27. An MCS instance with three tasks and two workers. Some possible worker trajectories for w_1 and w_2 are shown with solid and dashed lines, respectively, and task regions are enclosed with circles.

gers). This issue is partly investigated in the MCS literature [31, 30], but without considering the preferences of workers and task requesters.

(iii) *Capacity constraints:* In order to avoid disruptions to their daily schedule, workers in opportunistic MCS campaigns may choose to bound the number of tasks they accept to perform for each assignment period. The classic deferred-acceptance mechanism proposed by Gale and Shapley [42] can be used to find preference-aware assignments in general matching problems even in presence of capacity constraints, but it works only in offline settings, where the eligibility of every (worker-task) pair is known and fixed. This is not the case in our setting, where the trajectories of workers, hence which tasks they can perform, are uncertain. On the other hand, the studies that consider capacity (or budget) constraints for workers in the MCS literature [38, 87] neglect to address the previous two issues.

As summarized above, these three issues have yet to be studied together, de-

spite being crucial for the success of an opportunistic MCS campaign. To fill this gap, in this chapter, we study the preference-aware task assignment problem within an opportunistic MCS model with uncertain worker trajectories and given capacity constraints. To point out some of the challenges this problem entails, let us analyze a few different scenarios in the MCS instance illustrated in Fig. 27, which consists of two workers (w_1, w_2) and three tasks (t_1, t_2, t_3) scattered in the area. Assume that w_1 can potentially visit all three task regions, while w_2 can visit only t_3's region, but it is not known in advance if they will actually do so. Consider the three (p_1, p_2, p_3) and two (p_4, p_5) of possible trajectories and visit scenarios for w_1 and w_2, respectively, shown in Fig. 27. If the workers do not have a capacity constraint (i.e., can perform every task on their way) or have a capacity of at least three, then w_1 should always be matched with tasks t_1 and t_2 if he visits their regions, as w_1 is the only worker that can perform these tasks. However, since the region of t_3 can be visited by both workers, it is not trivial to decide an assignment for t_3 even if there is no capacity constraint for workers. This is because the preference of t_3 based on the worker qualities needs to be considered along with how likely the workers will be visiting the region of t_3.

On the other hand, the preferences of workers become important when they are constrained by a capacity. For instance, assume that the capacity of w_1 is one, and the probability of p_3 is negligible. Then, when worker w_1 visits the region of t_1 and a matching decision needs to be made between w_1 and t_1, we need to consider the preference of w_1 on tasks t_1 and t_2 based on their rewards as well as the likelihood of p_1 and p_2. For example, even if p_1 is more probable than p_2, it may still be more profitable to skip t_1 if the reward of t_2 is significantly larger than that of t_1. In such scenarios, the matching decisions should be made according to the expected utilities of workers and task requesters to consider their preferences.

Let us consider another scenario, in which worker w_1 has a capacity of one and is

more likely to take path p_3, and the reward of t_3 is much greater than that of t_1 and t_2. Then, in order for the decision of skipping the potential matching opportunities with t_1 and t_2 to be justifiable for w_1, either the probability of p_5 should be low, or the quality of w_1 should be substantially better than that of w_2 so that task t_3 would be willing to skip the opportunity to match with w_2 if he ended up taking path p_5. Here, the timeliness of the visits also plays a major role. If the quality scores of the workers are very close to each other, the best strategy for the tasks would be to get matched with the first worker that visits their regions, because the risk of losing a matching opportunity at hand to wait for another worker would not be worth the extra benefit that they may possibly get by waiting. Therefore, the actual number of scenarios that needs to be examined to make an optimal task assignment gets much larger when we take the timeliness of worker visits into consideration.

Moreover, in real instances, the uncertainty in worker visits may be even more severe, in which case the number of possible scenarios is likely to grow exponentially with the number of participants and the campaign duration. In this study, we address these issues and provide polynomial-time algorithms that produce task assignments that maximize the happiness of users with their assignments with respect to their preferences by considering all possible scenarios in an efficient manner. Our primary contributions in this study can be summarized as follows:

- We introduce the preference-aware task assignment problem in opportunistic MCS systems, where task assignments need to be made in an online manner due to uncertain worker trajectories.

- We formulate the criteria for preference-awareness in this problem after showing that the existing preference-awareness objectives used in the literature (e.g., minimizing the number of unhappy pairs) do not work when worker trajectories

are uncertain.

- We study the problem in MCS systems with and without capacity constraints, and propose a polynomial-time online algorithm for each case, which we then show to be preference-aware by theoretical analysis.

- We perform extensive simulations with both real and synthetic data sets, and empirically show the superiority of the proposed algorithms over the existing task assignment algorithms.

5.2 System Model

5.2.1 Assumptions

At the center of our system model is a service provider (SP) that receives the sensing task inquiries from different requesters and assigns them to appropriate participants. Formally, in each (hourly, daily, weekly) assignment period that is divided into discretized time-steps $(0, 1, .., T)$, the responsibility of SP is to assign a set of sensing tasks $\mathcal{T} = \{t_1, t_2, \ldots, t_m\}$ to a set of workers registered to the system $\mathcal{W} = \{w_1, w_2, \ldots, w_n\}$ in a way that will satisfy both parties (user satisfaction criteria will be described below).

Each task t has spatio-temporal constraints for successful completion. Let $t.r$ and $[t.b, t.d]$ denote the geographic region and the time frame (between the **b**eginning time and **d**eadline) in which task t should be performed, respectively. Tasks are assumed to require simple sensing activity such as taking pictures [88], recording noise levels [89], and reporting traffic volume [78] or crowdedness [90]. Thus, they take a few seconds to complete (thus neglected for simplicity), and they can be completed anytime during the specified time frame. Each task t is also associated with a monetary reward $m(t)$ that is paid to the worker who performs it by the task requester upon successful

delivery of the sensed data.

The workers in our system model perform opportunistic sensing, so they do not travel to the task locations by interrupting their own schedules. Instead, they get assigned to a task only when they happen to be in the task region. Therefore, there is no travel cost associated with the tasks. Each worker w has a capacity $c(w)$, which indicates the maximum number of tasks worker w is willing to perform in a single assignment period. This can be a necessary constraint if the tasks require a certain level of involvement, causing the workers to lose some time. However, we also investigate task assignments in MCS systems that do not require involvement of workers and hence have no capacity constraints. Each worker w also has a quality score $q(w)$, which may refer to the likelihood of completing the assigned tasks as in [91], the expected quality of the sensed data [44], or the trustworthiness of the worker [92]. Some real-world mobile crowdsensing/sourcing systems that use a single numerical value to specify the qualification of workers include Waze [78] and Uber [93], which, respectively, utilize what is called Waze points and a five-star quality rating system.

In order to simplify our analysis, we rearrange the worker and task sets in decreasing order of the quality scores of workers and rewards of tasks, respectively. Thus, hereafter we have

$$q(w_i) > q(w_{i+1}),\ 1 \leq i < n, \tag{5.1}$$

and

$$m(t_j) > m(t_{j+1}),\ 1 \leq j < m. \tag{5.2}$$

If there are ties, we assume they are either broken by secondary factors such as registration time, or in an arbitrary manner so that there is only one possible order for both sets. Here, we note that the results of this study hold as long as the sets of

workers and tasks can be ordered as in (5.1) and (5.2). Within this constraint, it is possible to have non-uniform worker qualities and task rewards. For example, a task requester can offer a larger reward to workers with a higher quality score, as long as the rewards offered for a task t are always greater or always smaller than those offered for another task t'. This makes it possible to form a universal preference list for workers based on task rewards.

We assume that the trajectories of workers are uncertain but predictable, and revealed in real-time during the assignment period, so the task assignments have to be made in an online manner. However, the set of agents (i.e., workers and tasks) and all other parameters in the system such as task rewards and worker quality scores are certain and known to SP by the beginning of each assignment period. Let $\lambda_{i,j}$ be the average inter-visit time of worker w_i to the region $t_j.r$. Then, assuming an exponential distribution (similar to [30, 50, 51, 52]), the probability that worker w_i visits $t_j.r$ in a time frame of length L is computed as follows:

$$V_{i,j}(L) = 1 - e^{-L/\lambda_{i,j}} \tag{5.3}$$

The results of this study, however, do not depend on the underlying distribution model, and other probability functions including those produced by machine learning methods [94], which can integrate any dependency between the visits of a worker to different regions, can be used as well.

Once the task set for the current assignment period is determined, each worker w_i will be asked to provide SP with $\lambda_{i,j}$ values for the region of each task $t_j \in \mathcal{T}$. To this end, workers should be maintaining their visit records with a sufficient geographic density, as task regions may differ between assignment periods. They can submit arbitrarily large numbers for regions they have not visited, or for which they do not feel comfortable disclosing their true visit frequency for privacy-related reasons. Also,

they can always inform SP of the regions they will definitely visit if certain parts of their trajectories are (or become) fixed. A legitimate concern here would be the possibility of receiving fabricated $\lambda_{i,j}$ values from some workers aiming to increase their gains from the system in a malicious manner. However, workers are required to inform SP when they enter one of the task regions to be considered for the assignment of the corresponding task, as a task may be assigned to a worker only when the worker is in the task region. Thus, SP can easily verify the accuracy of the received $\lambda_{i,j}$ values based on the visit frequencies of worker w_i, and reduce the quality scores of dishonest workers.

5.2.2 Problem Formulation

We represent the task assignments in our model with a matching $\mathcal{M}$ between the sets $\mathcal{W}$ and $\mathcal{T}$, where $\mathcal{M}(w)$ and $\mathcal{M}(t)$ denote the set of tasks assigned to worker w and the worker assigned to task t, respectively. If user (worker or task) u is unassigned in $\mathcal{M}$, then $\mathcal{M}(u) = \varnothing$. For a matching to be feasible according to our system model, it should satisfy the following constraints for each $w \in \mathcal{W}$ and $t \in \mathcal{T}$:

- $\mathcal{M}(t) \in \mathcal{W} \cup \{\varnothing\}$,
- $\mathcal{M}(w) \subseteq \mathcal{T}$,
- $|\mathcal{M}(w)| \leq c(w)$,
- $\mathcal{M}(t) = w \Leftrightarrow t \in \mathcal{M}(w)$.

We assume a transparent SP whose decisions are visible to the users so that each user can see with whom they could be matched, but did not. In such a setting, it is crucial to produce impartial and satisfactory task assignments that do not sacrifice the benefit of some users for the others or for maximizing the overall matching utility

according to some system-level metric such as the number of completed tasks. We will use the following definitions to refer to the user happiness in a matching.

Definition 13 (Unhappy pair)**.** *A worker-task pair (w, t) is said to be unhappy in a matching $\mathcal{M}$ if*

- *worker w has visited region $t.r$ between the time frame $[t.b, t.d]$,*
- *task t is either unmatched or matched to a worker w' with a smaller QoS score than worker w (i.e., prefers w to w'),*
- *worker w has unused capacity (i.e., $|\mathcal{M}(w)| < c(w)$), or the reward of at least one task t' in $\mathcal{M}(w)$ is smaller than that of task t (i.e., prefers t to t').*

From the perspective of worker w and task t, the first condition in the definition indicates that there was in fact an opportunity for them to get matched, and the last two indicate that SP instead matched them with some other users that they rationally prefer less, or left them unmatched/with an unused capacity.

Definition 14 (Stable matching)**.** *A matching $\mathcal{M}$ is stable if it contains no unhappy pairs.*

Although we can always find a stable matching in an offline setting (as it will be shown in the next section), it may not be possible to do so in an online setting where we do not know whether and when a worker will visit a region. Consider the instance in Fig. 28. Given that worker w_1 is currently in $t_2.r$, SP should decide whether to assign him to task t_2.

- If it assigns worker w_1 to task t_2, but then worker w_1 visits $t_1.r$, (w_1, t_1) will be an unhappy pair because worker w_1 prefers task t_1 to task t_2 as $m(t_1) > m(t_2)$, and task t_1 prefers being matched to worker w_1 to being unmatched.

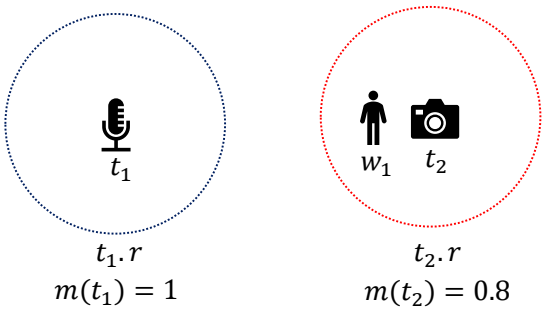


Fig. 28. An instance with a worker and two tasks. Let the probability of w_1 visiting $t_1.r$ before the deadline of t_1 be 0.6, and the probability of w_1 revisiting $t_2.r$ before the deadline of t_2 be 0. Also, let $q(w_1) = c(w_1) = 1$.

- If it does not assign worker w_1 to task t_2, and worker w_1 does not visit $t_1.r$, then (w_1, t_2) will be an unhappy pair because worker w_1 and task t_2 prefer being matched to each other to being unmatched.

Thus, it is not possible to ensure perfect user happiness without knowing the exact worker trajectories.

Besides, in an online setting, minimizing the *expected* number of unhappy pairs may not actually maximize user happiness. Again, in the instance in Fig. 28, SP should avoid matching worker w_1 to task t_2 in order to minimize the expected number of unhappy pairs, because as shown in Table 8, the expected number of unhappy pairs is larger when they get matched ($w_1 \Rightarrow t_2$). Yet the expected profit of worker w_1 in case he is not matched to task t_2 is $m(t_1) \times 0.6 = 0.6$, which is smaller than the profit he would make if he was matched to t_2 ($m(t_2) = 0.8$). So, worker w_1 would prefer to be matched to task t_2 despite the increase in the expected number of unhappy pairs he will form.

The example discussed above demonstrates that, in an online setting, user happiness should be measured in an online manner and by considering the impact of

Decision	Scenario	# of UPs	Expected # of UPs
$w_1 \Rightarrow t_2$	w_1 visits $t_1.r$	1	0.6
	otherwise	0	
$w_1 \not\Rightarrow t_2$	w_1 visits $t_1.r$	0	0.4
	otherwise	1	

Table 8. Analysis of all possible scenarios in the instance illustrated in Fig. 28. (UP is short for unhappy pair.)

each matching-related decision of SP on the overall benefit that users will get from the system. In MCS systems without capacity constraints, there is no competition among task requesters, because workers would like to and can get matched with all tasks on their trajectory. Therefore, the stability of the assignments and preference-awareness in such systems can be ensured by maximizing the expected assignment quality of each task based on the visit probabilities of the workers, which can formally be expressed as

$$\text{maximize} \sum_{t_j \in \mathcal{T}} E_j, \tag{5.4}$$

where E_j is the expected assignment quality of task t_j. Then, a task assignment mechanism is said to be optimal in terms of preference-awareness in these systems if its all matching decisions for each task t in the system maximizes the expected assignment quality of t based on the quality score of the readily available worker and the quality scores of the workers that could probably visit $t.r$ in future.

However, in the presence of capacity constraints, there is a competition among both workers and task requesters, because the fact that workers are able to perform only a limited number of tasks transforms the task assignment problem into a limited resource allocation problem. In this setting, the expected utilities of users become

interdependent, and get affected by each matching decision of SP. Consequently, the expected utility of a worker or a task after a certain time-step depends on the decision mechanism that will be used by SP in that time frame. Besides, the number of possible visit scenarios increases exponentially with respect to the length of the assignment period and the number of users. To address these challenges, we map all possible (exponentially many) visit scenarios that can happen after time-step s to all possible stable matchings in these scenarios along with their likelihood of occurrence, which we can analyze in polynomial-time and use to estimate the expected user utilities for the time period $[s, T]$.

Suppose that a worker w_i with a remaining capacity of $c_i^s \geq 1$ is in the region of a currently unassigned task t_j at time-step $s : t_j.b \leq s \leq t_j.d$, so SP has to make a matching decision for the pair. Let $\mathcal{A}_s = \{A_s^1, A_s^2, \ldots, A_s^k\}$ be the set of all possible worker visit scenarios that can happen in the time frame $[s, T]$ given the visit probabilities of the workers for all task regions. That is,

$$\mathcal{A}_s = R_1^s \times R_2^s \times .. \times R_n^s, \tag{5.5}$$

where R_i^s is the set of all possible spatiotemporal trajectories of worker w_i after time-step s. Let $p(A_s^l)$ denote the probability that the scenario $A_s^l \in \mathcal{A}_s$ will occur. Then, we have

$$\sum_{l=1}^{k} p(A_s^l) = 1. \tag{5.6}$$

Let M_s^l be the stable matching in the scenario A_s^l between the tasks that are unassigned and the workers that have a positive remaining capacity at time-step s (since the visits in A_s^l are known, a stable matching can be found using the offline stable matching algorithm that will be described in Section 5.3.2.1). Also, let $\hat{M}_s^l$ be the stable matching in the same scenario assuming that w_i is matched to t_j. Then, as-

suming SP is making optimal assignments in terms of stability, the expected total reward worker w_i would get in time frame $[s, T]$ if he was not assigned to task t_j at time-step s can be computed by:

$$\mathbf{W}_i(s) = \sum_{l=1}^{k} p(A_s^l) \times \sum_{t \in M_s^l(w_i)} m(t), \tag{5.7}$$

and that if he was assigned to task t_j by:

$$\mathbf{W}'_{i,j}(s) = m(t_j) + \sum_{l=1}^{k} p(A_s^l) \times \sum_{\hat{t} \in \hat{M}_s^l(w_i)} m(\hat{t}). \tag{5.8}$$

Analogously, the expected sensing quality to be received by task t_j if it is not assigned to worker w_i at time-step s and otherwise can be, respectively, computed by:

$$\mathbf{T}_j(s) = \sum_{l=1}^{k} p(A_s^l) \times q(M_s^l(t_j)), \tag{5.9}$$

and

$$\mathbf{T}'_{j,i}(s) = q(w_i). \tag{5.10}$$

Then, we can define a decision-time unhappy pair as follows.

Definition 15 (Decision-time unhappy pair)**.** *A worker-task pair (w_i, t_j) is said to be a decision-time unhappy pair if the following conditions hold for any time-step s in $[t_j.b, t_j.d]$:*

- *worker w_i has a positive remaining capacity,*
- *task t_j is unassigned,*
- *worker w_i is in region $t_j.r$, and*
- *either (i) SP matches worker w_i to task t_j, but at least one of them would be*

better off otherwise, i.e.,

$$\mathbf{W}_i(s) > \mathbf{W}'_{i,j}(s) \quad or \quad \mathbf{T}_j(s) > \mathbf{T}'_{j,i}(s), \tag{5.11}$$

- *or (ii) SP does not match worker w_i to task t_j, but they both would be better off otherwise, i.e.,*

$$\mathbf{W}'_{i,j}(s) > \mathbf{W}_i(s) \quad and \quad \mathbf{T}'_{j,i}(s) > \mathbf{T}_j(s). \tag{5.12}$$

In our example illustrated in Fig. 28, assuming s is the current time-step, we have two possible trajectories that can be seen after s (i.e., w_1 visits $t_1.r$ or he does not; $|\mathcal{A}_s| = 2$) with the given probabilities. This yields $\mathbf{W}'_{1,2}(s) = 0.8 > \mathbf{W}_1(s) = 0.6$ and $\mathbf{T}'_{2,1}(s) = 1 > \mathbf{T}_2(s) = 0$. Hence, worker w_1 and task t_2 will, as desired, form a decision-time unhappy pair due to (5.12) if SP fails to match them.

Definition 16 (Online stable matching)**.** *A matching $\mathcal{M}$ is called an online stable matching if it does not admit any decision-time unhappy pairs.*

Consequently, our objective in the MCS systems with capacity constraints is to find an online stable matching, and we call such a matching *optimal* in terms of preference-awareness. It is straightforward to see that the optimal matching strategy to this end would be to match a worker-task pair if (5.12) holds. However, the difficult part is to compute the values of $\mathbf{W}_i(s)$, $\mathbf{W}'_{i,j}(s)$ and $\mathbf{T}_j(s)$, because $\mathcal{A}_s$ grows exponentially with the number of workers (n) and length of the assignment period (T). In the following section, we will show how to compute these values efficiently without actually forming the set $\mathcal{A}_s$.

The key notations used throughout this chapter are summarized in Table 9 for convenience.

Notation	Description
$\mathcal{T}$, $\mathcal{W}$	Set of tasks and workers, respectively
m, n	Number of tasks and workers, respectively
$\mathcal{M}$	A many-to-one matching between $\mathcal{T}$ and $\mathcal{W}$
$\mathcal{M}(u)$	Assigned worker (task set) to task (worker) u in $\mathcal{M}$
$[0, T]$	Current assignment period
$t.r$	Region of task t
$[t.b, t.d]$	Time interval in which t should be performed
$m(t)$	Reward associated with task t
$c(w)$	Capacity of worker w
$q(w)$	Quality score of worker w
$\lambda_{i,j}$	Average time between the visits of w_i to $t_j.r$
$V_{i,j}(L)$	Probability that w_i visits $t_j.r$ in a time frame of length L

Table 9. Notations used in Chapter 5.

5.3 Proposed Solution

In this section, we begin by considering a simpler, but still practical version of the problem. Then, we investigate the generic version of the problem, and provide preference-aware task assignment algorithms and their theoretical analysis for both versions.

5.3.1 Task Assignment in Systems without Capacity Constraints

In MCS systems with small sensing tasks that require no interaction from the workers, it is safe to disregard the capacity constraints of workers as carrying out a task does not put a load on them. Moreover, in the case of uniformly distributed

tasks in an area or short assignment periods, since workers could visit only a limited number of task regions, it would still be safe to disregard the capacity constraints. In other words, even if workers had capacity constraints in any of these cases, they would be overshadowed by the spatiotemporal constraints and can hence be ignored during the task assignment process (at least until the point where assigning another task to a worker would violate his capacity constraint).

An example of an MCS system without capacity constraints would be a traffic monitoring system such as Waze [78], where the speed of traffic, which can be estimated by the speed of change in the GPS coordinates of workers, can be sensed and transmitted to SP automatically by workers' mobile devices without requiring active involvement of workers.

In this type of MCS systems, workers would like to perform each and every task that is on their trajectory and does not conflict with their preferences in order to maximize their profits. However, task requesters would still desire to have their sensing tasks performed by workers with the highest quality scores. Thus, the problem transforms into a one-sided matching problem in terms of user preferences. That is, to find optimal task assignments we just need to maximize the sensing quality received by task requesters. Moreover, we can consider each task separately, because the assignment quality of a task t_j depends only on which workers will visit the task region $t_j.r$ and the time of their visits, and is independent of the visits of workers to the other task regions due to the absence of capacity constraints.

To solve this problem, we utilize *Optimal Stopping Theory (OST)* [95], which provides a dynamic programming based framework for the decision problems with a finite horizon (e.g., the secretary hiring problem). This is suitable for our problem, because for each task t_j there will be a number of decision points at the times $t_j.r$ is visited by any worker, and at each of these we should decide whether to wait for

a higher quality worker or to assign task t_j to the worker w_i who is currently in the region $t_j.r$ based on the quality of w_i and the expected quality to be achieved if we choose to wait instead.

Let $\mathbf{E}_j(s)$ be the expected assignment quality for task t_j after the time-step s. Since task t_j can only be performed between $[t_j.b, t_j.d]$, we have

$$\mathbf{E}_j(s) = \mathbf{E}_j(t_j.b),\ s < t_j.b, \tag{5.13}$$

and

$$\mathbf{E}_j(s) = 0,\ s \geq t_j.d. \tag{5.14}$$

Since each worker w_i will visit the region $t_j.r$ in the time frame $[s, s+1)$ with the probability $V_{i,j}(1)$, we have the following recursive relation between $\mathbf{E}_j(s)$ and $\mathbf{E}_j(s+1)$:

$$\begin{aligned}\mathbf{E}_j(s) = \sum_{i=1}^{n} \Big(\max\big(q(w_i), \mathbf{E}_j(s+1)\big) \times V_{i,j}(1) \times \rho_j(i) \Big) \\ + \mathbf{E}_j(s+1) \times \rho_j(n+1),\end{aligned} \tag{5.15}$$

where $\rho_j(i)$ is the probability that no worker with an index smaller than i visits $t_j.r$ within a time frame of length 1. Since the smallest worker index is 1, we have $\rho_j(1) = 1$, and the value of $\rho_j(i)$ for $2 \leq i \leq n+1$ can be computed by:

$$\rho_j(i) = \rho_j(i-1) \times \big(1 - V_{i-1,j}(1)\big) \tag{5.16}$$

Note that although there are 2^n possible scenarios (i.e., each worker being within or outside of the task region) for each time frame of length 1 in terms of worker visits to a task region, we consider only n of them to calculate (5.15), because if w_i is in the region, whether $w_{i+1}, .., w_n$ are within or outside of the region is irrelevant as they are preferred less than w_i. Therefore, using the base cases $\mathbf{E}_j(t.d) = 0$ and $\rho_j(1) = 1$, we can recursively compute all values of $\mathbf{E}_j(s)$ for $t_j.b \leq s < t_j.d$ in polynomial time

Algorithm 9: Calculation of $\mathbf{E}_j(s)$ for all practical values of time-step s

```
1  E_j(t_j.d) ← 0
2  for s ← t_j.d − 1 down to t_j.b do
3      E_j(s) ← q(w_1) × V_{1,j}(1)
4      ρ ← 1 − V_{1,i}(1)
5      for i ← 2 to n do
6          if q(w_i) ≥ E_j(s + 1) then
7              E_j(s) ← E_j(s) + q(w_i) × V_{i,j}(1) × ρ
8              ρ ← ρ × (1 − V_{i,j}(1))
9          else
10             break
11     E_j(s) ← E_j(s) + E_j(s + 1) × ρ
```

as described in Algorithm 9.

In this algorithm, when we calculate $\mathbf{E}_j(s)$, we utilize the fact that task t_j would like to match only with workers with a quality score that is greater than or equal to $\mathbf{E}_j(s+1)$ (line 6) at time-step s because, otherwise, it would be more advantageous for it to wait for the next time-step. Thus, we consider only these workers in lines 3-10 in decreasing order of their quality scores to compute the expected utility of task t_j based on the visit probabilities of these workers to its region in case it will be matched with one of these workers at time-step s. Since the expected utility of task t_j at time-step s will be the same as that at time-step $s+1$ if none of these workers visits the region of task t_j between time-steps s and $s+1$, we finally increase the value of $\mathbf{E}_j(s)$ by $\mathbf{E}_j(s+1) \times \rho$ in line 11, where ρ is calculated between lines 4-10 as the probability that $t_j.r$ will not be visited by any of these workers between time-steps s

Algorithm 10: OST-based Algorithm (OSTA) at time-step s

1 **if** $q(w_i) \geq \mathbf{E}_j(s)$ **then**
2 match w_i to t_j
3 terminate the algorithm for t_j

and $s + 1$.

A summary of the optimal decision mechanism that will be run for each task t_j whenever $t_j.r$ is visited by a worker w_i is given in Algorithm 10. We assume all $\mathbf{E}_j(s)$ values for $s : t_j.b \leq s \leq t_j.d$ are precomputed and stored in a lookup table, but it is also possible to compute only $\mathbf{E}_j(s)$ for $s : \hat{s} \leq s \leq t_j.d$ at the first time ($\hat{s}$) a worker visits $t_j.r$ to avoid computing $\mathbf{E}_j(s)$ values that will never be used. The algorithm simply checks whether it is more advantageous to match with the visiting worker or to skip the opportunity (line 1), and makes a matching decision accordingly. Due to sparse nature of visits in mobile networks, we assume that there will be a single matching decision to make at each time-step. However, if there are multiple workers that visit the region of a task at a certain time-step, it suffices to run Algorithm 10 for the worker with the highest quality score. For each task t_j, the algorithm will be run until either task t_j gets matched, or it expires. Since we are able to compute the expected utilities of task requesters precisely, and make decisions in a way to maximize their utilities during the matching process, we have the following result.

Corollary 10.1. *Algorithm 10 always makes the optimal matching decisions for task requesters when workers do not have capacity constraints (i.e., maximizes the expected assignment quality $q(\mathcal{M}(t))$ for each task t).*

Running time. Algorithm 10 obviously has a time complexity of $O(1)$, however $\mathbf{E}_j(s)$ needs to be precomputed for all feasible j and s values by running Algorithm 9.

For each task t_j, we precompute $\mathbf{E}_j(s)$ values for all $s : t_j.b \leq s \leq t_j.d$, and computing each $\mathbf{E}_j(s)$ value takes $O(n)$ time. Thus, the total time complexity becomes $O(n\tau)$, where $\tau = \sum_{t \in \mathcal{T}}(t.d - t.b)$.

5.3.2 Task Assignment in Systems with Capacity Constraints

In this section, we first describe an optimal algorithm to find stable matchings in offline settings where the trajectory of each worker is known in advance. Then, exploiting the ideas behind the offline algorithm, we provide our algorithm for the online settings. We begin with the following definition.

Definition 17 (Pair priority)**.** *The priority of a worker-task pair (w_i, t_j) refers to the relative importance of the pair in terms of stability, and can be defined as*

$$\phi(w_i, t_j) = \max(m, n) \times \min(i, j) + \max(i, j) \tag{5.17}$$

where a smaller value indicates a higher priority.

If the worker and task indices were to start at 0 in the sets $\mathcal{W}$ and $\mathcal{T}$, the priority of a worker-task pair p would be equal to the number of worker-task pairs with a higher priority than p.

5.3.2.1 Offline Algorithm

In Algorithm 11, we present a pseudo-code description of the offline algorithm. In line 1, it finds the set $\mathcal{A}$ of all eligible worker-task pairs that can be matched to each other (i.e., the task region visited by the worker). Then, in each step, it finds (line 3) and matches (lines 4-5) the worker-task pair with the highest priority in $\mathcal{A}$, which is followed by removing all pairs that become infeasible due to the most recent pair assignment (lines 6-8). This continues until the set $\mathcal{A}$ becomes empty. In the following theorem, we prove the optimality of this algorithm.

Algorithm 11: Offline SM Algorithm

$\mathcal{A} \leftarrow \{(w, t) : w \text{ visits } t.r \text{ in } [t.b, t.d]\}$
while $\mathcal{A} \neq \varnothing$ **do**
  $(w_i, t_j) \leftarrow \arg\min_{(w,t)\in\mathcal{A}} \phi(w, t)$
  $\mathcal{M}(w_i) \leftarrow \mathcal{M}(w_i) \cup t_j$
  $\mathcal{M}(t_j) = w_i$
  if $|\mathcal{M}(w_i)| = c(w_i)$ **then**
    $\mathcal{A} \leftarrow \mathcal{A} \setminus \{(w, t) : w = w_i\}$
  $\mathcal{A} \leftarrow \mathcal{A} \setminus \{(w, t) : t = t_j\}$
return $\mathcal{M}$

Theorem 11. *Algorithm 11 always produces a stable matching in offline settings.*

Proof. We prove this by contradiction. Assume that there is an unhappy pair (w_i, t_j) in the final matching $\mathcal{M}$ produced by the algorithm. According to Definition 13, worker w_i has visited $t_j.r$ within the time frame of task t_j, so (w_i, t_j) is in $\mathcal{A}$ in the beginning.

- If $|\mathcal{M}(w_i)| < c(w_i)$ and $\mathcal{M}(t_j) = \varnothing$, then the pair (w_i, t_j) should still be in $\mathcal{A}$, which indicates that $\mathcal{A}$ is non-empty, contradicting the termination condition of the algorithm.
- If $|\mathcal{M}(w_i)| < c(w_i)$ and $\mathcal{M}(t_j) = w_k$, then for (w_i, t_j) to be an unhappy pair, we should have $q(w_i) > q(w_k)$, hence

$$i < k \text{ and } \phi(w_i, t_j) < \phi(w_k, t_j). \tag{5.18}$$

  Since at the time the pair (w_k, t_j) was selected by the algorithm, the pair (w_i, t_j) was still in $\mathcal{A}$ (as $|\mathcal{M}(w_i)| < c(w_i)$) and has a higher priority than (w_k, t_j), the

algorithm should have selected (w_i, t_j), which is a contradiction.

- If $|\mathcal{M}(w_i)| = c(w_i)$ and $\mathcal{M}(t_j) = \varnothing$, then for (w_i, t_j) to be an unhappy pair, we should have $m(t_k) < m(t_j)$ for at least one task $t_k \in \mathcal{M}(w_i)$. Thus,

$$j < k \text{ and } \phi(w_i, t_j) < \phi(w_i, t_k). \tag{5.19}$$

  As in the previous scenario, this indicates that the pair (w_i, t_j) should have been selected prior to (w_i, t_k), which is a contradiction.

- If $|\mathcal{M}(w_i)| = c(w_i)$ and $\mathcal{M}(t_j) = w_k$, then for (w_i, t_j) to be an unhappy pair, we should have

$$q(w_i) > q(w_k) \text{ and } m(t_j) > m(t_l) \tag{5.20}$$

  for at least one task $t_l \in \mathcal{M}(w_i)$. This yields

$$i < k \text{ and } j < l, \tag{5.21}$$

  and hence

$$\phi(w_i, t_j) < \phi(w_k, t_j) \text{ and } \phi(w_i, t_j) < \phi(w_i, t_l). \tag{5.22}$$

  This means the pair (w_i, t_j) should have been selected prior to both (w_i, t_l) and (w_k, t_j), which is also a contradiction and completes the proof.

□

5.3.2.2 Online Algorithm

Theorem 11 shows that in the presence of capacity constraints, a worker-task pair dominates the pairs with lower priority scores if it is in the set $\mathcal{A}$, which implies that the highest priority pair will be matched with the same probability of being in the set $\mathcal{A}$. Likewise, the next highest priority pair will be matched with the probability of being in the set $\mathcal{A}$ in case it is not eliminated by the higher priority pair, and so

on. We utilize this observation to find the matching probability of a worker-task pair in the stable matchings for all possible scenarios ($\mathcal{A}_s$) that can occur after a certain time-step (s), and use it to make decisions in online setting.

Lemma 1. *Given a worker-task pair (w_i, t_j) and time-step s at which w_i has a remaining capacity of c_i^s and t_j is unmatched, the probability of w_i and t_j being matched in a stable matching in any of the possible scenarios that can occur between time-steps $s < t_j.d$ and T can be computed by*

$$P_s(i,j) = \sum_{k=1}^{c_i^s} Q_{i,j}^s[k] \times \overbrace{\prod_{k=1}^{i-1} \big(1 - P_s(k,j)\big)}^{\eta_{i,j}^s} \times V_{i,j}(t_j.d - s), \tag{5.23}$$

where

$$Q_{i,j}^s[k] = \begin{cases} 1, & \text{if } j = 1,\ k = c_i^s \\ 0, & \text{if } j = 1,\ k \neq c_i^s \\ Q_{i,j-1}^s[k] + Q_{i,j-1}^s[k+1] \times \eta_{i,j-1}^s, & \text{if } j > 1,\ k = 0 \\ Q_{i,j-1}^s[k] \times \overline{\eta}_{i,j-1}^s, & \text{if } j > 1,\ k = c_i^s \\ Q_{i,j-1}^s[k+1] \times \eta_{i,j-1}^s + Q_{i,j-1}^s[k] \times \overline{\eta}_{i,j-1}^s, & \text{otherwise} \end{cases} \tag{5.24}$$

and $\overline{\eta}_{i,j}^s = 1 - \eta_{i,j}^s$. If t_j was matched before time-step s, or $s \geq t_j.d$, $P_s(i,j) = 0$.

Proof. In order for worker w_i and task t_j to be matched in a stable matching $\mathcal{M}$ in a given scenario (e.g., $A_s^l \in \mathcal{A}_s$), the following three conditions should be satisfied:

- at most $c_i^s - 1$ of the higher priority pairs in the set

$$\begin{aligned} F_{i,j} &= \{(w_i, t_k) : \phi(w_i, t_k) < \phi(w_i, t_j)\} \\ &= \{(w_i, t_k) : k < j\} \end{aligned} \tag{5.25}$$

  should be matched in $\mathcal{M}$ (i.e., w_i should be matched with at most $c_i^s - 1$ of the

tasks that he prefers more than t_j), because, otherwise, the pair (w_i, t_j) will be eliminated. In (5.23), the probability of this is given by

$$\sum_{k=1}^{c_i^s} Q_{i,j}^s[k], \tag{5.26}$$

where $Q_{i,j}^s[k]$ is the probability that worker i will be matched to exactly $c_i^s - k$ of the tasks in $F_{i,j}$, and thus will have a remaining capacity of k. Thus, (5.26) is the probability that worker w_i will have a positive remaining capacity and be able to match with task t_j. The calculation of the $Q_{i,j}^s[k]$ values is realized in a recursive manner as described in (5.24). Since $F_{i,1} = \varnothing$, we initially have $Q_{i,1}^s[c_i^s] = 1$ and $Q_{i,1}^s[k < c_i^s] = 0$. We can then calculate $Q_{i,j+1}^s$ from $Q_{i,j}^s$ based on the probability ($\eta_{i,j}^s$) that worker w_i matches with task t_j. An illustration of this recursive procedure is given in Fig. 29.

- none of the higher priority pairs in

$$\begin{aligned} G_{i,j} &= \{(w_k, t_j) : \phi(w_k, t_j) < \phi(w_i, t_j)\} \\ &= \{(w_k, t_j) : k < i\} \end{aligned} \tag{5.27}$$

should be matched in $\mathcal{M}$, because, otherwise, task t_j will already be matched with a more favorable worker, hence the pair (w_i, t_j) will be eliminated. In (5.23), this is given in a recursive fashion as follows:

$$\prod_{k=1}^{i-1} 1 - P_s(k, j). \tag{5.28}$$

- worker w_i should visit the region of task t_j between $[s, t_j.d]$. That is, the pair (w_i, t_j) should be in the set $\mathcal{A}$ of Algorithm 11. This occurs with the probability of $V_{i,j}(t_j.d - s)$, which is the last term in (5.23).

□

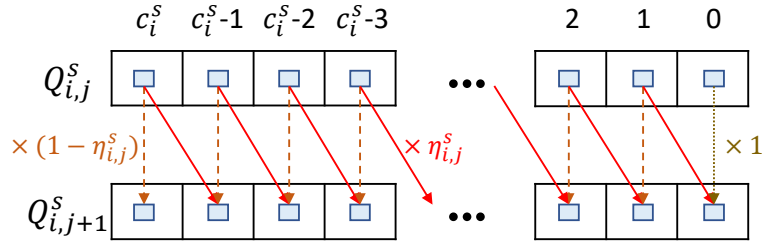


Fig. 29. Illustration of the update procedure for the remaining capacity probabilities defined in (5.24). Each of the dashed and solid edges indicates a contribution with a factor of $1-\eta_{i,j}^s$ and $\eta_{i,j}^s$, respectively, while the rightmost, dotted edge indicates a direct addition.

Algorithm 12 summarizes the procedure to calculate $P_s(i,j)$ values for all $1 \leq i \leq n, 1 \leq j \leq m$ values. In this algorithm, we maintain a variable u_j for each task t_j, which refers to the probability of task t_j not being matched to any of the workers that considered so far in the algorithm, hence initialized to be 0 in line 1 if t_j is already matched before time-step s, and 1 otherwise. Note that once $P_s(i,j)$ is calculated, the values of $P_s(k,j)$ for $k > i$ and $P_s(i,l)$ for $l > j$ are independent of each other. Thus, we can first compute $P_s(1,j)$ starting from $j = 1$ to $j = m$, then $P_s(2,j)$ for all j values in the same order, and so on. This ensures that the matching probabilities of all interdependent worker-task pairs will be calculated following the priority order.

According to Lemma 1, we can express $P_s(i,j)$ as the ratio of the number of stable matchings that worker w_i and task t_j are matched to each other to the total number of stable matchings in all possible scenarios after time-step s. Thus, given the $P_s(i,j)$ values for all i,j pairs, we can compute $\mathbf{W}_i(s)$ and $\mathbf{T}_j(s)$ as follows:

$$\mathbf{W}_i(s) = \sum_{k=1}^{m} P_s(i,k) \times m(t_k), \tag{5.29}$$

$$\mathbf{T}_j(s) = \sum_{k=1}^{n} P_s(k,j) \times q(w_k). \tag{5.30}$$

Algorithm 12: Calculation of $P_s(i,j)$ for all i,j

```
1 for j ← 1 to m do u_j ← 1 − |M(t_j)|
2 for i ← 1 to n do
3     compute Q^s_{i,1} according to (5.24)
4     for j ← 1 to m do
5         vp ← V_{i,j}(t_j.d − s)
6         η^s_{i,j} ← vp × u_j
7         P_s(i,j) ← η^s_{i,j} × Σ_{k=1}^{c^s_i} Q^s_{i,j}[k]
8         compute Q^s_{i,j+1} from Q^s_{i,j} according to (5.24)
9         u_j ← u_j − P_s(i,j)
```

On the other hand, the value of $\mathbf{W}'_{i,j}(s)$ depends on the probability of worker w_i being matched with each task t_k in the stable matchings that can be seen after time-step s assuming worker w_i and task t_j will be matched at s. This probability is denoted by $\hat{P}_s(i,k)$, and can simply be calculated by assuming task t_j is already matched and replacing c^s_i with $c^s_i - 1$ in (5.23) and (5.24) (and running Algorithm 12 accordingly). Then, we can compute $\mathbf{W}'_{i,j}(s)$ as:

$$\mathbf{W}'_{i,j}(s) = m(t_j) + \sum_{k \in \{1..m\} \setminus \{j\}} \big(\hat{P}_s(i,k) \times m(t_k)\big). \tag{5.31}$$

Lastly, we have $\mathbf{T}'_{j,i}(s) = q(w_i)$. Using these values, we can make an optimal matching decision for the worker-task pair (w_i, t_j) in terms of online stable matchings at any time-step s worker w_i is in the region of task t_j.

A summary of the decision process is described in Algorithm 13. In line 1, we compute the matching probabilities for all worker-task pairs by calling Algorithm 12, and then, based on these probabilities, we compute the expected utilities of worker

Algorithm 13: PRobabilistic Stable Task Assignment (PRSTA)$_\alpha(w_i, t_j)$ at time-step s

1 Compute $P_s(k,l)$ and $\hat{P}_s(k,l)$, $\forall k,l$, via Algorithm 12
2 Compute $\mathbf{W}_i(s)$ according to (5.29)
3 Compute $\mathbf{T}_j(s)$ according to (5.30)
4 Compute $\mathbf{W}'_{i,j}(s)$ according to (5.31)
5 $\mathbf{T}'_{j,i}(s) \leftarrow q(w_i)$
6 **if** $\mathbf{W}'_{i,j}(s) > \alpha \times \mathbf{W}_i(s)$ **then**
7 **if** $\mathbf{T}'_{j,i}(s) > \alpha \times \mathbf{T}_j(s)$ **then**
8 match w_i to t_j

w_i and task t_j at time-step s for the scenarios they do and do not get matched with each other in lines 2-5. If getting matched with each other is more preferable for both worker w_i (line 6) and task t_j (line 7) by a constant factor α (which will be discussed below), then a positive matching decision is made in line 8. As earlier, we assume at most one matching decision is being made at each time-step, but multiple worker-task pairs can be processed in the order of pair priority if needed. Given Definition 15 & 16, since the proposed algorithm makes a positive matching decision for a worker-task pair if (5.12) holds when $\alpha = 1$, we have the following result.

Corollary 11.1. *$PRSTA_{1.0}$ algorithm always produces online stable matchings.*

It should, however, be noted that the proposed method to compute expected user utilities does not consider the order of visits of workers to the task regions. Let us consider the instance given in Fig. 30 to explain this issue and why it is necessary to incorporate a constant α factor in the matching decisions of Algorithm 13 as a heuristic to address it. In this example, we assume that the visit probability

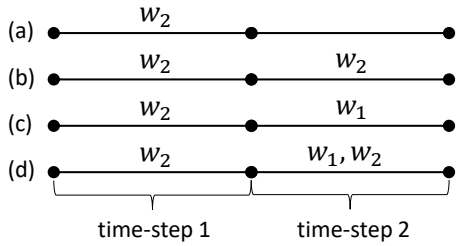


Fig. 30. Four of the possible visit orders of two workers (w_1, w_2) to the region of task t in an MCS instance with an assignment period of length two. For example, in scenario (c), the task region is visited by w_2 in time-step 1, and by w_1 in time-step 2.

of worker w_1 to the region of task t is quite high, and his quality score is significantly larger than that of worker w_2 so that even if worker w_2 visits the task region in time-step 1, it is advantageous for task t to wait for worker w_1. Thus, in all four scenarios, the decision at time-step 1 when worker w_2 visits the task region should be not to assign him to the task.

However, when we compute the expected utilities of users considering all possible stable matchings and their probability of occurrence, worker w_2 should be assigned to task t in scenarios (a) and (b) given that worker w_1 does not visit the task region in either time-step in these scenarios. Therefore, Algorithm 13 considers the matching (w_2, t) for scenarios (a) and (b) during computation of expected user utilities, and the matching (w_1, t) for the other scenarios. Yet when we see a similar visit pattern for time-step 1 in the online setting, since we do not in advance know the visit pattern for time-step 2, we need to either assign worker w_2 to task t or not. Consequently, the utility of the task will inevitably be overestimated, because if it gets assigned to worker w_2, its actual utility will also be $q(w_2)$ in scenarios (c) and (d), which is smaller than its expected utility $q(w_1)$ based on the stable matching (w_1, t) of these scenarios. On the other hand, if it does not get assigned to worker w_2, then it will

be left unmatched in scenario (a), and its actual utility (0) will be worse than its expected utility $q(w_2)$ based on the stable matching (w_2, t) of this scenario.

To alleviate the impact of these overestimations on the performance of the algorithm, we require (in lines 6-7 of Algorithm 13) that the expected utility of a user after the current time-step s is at least $1/\alpha$ times better than the utility he can get at time-step s to skip the existing matching opportunity. Here, we note that using such a constant factor does not favor any groups of users in the system, and hence does not invalidate its preference-awareness, in general, as long as a single, universal α factor is used for all decisions to ensure fairness towards different users. We empirically examine the algorithm's performance with various α values in the next section.

Running time. The time complexity of Algorithm 12 is $O(mnc_{max})$, where $c_{max} = \max_{w \in \mathcal{W}} c(w)$. Since the most expensive operation in Algorithm 13 is to run Algorithm 12 to find the matching probabilities, the worst-case running time of Algorithm 13 is also $O(mnc_{max})$. This can also be expressed as $O(nm^2)$, as the largest feasible $c_{max} = m$.

5.4 Evaluation

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

5.4.1 Simulation Settings

We perform simulations utilizing both a real data set and a synthetic data set. The latter is performed to understand the impact of the accuracy of the distribution model used to estimate the visit probabilities of workers on the performance of the proposed algorithms.

The synthetic data set is generated using 60 workers and 100 tasks in a 4 hours long assignment period. We randomly set the quality scores of the workers and the

task rewards from the range $(0, 1)$, and assign a capacity to each worker between 1 and 10 (we also look at the case without capacity constraints). For each worker-task pair (w_i, t_j), we randomly set the value of $\lambda_{i,j}$ between 8 to 24 hours. This generates instances where each worker visits, on average, 23% of all task regions in an assignment period. We examine the performance of the algorithms in instances with different worker/task counts, capacity constraints, and inter-visit times as well.

For the real data set, we utilize the San Francisco taxi data set [96], which contains the traces of 536 yellow cabs during May of 2018. For each instance, we randomly select a day as the assignment period, and then pick 60 taxis and use their traces on that day as worker trajectories. We divide the SF city into 121×100 regions of approximately $10^2 \times 10^2$ square meters, and create a task on randomly selected 100 regions that have at least 1000 traces in the whole data set. The average daily travel by the cabs is approximately 280 km, and the cabs visit about 20% of all regions at least once, on average. The other parameters are assigned similarly with the synthetic data set, and for each worker-task pair (w_i, t_j), the value of $\lambda_{i,j}$ is extracted from the traces.

To avoid introducing arbitrary random values for parameters that do not affect the performance of the algorithms in a notable way, we let the time frame of each task be the same as the duration of the assignment period in both data sets. Also, the assignment period is divided into a minute long time-steps in both data sets. Following the procedures described above, we generate 100 different instances of both synthetic and real data sets, and present the averaged results.

5.4.1.1 Benchmark Algorithms

We compare the performance of the proposed algorithms with a greedy algorithm and the well-known Gale-Shapley (GS) algorithm [42] by adapting it to our setting.

The former simply matches a task greedily with the first worker that visits its region with a positive remaining capacity. The latter is normally used to find stable matchings when the preference lists of individuals are static and known in advance. In our setting, however, it is neither possible nor desirable to match a worker-task pair if the worker does not visit the task region even if they happen to prefer each other the most. Since worker visits are uncertain in our setting, user preferences change dynamically based on worker trajectories, thus the GS algorithm cannot be used directly. We adapt it to our setting as follows. When a matching decision needs to be made at a time-step s for a worker-task pair, we form the preference lists of all workers with a positive remaining capacity and all unmatched tasks based on how likely they will have a chance to match and how beneficial they are to each other. Specifically, the preference list of each worker w_i is formed as $t_{\sigma_1}, t_{\sigma_2}, .., t_{\sigma_k}$ in order of non-increasing preference such that

$$m(t_{\sigma_j}) \times V_{i,\sigma_j}(t_{\sigma_j}.d - s) \geq m(t_{\sigma_{j+1}}) \times V_{i,\sigma_{j+1}}(t_{\sigma_{j+1}}.d - s) \tag{5.32}$$

for all $j : 1 \leq j < k$. The preference lists of tasks are formed similarly using the quality scores of the workers. Then, the GS algorithm is run to find a stable matching for these preference lists. If the currently examined worker-task pair is matched in this stable matching, we also match them in the real matching problem, otherwise we leave them unmatched for that time-step. For the PRSTA$_\alpha$ algorithm, we present the results for $\alpha = 1.0$ and $\alpha = 0.9$ in general as PRSTA$_{1.0}$ guarantees to produce online stable matchings, and PRSTA$_{0.9}$ is empirically shown to produce high quality final assignments with respect to the other performance metrics. However, we also examine the performance of the PRSTA$_\alpha$ algorithm with different values of α.

5.4.1.2 Performance Metrics

In order to evaluate and compare the performance of the algorithms, we utilize the following metrics, which capture different aspects of user satisfaction and efficiency.

- *Pairwise user happiness (%):* This is calculated as

$$100 \times \frac{b-a}{b}, \tag{5.33}$$

where a is the number of unhappy pairs, and b is the number of worker-task pairs (w, t) that had at least one matching opportunity during the assignment period, i.e., w visits $t.r$ between $[t.b, t.d]$, and at the time of the visit, w has a non-zero remaining capacity and t is unmatched.

- *Average user happiness (%):* Given a matching $\mathcal{M}$, let $\mathcal{S}_u$ be the set of tasks (workers) with whom worker (task) u forms an unhappy pair. Then, we can define the happiness ratio of user u as follows:

$$\theta_u = \begin{cases} 1, & \text{if } \mathcal{S}_u = \varnothing \\ 0, & \text{if } \mathcal{S}_u \neq \varnothing,\ \mathcal{M}(u) = \varnothing \\ \min\limits_{v \in \mathcal{S}_u} \left\{ \frac{\hat{f}(u)}{f(v)} \right\}, & \text{otherwise} \end{cases} \tag{5.34}$$

where

$$\hat{f}(u) = \begin{cases} q(\mathcal{M}(u)), & \text{if } u \in \mathcal{T} \\ \min\limits_{t \in \mathcal{M}(u)} \{m(t)\}, & \text{if } u \in \mathcal{W} \end{cases} \tag{5.35}$$

and $f(v) = m(v)$ if v is a task, and $f(v) = q(v)$ if it is a worker. Here, $\theta_u = 1$ if user u does not form any unhappy pairs, and $\theta_u = 0$ if he forms unhappy pairs

and is unmatched (i.e., since his current utility is 0, he is infinitely unhappy). Otherwise, its happiness is computed as the ratio of his current utility to the maximum utility he could achieve if he was matched to one of the users in the unhappy pairs he forms. Accordingly, the average user happiness is computed by

$$\frac{100}{m+n} \times \sum_{u \in \mathcal{W} \cup \mathcal{T}} \theta_u. \tag{5.36}$$

- *Average quality of sensing:* This is the average quality of sensing/service provided to the task requesters, and is computed by

$$\frac{1}{m} \times \sum_{t \in \mathcal{T}} q(\mathcal{M}(t)), \tag{5.37}$$

  where $q(\mathcal{M}(t)) = 0$ if $\mathcal{M}(t) = \varnothing$.

- *Online user happiness:* To show the optimality of the $\text{PRSTA}_{1.0}$ algorithm empirically, we look at the happiness of the users with the matching decisions in capacity-constrained settings. This is computed similarly to pairwise user happiness, but a and b in (5.33) are set, respectively, as the number of decision-time unhappy pairs and the number of times the algorithm is run to make a matching decision, which can be different for each algorithm.

- *Running time:* We also look at the running times of the algorithms to analyze how quickly they make the matching decisions, which is particularly important in MCS systems with high mobility.

5.4.2 Results

We first look at the results in the synthetic data set without capacity constraints. Fig. 31 shows the impact of the number of workers on the performance of the al-

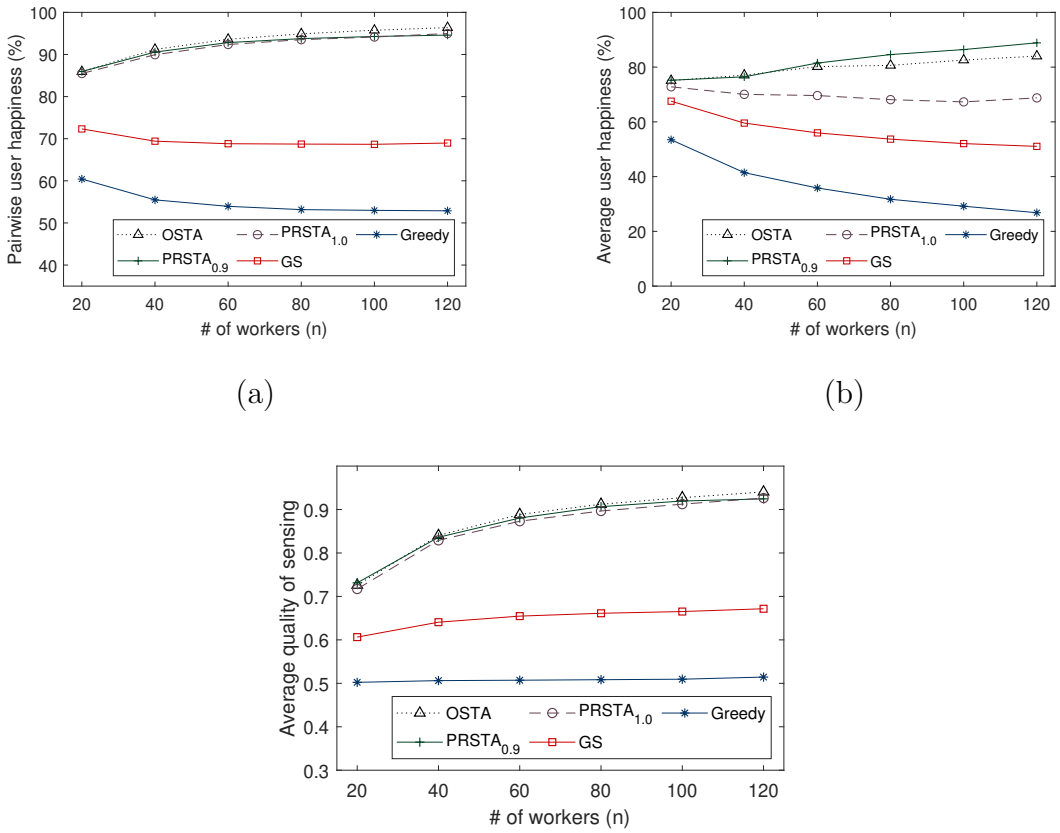


Fig. 31. Performance comparison of the algorithms against varying number of workers in systems without capacity constraints in the synthetic data set ($m = 100$).

gorithms. We see that the proposed algorithms substantially outperform the others, and the OSTA algorithm has the best performance for the most part, as expected. In fact, it is only slightly outperformed by the $\text{PRSTA}_{0.9}$ algorithm in terms of average user happiness. This indicates that despite producing matchings with marginally worse pairwise user happiness, the $\text{PRSTA}_{0.9}$ algorithm can produce more balanced matchings, in which the degree of unhappiness of the users that form at least one unhappy pair is lower. This is simply because of reducing the risk levels by setting $\alpha = 0.9$, and seeking to match users with possibly not perfect, but good enough candidates.

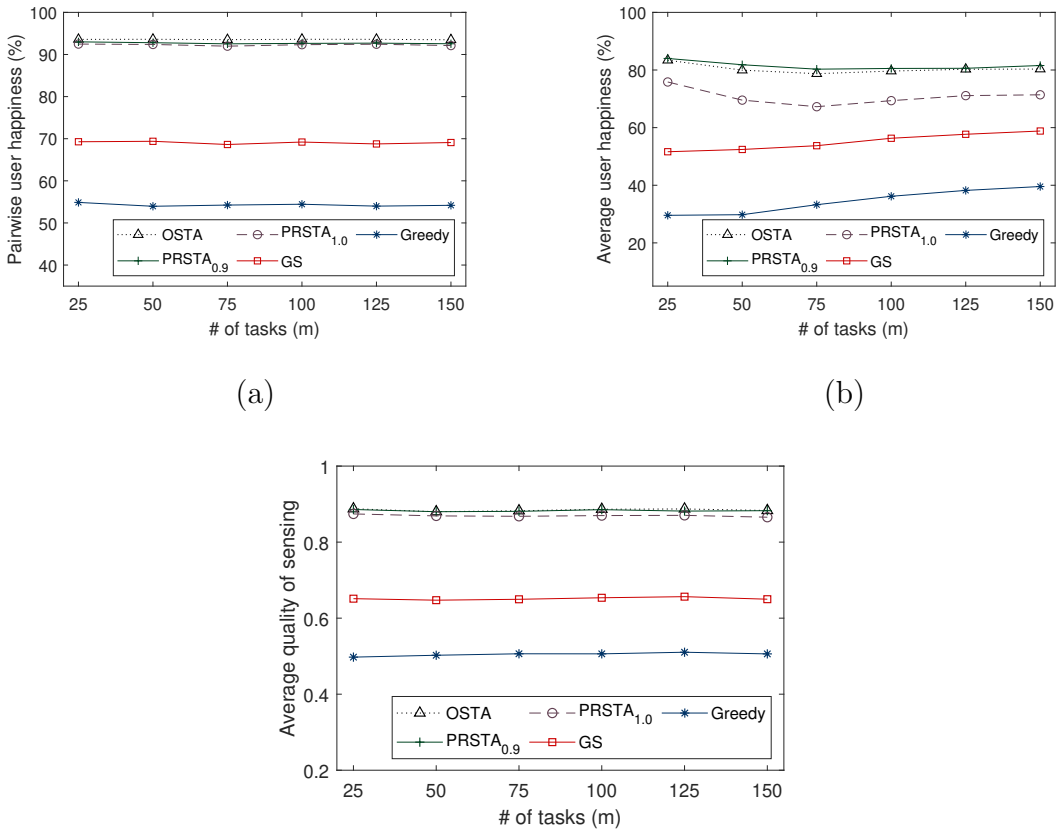


Fig. 32. Performance comparison of the algorithms against varying number of tasks in systems without capacity constraints in the synthetic data set ($n = 60$).

In Fig. 31c, we see that the proposed algorithms achieve better average quality of sensing scores with increasing number of workers, because as the number of workers increases, there will also be more high-quality workers. However, the GS and Greedy algorithms do not benefit much from this significantly as the former uses an inaccurate approximation for the expected user utilities, and the latter simply ignores the matching opportunities that may come in the future.

In Fig. 32, we examine the performance of the algorithms with various task counts in the systems without capacity constraints. Since the workers do not have a capacity constraint, increasing the number of tasks does not escalate the competition

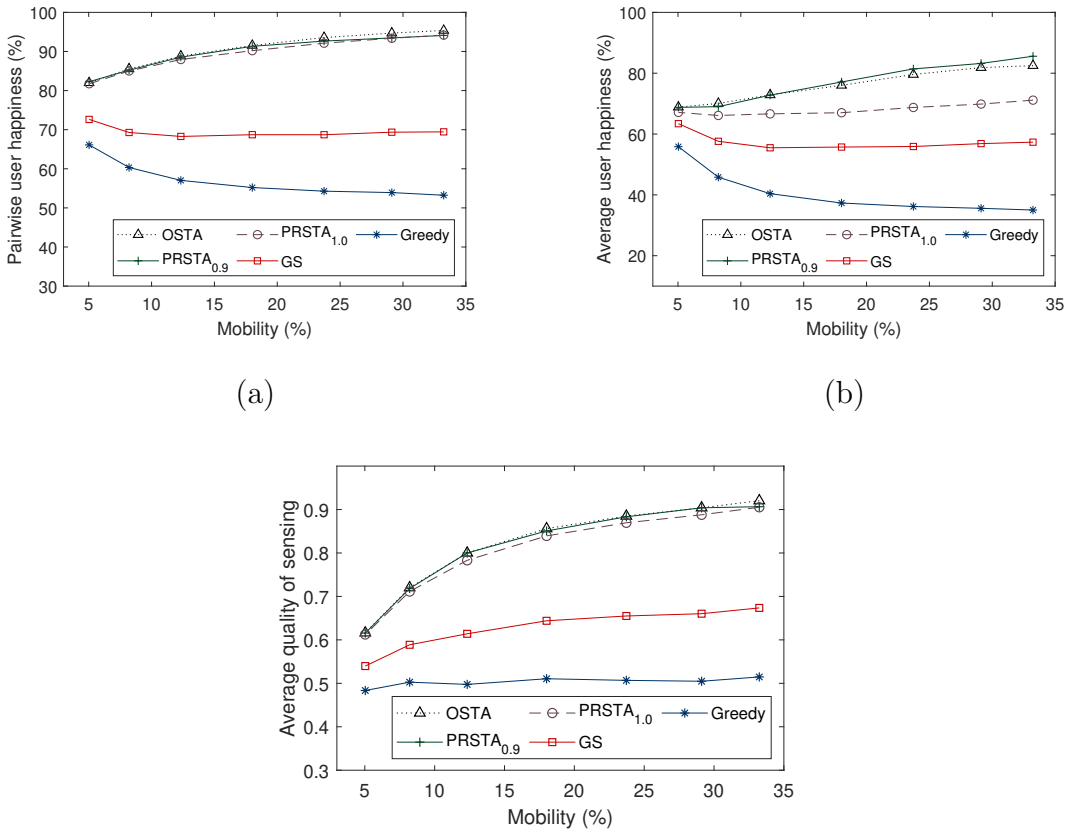


Fig. 33. Performance comparison of the algorithms against varying degree of mobility in systems without capacity constraints in the synthetic data set $(m = 100, n = 60)$.

between tasks (unlike what we will see in the presence of capacity constraints), thus we do not see big differences in the performance of the algorithms with the exception that the proposed algorithms perform slightly worse, and the others slightly better in terms of average user happiness.

Next, in Fig. 33, we look at the performance of the algorithms against varying degree of mobility, which is defined as the average percentage of the task regions visited by each worker. We generate instances with different mobility levels (i.e., percentage of all task regions visited by each worker, on average) by adjusting the

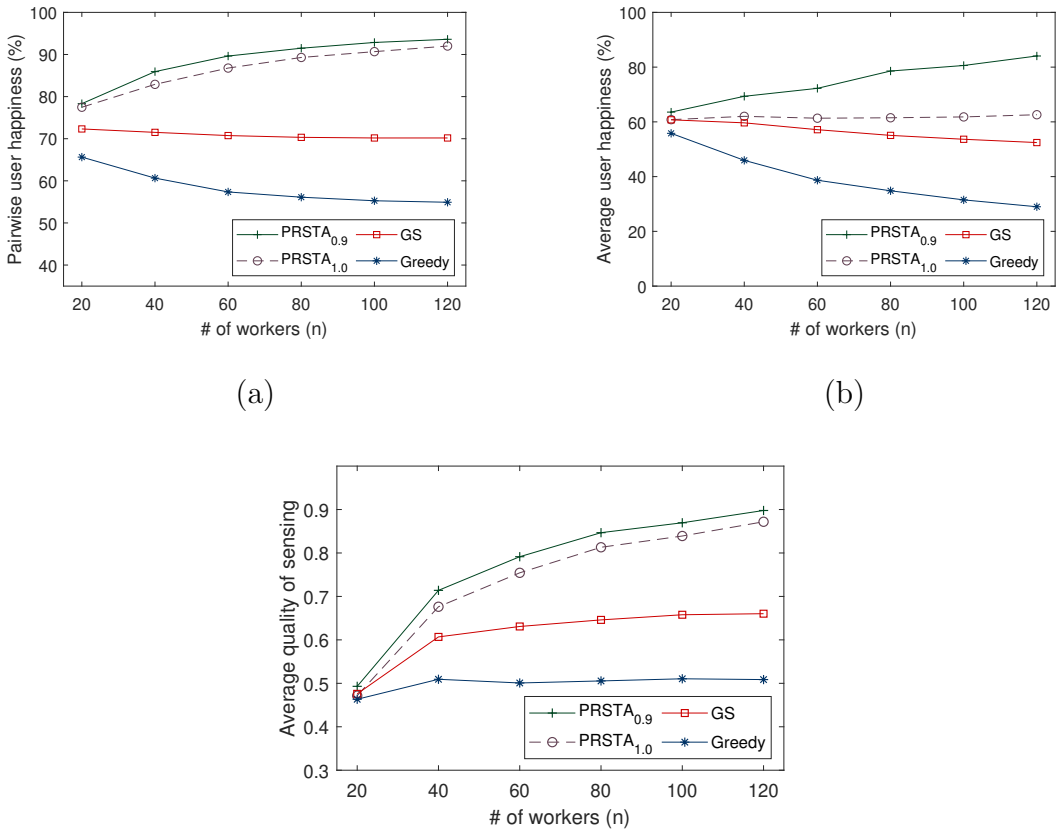


Fig. 34. Performance comparison of the algorithms against varying number of workers in systems with capacity constraints in the synthetic data set ($m = 100$).

range of the $\lambda_{i,j}$ values for worker-task pairs (e.g., increasing the average value of $\lambda_{i,j}$ results in lower mobility). As expected, with higher mobility, the high-quality workers visit more task regions, hence we see a profound increase in the average quality of sensing scores of the proposed algorithms. A remarkable point is that the GS and Greedy algorithms produce matchings with worse pairwise/average user happiness scores with increasing mobility, because the amount of better matching opportunities to be seen in the future, which are mostly neglected by these algorithms, becomes larger with increasing mobility.

In Fig. 34, 35, and 36, we present the performance comparison of the algorithms

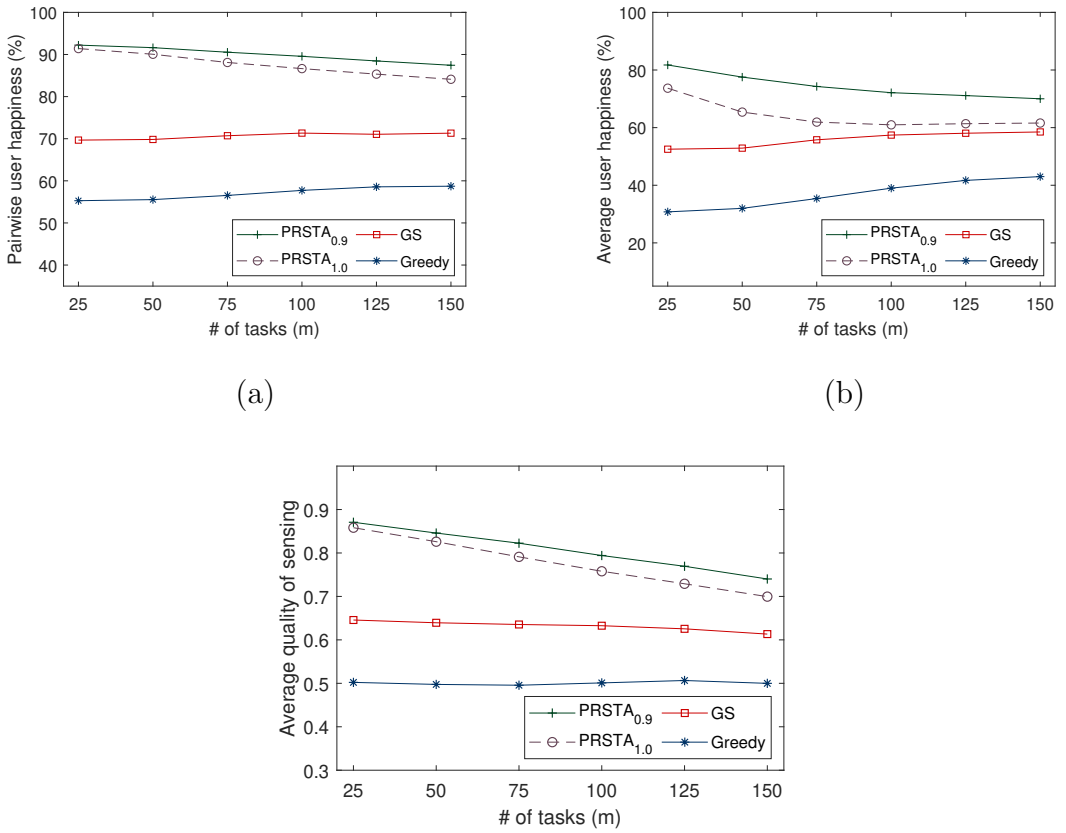


Fig. 35. Performance comparison of the algorithms against varying number of tasks in systems with capacity constraints in the synthetic data set ($n = 60$).

in the MCS systems with capacity constraints (note that there is no result for the OSTA algorithm, as it can only be run in the systems without capacity constraints). Fig. 34 shows the performance of the algorithms with various worker counts. Although the relative performance of the algorithms is similar to the case without capacity constraints (Fig. 31), the quality of the produced matchings is generally slightly worse in terms of all performance metrics. This is because the high-quality workers will not be able to perform as many tasks as possible in this scenario, and the propriety of each matching decision becomes more important as there will be only a limited number of opportunities to make up for the previous decisions.

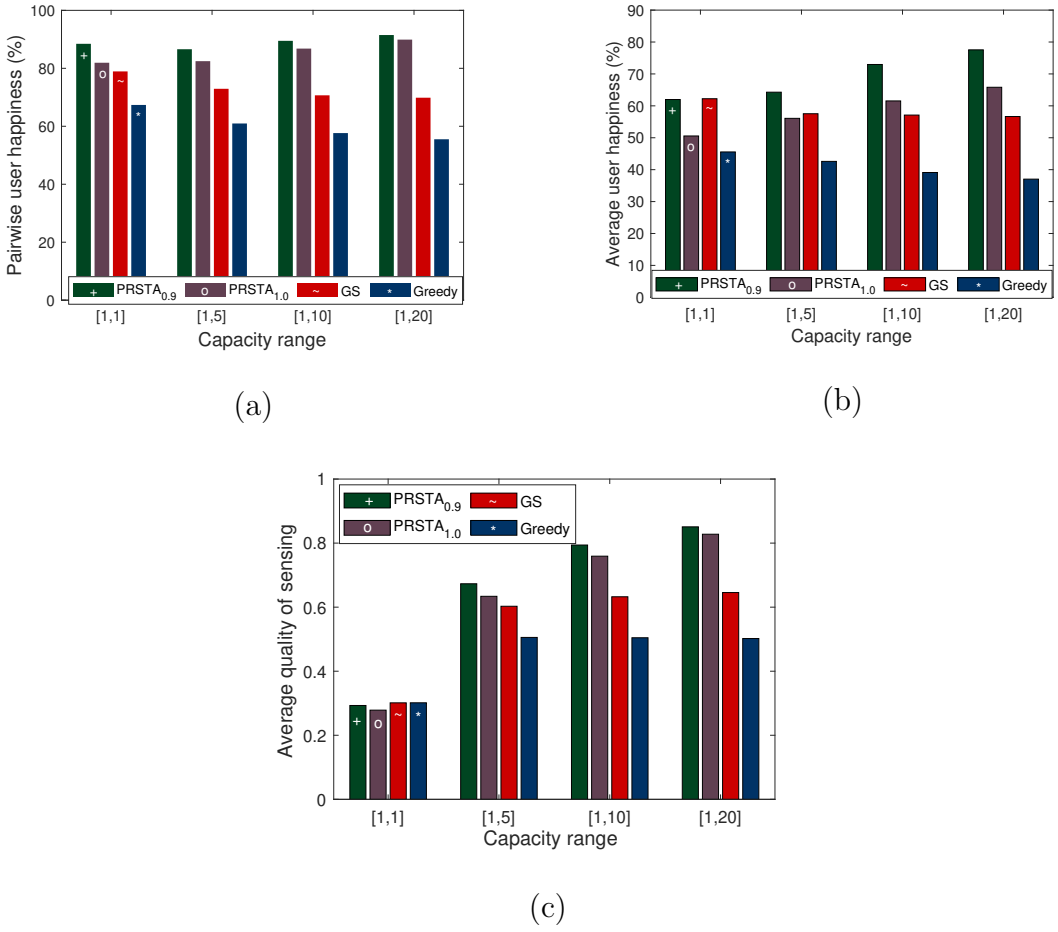


Fig. 36. Performance comparison of the algorithms against varying ranges of worker capacities in the synthetic data set ($m = 100, n = 60$).

We inspect how the algorithms perform with varying number of tasks in presence of capacity constraints in Fig. 35. Different from the case without capacity constraints (Fig. 32), the user happiness and average quality of sensing achieved by the proposed algorithms get worse with increasing task counts, because, in this case, there is a competition between tasks as the high-quality workers can be matched to only a small number of tasks.

Another noteworthy point is that increasing the number of tasks has a different impact on the performance of the proposed algorithms and the others in terms of

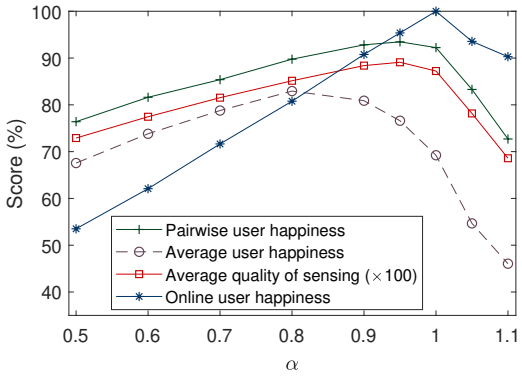


Fig. 37. The impact of α on the performance of the PRSTA$_\alpha$ algorithm in the synthetic data set without capacity constraints ($m = 100, n = 60$).

average user happiness. That is, the proposed algorithms perform slightly worse, while the GS and Greedy algorithms perform slightly better. This is due to the fact that the tasks will be, on average, matched to the workers with low quality scores or will be even unmatched when the number of tasks is large. This makes the cost of missing a present matching opportunity in terms of user happiness bigger, and the proposed algorithms consequently suffer as they frequently disregard the present matching opportunities to wait for better ones.

In Fig. 36, we analyze the effect of extending the worker capacity ranges on the performance of the algorithms. We observe that the proposed algorithms always outperform the others in terms of pairwise user happiness, and the performance difference becomes more significant with increasing worker capacities. However, when each worker can be matched with only a single worker, the GS algorithm has a similar performance with the PRSTA$_{0.9}$ algorithm in terms of average user happiness. Besides, the GS and Greedy algorithms achieve comparable average quality of sensing scores with the proposed algorithms when each worker has a capacity of one. This is because they have a lower risk of leaving the workers completely unmatched by skipping the existing matching opportunities, and this compensates for the loss of quality

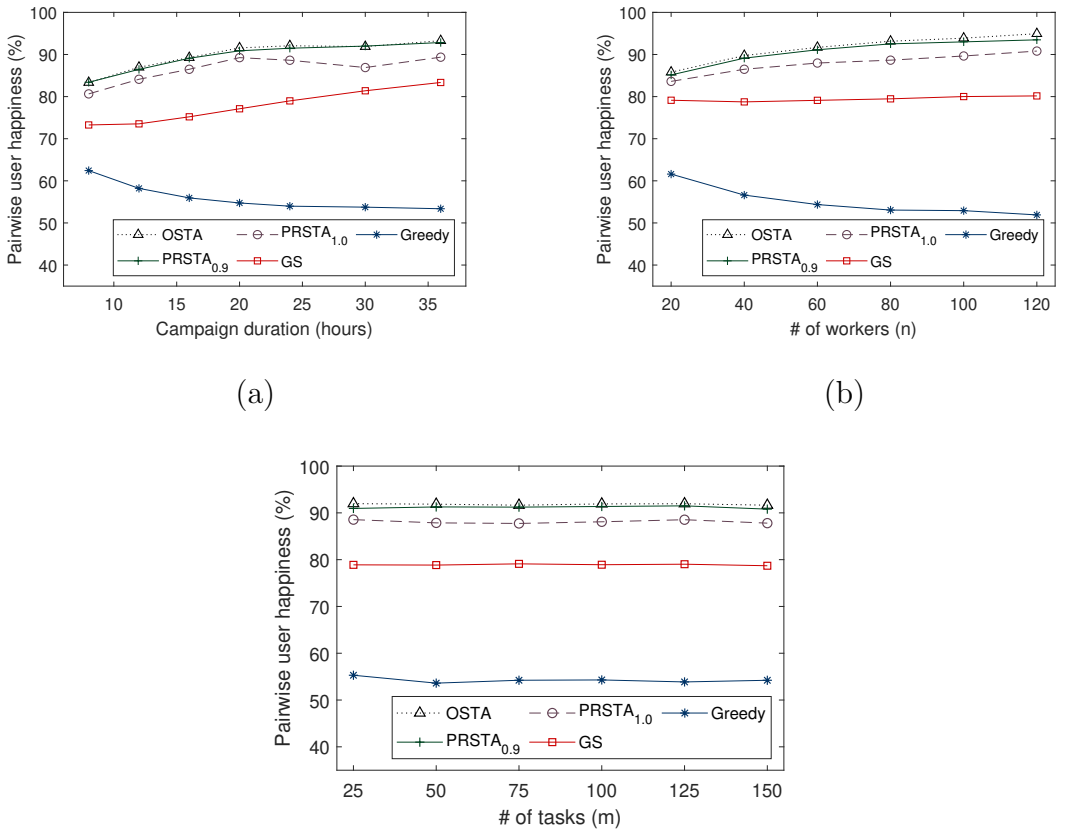


Fig. 38. Performance comparison of the algorithms in the SF taxi data set without capacity constraints ($m = 100, n = 60$).

of sensing caused by the poor matching decisions they otherwise tend to make (as we see with larger worker capacities).

In the synthetic data set, we lastly look at the performance of the PRSTA$_\alpha$ algorithm with various values of α parameter as shown in Fig. 37. We see that the algorithm produces optimal task assignments in terms of online user happiness when $\alpha = 1$ (as proven in Corollary 11.1), and that it achieves the best performance in terms of all other metrics when α is between 0.8 and 1. When we further decrease the value of α, the algorithm starts to match the worker-task pairs greedily (when $\alpha = 0$, it is in fact practically the same with the Greedy algorithm), while it misses

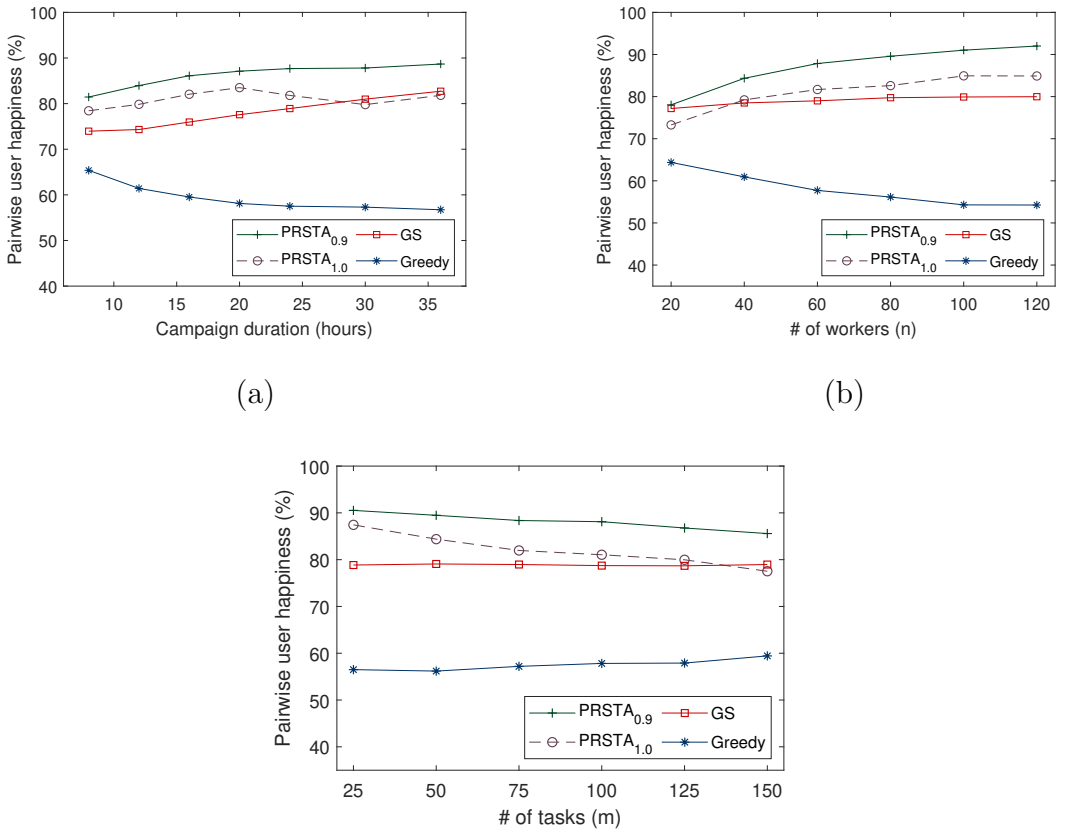


Fig. 39. Performance comparison of the algorithms in the SF taxi data set with capacity constraints ($m = 100, n = 60$).

too many existing matching opportunities to wait for substantially better ones when we use an α value greater than 1.

In Fig. 38, 39, & 40, we present the performance of the algorithms in terms of pairwise user happiness in the real data set without and with capacity constraints, respectively. In both figures, the performance of the proposed algorithms and GS algorithm mostly improve with the extended campaign duration, yet that of the Greedy algorithm gets consistently worse as it makes almost all of the assignments right in the beginning of the campaign without considering potential opportunities that may come later. In Fig. 38a & 39a, we see a slight fluctuation in the performance

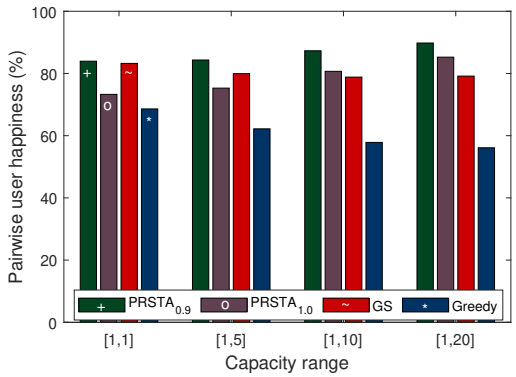


Fig. 40. Performance comparison of the algorithms with varying capacity ranges in the SF taxi data set ($m = 100, n = 60$).

of the $\text{PRSTA}_{1.0}$ algorithm when the campaign duration is between 20-30 hours. This is mostly because of the changes in the movement patterns of the taxis between two consecutive days. For instance, the taxis are likely to have different visit patterns on Friday and Saturday as the former is a business day and the latter is not.

In Fig. 38, we observe that the relative performance of the algorithms and the impact of worker/task counts on the performance of all algorithms are quite similar to what we have seen in the synthetic data set (Fig. 31 & 32). In fact, the only major difference is that the GS algorithm achieves notably higher pairwise user happiness scores (by about 10%) in the real data set. Moreover, its performance is also significantly better with capacity constraints (Fig. 39) so that it even slightly outperforms the $\text{PRSTA}_{1.0}$ algorithm when the ratio of the number of tasks to the number of workers is larger than 2. Yet it should be noted that it is always outperformed by the $\text{PRSTA}_{0.9}$ algorithm.

Finally, in Fig. 41, we look at the running times of the algorithms with varying worker/task counts and capacity ranges on an Intel core i7 processor with a memory

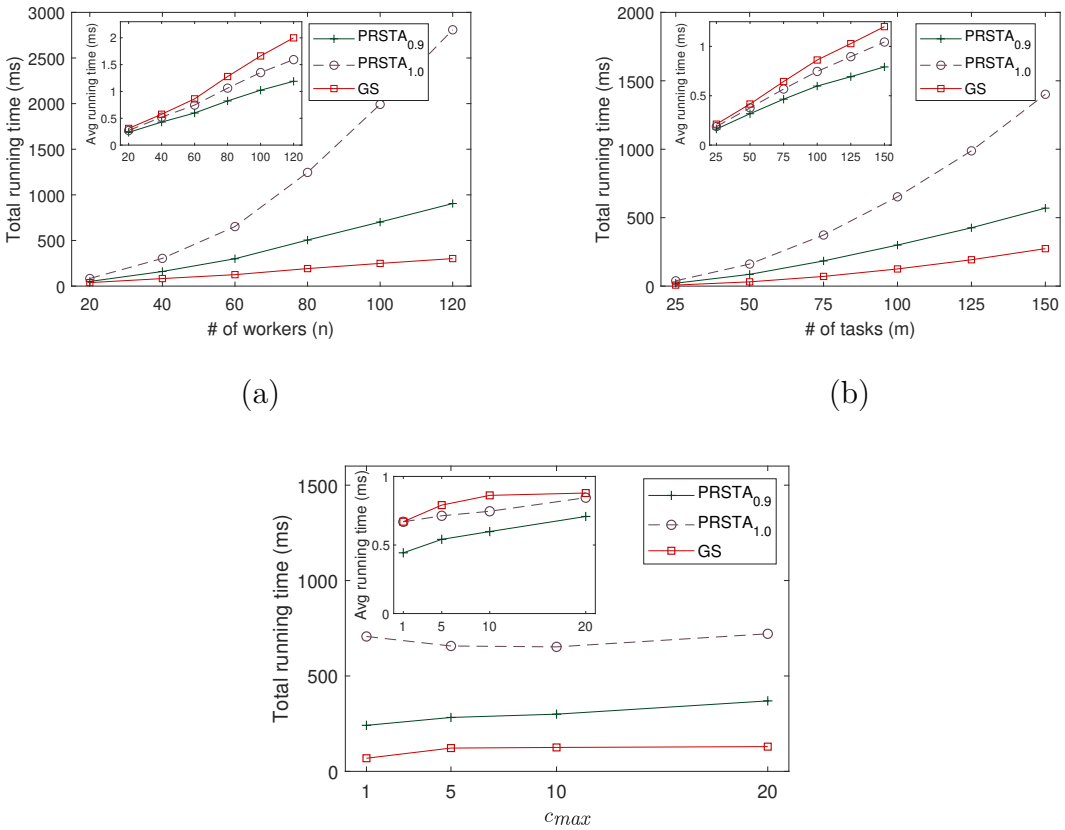


Fig. 41. Running times of the algorithms with varying worker (a) and task (b) counts; and varying ranges [1,c_{max}] of worker capacities with $m = 100, n = 60$ (c) . The total and average running times refer to the total time spent in making matching decisions throughout the campaign, and the average time spent per matching decision, respectively.

of 16 GB and a speed of 2.5 GHz[7]. We only present the running times for the synthetic data set as the comparison of running times of algorithms in the real data set does not exhibit any remarkable difference (except for the naturally larger total running

[7]Since the OSTA and Greedy algorithms make the matching decisions in constant time, we do not present their running times. One-time cost of obtaining $\mathbf{E}_j(s)$ values for the OSTA algorithm is also very small (e.g., 32 ms, which is about 10% of the total running time of the GS algorithm, when $m = 150$ and $n = 60$) in the setting without capacity constraints.

times due to the longer campaign duration). Recall that the worst-case running time of the PRSTA$_\alpha$ algorithm is $O(nmc_{max})$ (or $O(nm^2)$), and that of the GS algorithm is $O(mn)$. We first note that in all cases the total running time of the GS algorithm is lower than the PRSTA$_\alpha$ algorithms, while its average running time per decision is higher. This is because the GS algorithm makes the matching decisions more greedily compared to the PRSTA$_\alpha$ algorithms, thus it matches most of the workers and tasks in the beginning of the campaign, which means that it will not be run again for these users, reducing the number of times it will be run in total. On the other hand, the PRSTA$_\alpha$ algorithms have smaller average running times per decision, because they are run much more frequently after the first part of the campaign where there are generally fewer worker-task pairs that can still get matched.

Moreover, the PRSTA$_{1.0}$ algorithm has a significantly larger total running time than the PRSTA$_{0.9}$ algorithm as the former has a stronger requirement to match a pair, and consequently will be run considerably more times compared to the latter. This is also the reason behind why we see an almost quadratic increase in the total running time of the PRSTA$_{1.0}$ algorithm with increasing worker and task counts. That is, when the number of workers/tasks increases, there will be more visits, and the PRSTA$_{1.0}$ algorithm will need to be run even more frequently. Lastly, since workers will be less selective when they have higher capacities, and accordingly tasks will end up getting matched earlier, the total/average running times of the algorithms do not get significantly larger with increasing capacities as seen in Fig. 41c, even though the worst-case running time of the PRSTA$_\alpha$ algorithm (i.e., $O(nmc_{max})$) hints at a linear grow with increasing worker capacities.

5.5 Conclusion

In this chapter, we studied the task assignment problem in opportunistic MCS. First, we presented a complete system model considering the uncertainty in worker trajectories and capacity constraints of workers, and formally defined the preference-aware/stable task assignment problem. We then demonstrated how to efficiently examine all practical scenarios for assignment opportunities, which arise when the workers visit the task regions, to compute the expected utilities of the task requesters and workers with and without capacity constraints. Finally, we proposed polynomial-time task assignment algorithms that are proven to be preference-aware, and showed via extensive simulations that they significantly outperform the existing solutions in terms of worker/task requester happiness and quality of sensing.

CHAPTER 6

THREE-DIMENSIONAL TASK ASSIGNMENT IN SEMI-OPPORTUNISTIC MOBILE CROWDSENSING

6.1 Introduction

As we have seen in the previous chapters, the key issue in the participatory sensing is that the paths assigned to workers are likely to disturb their daily schedules and to introduce significant additional travel costs, whereas the opportunistic sensing 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 sensing, a new sensing mode, namely *semi-opportunistic*, has been proposed recently [38]. In this novel mode, workers provide the matching platform with alternative paths they would be willing to take within their comfort zones in addition to the path they would normally take (e.g., dashed lines in Fig. 42). 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 workers can carry out, allowing them to increase their profits by performing more tasks.

In this chapter, we study the preference-aware 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

Fig. 42. Example paths of a user for different sensing modes.

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 (i.e., 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 chapter can be summarized as follows:

- We provide a formal definition of the preference-aware task assignment problem in an SO-MCS system, and 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.

6.2 System Model

6.2.1 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

$$\mathcal{T}_{i,j} = \{t_k : t_k \in \mathcal{T} \text{ and } t_k \text{ is on } p_{i,j}\}. \tag{6.1}$$

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. 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. 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:

$$L_j^t = w_{\sigma_1}, w_{\sigma_2}, \ldots, w_{\sigma_k} \text{ where } q_{\sigma_i,j} \geq q_{\sigma_{i+1},j}. \tag{6.2}$$

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 (6.2) 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:

$$L_{\mathcal{T}} = \hat{L}_1^t = \hat{L}_2^t = \cdots = \hat{L}_n^t. \tag{6.3}$$

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:

$$L_i^w = t_{\sigma_1}, t_{\sigma_2}, \ldots, t_{\sigma_k} \text{ where } r_{\sigma_i,i} \geq r_{\sigma_{i+1},i}. \tag{6.4}$$

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.

6.2.2 Problem Statement

Our main objective in this chapter 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 18 (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.,*

$$1 \leq |S| \leq c_{i,j}, S \subseteq L_i^w, \text{ and } S \subseteq \mathcal{T}_{i,j}, \tag{6.5}$$

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

$$w_i \in L_k \text{ and } t_k \in \mathcal{T}_{i,j}, \tag{6.6}$$

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

$$q_{i,k} > q_{h,k} \text{ where } q_{h,k} = 0 \text{ if } w_h = \varnothing, \tag{6.7}$$

  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.,*

$$\sum_{t_h \in S} r_{h,i} > \sum_{t_k \in \mathcal{M}_u(w_i)} r_{k,i}, \tag{6.8}$$

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 19 (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.

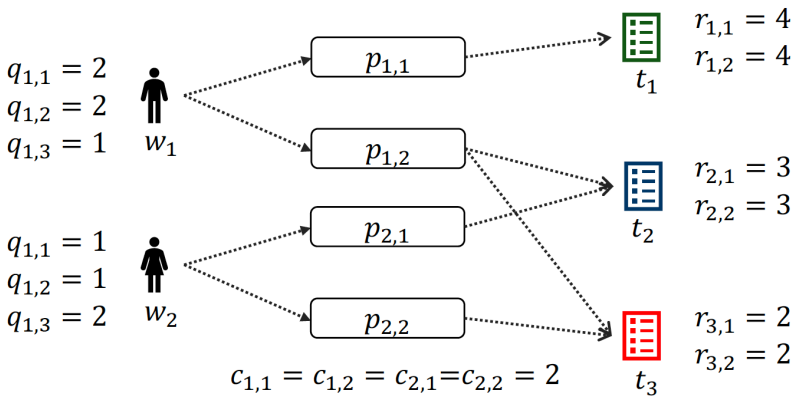


Fig. 43. An MCS instance for which no stable matching exists. There is an edge from a path $p_{i,j}$ to a task t_k if $t_k \in \mathcal{T}_{i,j}$.

Theorem 12. *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. 43. There are 11 possible task assignments in this instance, and, as shown in Fig. 44, 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:

$$\textit{maximize} \min_{x \in U(\mathcal{M})} \frac{1}{\delta_x}, \tag{6.9}$$

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

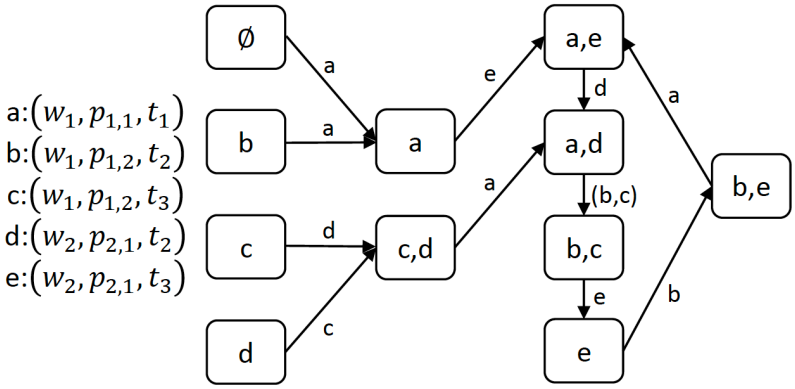


Fig. 44. Proof of Theorem 12. All possible matchings for the instance in Fig. 43 are shown with boxes. There is an edge k from matching (box) m to matching m', if k is an unhappy triad in m due to a more favorable assignment in m'.

computed by

$$\delta_x = \frac{\sum_{t_h \in S} r_{h,i}}{\sum_{t_k \in \mathcal{M}_u(w_i)} r_{k,i}}, \tag{6.10}$$

if $\mathcal{M}_u(w_i) \neq \varnothing$, otherwise $\delta_x = \infty$. 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. Note that the value of (6.9) ranges between 0 and 1, where it is 1 when the matching is perfect/stable (i.e., $U(\mathcal{M}) = \varnothing$), and decreases as the unhappiness of the users in the matching grows.

Definition 20 (α-stable matching)**.** *A matching $\mathcal{M}$ is said to be α-stable if*

$$\max_{x \in U(\mathcal{M})} \delta_x \leq \alpha. \tag{6.11}$$

A summary of the notations used throughout this chapter is presented in Table 6.2.2.

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 $\mathcal{M}$

Table 10. Notations used in Chapter 6.

6.3 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.

6.3.1 Stable Task Assignment in Uniform MCS Systems

In Algorithm 14, 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 (6.3) 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).

Theorem 13. *Algorithm 14 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 $T_1' = \mathcal{T}$.

Algorithm 14: UniformSTA

let $\mathcal{M}(u) = (\varnothing, -)$ for all $u \in \mathcal{W} \cup \mathcal{T}$
form $L_{\mathcal{T}}$ by (6.3)
for $i \leftarrow 1$ *to* m **do**
    let w_h be the ith worker in $L_{\mathcal{T}}$
    form L_h^w by (6.4)
    $A \leftarrow \{\}$, $s \leftarrow 0$, $r \leftarrow 0$
    for $j \leftarrow 1$ *to* a_h **do**
        $A' \leftarrow \{\}$, $s' \leftarrow 0$
        for $l \leftarrow 1$ *to* $|L_h^w|$ **do**
            let t_k be the lth task in L_h^w
            if $t_k \in \mathcal{T}_{h,j}$ *and* $\mathcal{M}_u(t_k) = \varnothing$ **then**
                append t_k to A'
                $s' \leftarrow s' + r_{k,h}$
            if $|A'| = c_{h,j}$ **then**
                break
        if $s' > s$ **then**
            $A \leftarrow A'$, $s \leftarrow s'$, $r \leftarrow j$
    $\mathcal{M}(w_h) \leftarrow (A, p_{h,r})$
    foreach $t \in A$ **do**
        $\mathcal{M}(t) \leftarrow (w_h, p_{h,r})$
return $\mathcal{M}$

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

$$S \cap (\mathcal{T} \setminus T'_k) = \varnothing. \tag{6.12}$$

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 (6.2) and (6.3), which contradicts the unhappy triad definition due to (6.7). Then, by (6.12), 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

$$\sum_{t_x \in \mathcal{M}_u(w_h)} r_{x,h} \geq \sum_{t_y \in S} r_{y,h}. \tag{6.13}$$

This also contradicts the unhappy triad definition due to (6.8), hence we conclude that such an unhappy triad cannot exist in the matching produced by Algorithm 14. □

As a result of Theorem 13, we obtain the following corollary.

Corollary 13.1. *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 14 is $O(mna_{max} + mn \log n + m \log m)$.

6.3.2 Stable Task Assignment in General MCS Systems

In Algorithm 15, 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 line 2. 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 the execution of the algorithm, all tasks that are currently unmatched and are not yet attempted to be matched to each worker in their preference lists, i.e.,

$$\forall t_k \in \mathcal{T} : \mathcal{M}_u(t_k) = \varnothing \text{ and } index_k \leq |L_k^t|, \tag{6.14}$$

Algorithm 15: GeneralSTA ($\mathcal{W}$, $\mathcal{T}$)

let $\mathcal{M}(u) = (\varnothing, -)$ for all $u \in \mathcal{W} \cup \mathcal{T}$
let $x_i = \sigma_i = 0$ for all $1 \le i \le m$, and $index_k = 1$ for all $1 \le k \le n$
Stack.push($\mathcal{T}$)
while *Stack is not empty* **do**
    $t_k \leftarrow$ *Stack.pop()*
    if $index_k \le |L_k^t|$ **then**
        let w_i be the ($index_k$)th worker in L_k
        $index_k \leftarrow index_k + 1$
        $A \leftarrow \{\}$, $R \leftarrow \{\}$, $r \leftarrow 0$
        for $j \leftarrow 1$ *to* a_i **do**
            if $t_k \notin \mathcal{T}_{i,j}$ **then continue**;
            A', R', $\sigma' \leftarrow$ FindPathOptimal$(i, j, k, \mathcal{M})$
            let t_h be the first task in A'
            if $r_{h,i} \times c_{i,j} > x_i$ *or* $(r_{h,i} \times c_{i,j} = x_i$ *and* $\sigma' > \sigma_i)$ **then**
                $x_i \leftarrow r_{h,i} \times c_{i,j}$, $A \leftarrow A'$, $R \leftarrow R'$, $\sigma_i \leftarrow \sigma'$, $r \leftarrow j$
        if $|A| > 0$ **then**
            let $\mathcal{M}(w_i) = (A, p_{i,r})$, and $\mathcal{M}(t) = (w_i, p_{i,r})$ for all $t \in A$
            let $\mathcal{M}(t) = (\varnothing, -)$ for all $t \in R$
            Stack.push(R)
        else
            Stack.push(t_k)
return $\mathcal{M}$

Algorithm 16: FindPathOptimal $(i, j, k, \mathcal{M})$

```
1  A' ← {}, R' ← {}, σ' ← 0
2  for l ← 1 to |M_u(w_i)| do
3      let t_h be the lth task in M_u(w_i)
4      if t_h ∈ T_{i,j} and |A'| < c_{i,j} then
5          append t_h to A'
6          σ' ← σ' + r_{h,i}
7      else
8          append t_h to R'
9  insert t_k into A' by maintaining non-increasing order of task rewards
10 σ' ← σ' + r_{k,i}
11 if |A'| > c_{i,j} then
12     let t_h be the last task in A
13     remove t_h from A'
14     append t_h to R'
15     σ' ← σ' − r_{h,i}
16 return (A', R', σ')
```

reside in a stack that is initialized in line 3. In the while loop starting in line 4, 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 10. 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 10-11), 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 $p_{i,j}$ by running an algorithm called FindPathOptimal (line 12), which is described in Algorithm 16. We then choose the task set A' over the task set A, and update the variables A, R, x_i, σ_i, and r accordingly (lines 14-15) if one of the following two conditions is satisfied:

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

Finally, in lines 16-21, 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 14. *Algorithm 15 always produces a κ-stable matching for a general MCS instance, where κ is the maximum path capacity in the instance, i.e.,*

$$\kappa = \max_{\substack{1 \le i \le m \\ 1 \le j \le a_i}} c_{i,j}. \tag{6.15}$$

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

$$\sum_{t_x \in \mathcal{M}_u(w_i)} r_{x,i} \times \kappa < \sum_{t_y \in S} r_{y,i}. \tag{6.16}$$

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 (6.7), 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 7), 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 14) 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 15), which will have the maximum value possible in the end. Thus, we have

$$\forall t_e \in E : x_i \geq r_{e,i} \times \max_{1 \leq h \leq a_i} c_{i,h}, \tag{6.17}$$

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

$$\begin{aligned} \forall t_s \in S : x_i &\geq r_{s,i} \times \max_{1 \leq h \leq a_i} c_{i,h}, \\ &\geq r_{s,i} \times c_{i,j}. \end{aligned} \tag{6.18}$$

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 (6.18), we get

$$\forall t_s \in S : r_{m,i} \times c_{i,g} \geq r_{s,i} \times c_{i,j}. \tag{6.19}$$

For the condition in (6.16) to hold, we must have

$$\sum_{t_x \in \mathcal{M}_u(w_i)} r_{x,i} \times \max\{c_{i,g}, c_{i,j}\} < \sum_{t_y \in S} r_{y,i}, \tag{6.20}$$

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

$$\sum_{t_x \in \mathcal{M}_u(w_i)} r_{x,i} \times c_{i,g} < \sum_{t_y \in S} r_{y,i} \tag{6.21a}$$

$$r_{m,i} \times c_{i,g} < \sum_{t_y \in S} r_{y,i} \text{ (by (i))} \tag{6.21b}$$

$$\forall t_s \in S : r_{s,i} \times c_{i,j} < \sum_{t_y \in S} r_{y,i} \text{ (by (6.19))}. \tag{6.21c}$$

For the task $t_{s'}$ with the highest reward in S, (6.21c) yields

$$r_{s',i} \times c_{i,j} \quad < \quad \sum_{t_y \in S} r_{y,i}, \tag{6.22}$$

which is a contradiction, because 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) $c_{i,g} \leq c_{i,j}$, we would have

$$\sum_{t_x \in \mathcal{M}_u(w_i)} r_{x,i} \times c_{i,j} < \sum_{t_y \in S} r_{y,i} \tag{6.23a}$$

$$r_{m,i} \times c_{i,j} < \sum_{t_y \in S} r_{y,i} \text{ (by (i))} \tag{6.23b}$$

$$r_{m,i} \times c_{i,g} < \sum_{t_y \in S} r_{y,i} \text{ (by (ii))} \tag{6.23c}$$

which also leads to a contradiction, as (6.23c) is identical to (6.21b). Therefore, we conclude that there cannot exist any unhappy triad that violates κ-stability in the matching produced by Algorithm 15, and it is always κ-stable. □

Running time. Algorithm 15 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 $|L_k^t| \leq m$ times, so the while loop starting in line 4 will iterate $O(mn)$ times. The for loop starting in line 10 will iterate at most $a_{max} = \max_{1 \leq i \leq m} a_i$ times, and the most expensive operation in it is running

Algorithm 16, which has a cost of $O(c_{max})$, where $c_{max} = \max_{1 \leq i \leq m} a_i$, as the size of the assignment set $\mathcal{M}_u(w_i)$ iterated in line 2 of this algorithm cannot be larger than the maximum path capacity c_{max} in the given instance. Thus, the worst-case running time of Algorithm 15 is $O(nm \log m + nmc_{max}a_{max})$.

6.4 Evaluation

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

6.4.1 Simulation Settings

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 [97] 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 [98]. 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

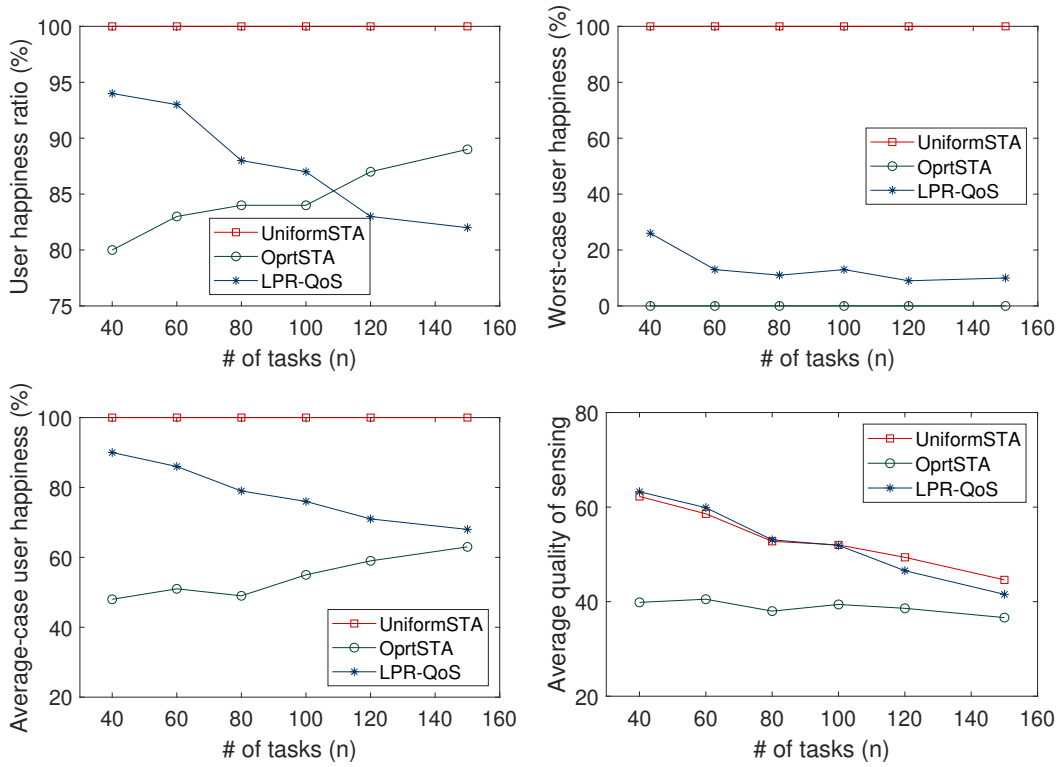


Fig. 45. Performance of algorithms with varying task counts in uniform MCS instances (m=30).

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.

6.4.1.1 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 [42] algorithm in $O(mn)$ time.

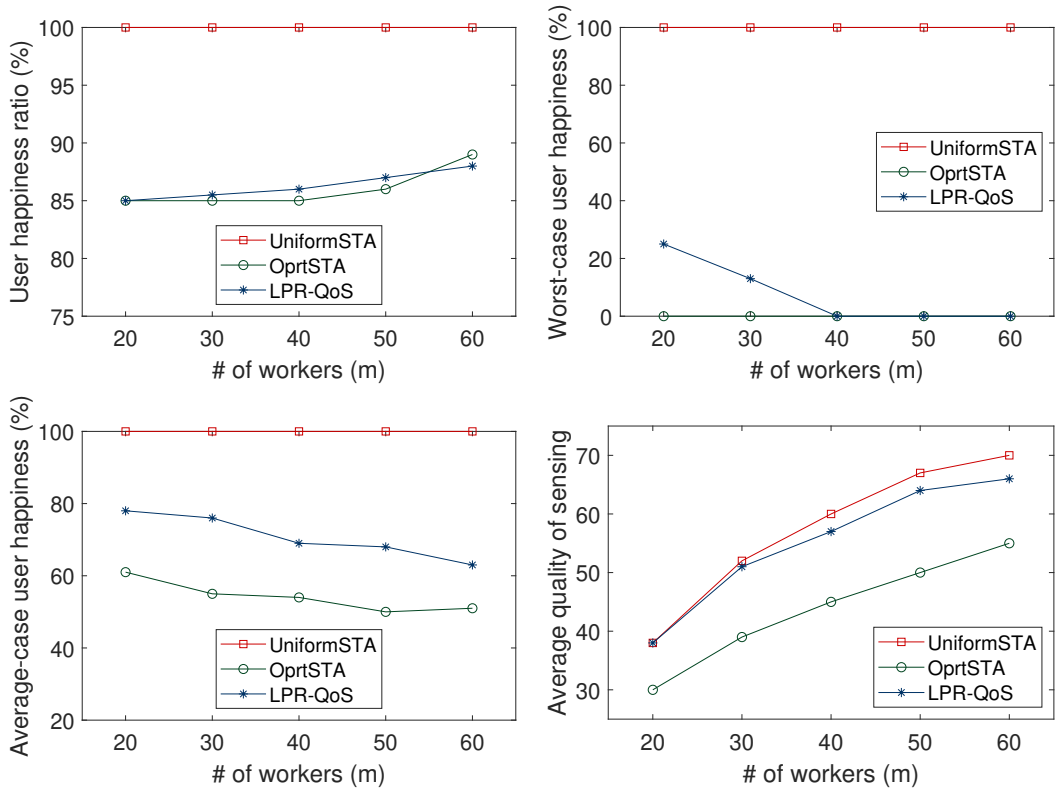


Fig. 46. Performance of algorithms with varying worker counts in uniform MCS instances (n=100).

- *LPR-QoS* [38]: 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 [99] to implement this algorithm.

6.4.1.2 Performance Metrics

- *User happiness ratio:* The ratio of the number of unhappy triads 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

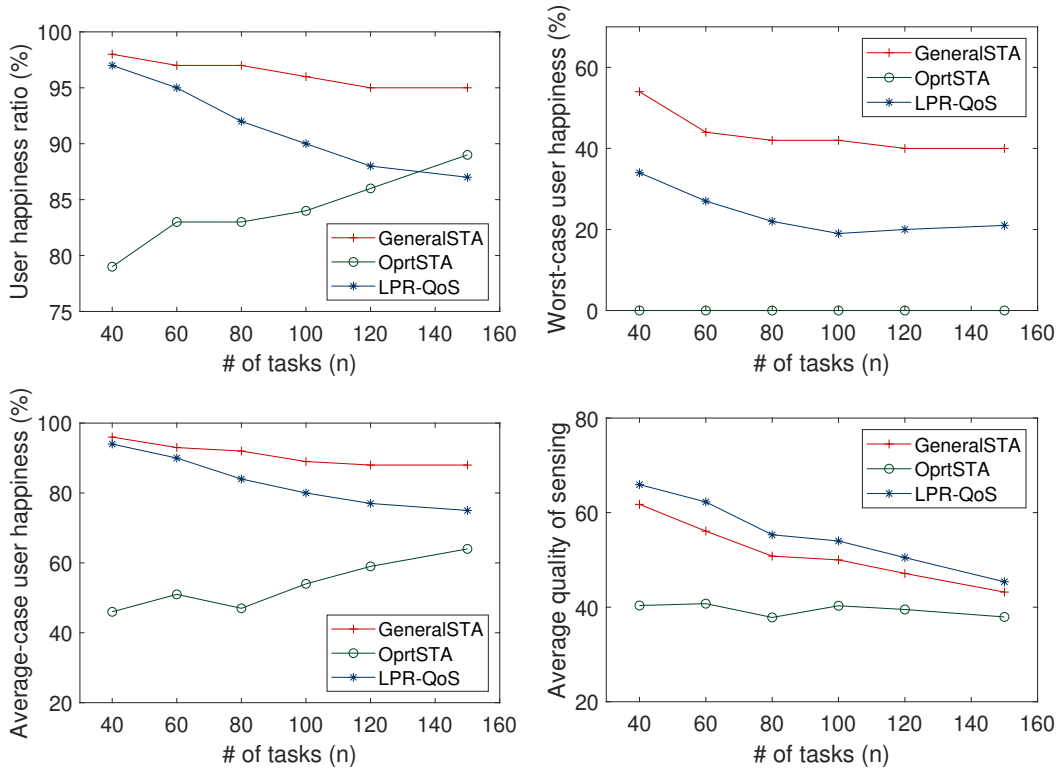


Fig. 47. Performance of algorithms with varying task counts in general MCS instances (m=30).

in (6.9), 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.

6.4.2 Results

We first look at the performance of the algorithms in the uniform instances with varying numbers of tasks (Fig. 45) and workers (Fig. 46). As expected (due

to Theorem 13), 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. However, with increasing task counts, we observe a notable decrease in the ratio of unhappy triads in the matchings produced by the OprtSTA algorithm, as each worker is likely to have more assignment opportunities on their optimal path (i.e., only path considered in opportunistic sensing), hence is less likely to form unhappy triads with the tasks on their alternative paths. We observe the exact opposite of this for the LPR-QoS algorithm, because this algorithm does not consider worker preferences whatsoever, and consequently produces task assignments with more unhappy worker-task pairs as the number of tasks in the instance increases, and it ends up disregarding worker preferences over a larger set of tasks.

In Fig. 47 & 48, 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 achieved by 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 al-

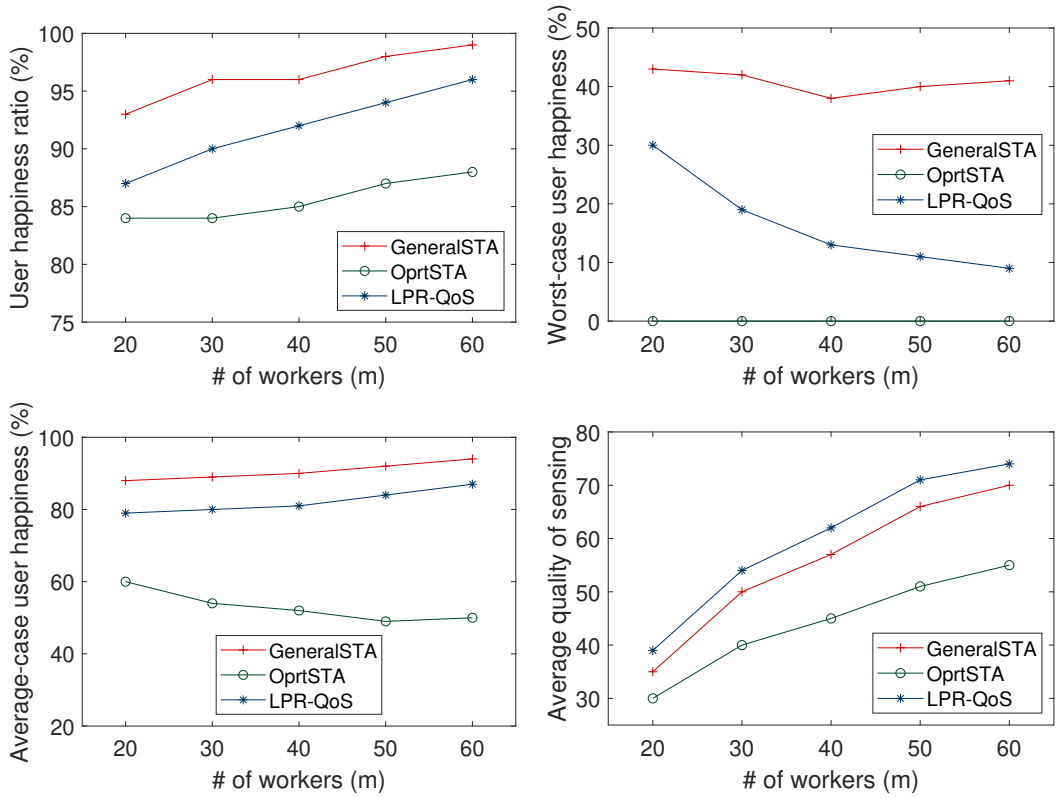


Fig. 48. Performance of algorithms with varying worker counts in general MCS instances (n=100).

ternative path counts ($a_i - 1$) and path capacities ($c_{i,1}$) in Fig. 49. 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. 50, 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

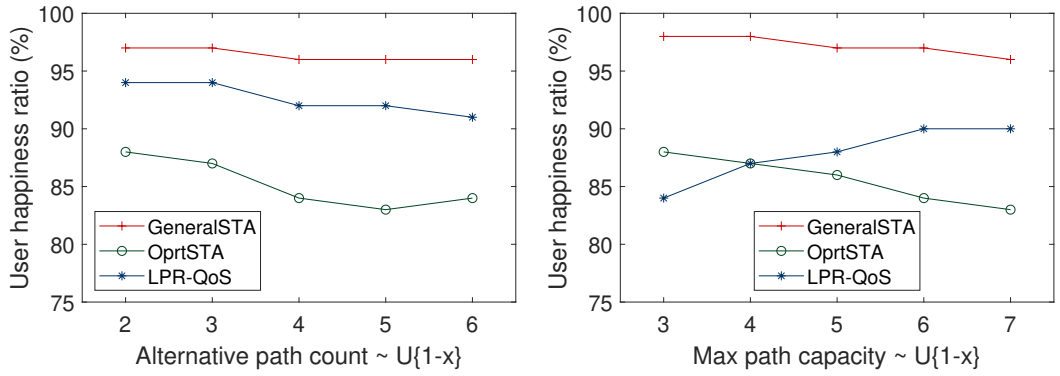


Fig. 49. User happiness (n=80, m=30) with varying path counts and capacities.

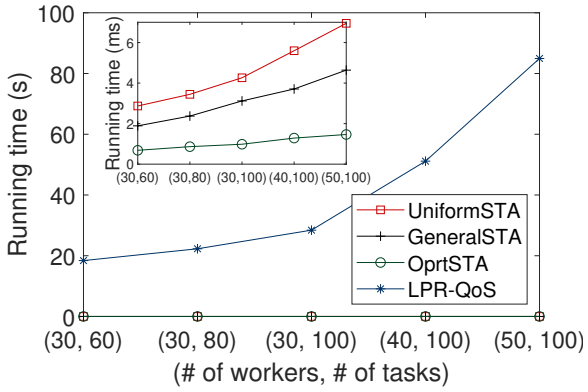


Fig. 50. Running times of algorithms with varying worker/task counts.

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.

6.5 Conclusion

In this chapter, 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, which are performed on instances using real routes from Google Maps, have shown that the proposed algorithms achieve to produce task assignments with significantly larger user happiness scores compared to the benchmark algorithms.

CHAPTER 7

PREFERENCE-AWARE MAXIMUM SYSTEM UTILITY TASK ASSIGNMENT

7.1 Introduction

In most of the studies in the MCS literature, the objective in the task allocation process is set as finding a maximum system utility (e.g., number of completed tasks, quality of completed tasks, time needed to complete given tasks) assignment, and user preferences are overlooked. However, users may not want to sacrifice their individual convenience for the system utility, and thus such task assignments may not be appealing to users (i.e., both task requesters and workers) and impair their future participation. On the other hand, satisfying user preferences perfectly may make it impossible to achieve a maximum system utility assignment (e.g., the number of matched users may decrease in preference-aware task assignments).

We illustrate this trade-off between user happiness and system utility through an example MCS scenario with 5 tasks and 5 workers shown in Fig.51a, which will be referenced throughout this chapter to describe the problem and proposed solutions for convenience. We assume that workers have some serving region and they are only eligible for the tasks in that region. The preference orders of the workers and the task requesters are also provided in Fig.51b (we will talk about how users define their preferences in Section 7.2). A matching that satisfies all users based on their preferences in the sense that they cannot claim to have deserved a better assignment than their assigned partners can be found via the well-known Gale-Shapley algorithm [42]. There can be many such *stable matchings* in a single matching instance with the same

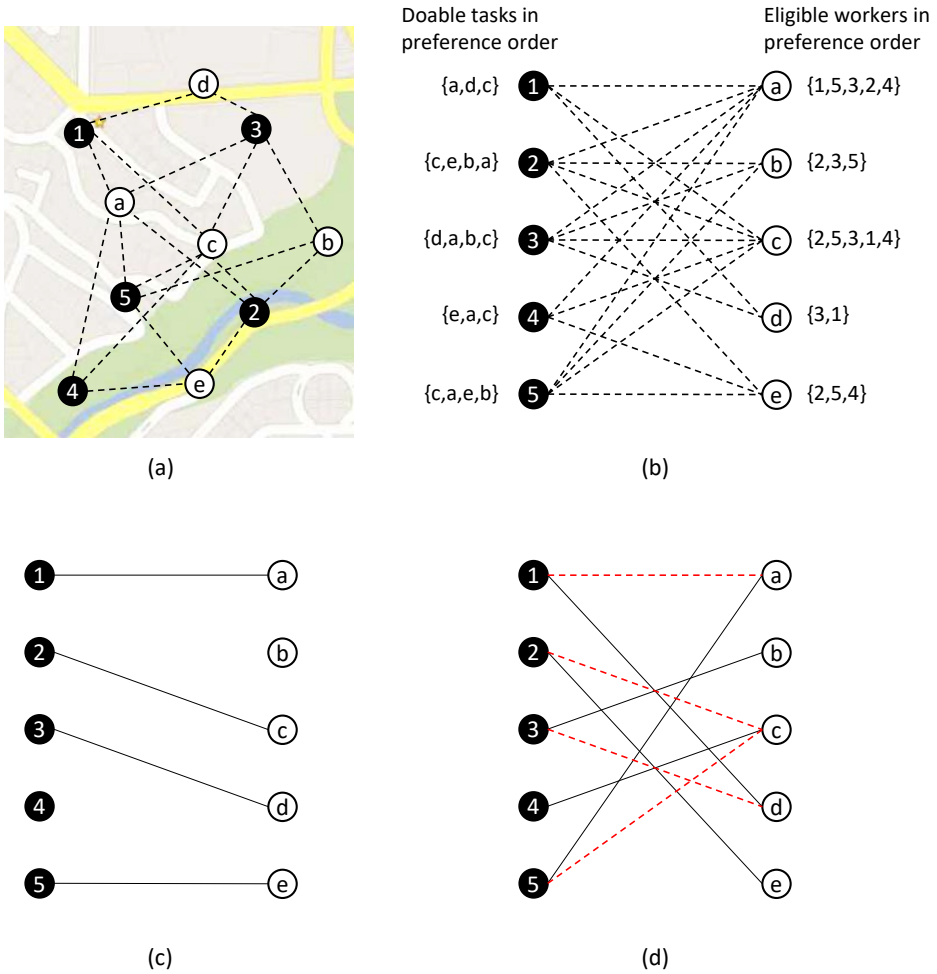


Fig. 51. An MCS scenario with 5 workers and 5 tasks, which are respectively denoted by numbers and letters. (a) Task and worker locations on the map (workers eligible to perform a task is connected with an edge to that task); (b) corresponding bipartite graph with the preference lists (from left to right) of workers and tasks; (c) a stable matching that leaves 4 and b unassigned, yielding a lower system utility; and (d) a task assignment that maximizes the system utility but yielding unhappy users (shown with dashed edges).

size, but there is only one in our example which is shown in Fig.51c. Thus, any other matching will make at least two users unhappy. On the other hand, the issue with this matching is that it leaves a worker (4) and a task (b) unassigned and hence diminishes the system utility. However, the foremost objective of a reasonable platform would be to maximize its own utility by assigning as many tasks as possible (as in Fig.51d), since it is typically paid a brokerage fee for each assignment it makes. Yet it is for the platform's own benefit to also take the preferences of users into consideration and aim to decrease the number of unhappy users with their assignments, because a user that continuously gets unhappy with his assignments is likely to abandon the platform at some point, which might have a more significant and permanent detrimental effect on the system utility.

Therefore, the platform should aim to find the matching with the minimum number of unhappy pairs (i.e., a worker-task pair preferring each other more than their current partners) without sacrificing from its own utility. For example, a matching that also achieves the maximum system utility, but with only one unhappy pair is possible in the given scenario; thus, the platform should try to produce this matching instead of the one given in Fig.51d, which contains 4 unhappy pairs. In this chapter, we address this problem of finding a maximum size task assignment with as few unhappy pairs as possible, which turns out to be NP-complete, and propose two polynomial time heuristic algorithms. Our key contributions are listed below:

- We formulate the user satisfaction aware maximum utility task assignment problem, and describe an Integer Linear Programming (ILP) model to solve it optimally.

- We propose a method that reduces the user unhappiness in a given task assignment without affecting the system utility (or the size of the assignment), and

another method that improves the system utility in a given task assignment in a way that causes as little increase in user unhappiness as possible. Then, we present two different polynomial-time task assignment algorithms based on these two new methods.

- We perform extensive simulations using a real data set, and show that our algorithms provide near-optimal results, and complement each other in different scenarios.

7.2 System Model

7.2.1 Assumptions

Let $\mathcal{W} = \{w_1, w_2, \ldots, w_n\}$ denote the set of $|\mathcal{W}| = n$ workers and $\mathcal{T} = \{t_1, t_2, \ldots, t_m\}$ denote the set of $|\mathcal{T}| = m$ tasks in the system. Also, let c_{ij} denote the cost[8] of assigning worker w_i to task t_j and r_j denote the reward of completing the task t_j. We assume that workers are rational, hence they do not perform a task if its cost is higher than the reward of the task. The set of *eligible* tasks that worker w_i can perform are defined as:

$$\mathcal{E}(w_i) = \{t_j | r_j \geq c_{ij},\ \forall j \in [1, \ldots, m]\} \tag{7.1}$$

As workers aim to increase the profit from the tasks they complete, they prefer the tasks with higher $r_j - c_{ij}$ value. We use $t_j \succ_{w_i} t_{j'}$ notation to express that w_i prefers t_j to $t_{j'}$, which happens when $r_j - c_{ij} > r_{j'} - c_{ij'}$.

[8]Note that cost can be defined with a complicated function that considers the worker's traveling and task completion duration due to spatiotemporal constraints, energy consumption on the worker's device due to sensing, and privacy risks to the worker. Similarly, reward can be defined based on several factors such as the quality of sensed data and the trustworthiness of the users.

The task requesters cannot also hire a worker if the cost of hiring that worker is more than the reward the requester can provide (which could also be considered as the budget of the requester). The set of *eligible* workers that can perform the task t_j is then similarly defined as:

$$\mathcal{E}(t_j) = \{w_i | r_j \geq c_{ij},\ \forall i \in [1, \ldots, n]\} \tag{7.2}$$

Similarly, the task requester can have preferences on the eligible set of workers. For example, if the cost of assigning a worker to a task is dependent on the traveling distance from the worker location to the task location [100, 101], the task requester may prefer the workers who have less cost, as they indicate quicker arrival of the worker to the task location and early completion of the task. It could also be a totally location-independent cost function and the preference of the task requester can be determined by other factors such as the quality of the sensed data the worker can provide. Given the eligibility relations, the corresponding undirected bipartite graph $G = (V, E)$ can be defined as

$$\begin{gathered} G.V = \mathcal{W} \cup \mathcal{T} \\ G.E = \{(u, v) | u, v \in G.V, u \in \mathcal{E}(v), v \in \mathcal{E}(u)\} \end{gathered} \tag{7.3}$$

Lastly, we use $w_i \succ_{t_j} w_{i'}$ notation to express that t_j prefers w_i to $w_{i'}$, and we assume that the preference list of a user u is the ordered list of $\mathcal{E}(u)$ in which a more favorable candidate precedes the less favorable ones, and is denoted by P_u. The notations used throughout this chapter are summarized in Table 11.

7.2.2 Problem Formulation

Given the set of eligible workers for each task and eligible tasks for each worker, the platform can assign the tasks to the workers with some optimization goal. An as-

Notation	Description
$\mathcal{W}$, $\mathcal{T}$	Set of workers and tasks, respectively.
n, m	Number of workers and tasks, respectively.
N	$\max\{m, n\}$.
$\mathcal{M}$	Matching between workers and tasks.
$\mathcal{M}(u)$	Task/worker assigned for user (worker/task) u.
$U(\mathcal{M})$	The set of unhappy pairs in matching $\mathcal{M}$.
$\lvert U(\mathcal{M})\rvert$	Unhappiness Index (UI).
$\mathcal{E}(u)$	Eligible tasks/workers for worker/task u.
$\lvert\mathcal{E}\rvert$	Average eligible task/worker size.
P_u	Preference list of user u.
c_{ij}	Cost for worker w_i to perform task t_j.
r_j	Reward of completing task t_j.
$w_i \succ_{t_j} w_{i'}$	Task t_j prefers worker w_i to worker $w_{i'}$.
$G = (V, E)$	Bipartite graph between workers and tasks.

Table 11. Notations used in Chapter 7.

signment aiming to maximize the system utility[9] can be obtained by constructing the corresponding maximum bipartite matching instance between workers and tasks, and solving it using the Hungarian [102] algorithm or the Ford-Fulkerson [103] method. Similarly, an assignment aiming to satisfy users with their assignments can be obtained using the deferred acceptance mechanism in the Gale-Shapley algorithm [42]. However, achieving both may not be possible at the same time, and there is a trade-off

[9]Since we assume that the platform is paid a brokerage fee for each assignment it makes, this refers to the number of worker-task pairs assigned.

between system utility and user satisfaction.

Let $\mathcal{M} = \{(w_{i_1}, t_{j_1}), \ldots, (w_{i_k}, t_{j_k})\}$, $k \leq \min\{m, n\}$ denote the set of (worker, task) pairs assigned to each other depending on the task requirements and worker skills. We denote the task assigned to a worker w in a matching $\mathcal{M}$ by $\mathcal{M}(w)$. We say $\mathcal{M}(w) = \varnothing$, if w is not matched in $\mathcal{M}$. Analogously, we denote the user assigned to a task t by $\mathcal{M}(t)$.

In order for a matching $\mathcal{M}$ to be stable it should not admit any unhappy (i.e., blocking) pair $\langle w, t \rangle$ such that $t \in \mathcal{E}(w)$, $w \in \mathcal{E}(t)$, and

- $t \succ_w \mathcal{M}(w)$ and $w \succ_t \mathcal{M}(t)$, or
- $t \succ_w \mathcal{M}(w)$ and $\mathcal{M}(t) = \varnothing$, or
- $w \succ_t \mathcal{M}(t)$ and $\mathcal{M}(w) = \varnothing$, or
- $\mathcal{M}(w) = \varnothing$ and $\mathcal{M}(t) = \varnothing$.

If $\mathcal{M}$, however, contains such pairs, we say that $\mathcal{M}$ is *unstable* and denote the set of unhappy pairs in $\mathcal{M}$ by $U(\mathcal{M})$. The number of unhappy pairs, $|U(\mathcal{M})|$, (which we also call as *unhappiness index (UI)*) in a matching has been a recognized way of measuring the instability of the matching [62].

In Section 7.1, the instance in Fig. 51 is used to show that there can be a trade-off between system utility and user satisfaction, which are respectively measured by the number of assigned users and *UI*. In order to quantify the loss in system utility and user satisfaction, respectively, in stable matchings and maximum system utility matchings in general, we run a series of experiments with 50 workers and 50 tasks randomly deployed in a 1 km by 1 km region. Eligibility conditions for workers and tasks are defined in two ways. In the *local* case, we assume that each worker can only travel up to a distance with travel cost less than the task reward and a worker prefers

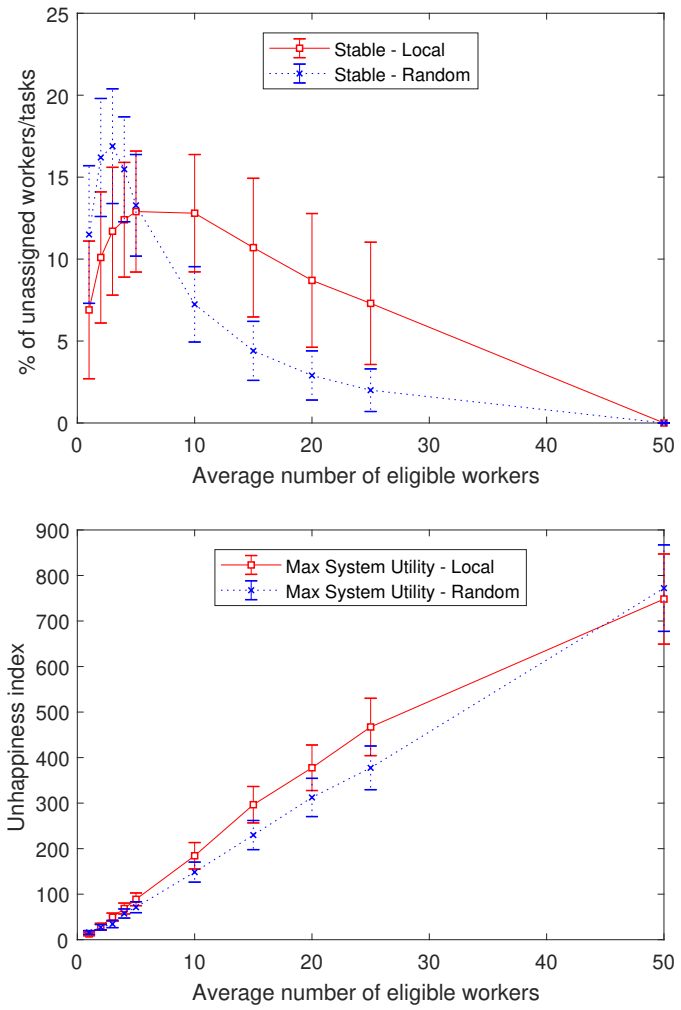


Fig. 52. Percentage of decrease in the number of assigned workers/tasks in stable matching compared to maximum system utility matching (upper) and unhappiness index in maximum system utility matching (lower) with varying size of eligible worker/task sets in the *local* and *random* settings.

the task closer to the worker's location and vice versa. In the *random* case, since each user may have a distinct and unique set of criteria to determine the eligibility, we randomly decide the eligible user sets. We then obtained the task assignments with maximum system utility and stable matching procedures for eligibility sets of different density (obtained by adjusting the rewards in the local case and the probability of

eligibility in the random case).

Fig. 52 shows the unhappiness index obtained with maximum system utility matching and the percentage of decrease in the number of assigned workers and tasks in stable matching ($\mathcal{M}_{SM}$) compared to maximum system utility matching ($\mathcal{M}_{MM}$), which can formally be defined as

$$100 \times \frac{|\mathcal{M}_{MM}| - |\mathcal{M}_{SM}|}{|\mathcal{M}_{MM}|}. \tag{7.4}$$

For all results in this and the following sections, we take the average of 100 different runs for statistical significance. We observe that up to 17% more users are left unassigned with stable matching, while maximum system utility matching yields a massively larger unhappiness index (i.e., by definition, the unhappiness index is 0 in stable matching). Although one can carefully use the appropriate algorithm in the extreme cases (e.g., stable matching when all workers are eligible for all tasks, and maximum system utility matching when only a few workers are eligible for each task provided that small number of unhappy pairs is acceptable), neither algorithm provides efficient results for most scenarios. In Fig. 53, the same trade-off is also obtained for different ratios of worker and task ratios with an average eligible worker/task size of 3. The highest decrease in the number of unassigned workers/tasks by stable matching is observed when the ratio is 1, where we see the minimum unhappiness index obtained by maximum system utility matching.

In this study, we aim to address this trade-off and develop a task assignment algorithm that reaches the maximum possible system utility (i.e., number of matched workers/tasks) while satisfying the users as much as possible, thus minimizing the unhappiness index. A brute force method to solve this problem would be to enumerate all maximum cardinality matchings, and pick the one with the smallest unhappiness index. However, this would be too costly since the number of maximum cardinality

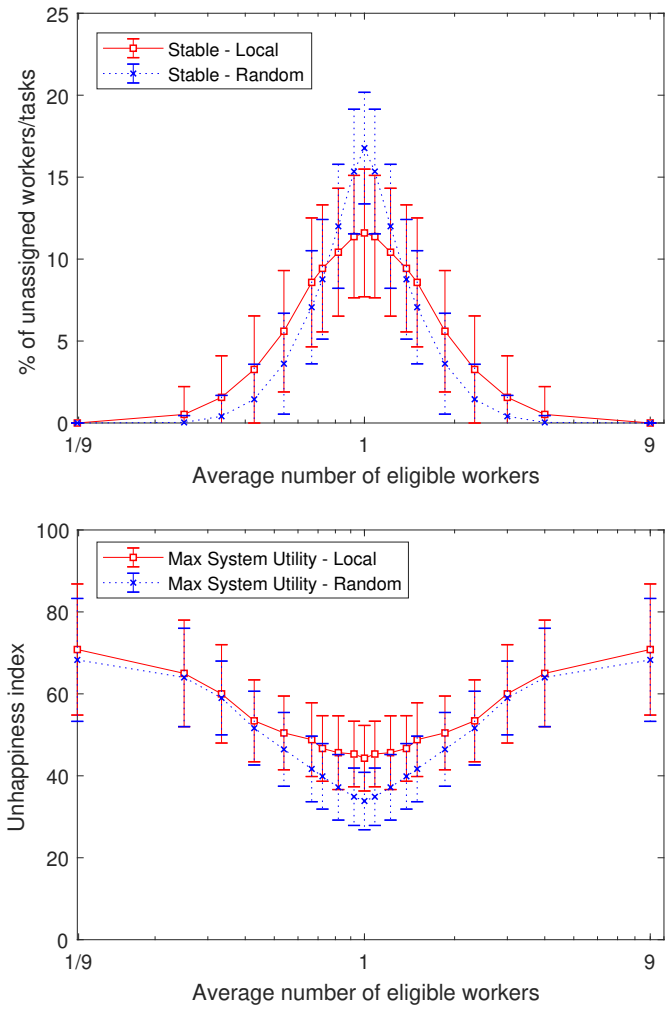


Fig. 53. Percentage of decrease in the number of assigned workers/tasks in stable matching compared to maximum system utility matching (upper) and unhappiness index in maximum system utility matching (lower) with different ratios of worker and task set sizes. We use an average eligible worker/task set size of 3 with the total number of tasks and workers fixed at 100.

matchings grows exponentially with the number of nodes. Moreover, this problem can be reduced to the problem of finding a maximum cardinality matching with minimum number of blocking pairs, which is proven to be NP-complete [62], even when the size of eligible worker/task sets is 3.

7.3 Proposed Solution

In this section, we first model the problem using Integer Linear Programming (ILP) to find the optimal solution for a given set of tasks and workers with their restrictions and eligibility. Then, we present two different heuristic-based cost-efficient solutions.

7.3.1 ILP Model

Our objective is to find a maximum size matching between workers and tasks with as few unhappy pairs as possible, which can be formally defined as follows:

$$\max \sum_{\forall i,j} \left(mn\mathcal{X}_{ij} - \mathcal{U}_{ij} \right) \tag{7.5}$$

with the constraints:

$$\sum_{\forall i} \mathcal{X}_{ij} \leq 1 \qquad \forall j \tag{7.6}$$

$$\sum_{\forall j} \mathcal{X}_{ij} \leq 1 \qquad \forall i \tag{7.7}$$

$$\mathcal{X}_{ij} \leq e_{ij} \qquad \forall i,j \tag{7.8}$$

where,

$$e_{ij} = \begin{cases} 1, & \text{if } w_i \text{ is eligible to perform } t_j \\ 0, & \text{otherwise} \end{cases} \tag{7.9}$$

$$\mathcal{X}_{ij} = \begin{cases} 1, & \text{if } w_i \text{ is assigned to } t_j \\ 0, & \text{otherwise} \end{cases} \tag{7.10}$$

$$\mathcal{U}_{ij} = \begin{cases} 1, & \text{if } (w_i, t_j) \text{ is an unhappy pair} \\ 0, & \text{otherwise} \end{cases} \tag{7.11}$$

Note that the number of unhappy pairs can be at most mn. Increasing the assigned pair count by one will increase the value of objective function (7.5) more than removing all unhappy pairs. Thus, it produces an assignment with maximum system utility, then reduces the unhappiness index as much as possible.

7.3.2 Maximum to Stable Reduction Algorithm

Our first algorithm initially finds a maximum system utility matching, and then attempts to decrease the number of unhappy pairs in it one by one without altering the total utility of the matching. Before elaborating on the algorithm steps, we first describe *happify* procedure, which constitutes the core part of the algorithm.

7.3.2.1 Happify Procedure

The purpose of the happify procedure is to get rid of a specific unhappy pair by re-matching the worker and the task that form it with each other. Consider the example in Fig. 54a, in which worker 1 and task a form an unhappy pair, denoted by $\langle 1, a\rangle$. We happify $\langle 1, a\rangle$ by matching 1 with a. In order to maintain the utility of the matching, we also attempt to match their former partners, b and 2, with each other (and form the matching $\mathcal{M}'$). Yet this is not always feasible, because b and 2 may be considering each other unacceptable (i.e., $2 \notin \mathcal{E}(b)$ and $b \notin \mathcal{E}(2)$). In this case, since leaving b and 2 unmatched would decrease the utility of the matching, we avoid performing the happify procedure on such pairs.

On the other hand, even if b and 2 consider each other as acceptable, happifying $\langle 1, a\rangle$ would not always yield a matching that contains fewer unhappy pairs. In fact,

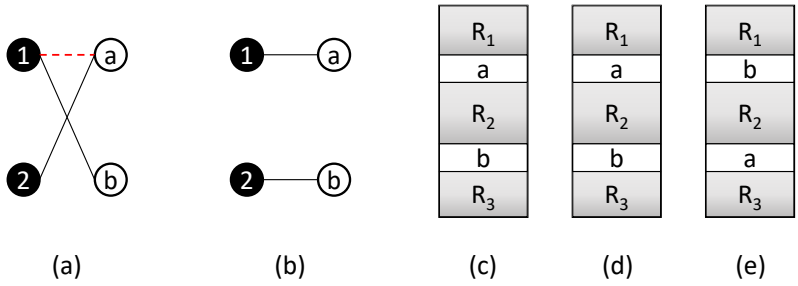


Fig. 54. An instance of happify procedure. (a) the initial matching $\mathcal{M}$; (b) the matching $\mathcal{M}'$ after happifying the unhappy pair $\langle 1, a\rangle$ in $\mathcal{M}$; (c) 1's preference list, P_1; (d) 2's possible preference list, P_2'; (e) 2's alternative preference list, P_2''. $\mathcal{R}_i$'s are defined in (7.12).

the number of unhappy pairs can decrease, remain unchanged, or even increase. To figure that out, we need to check the preference lists of these four nodes, and identify the nodes in their preference lists, which can be potentially affected by partner change. To illustrate this, we will analyze the possible scenarios that can arise after happifying $\langle 1, a\rangle$. Since the relationship between the tasks and workers is symmetric as seen in Fig. 54a, 1 and a will have similar scenarios, as do 2 and b. Therefore, the examination of scenarios for nodes 1 and 2 should be sufficiently descriptive.

First of all, since $\langle 1, a\rangle$ is given as an unhappy pair, we can deduce that $a \succ_1 (\mathcal{M}(1) = b)$. Then, we divide P_1 (i.e., preference list of worker 1 on eligible tasks in $\mathcal{E}(1)$) into regions as $\mathcal{R}_1 \cup \{a\} \cup \mathcal{R}_2 \cup \{b\} \cup \mathcal{R}_3$ such that

$$(\forall x \in \mathcal{R}_1) \succ_1 a \succ_1 (\forall x \in \mathcal{R}_2) \succ_1 b \succ_1 (\forall x \in \mathcal{R}_3) \tag{7.12}$$

as illustrated in Fig. 54c. Note that the partner change of 1, from $\mathcal{M}(1) = b$ to $\mathcal{M}'(1) = a$, will result in clearing all unhappy pairs in

$$\{\langle 1, x\rangle \mid x \in \mathcal{R}_2, \langle 1, x\rangle \in U(\mathcal{M})\}, \tag{7.13}$$

if any, because for all $x \in \mathcal{R}_2$, $a \succ_1 x$. The set of other unhappy pairs formed as

$$\{\langle 1, x\rangle \mid x \in \mathcal{R}_1, \langle 1, x\rangle \in U(\mathcal{M})\} \tag{7.14}$$

will remain unchanged in $\mathcal{M}'$, as $\forall x \in \mathcal{R}_1$, $x \succ_1 (a = \mathcal{M}'(1))$. Lastly, there cannot exist any unhappy pairs

$$\{\langle 1, x\rangle \mid x \in \mathcal{R}_3\} \tag{7.15}$$

in neither $\mathcal{M}$ nor $\mathcal{M}'$, since $(b = \mathcal{M}(1)) \succ_1 x$ and $(a = \mathcal{M}'(1)) \succ_1 x$, for all $x \in \mathcal{R}_3$.

Although we know how a and b are ranked in P_1, we do not have any data to infer that for P_2. Therefore, we must consider both possibilities, namely P_2' if $a \succ_2 b$ and P_2'' if $b \succ_2 a$, which are also partitioned into regions as shown in Fig. 54d and Fig. 54e. Note that, regardless of P_2' or P_2'', happifying $\langle 1, a\rangle$ will not affect the unhappy pairs in

$$\{\langle 2, x\rangle \mid x \in \mathcal{R}_1, \langle 2, x\rangle \in U(\mathcal{M})\}, \tag{7.16}$$

so that they will still be present in $\mathcal{M}'$, and

$$\{\langle 2, x\rangle \mid x \in \mathcal{R}_3, \langle 2, x\rangle \in U(\mathcal{M}) \cup U(\mathcal{M}')\} = \varnothing, \tag{7.17}$$

due to the same reasons pointed out above. As for $\mathcal{R}_2$, we face two different scenarios. Considering $P_2 = P_2'$, since happifying $\langle 1, a\rangle$ forces 2 to match with b, which it prefers less than its former partner a, a new set of unhappy pairs

$$\{\langle 2, x\rangle \mid x \in \mathcal{R}_2, 2 \succ_x \mathcal{M}'(x)\} \tag{7.18}$$

will arise in $\mathcal{M}'$. Contrary to this, matching 2 with b is for the benefit of 2 if $P_2 = P_2''$

and will indirectly happify all the unhappy pairs, if any, in

$$\{\langle 2, x\rangle \mid x \in \mathcal{R}_2, \langle 2, x\rangle \in U(\mathcal{M})\}. \tag{7.19}$$

We next show how the set of unhappy pairs in $\mathcal{M}$ and $\mathcal{M}'$ are related. Let U denote the subset of unhappy pairs in $\mathcal{M}$ that will be happified and $\mathcal{M}'$ be the resulting matching. The set of unhappy pairs, which were not present in $\mathcal{M}$, however will arise in $\mathcal{M}'$ is

$$U_N = \Big\{\langle x, y\rangle \notin U(\mathcal{M}) \mid y \succ_x \mathcal{M}'(x), x \succ_y \mathcal{M}'(y)\Big\}, \tag{7.20}$$

and the set of unhappy pairs that were found in $\mathcal{M}$, but will disappear in $\mathcal{M}'$ is

$$U_O = \Big\{\langle x, y\rangle \in U(\mathcal{M}) \mid \mathcal{M}'(x) \succ_x y \text{ or } \mathcal{M}'(y) \succ_y x\Big\}. \tag{7.21}$$

Then, the set of unhappy pairs in the new matching becomes

$$U(\mathcal{M}') = \Big(U(\mathcal{M}) \cup U_N\Big) \setminus U_O \tag{7.22}$$

Thus, to find the new set of unhappy pairs, $U(\mathcal{M}')$, we need to identify U_N and U_O, for which we just need to check whether the users (i.e., x) whose partners have changed due to the happify procedure form an unhappy pair with those (i.e., y) who are between $\mathcal{M}(x)$ and $\mathcal{M}'(x)$ in P_x. Note that only the users that are in at least one of the pairs in U will get matched with a different user. Thus, for each worker w and task t, for which $\mathcal{M}(w) \neq \mathcal{M}'(w)$ and $\mathcal{M}(t) \neq \mathcal{M}'(t)$ (i.e., there are at most 4 of them within a single round of happify procedure), we need to check at most $|\mathcal{T}| - 2$ and $|\mathcal{W}| - 2$ worker-task pairs to find $U(\mathcal{M}')$, respectively. Thus, each happify operation has $O(N)$ complexity, where $N = \max\{m, n\}$.

The proposed algorithm aims to reduce the number of unhappy pairs greedily through consecutive happify operations. To this end, in each iteration, we find the

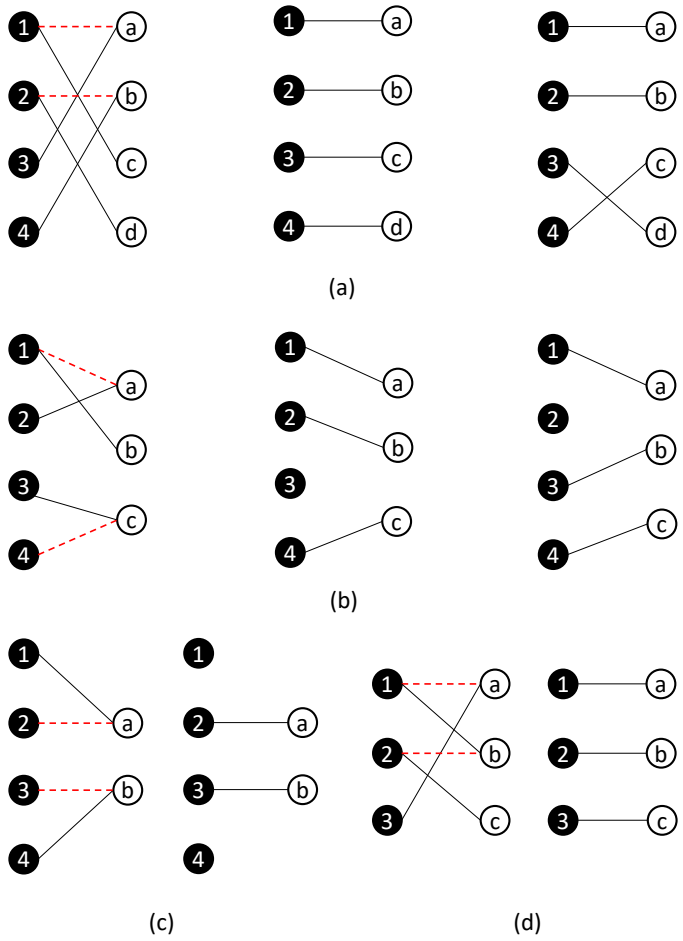


Fig. 55. Some possible happify attempts that can occur in Phase 2. Unhappy pairs are shown with red dotted lines.

unhappy pair that reduces the total number of unhappy pairs the most when it gets happified (if possible), and happify it. However, it is possible that none of the happify operations at the current iteration is able to reduce the unhappy pair count as the result of hitting a local minimum. To address this, we introduce a hop-based approach and give chance to reduction in the unhappy pair count up to k consecutive happify operations. That is, even though the happify operation that results in the minimum unhappy pair count increases the current unhappy pair count, the process continues up to k tries expecting that there will be a decrease.

Algorithm 17: Maximum to Stable Reduction ($\mathcal{W}$, $\mathcal{T}$, k)

Input: $\mathcal{W}$, $\mathcal{T}$, k: Set of workers, set of tasks, and number of hops

$\mathcal{M} \leftarrow$ Find a maximum cardinality matching between $\mathcal{W}$ and $\mathcal{T}$.

$U(\mathcal{M}) \leftarrow$ Identify the unhappy pairs in $\mathcal{M}$.

$\mathcal{M}_{best} \leftarrow \mathcal{M}$

for $i \leftarrow 1$ *to* 2 **do**

  if $i == 2$ **then**

    $j \leftarrow k$

  else

    $j \leftarrow 1$

  while $j > 0$ **do**

    $\mathcal{M}' \leftarrow \varnothing$; ▷ $|U(\mathcal{M}')| = \infty$

    $\mathcal{S} \leftarrow \{\mathcal{A} \subseteq U(\mathcal{M}) : |\mathcal{A}| = i\}$

    for $U \in \mathcal{S}$ **do**

      for $\mathcal{M}_{new} \in$ *Happify*$(\mathcal{M}, U)$ **do**

        if $|U(\mathcal{M}_{new})| < |U(\mathcal{M}')|$ **then** $\mathcal{M}' \leftarrow \mathcal{M}_{new}$

    if $|U(\mathcal{M}')| < |U(\mathcal{M}_{best})|$ **then**

      $j \leftarrow k$

      $\mathcal{M}_{best} \leftarrow \mathcal{M}'$

    else

      $j \leftarrow j - 1$

    $\mathcal{M} \leftarrow \mathcal{M}'$

  $\mathcal{M} \leftarrow \mathcal{M}_{best}$

return $\mathcal{M}_{best}$

Algorithm 18: Happify ($\mathcal{M}$, U)

Input: $\mathcal{M}$: A matching between $\mathcal{W}$ and $\mathcal{T}$

U: The set of unhappy pairs to be happified

1 Let $\mathcal{M}_S$ be the set of all matchings that can be obtained by happifying the unhappy pairs in U (as shown in Fig. 54 & 55).

2 **foreach** $\mu \in \mathcal{M}_S$ **do**

3 Find U_N and U_O by Eq. 7.20 & 7.21.

4 $|U(\mu)| = |U(\mathcal{M})| + |U_N| - |U_O|$

5 **return** $\mathcal{M}_S$

Another consideration is rather than happifying the unhappy pairs individually, we can happify them in groups simultaneously. In that case, former partners of nodes comprising the unhappy pairs will have more options to be matched. For example, Fig. 55 shows some possible re-matchings of former partners for different cases observed when two unhappy pairs are happified simultaneously. While this extension will increase the likelihood of reducing the unhappy pair count without affecting the matching utility at each iteration, it increases the complexity of the algorithm due to more permutations to be checked.

In order to address all these points, we propose a phased approach. That is, we begin by considering unhappy pairs individually in the happify procedure, and when this fails to provide further improvement, we start to happify them in groups of two (it can also be extended to groups of three or more). However, with the phased approach, we consider the hop-based happify operations only for the last phase to avoid hitting the local minimum earlier. Algorithm 17 shows a two-phase instance of the proposed solution. The phases are iterated by the for loop in line 4. The algorithm makes use of a subroutine, *happify*, that takes a matching $\mathcal{M}$ and a set U

of unhappy pairs in $\mathcal{M}$ as input and returns the set of all possible matchings that can arise by happifying U. A pseudo-code of the happify procedure is given in Algorithm 18.

Maximum to Stable Reduction algorithm, shown in Algorithm 17, begins with finding a maximum utility (i.e., cardinality) matching $\mathcal{M}$. In the first phase ($i = 1$), we find the best matching, $\mathcal{M}'$, amongst a set of matchings, each of which is obtained from $\mathcal{M}$ by happifying a single, different unhappy pair in $U(\mathcal{M})$ (i.e., the set $\mathcal{S}$ in line 12 consists of the subsets $\mathcal{A}$ of unhappy pairs with size 1). We update $\mathcal{M}_{best}$, which denotes the best matching that is ever reached by the algorithm, if $\mathcal{M}'$ is better than $\mathcal{M}_{best}$. Note that since all the matchings that are scanned by the algorithm are of maximum utility, the goodness of a matching depends only on the number of unhappy pairs it has. The same process is then repeated for the new matching $\mathcal{M}'$ in the same manner as long as an improvement in the number of unhappy pairs is observed in at least one of k consecutive steps. Note that in the first phase, k is set to 1 as explained above. In the second phase ($i = 2$), our algorithm tries to relax two unhappy pairs simultaneously (*happify* in line 13 returns all possible variations). If no improvement is achieved in the unhappy pair count in k hops, the algorithm terminates.

Example. We provide a sample run of Algorithm 17 on the instance in Fig. 51 to demonstrate how it gradually decreases the number of unhappy pairs while preserving the maximum system utility. Firstly, a maximum matching is found, which, as shown in Fig. 56a, turns out to have 4 unhappy pairs, namely $\langle 1, a\rangle$, $\langle 2, c\rangle$, $\langle 3, d\rangle$, and $\langle 5, c\rangle$. We try to happify each of these unhappy pairs individually and find the one that leads to a better matching when happified. The new set of unhappy pairs that could be obtained by happifying each unhappy pair is given in Table 12.

Note that we cannot happify $\langle 1, a\rangle$ and $\langle 3, d\rangle$, because their current partners in the initial matching, $(d, 5)$ for $\langle 1, a\rangle$, and $(b, 1)$ for $\langle 3, d\rangle$, consider each other

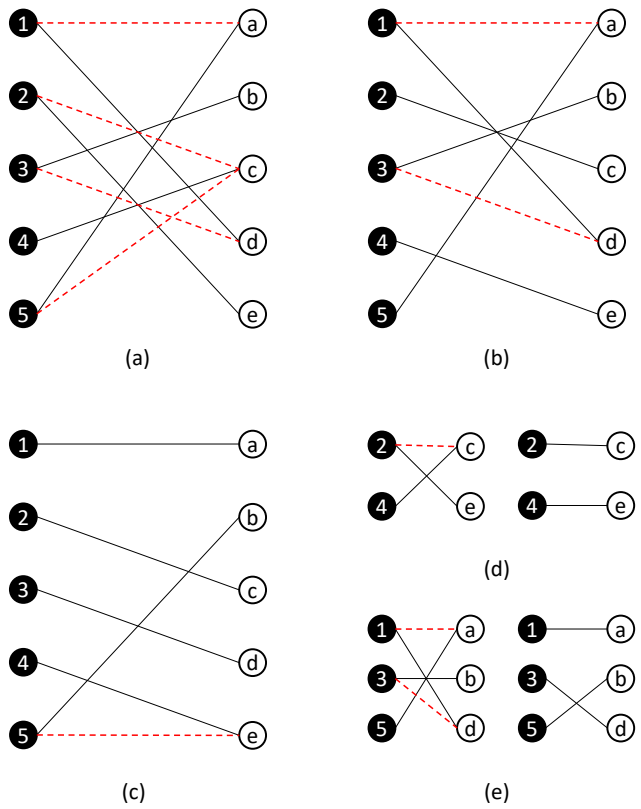


Fig. 56. Steps of running Algorithm 17 on the instance given in Fig. 51. (a) the initial maximum matching; (b) the best matching reached by the end of the first phase; (c) the best matching ever found by the algorithm, also an optimal solution; (d) happifying $\langle 2, c\rangle$ on the matching in (a); (e) happifying $\{\langle 1, a\rangle, \langle 3, d\rangle\}$ on the matching in (b).

Table 12. Matchings to be obtained by happifying each unhappy pair in Fig. 56a.

Unhappy pair	$U(\mathcal{M}_{new})$
$\langle 1, a\rangle$	cannot be happified
$\langle 2, c\rangle$	$\langle 1, a\rangle$, $\langle 3, d\rangle$
$\langle 3, d\rangle$	cannot be happified
$\langle 5, c\rangle$	$\langle 1, a\rangle$, $\langle 2, c\rangle$, $\langle 3, a\rangle$, $\langle 3, d\rangle$

unacceptable, therefore, we skip these unhappy pairs. Since the matching obtained by happifying $\langle 2, c\rangle$ has the minimum number of unhappy pairs among all, we proceed with it (Fig. 56b) to the second phase (any further attempt in phase 1 would not decrease the unhappy pair count, as neither $\langle 1, a\rangle$ nor $\langle 3, d\rangle$ can be happified due to partner incompatibility as above). In the second phase, since there are only two unhappy pairs in the matching, we will have only one subset of unhappy pairs of size 2 that we will, if possible, happify simultaneously, which is $\{\langle 1, a\rangle, \langle 3, d\rangle\}$. Indeed, these two unhappy pairs, which could not be happified separately, can be jointly happified as in Fig. 56e. Besides, this yields a matching with just one unhappy pair, $\langle 5, e\rangle$, as shown in Fig. 56c, which, having less than 2 unhappy pairs, cannot be improved furthermore by the second phase. Even if we run the first phase again on this final matching, it would make no difference since $\langle 5, e\rangle$ cannot be happified due to partner incompatibility. Actually, this final matching is identical to the optimal solution found via ILP, and is the only optimal solution possible.

Running time. Let E be the number of eligible pairs in a given matching instance, and $N = \max(m, n)$. In line 1, we can find a maximum size matching in $O(NE)$ time using the Ford-Fulkerson method [103]. Then, for the first phase, since the loop starting in line 12 may iterate at most E times, as the number of unhappy pairs in any matching is at most E, and the computation of the benefit that can be obtained by happifying each unhappy pair takes $O(N)$ time (see Section 7.3.2.1), the time complexity of the first phase of our algorithm is $O(NE^2)$ or $O(N^5)$. Similarly, the time complexity of the second phase is $O(B^3N)$, where B is the number of unhappy pairs in the final matching produced by the first phase. By using an extra $O(N^2)$ space to store the benefits of happifying each unhappy pair and updating these benefits after each happify procedure in constant time for $N^2 - 4N$ pairs that are *indirectly* affected by the last happify procedure and in $O(N)$ time for $4N$ pairs that are *directly* affected

by the last happify procedure (i.e., the pairs whose partners have changed), we can reduce[10] the time complexity of the first phase to $O(N^4)$. Using the same strategy, we can also reduce the time complexity of the second phase to $O(B^2N)$ by using an extra $O(B^4)$ space (to store the benefit of happifying every pair of unhappy pairs). However, this may not be practical for large MCS instances, as B can be as large as N^2.

7.3.3 Stable to Maximum Convergence Algorithm

In our second algorithm, we propose a reversed approach. That is, we first obtain a stable matching, where every user is perfectly happy. Then, we update it iteratively to obtain an assignment with maximum system utility while keeping the number of unhappy pairs as low as possible. We find paths with certain properties in the given bipartite graph (7.3) at every step that will increase the number of assigned pairs with respect to the current assignment. We simply call these paths *beneficial* paths. Given a matching $\mathcal{M}$ in a given bipartite graph G, a path $p = \{p_1, p_2, \dots p_{2j+2}\}$ is considered a beneficial path if its both endpoints are not matched with any node in $\mathcal{M}$, and its edges alternate between the edges in $\mathcal{M}$ and the other edges not included $\mathcal{M}$. More formally,

$$\begin{aligned}
&\mathcal{M}(p_1) = \varnothing, \text{ i.e., } (p_1, p_2) \in G.E \setminus \mathcal{M}, \\
&\mathcal{M}(p_{2j+2}) = \varnothing, \text{ i.e., } (p_{2j+1}, p_{2j+2}) \in G.E \setminus \mathcal{M}, \\
&\mathcal{M}(p_{2i}) = p_{2i+1}, \text{ i.e., } (p_{2i}, p_{2i+1}) \in \mathcal{M}, && \forall i \in [1, \dots, j] \\
&\mathcal{M}(p_{2i-1}) \neq p_{2i}, \text{ i.e., } (p_{2i-1}, p_{2i}) \in G.E \setminus \mathcal{M}. && \forall i \in [1, \dots, j]
\end{aligned} \tag{7.23}$$

[10] I would like to thank Dr. Preetam Ghosh for suggesting this method to improve the time complexity of the proposed algorithm.

Algorithm 19: Stable To Maximum Convergence ($\mathcal{W}$, $\mathcal{T}$)

Input: $\mathcal{W}$: The set of workers

$\mathcal{T}$: The set of tasks

```
1  M ← Find a stable matching via Gale-Shapley algorithm between W and T.
2  while true do
3      set all t ∈ T as unvisited
4      foreach unmatched w ∈ W do
5          p = {w}
6          p ← FindBeneficialPath(p)
7          if p.isBeneficialPath then
8              break
9      if a beneficial path p = {p_1, p_2, .., p_{2j+2}} is found then
10         for i ← 1 to j + 1 do
11             M(p_{2i-1}) ← p_{2i}
12             M(p_{2i}) ← p_{2i-1}
13     else
14         break
15 return M
```

By definition, note that there cannot be a beneficial path of even length, and for a path $p = \{p_1, p_2\}$ of length 1 to be beneficial, both p_1 and p_2 should be unmatched in $\mathcal{M}$.

The proposed algorithm is given in Algorithm 19. We first find a stable matching $\mathcal{M}$ between the given workers and tasks using the deferred acceptance mechanism in Gale-Shapley algorithm [42]. Then, in each iteration of the while loop in line 2, we

Algorithm 20: *FindBeneficialPath(p)*

Input: p: Current path

```
1  w ← p.last() ;                          ▷ last node on current path
2  foreach t ∈ P_w in the preference order do
3      if M(t) = ∅ then
4          p ← p ∪ {t}
5          p.isBeneficialPath ← true
6          return p
7  foreach t ∈ P_w in the preference order do
8      if t is unvisited then
9          set t as visited
10         p' ← FindBeneficialPath(p ∪ {t, M(t)})
11         if p'.FindBeneficialPath then
12             return p'
```

try to find a beneficial path p in $\mathcal{M}$. If we find one, we update $\mathcal{M}$ as follows

$$\mathcal{M} \leftarrow (\mathcal{M} \setminus E(p)) \cup (E(p) \setminus \mathcal{M}), \tag{7.24}$$

where $E(p)$ is the set of edges in p. Note that in a beneficial path p of length $2j+1$ (with $2j+2$ nodes), there are j edges that are in $\mathcal{M}$, and $j+1$ edges that are not. Thus, replacing the former j edges in $\mathcal{M}$ with the latter $j+1$ edges will increase the system utility by 1, which is performed between lines 9-12. However, if we cannot find a beneficial path, it means $\mathcal{M}$ has reached the maximum possible assignment [103] and will be returned as the final matching.

The procedure of finding a beneficial path is shown in Algorithm 20. Starting

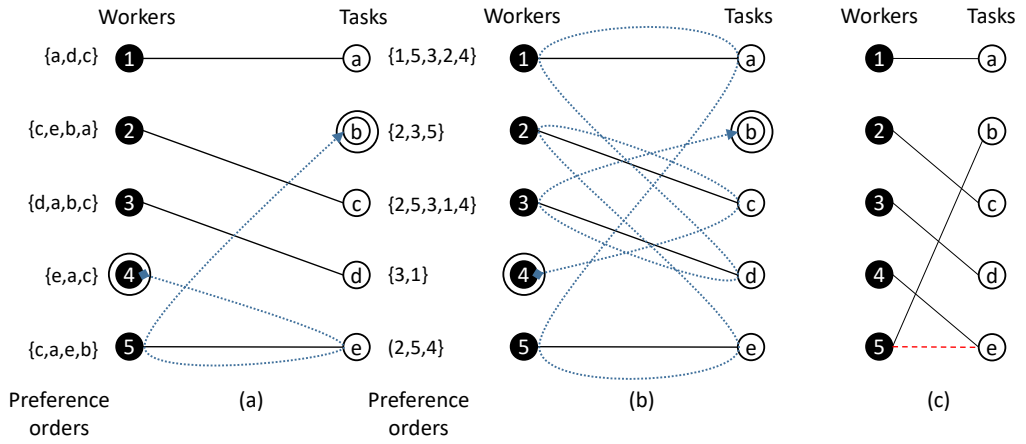


Fig. 57. Two example beneficial paths (shown with dotted lines) in the initial stable matching (shown with solid lines) generated in the first line of Algorithm 19 when it is run on the example illustrated in Fig. 51. (a) Beneficial path ($4 \to e \to 5 \to b$) found by Algorithm 19; (b) An alternative beneficial path ($4 \to c \to 2 \to e \to 5 \to a \to 1 \to d \to 3 \to b$) that could be found if the search was done without considering preference orders; (c) Matching obtained by Algorithm 19 using the beneficial path shown in (a). The only remaining unhappy pair is shown with a dashed line.

from each worker w not matched currently in $\mathcal{M}$, we attempt to find a beneficial path (lines 4-8 in Algorithm 19). If w can be matched directly with an unmatched task in P_w, a beneficial path of length 1 is obtained immediately (lines 2-6 in Algorithm 20). Otherwise, the tasks that are currently matched in $\mathcal{M}$ are processed in their preference order. For each such task t, a new potential path is created by extending the current path with task t and its partner $\mathcal{M}(t)$, and the same process is repeated recursively (lines 7-12 in Algorithm 20).

We run Algorithm 19 on the same toy example given in Fig. 51. We first obtain the stable matching given in Fig. 57a. Then, we look for a beneficial path in this matching. The process in Algorithm 20 finds the beneficial path $4 \to e \to 5 \to b$ (of length 3). Executing the lines 10-12 in Algorithm 19 will result in the opti-

mal solution (with an unhappiness index of 1 caused by (5,e)) shown in Fig. 57c. Since this matching is maximum, Algorithm 19 will return it as the final matching. However, note that there can be multiple beneficial paths in the initial stable matching, as illustrated in Fig. 57, and any of them might be returned first based on the implementation. For example, assume this time that we find the beneficial path $4 \rightarrow c \rightarrow 2 \rightarrow e \rightarrow 5 \rightarrow a \rightarrow 1 \rightarrow d \rightarrow 3 \rightarrow b$. It gives us an assignment with an unhappiness index of 4, and hence is not an optimal solution. In our implementation, we visit the neighbors greedily in their preference order to find a beneficial path that causes as little increase in the unhappiness index as possible.

Running time. Gale-Shapley algorithm that is executed in the first line of Algorithm 19 runs in $O(E)$ time, where E is the number of eligible pairs in the matching instance. There can be at most $O(N)$ cardinality difference between a stable matching and a maximum matching in a bipartite graph, where $N = \max(m, n)$. Since finding a beneficial path, as well as updating the matching accordingly, has $O(N + E)$ complexity (as we need to visit every edge at most once), the total running time of Algorithm 19 becomes $O(N^2 + NE)$ or $O(N^3)$.

7.4 Evaluation

In this section, we evaluate the performance of the proposed algorithms using a real world dataset.

7.4.1 Simulation Settings

In order to have a realistic set of user locations, we have used a taxi trip dataset [81] in a city (i.e., New York City (NYC)) similar to previous work [29, 80, 104]. Previous work mostly consider taxi driver locations as workers and assign task locations randomly. In order to have more realistic task locations as well, we

have used the pick up locations of passengers as task locations. Specifically, we generate the user set for each of the 100 runs of an experiment by selecting the taxis that dropped off their passengers between 1-2 pm on a randomly selected day in 2015 as workers at the corresponding drop-off locations, and by creating a task at the pick up location of each passenger who requested a taxi in the next hour of the same day. Then, from this set we randomly sample a certain number of workers and tasks according to the experiment specifications.

In the first part of the simulations, we use 50 workers and 50 tasks as smaller and equal set sizes represent the hardest scenario. This is because, as it is shown in Fig. 53, the largest difference in the matching cardinality between stable and maximum system utility matching happens when $|\mathcal{W}|/|\mathcal{T}|$=1. That is, the trade-off between user satisfaction and system utility becomes more important and harder to handle when the size of the worker and task sets are equal. Nonetheless, in the following simulations, we also examine the scenarios with different $|\mathcal{W}|/|\mathcal{T}|$ ratios. Moreover, we provide results regarding the scalability of proposed algorithms with up to 1000 workers/tasks. The preference lists of workers and tasks are defined either locally (i.e., based on the ascending order of distances) or randomly, as described in Section 7.2.

7.4.2 Results

We first look at the effectiveness of the proposed approaches by comparing them with ILP results in terms of unhappiness index. Throughout the section, we use the notation *Ph***x**-*Hop***k** to denote the *Maximum to Stable* reduction algorithm with x phases in which the first $(x-1)$ phases run only 1 hop and the x^{th} phase runs up to k hops. Fig. 58 shows the performance comparison of *Ph***1**-*Hop***1**, *Ph***2**-*Hop***1**, *Stable to*

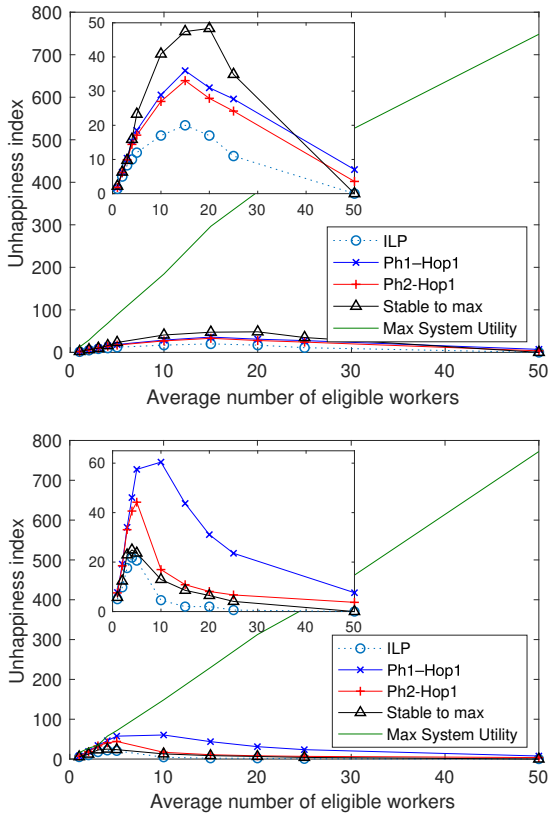


Fig. 58. Performance comparison of the heuristic algorithms with optimal results (ILP) and maximum system utility matching in terms of unhappiness index (UI) in the local (upper) and random (lower) settings, respectively.

Max and *Max System Utility*[11] algorithms with ILP results in the local and random settings, respectively. First of all, note that, as expected, the unhappiness index in the initial maximum matching grows linearly with increasing average eligible worker/task set size (simply denoted as $|\mathcal{E}|$). *Ph***1**-*Hop***1** algorithm gives a very close result to ILP, and *Ph***2**-*Hop***1** can further improve the result. The improvement is, however, more in the random setting. *Stable to Max* algorithm also performs differently. It performs better (i.e., fewer unhappy pairs) than other algorithms in the random setting, while it results in a greater unhappiness index in the local setting. With larger $|\mathcal{E}|$, it

[11]It refers to the solution found by the Ford-Fulkerson method [103].

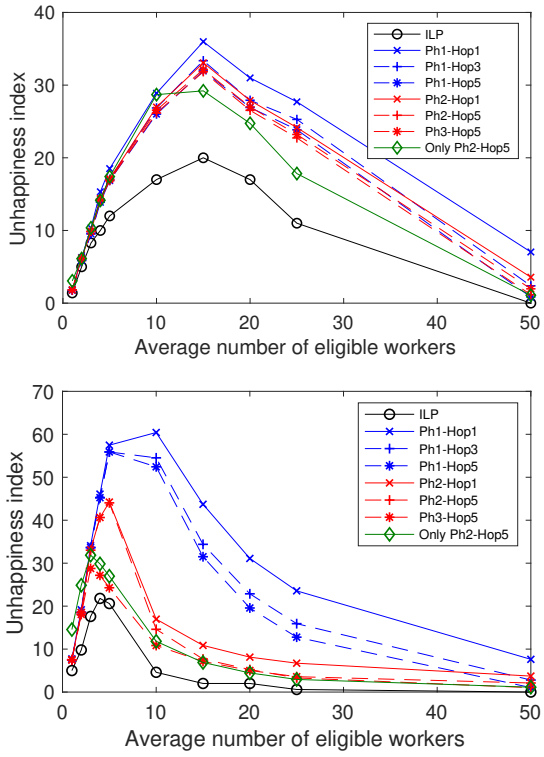


Fig. 59. The impact of number of hops and different phases on the performance of the Maximum to Stable Reduction algorithm in the local (upper) and random (lower) settings, respectively.

also performs better in the local setting and always reaches complete perfect user satisfaction and stability (an unhappiness index of 0). The maximum gap between the proposed algorithms and the ILP results occurs with $|\mathcal{E}|$ around 10-20 and gets smaller as it increases or decreases.

Next, in Fig. 59, we look at the impact of the number of hops and different phases on the performance of the *Maximum to Stable* reduction algorithm variants in both local and random settings. Note that, in the local setting, the unhappiness index in the optimal assignment increases until $|\mathcal{E}|$ is 15 and then starts to decline, while it peaks at around 4-5 in the random setting. Besides, a sharper decrease is observed after the peak in the random setting compared to the local setting. The results of

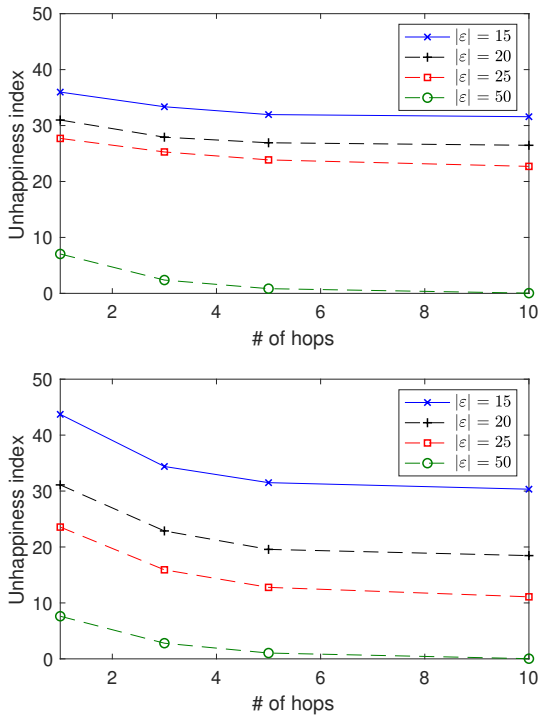


Fig. 60. The change in the unhappiness index (in Ph**1**-Hop#) with different number of hops in the local (upper) and random (lower) settings, respectively, for different eligible worker/task sizes ($|\mathcal{E}|$).

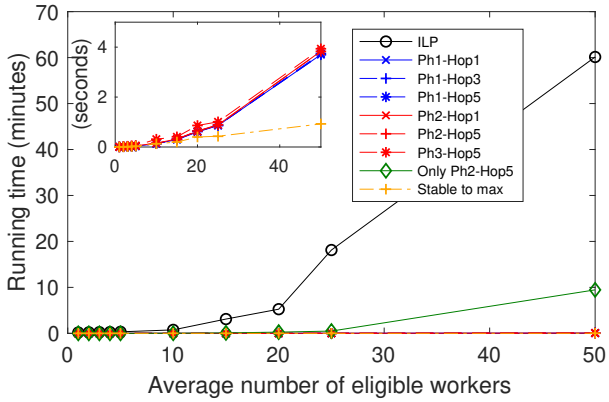


Fig. 61. Running times of the proposed algorithms.

our algorithms are also in accordance with these trends in both settings.

As for the usefulness of Phase 2 or 3, we observe that more phases offer more

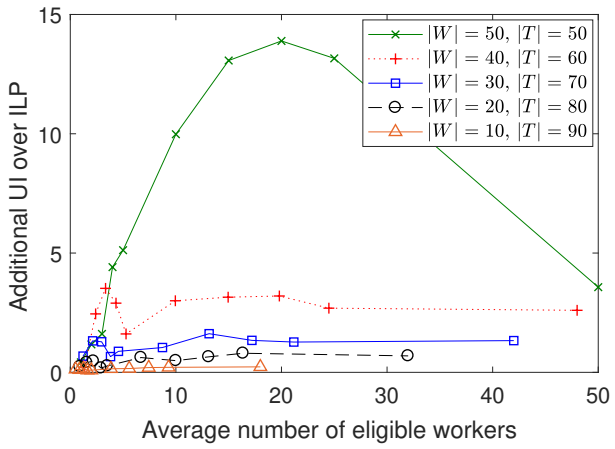


Fig. 62. The difference in the unhappiness index between ILP and *Ph***2**-*Hop***1** for different number of workers/tasks ratios.

benefit in the random setting compared to the local setting. However, there is not much benefit in running Phase 2 (or Phase 3) before the peak in neither setting. This is because the likelihood of finding a set of unhappy pairs that can be happified simultaneously is quite low when $|\mathcal{E}|$ is small given that happifying multiple unhappy pairs at the same time necessitates that the current partners of the nodes forming those unhappy pairs have each other in their eligibility lists.

Note that we also run a special version called *Only Ph***2**-*Hop***5** in which Phase 2 is directly run by skipping Phase 1. This was to show the benefit of phased approach as it can provide results as good as *Only Ph***2**-*Hop***5** with a much less running time than it (as it is shown in Fig. 61). Another point is that the difference between the performance of same phases with different number of hops is more profound in the random setting than it is in the local setting. In fact, as it is shown in Fig. 60, running the algorithm with higher number of hops reduces the unhappiness index by 1.12 per hop in the random setting and by only 0.52 per hop in the local setting, on average. Nonetheless, increasing the number of hops does not seem to be very beneficial after

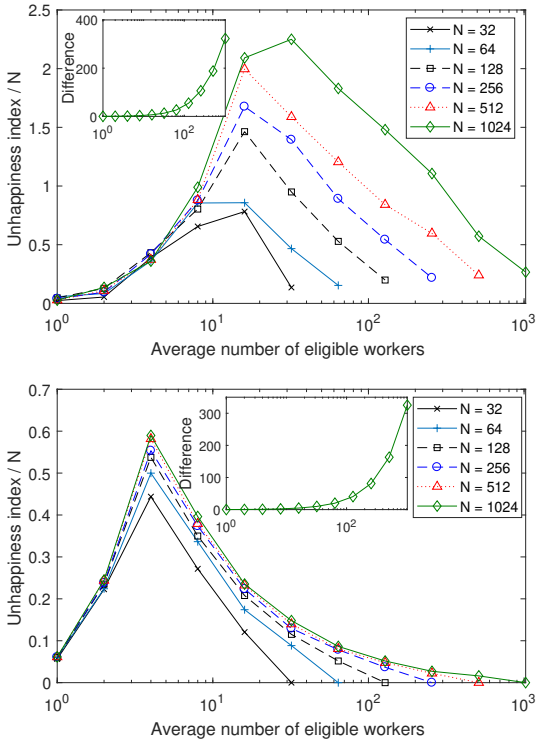


Fig. 63. The ratio of the unhappiness index (UI) to $N = |\mathcal{W}| = |\mathcal{T}|$ in *Ph***1**-*Hop***1** and *Stable to Max* algorithms, respectively, for different number of workers and tasks (in the local and random settings, respectively). The inner graphs show the difference of UI/N in Max System Utility matching from the compared algorithms when $N = 1024$.

a certain point (around 5-10 hops) in either setting.

In Fig. 61, we compare the running time of the proposed algorithms in the local setting (since the results are almost identical in the random setting, the corresponding figure is not shown here for brevity). Unsurprisingly, ILP has a very long running time (e.g., approximately an hour when $|\mathcal{E}| = 50$), which makes it infeasible to find the optimal solutions for applications that demand timely response. The running time of *Only Ph***2**-*Hop***5** also increases substantially as the average eligible worker/task set size, $|\mathcal{E}|$, grows, which actually confirms the idea behind phased approach. Indeed, all the other variations of *Maximum to Stable* reduction algorithm and the *Stable*

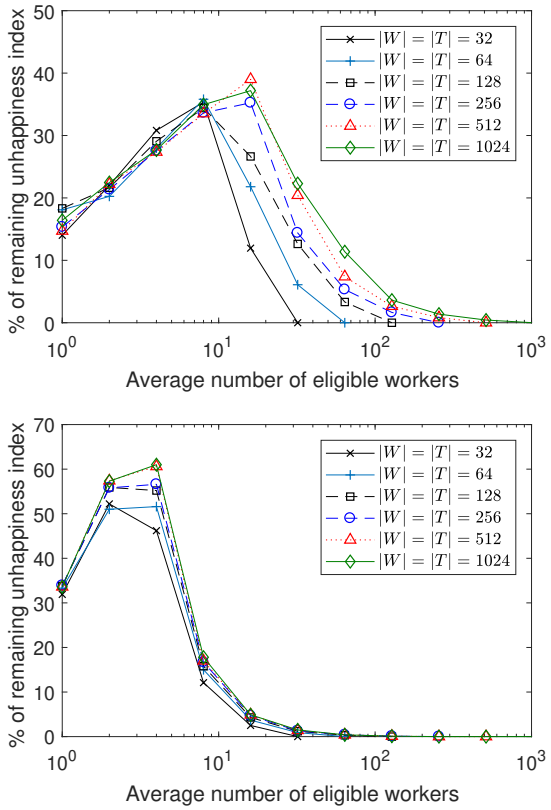


Fig. 64. The percentage of the unhappiness index in the *Ph***1**-*Hop***1** and *Stable to Max* algorithms, respectively, to the unhappiness index in the maximum system utility based assignment for different number of workers and tasks (in the local and random settings, respectively).

to Max algorithm take less than 4 seconds even when all workers are eligible for all tasks. It should also be noted that the running time is not much affected by number of phases and hops. For example, *Ph***1**-*Hop***1** and *Ph***2**-*Hop***1** take almost equal time despite the fact that *Ph***2**-*Hop***1** involves *Ph***1**-*Hop***1** in it and additionally runs Phase 2 of the algorithm. This is due to the fact that the large part of the reduction in the number of unhappy pairs occurs during Phase 1. For instance, in Fig. 58, when the average number of eligible workers is 10 in the local setting, Phase 1 decreases the number of unhappy pairs by around 155 (from 185 to 30), while Phase 2 decreases it

by only about 2 and hence takes a lot less.

In Fig. 62, we analyze the performance of the proposed algorithms when there are unequal number of workers and tasks in the system. Specifically, we calculate the difference between the unhappiness index in the optimal (i.e., ILP) matching and in the final matching produced by *Ph***2**-*Hop***1** (others perform similar). We observe that the difference in the unhappiness index gets smaller as the disparity between the number of workers and tasks grows[12]. This is also consistent with the results in Fig. 53, since the decrease in system utility is maximum when there are similar number of workers and tasks, which indicates that a larger number of users' happiness will have to be sacrificed in order to reach the maximum system utility, in general.

Next, we look at the scalability results using both a *Maximum to Stable* algorithm and *Stable to Maximum* algorithm. More specifically, we have used *Ph***1**-*Hop***1** algorithm in local case (as it performs better than *Stable to Maximum* as shown in Fig. 58) and *Stable to Maximum* algorithm in random case. Fig. 63 shows the ratio of the unhappiness index to the total number of workers/tasks (N) for different but equal number of workers and tasks with *Ph***1**-*Hop***1** and *Stable to Maximum* algorithms. The results show that the proposed algorithms scale very well and produce only a few additional unhappy pairs per user for larger networks, and that they greatly outperform the Max System Utility matching by achieving up to more than 300 less unhappy pairs per user. Moreover, as shown in Fig. 64 when we calculate the percentage of the unhappiness index compared to the unhappiness index in the Max System Utility matching, we obtain a similar percentage regardless of the number of workers and tasks with *Stable to Maximum* algorithm. With *Ph***1**-*Hop***1** algorithm, the percentage shifts a bit with increasing user count, however the peak stays similar.

[12]For unequal number of tasks and workers, there is a limit on the maximum average eligible worker/task set size achievable. Thus, data is available up to this maximum.

It is also worth noting that as the average eligible worker/task set size, $|\mathcal{E}|$, increases, we achieve a better performance in both scenarios.

7.5 Conclusion

In this chapter, we studied the problem of finding a maximum size task assignment that satisfies workers and task requesters as much as possible based on their preferences. Since it is an NP-hard problem, we proposed two different heuristic algorithms. In the first one, we initially find a maximum size (or system utility) matching, and then reduce the unhappy pairs in it through a novel method called *happify*. In the second algorithm, however, we first obtain a stable matching, and then transform it into a maximum system utility matching by finding what we call beneficial paths and reassigning the workers and tasks on these paths accordingly. The results have shown that the proposed algorithms run very fast compared to the ILP-based fully optimal solution, produce near-optimal task assignments, and complement each other in different settings. Note that the findings of this study can be applied to any matching problem, in which the goal is to maximize both system utility and happiness of the users, but the former has a higher priority than the latter.

CHAPTER 8

CONCLUDING REMARKS

In this dissertation, we have studied the preference-aware task assignment problem in various MCS settings, and explored how the conditions for preference-awareness (or user happiness) are affected by the characteristics of the underlying system model. Due to the nonexistence of optimal solutions and/or hardness of the problem, we have proposed efficient approximation and heuristic algorithms, which we have shown to produce near-optimal solutions through theoretical and empirical analysis.

Empirical results have shown that compared to the preference-aware solutions, task assignments that are obtained by disregarding user preferences make a larger number of users unhappy with their assignments even if they maximize a system-level utility function, and that satisfying user preferences does not necessarily yield a task assignment with a poor system-level utility score. In fact, as we have seen in Chapter 4, our algorithms, which are designed to maximize user happiness, achieved better coverage quality scores in most cases than the benchmark algorithms, which were exclusively designed to maximize coverage quality. This was partly because of the fact that our algorithms, by satisfying the coverage-based preferences of task requesters, implicitly maximize the overall coverage quality as well.

We note that the results of this dissertation, especially those presented in Chapter 3 & 4, are of vital importance for various theoretical preference-aware matching problems in the economics literature as well. For instance, despite the nonexistence results in general settings, this dissertation is the first to show that, in many-to-one matching problems with additive utility functions, a pairwise stable matching always

exists and can be found in pseudo-polynomial time when there is a proportional relationship between the cost and utility functions. This dissertation also provides some significant theoretical results on coalitional stability for both additive (Chapter 3) and non-additive (Chapter 4) utility functions.

There are however still many open problems waiting to be solved in the field of mobile crowdsensing, especially concerning preference-awareness in the task assignment process. For instance, a key aspect that has been overlooked so far is the benefit of cooperation between workers. In MCS systems with non-trivial tasks, it may be the case that two workers who cannot carry out a certain task individually can do so if they are both assigned to the task and work in a cooperative manner. Therefore, their total utility for the task would be larger than the sum of their individual utilities. Additional costs, however, may need to be incurred to make them work cooperatively, which need to be considered in the assignment process, along with the potential benefits to be reaped. This is similar to the assignment problem with non-additive utility functions studied in Chapter 4, but a major difference is that the total (coverage-based) utility of two (or more) workers for a task in the model considered in Chapter 4 cannot be larger than the sum of their individual utilities.

In the online, preference-aware task assignment problem studied in Chapter 5, we assume a system model with uniform task rewards and worker qualities. If an MCS system contains different types of non-trivial tasks, this assumption may lose its practicality. In future work, we would like to extend the results of this study by considering a more general system model.

Another interesting problem is to find preference-aware task assignments in an MCS system that contains both participatory and opportunistic workers. From the perspective of task requesters, it may be desirable to first hire opportunistic workers to minimize costs, and then to hire participatory workers to maximize coverage.

However, this would have to be done considering their budget or capacity constraints as well as the preferences of workers.

In Chapter 7, we have studied the problem of finding maximum size (coverage) task assignments with minimum user unhappiness. This is a special case of the problem of finding maximum weight task assignments with minimum unhappiness. The latter, general version of the problem has the potential to capture a wider range of MCS applications, as it would enable the platform to assign different priorities to each worker-task pair. Besides, in this chapter, we have made the simplifying assumption that the system utility is independent of the happiness of the users. However, continuous participation of users, which is critical in long-term system utility, naturally depends on the happiness of the users with the system. Thus, a new metric that accounts for this interdependency between system utility and user happiness can help formulate the problem more accurately.

In this dissertation, we have always assumed a system model with rational and reliable participants. However, there may be, for example, workers who are trying to spread misinformation by submitting fabricated data. When the possibility of having such malicious users are taken into consideration, user preferences may become uncertain. We have also assumed that the sets of workers and tasks were known to the matching platform before the sensing campaign actually starts. Yet for many real-world application, a more realistic model would allow users to join and leave the system, and allow task requesters to publish new tasks and withdraw some of their existing tasks in real time during the campaign. Lastly, we note that it is possible to improve the long-term efficiency of the proposed algorithms by forming the task assignments for an assignment period by modifying the assignments in the previous task assignment period(s) instead of creating a new task assignment from scratch in each assignment period. This has a potential to largely reduce the total running time

of the proposed algorithms, especially in MCS applications, where user preferences do not change significantly between consecutive assignment periods. We plan to investigate these issues in our future work.

REFERENCES

[1] Jinwei Liu et al. "A survey of mobile crowdsensing techniques: A critical component for the internet of things". In: *ACM Transactions on Cyber-Physical Systems* 2.3 (2018), pp. 1–26.

[2] Thomas J Matarazzo et al. "Crowdsensing framework for monitoring bridge vibrations using moving smartphones". In: *Proceedings of the IEEE* 106.4 (2018), pp. 577–593.

[3] Paula Fraga-Lamas et al. "A review on internet of things for defense and public safety". In: *Sensors* 16.10 (2016), p. 1644.

[4] Qiang Xu, Rong Zheng, and Ezzeldin Tahoun. "Detecting location fraud in indoor mobile crowdsensing". In: *Proceedings of the First ACM Workshop on Mobile Crowdsensing Systems and Applications*. 2017, pp. 44–49.

[5] José M Cecilia et al. "Mobile crowdsensing approaches to address the COVID-19 pandemic in Spain". In: *IET Smart Cities* 2.2 (2020), pp. 58–63.

[6] Yohan Chon, Yunjong Kim, and Hojung Cha. "Autonomous place naming system using opportunistic crowdsensing and knowledge from crowdsourcing". In: *2013 ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE. 2013, pp. 19–30.

[7] Bin Guo et al. "From participatory sensing to mobile crowd sensing". In: *2014 IEEE International Conference on Pervasive Computing and Communication Workshops (PERCOM WORKSHOPS)*. IEEE. 2014, pp. 593–598.

[8] Maotian Zhang et al. “Quality-aware sensing coverage in budget-constrained mobile crowdsensing networks”. In: *IEEE Transactions on Vehicular Technology* 65.9 (2015), pp. 7698–7707.

[9] Jiaoyan Chen and Jingsen Yang. “Maximizing coverage quality with budget constrained in mobile crowd-sensing network for environmental monitoring applications”. In: *Sensors* 19.10 (2019), p. 2399.

[10] Merkouris Karaliopoulos, Orestis Telelis, and Iordanis Koutsopoulos. “User recruitment for mobile crowdsensing over opportunistic networks”. In: *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE. 2015, pp. 2254–2262.

[11] Wei Gong, Baoxian Zhang, and Cheng Li. “Location-based online task assignment and path planning for mobile crowdsensing”. In: *IEEE Transactions on Vehicular Technology* 68.2 (2018), pp. 1772–1783.

[12] Tingting Hu et al. “A QoS-sensitive task assignment algorithm for mobile crowdsensing”. In: *Pervasive and Mobile Computing* 41 (2017), pp. 333–342.

[13] Menatalla Abououf et al. “Multi-worker multi-task selection framework in mobile crowd sourcing”. In: *Journal of Network and Computer Applications* 130 (2019), pp. 52–62.

[14] Tong Liu, Yanmin Zhu, and Liqun Huang. “TGBA: A two-phase group buying based auction mechanism for recruiting workers in mobile crowd sensing”. In: *Computer Networks* 149 (2019), pp. 56–75.

[15] Jiangtian Nie et al. “A stackelberg game approach toward socially-aware incentive mechanisms for mobile crowdsensing”. In: *IEEE Transactions on Wireless Communications* 18.1 (2018), pp. 724–738.

[16] Mingjun Xiao et al. “Secret-sharing-based secure user recruitment protocol for mobile crowdsensing”. In: *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE. 2017, pp. 1–9.

[17] Leye Wang et al. “Location privacy-preserving task allocation for mobile crowdsensing with differential geo-obfuscation”. In: *Proceedings of the 26th International Conference on World Wide Web*. 2017, pp. 627–636.

[18] Zhibo Wang et al. “Personalized privacy-preserving task allocation for mobile crowdsensing”. In: *IEEE Transactions on Mobile Computing* 18.6 (2018), pp. 1330–1341.

[19] Hui Cai et al. “Truthful incentive mechanisms for mobile crowd sensing with dynamic smartphones”. In: *Computer Networks* 141 (2018), pp. 1–16.

[20] Xi Tao and Wei Song. “Task allocation for mobile crowdsensing with deep reinforcement learning”. In: *2020 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE. 2020, pp. 1–7.

[21] Bowen Zhao et al. “itam: Bilateral privacy-preserving task assignment for mobile crowdsensing”. In: *IEEE Transactions on Mobile Computing* (2020).

[22] Ji Li et al. “Truthful incentive mechanisms for geographical position conflicting mobile crowdsensing systems”. In: *IEEE Transactions on Computational Social Systems* 5.2 (2018), pp. 324–334.

[23] Yufeng Zhan et al. “Incentive-aware time-sensitive data collection in mobile opportunistic crowdsensing”. In: *IEEE Transactions on Vehicular Technology* 66.9 (2017), pp. 7849–7861.

[24] Yufeng Zhan et al. “Incentive mechanism design in mobile opportunistic data collection with time sensitivity”. In: *IEEE Internet of Things Journal* 5.1 (2017), pp. 246–256.

[25] Leyla Kazemi and Cyrus Shahabi. “Geocrowd: enabling query answering with spatial crowdsourcing”. In: *Proceedings of the 20th international conference on advances in geographic information systems*. 2012, pp. 189–198.

[26] Zhenzhe Zheng et al. “A budget feasible incentive mechanism for weighted coverage maximization in mobile crowdsensing”. In: *IEEE Transactions on Mobile Computing* 16.9 (2016), pp. 2392–2407.

[27] Yan Liu et al. “TaskMe: Multi-task allocation in mobile crowd sensing”. In: *Proceedings of the 2016 ACM international joint conference on pervasive and ubiquitous computing*. 2016, pp. 403–414.

[28] Yan Zhao et al. “Destination-aware task assignment in spatial crowdsourcing: A worker decomposition approach”. In: *IEEE Transactions on Knowledge and Data Engineering* 32.12 (2019), pp. 2336–2350.

[29] En Wang et al. “An efficient prediction-based user recruitment for mobile crowdsensing”. In: *IEEE Transactions on Mobile Computing* 17.1 (2017), pp. 16–28.

[30] Mingjun Xiao et al. “Online task assignment for crowdsensing in predictable mobile social networks”. In: *IEEE Transactions on Mobile Computing* 16.8 (2016), pp. 2306–2320.

[31] Zongjian He, Jiannong Cao, and Xuefeng Liu. “High quality participant recruitment in vehicle-based crowdsourcing using predictable mobility”. In:

2015 IEEE Conference on Computer Communications (INFOCOM). IEEE. 2015, pp. 2542–2550.

[32] Chenxin Dai et al. "Stable task assignment for mobile crowdsensing with budget constraint". In: *IEEE Transactions on Mobile Computing* (2020).

[33] Menatalla Abououf et al. "Gale-shapley matching game selection—A framework for user satisfaction". In: *IEEE Access* 7 (2018), pp. 3694–3703.

[34] Xiaoyan Yin et al. "Matchmaker: Stable Task Assignment with Bounded Constraints for Crowdsourcing Platforms". In: *IEEE Internet of Things Journal* (2020).

[35] Yanjiao Chen and Xiaoyan Yin. "Stable job assignment for crowdsourcing". In: *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE. 2017, pp. 1–6.

[36] Bin Guo et al. "Activecrowd: A framework for optimized multitask allocation in mobile crowdsensing systems". In: *IEEE Transactions on Human-Machine Systems* 47.3 (2016), pp. 392–403.

[37] Jiangtao Wang et al. "HyTasker: Hybrid task allocation in mobile crowd sensing". In: *IEEE Transactions on Mobile Computing* 19.3 (2019), pp. 598–611.

[38] Wei Gong et al. "Task Allocation in Semi-Opportunistic Mobile Crowdsensing: Paradigm and Algorithms". In: *Mobile Networks and Applications* (2019), pp. 1–11.

[39] Jing Wang et al. "Towards energy-efficient task scheduling on smartphones in mobile crowd sensing systems". In: *Computer Networks* 115 (2017), pp. 100–109.

[40] Jiangtao Wang et al. "Task allocation in mobile crowd sensing: State-of-the-art and future opportunities". In: *IEEE Internet of Things journal* 5.5 (2018), pp. 3747–3757.

[41] Leigh Tesfatsion. *Gale-shapley matching in an evolutionary trade network game*. Tech. rep. 1998.

[42] D Gale and L Shapley. *College Admissions and Stability of Marriage. American Mathematicas Monthly, 69, 9-15*. 1962.

[43] Violet T Ho, Sze-Sze Wong, and Chay Hoon Lee. "A tale of passion: Linking job passion and cognitive engagement to employee work performance". In: *Journal of Management Studies* 48.1 (2011), pp. 26–47.

[44] Fatih Yucel, Murat Yuksel, and Eyuphan Bulut. "QoS-based budget constrained stable task assignment in mobile crowdsensing". In: *IEEE Transactions on Mobile Computing* (2020).

[45] Fatih Yucel, Murat Yuksel, and Eyuphan Bulut. "Coverage-aware Stable Task Assignment in Opportunistic Mobile Crowdsensing". In: *IEEE Transactions on Vehicular Technology* (2021).

[46] Fatih Yucel and Eyuphan Bulut. "Online Stable Task Assignment in Opportunistic Mobile Crowdsensing with Uncertain Trajectories". In: *IEEE Internet of Things Journal* (2021).

[47] Fatih Yucel and Eyuphan Bulut. "User satisfaction aware maximum utility task assignment in mobile crowdsensing". In: *Computer Networks* 172 (2020), p. 107156.

[48] Andrea Capponi et al. "A survey on mobile crowdsensing systems: Challenges, solutions, and opportunities". In: *IEEE communications surveys & tutorials* 21.3 (2019), pp. 2419–2465.

[49] David Manlove. *Algorithmics of matching under preferences*. Vol. 2. World Scientific, 2013.

[50] Jie Wu, Mingjun Xiao, and Liusheng Huang. "Homing spread: Community home-based multi-copy routing in mobile social networks". In: *2013 Proceedings IEEE INFOCOM*. IEEE. 2013, pp. 2319–2327.

[51] Aashish Dhungana and Eyuphan Bulut. "Energy sharing based content delivery in mobile social networks". In: *2019 IEEE 20th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE. 2019, pp. 1–9.

[52] Yong Li et al. "Contact-aware data replication in roadside unit aided vehicular delay tolerant networks". In: *IEEE Transactions on Mobile Computing* 15.2 (2015), pp. 306–321.

[53] *National Resident Matching Program*. 2021. URL: `https://www.nrmp.org/matching-algorithm/`.

[54] Zhenyu Zhou et al. "Energy-efficient stable matching for resource allocation in energy harvesting-based device-to-device communications". In: *IEEE access* 5 (2017), pp. 15184–15196.

[55] Tao Wang et al. "Dynamic SDN controller assignment in data center networks: Stable matching with transfers". In: *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE. 2016, pp. 1–9.

[56] Rongqing Zhang, Xiang Cheng, and Liuqing Yang. "Flexible energy management protocol for cooperative EV-to-EV charging". In: *IEEE Transactions on Intelligent Transportation Systems* 20.1 (2018), pp. 172–184.

[57] David Gale and Marilda Sotomayor. "Some remarks on the stable matching problem". In: *Discrete Applied Mathematics* 11.3 (1985), pp. 223–232.

[58] Kazuo Iwama, Shuichi Miyazaki, and Hiroki Yanagisawa. "Approximation algorithms for the sex-equal stable marriage problem". In: *ACM Transactions on Algorithms (TALG)* 7.1 (2010), pp. 1–17.

[59] Robert W Irving, Paul Leather, and Dan Gusfield. "An efficient algorithm for the "optimal" stable marriage". In: *Journal of the ACM (JACM)* 34.3 (1987), pp. 532–543.

[60] Dan Gusfield. "Three fast algorithms for four problems in stable marriage". In: *SIAM Journal on Computing* 16.1 (1987), pp. 111–128.

[61] Robert W Irving and Paul Leather. "The complexity of counting stable marriages". In: *SIAM Journal on Computing* 15.3 (1986), pp. 655–667.

[62] Péter Biró, David F Manlove, and Shubham Mittal. "Size versus stability in the marriage problem". In: *Theoretical Computer Science* 411.16-18 (2010), pp. 1828–1841.

[63] Patrik Floréen et al. "Almost stable matchings by truncating the Gale–Shapley algorithm". In: *Algorithmica* 58.1 (2010), pp. 102–118.

[64] Rafail Ostrovsky and Will Rosenbaum. "Fast distributed almost stable matchings". In: *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*. 2015, pp. 101–108.

[65] Anisse Ismaili et al. “Weighted matching markets with budget constraints”. In: *Journal of Artificial Intelligence Research* 65 (2019), pp. 393–421.

[66] Yasushi Kawase and Atsushi Iwasaki. “Approximately stable matchings with budget constraints”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[67] Boming Zhao et al. “Preference-aware task assignment in on-demand taxi dispatching: An online stable matching approach”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 2245–2252.

[68] Huanyang Zheng and Jie Wu. “Online to offline business: urban taxi dispatching with passenger-driver matching stability”. In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2017, pp. 816–825.

[69] Haris Aziz et al. “Stable matching with uncertain linear preferences”. In: *Algorithmica* 82.5 (2020), pp. 1410–1433.

[70] Robert Bredereck et al. “Adapting stable matchings to evolving preferences”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 02. 2020, pp. 1830–1837.

[71] Kimmo Eriksson, Jonas Sjöstrand, and Pontus Strimling. “Optimal expected rank in a two-sided secretary problem”. In: *Operations research* 55.5 (2007), pp. 921–931.

[72] Donald Ervin Knuth and NG De Bruijn. *Stable marriage and its relation to other combinatorial problems: An introduction to the mathematical analysis of algorithms*. Vol. 10. American Mathematical Soc., 1997.

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

[74] Kazuo Iwama, Shuichi Miyazaki, and Kazuya Okamoto. "Stable roommates problem with triple rooms". In: *Proc. 10th KOREA-JAPAN joint workshop on algorithms and computation (WAAC 2007)*. 2007, pp. 105–112.

[75] Robert W Irving. "An efficient algorithm for the "stable roommates" problem". In: *Journal of Algorithms* 6.4 (1985), pp. 577–595.

[76] Lin Cui and Weijia Jia. "Cyclic stable matching for three-sided networking services". In: *Computer Networks* 57.1 (2013), pp. 351–363.

[77] Wikipedia contributors. *Knapsack problem — Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/w/index.php?title=Knapsack_problem&oldid=942139230`. [Online; accessed 3-March-2020]. 2020.

[78] *Waze*. 2021. URL: `https://www.waze.com/`.

[79] Ismel Brito and Pedro Meseguer. "Distributed stable marriage problem". In: *6th Workshop on Distributed Constraint Reasoning at IJCAI*. Vol. 5. Citeseer. 2005, pp. 135–147.

[80] Yongjian Yang et al. "A prediction-based user selection framework for heterogeneous mobile crowdsensing". In: *IEEE Transactions on Mobile Computing* 18.11 (2018), pp. 2460–2473.

[81] *Taxi and limousine commission (tlc) trip record data.* NYC Taxi Limousine Commission. 2021. URL: `https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page`.

[82] Kazuo Iwama and Shiro Taketomi. "Removable online knapsack problems". In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2002, pp. 293–305.

[83] *Amazon mechanical turk*. 2021. URL: https://www.mturk.com/.

[84] Dror Rawitz and Adi Rosén. "Online budgeted maximum coverage". In: *24th Annual European Symposium on Algorithms (ESA 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2016.

[85] Injong Rhee et al. "On the levy-walk nature of human mobility". In: *IEEE/ACM transactions on networking* 19.3 (2011), pp. 630–643.

[86] Injong Rhee et al. *CRAWDAD dataset ncsu/mobilitymodels (v. 2009-07-23)*. Downloaded from https://crawdad.org/ncsu/mobilitymodels/20090723. 2009. DOI: 10.15783/C7X302.

[87] Ke Yan et al. "A comprehensive location-privacy-awareness task selection mechanism in mobile crowd-sensing". In: *IEEE Access* 7 (2019), pp. 77541–77554.

[88] Bin Guo et al. "Worker-contributed data utility measurement for visual crowd-sensing systems". In: *IEEE Transactions on Mobile Computing* 16.8 (2016), pp. 2379–2391.

[89] Rajib Rana et al. "Ear-Phone: A context-aware noise mapping using smart phones". In: *Pervasive and Mobile Computing* 17 (2015), pp. 1–22.

[90] Pruthvish Rajput, Manish Chaturvedi, and Vivek Patel. "Opportunistic sensing based detection of crowdedness in public transport buses". In: *Pervasive and Mobile Computing* 68 (2020), p. 101246.

[91] Yongxin Tong et al. "Online mobile micro-task allocation in spatial crowdsourcing". In: *2016 IEEE 32Nd international conference on data engineering (ICDE)*. IEEE. 2016, pp. 49–60.

[92] Jianbing Ni et al. "Enabling strong privacy preservation and accurate task allocation for mobile crowdsensing". In: *IEEE Transactions on Mobile Computing* 19.6 (2019), pp. 1317–1331.

[93] *Uber*. 2021. URL: https://www.uber.com/.

[94] Dejiang Kong and Fei Wu. "HST-LSTM: A Hierarchical Spatial-Temporal Long-Short Term Memory Network for Location Prediction." In: *IJCAI*. Vol. 18. 7. 2018, pp. 2341–2347.

[95] Michael D Lee. "A hierarchical bayesian model of human decision-making on an optimal stopping problem". In: *Cognitive science* 30.3 (2006), pp. 1–26.

[96] Michal Piorkowski, Natasa Sarafijanovic-Djukic, and Matthias Grossglauser. *CRAWDAD dataset epfl/mobility (v. 2009-02-24)*. URL: https://crawdad.org/epfl/mobility/20090224.

[97] *NYC Points of Interest*. Department of Information Technology & Telecommunications (DOITT). 2021. URL: https://data.cityofnewyork.us/City-Government/Points-Of-Interest/rxuy-2muj#revert.

[98] *Google Directions API*. 2021. URL: https://developers.google.com/maps/documentation/directions/overview.

[99] *Google OR-Tools*. 2021. URL: https://developers.google.com/maps/documentation/directions/overview.

[100] Xiong Wang et al. “Dynamic task assignment in crowdsensing with location awareness and location diversity”. In: *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE. 2018, pp. 2420–2428.

[101] Claudio Fiandrino et al. “Sociability-driven user recruitment in mobile crowdsensing internet of things platforms”. In: *2016 IEEE Global Communications Conference (GLOBECOM)*. IEEE. 2016, pp. 1–6.

[102] Harold W Kuhn. “The Hungarian method for the assignment problem”. In: *Naval research logistics quarterly* 2.1-2 (1955), pp. 83–97.

[103] Lester Randolph Ford and Delbert R Fulkerson. “Maximal flow through a network”. In: *Canadian journal of Mathematics* 8 (1956), pp. 399–404.

[104] Guoju Gao et al. “Truthful incentive mechanism for nondeterministic crowdsensing with vehicles”. In: *IEEE Transactions on Mobile Computing* 17.12 (2018), pp. 2982–2997.

VITA

Fatih Yucel received a B.S. degree in computer engineering from Gazi University in Turkey in 2017. He is currently pursuing a Ph.D. degree in the Computer Science Department of Virginia Commonwealth University under the supervision of Dr. Eyuphan Bulut, and working on development of stable task assignment algorithms for mobile crowdsensing applications.

Publications

1. **Fatih Yucel**, Kemal Akkaya, and Eyuphan Bulut. "Efficient and privacy preserving supplier matching for electric vehicle charging." Ad Hoc Networks 90 (2019): 101730.

2. **Fatih Yucel** and Eyuphan Bulut. "Clustered crowd GPS for privacy valuing active localization." IEEE Access 6 (2018): 23213-23221.

3. **Fatih Yucel**, Kemal Akkaya, and Eyuphan Bulut. "Privacy preserving distributed stable matching of electric vehicles and charge suppliers." 2018 IEEE 88th Vehicular Technology Conference (VTC-Fall), 2018.

4. **Fatih Yucel** and Eyuphan Bulut. "User satisfaction aware maximum utility task assignment in mobile crowdsensing." Computer Networks 172 (2020): 107156.

5. **Fatih Yucel**, Murat Yuksel, and Eyuphan Bulut. "QoS-based budget constrained stable task assignment in mobile crowdsensing." IEEE Transactions on Mobile Computing (2020).

6. **Fatih Yucel**, Arupjyoti Bhuyan, and Eyuphan Bulut. "Secure, resilient and stable resource allocation for D2D-based V2X communication." 2020 Resilience Week (RWS), IEEE, 2020.

7. **Fatih Yucel** and Eyuphan Bulut. "Location-dependent task assignment for opportunistic mobile crowdsensing." 2020 IEEE 17th Annual Consumer Communications Networking Conference (CCNC), 2020.

8. **Fatih Yucel** and Eyuphan Bulut. "Joint optimization of system and user oriented task assignment in mobile crowdsensing." 2019 IEEE Global Communications Conference (GLOBECOM), 2019.

9. Chathurika S. Wickramasinghe, Daniel Marino, **Fatih Yucel**, Eyuphan Bulut, and Milos Manic. "Data driven hourly taxi drop-offs prediction using tlc trip record data." 2019 12th International Conference on Human System Interaction (HSI), 2019.

10. **Fatih Yucel**, Murat Yuksel, and Eyuphan Bulut. "Coverage-aware stable task assignment in opportunistic mobile crowdsensing." IEEE Transactions on Vehicular Technology (2021).

11. **Fatih Yucel** and Eyuphan Bulut. "Time-dependent stable task assignment in participatory mobile crowdsensing." 2020 IEEE 45th Conference on Local Computer Networks (LCN), 2020.

12. **Fatih Yucel** and Eyuphan Bulut. "Online stable task assignment in opportunistic mobile crowdsensing with uncertain trajectories." IEEE Internet of Things Journal (2021).

13. **Fatih Yucel** and Eyuphan Bulut. "Three-dimensional stable task assignment in semi-opportunistic mobile crowdsensing." (Submitted for publication).