Computer Networks 172 (2020) 107156

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

User satisfaction aware maximum utility task assignment in mobile crowdsensing

Fatih Yucel, Eyuphan Bulut*

Department of Computer Science, Virginia Commonwealth University, 401 West Main, St. Richmond, VA 23284, USA

ARTICLE INFO

Keywords:
Mobile crowdsensing
Task assignment
Stable matching

ABSTRACT

In mobile crowdsensing systems (MCS) efficient task assignment is the key problem that defines the performance of the system. The current state-of-the-art solutions consider the problem from system's point of view and target an assignment that optimizes the overall system utility such as minimizing the cost of sensing or maximizing the collected data quality. However, users (i.e., task requesters and task performers or workers) may have individual preferences, hence the resulting assignment may not satisfy the users and can discourage them from participation in the future. Stable matching based solutions can help achieving satisfactory assignments for the users, but they may degrade the system utility especially when the number of eligible task performers for each task is limited, hence may not be desired for the MCS platform. To address this problem, in this paper, we study the task assignment problem that aims to maximize the system utility and user satisfaction simultaneously as much as possible. As the problem is NP-complete, we first solve the problem using Integer Linear Programming (ILP) and provide two different heuristic based polynomial solutions. We perform extensive simulations using real dataset and show that the proposed solutions provide close to optimal results, complementing each other at different scenarios.

1. Introduction

Mobile crowdsensing (MCS) has emerged as an effective approach to accomplish large scale sensing and computation tasks that are beyond the capabilities of the task requesters themselves [1]. Mobile users carrying devices equipped with various sensors (e.g., microphone, camera, and GPS) are recruited to efficiently carry out the tasks that require massive expenses and execution times when performed individually. There are many applications of MCS systems today in use such as traffic [2] and air quality monitoring [3].

An MCS system consists of a platform, requesters, tasks and workers[1]. Task requesters post a set of tasks with different requirements such as a deadline to complete the task and a reward for completing the task. The workers register to the system together with their capabilities and any applicable restrictions (e.g., can only perform tasks in a specific region, or can perform a task with at least a minimum reward). The platform defines the workers eligible for each task and either automatically matches them based on some optimization goal or let the workers and task requesters communicate and agree on a task allocation in a distributed manner based on their preferences, where the platform only serves as the advertiser. For example, a worker may prefer to take the tasks that will provide more profit with a minimal effort based on the worker's capabilities.

MCS can be performed either opportunistically or in a participatory way. In the opportunistic MCS, workers passively contribute to the completion of tasks (e.g., sensing humidity in some region of the city) without changing their mobility. On the contrary, in participatory sensing, a central authority usually assigns workers to the tasks, and the workers actively move to the task location to complete the tasks in return of some reward. Depending on the MCS application and its requirements, either one may have advantages over the other. In both scenarios, the main challenging problem is the assignment of tasks to users under some optimization goal. However, the nature of the task assignment process changes depending on the scenario.

There have been many studies [4–16] performed to propose solutions to this task assignment problem under different scenarios and limitations (e.g., budget, maximum distance to travel). In most of these studies, however, the problem is formulated as a maximum system utility (e.g., number of completed tasks, quality of completed tasks, average time needed to complete a task) problem and users' individual goals are overlooked. However, users may not want to sacrifice their individual

* Corresponding author.
E-mail addresses: yucelf@vcu.edu (F. Yucel), ebulut@vcu.edu (E. Bulut).

[1] We utilize the term "user" to refer to both task requesters and workers. Workers are the users who are recruited to perform tasks.

https://doi.org/10.1016/j.comnet.2020.107156
Received 14 September 2019; Received in revised form 19 December 2019; Accepted 17 February 2020
Available online 19 February 2020
1389-1286/

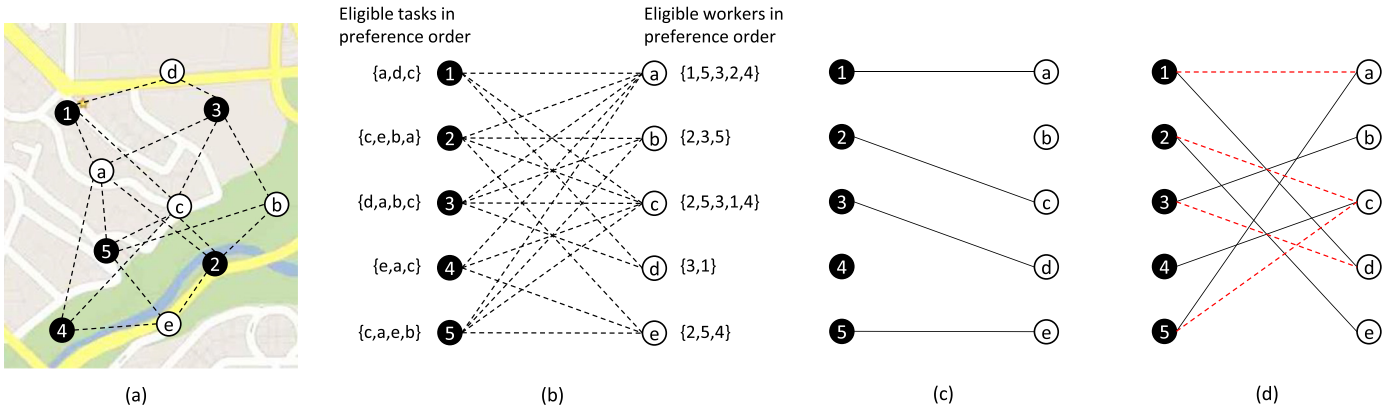

Fig. 1. An MCS scenario with 5 workers and 5 tasks, which are respectively denoted by numbers and letters. (a) The task and current worker locations on the map (workers eligible to perform a task is connected with an edge to that task); (b) corresponding bipartite graph with the list and preference order (from left to right) of workers and tasks; (c) Deferred acceptance based stable matching (Gale–Shapley algorithm [21]) leaving 4 and b unassigned, yielding a lower system utility; and (d) An assignment that maximizes the system utility but yielding unhappy users (shown with dashed edges).

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. Recently, stable matching based solutions are adopted to address this problem [17–20], and task assignments that are aware of user preferences are developed to make the users happy with their assignments. However, these solutions may reduce the system utility (e.g., some tasks and workers left unassigned) while trying to satisfy users in some cases.

We illustrate this tradeoff between user happiness and system utility through an example MCS scenario with 5 tasks and 5 workers shown in Fig. 1a, which will be referenced throughout the paper to describe the problem and the 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. 1b (we will describe how users define their preferences in Section 3). 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 [21]. There can be many such *stable matchings* in a single matching instance with the same size, but there is only one in our example which is shown in Fig. 1c. 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. 1d), 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. 1d, which contains 4 unhappy pairs. In this paper, we address this problem, which turns out to be NP-complete, and propose two polynomial time heuristic algorithms to find the matching that contains as few unhappy pairs as possible among all matchings with maximum system utility. The contributions of this paper can be summarized as follows:

- We formulate the user satisfaction aware maximum utility task assignment problem and solve it optimally using Integer Linear Programming (ILP).
- We provide two heuristic based efficient solutions that run in polynomial time using different approaches.
- We perform extensive real dataset based simulations covering numerous scenarios and show that the proposed solutions provide close to optimal results and complement each other at different scenarios.

In the preliminary version of this study [22], we only provide a single heuristic based solution (i.e., *Stable to Maximum*) and limited simulation results. The rest of the paper is organized as follows. In Section 2, we present an overview of related work. In Section 3, we provide the motivation of the study together with a background on stable matching and problem definition. In Section 4, we elaborate on the ILP solution and heuristic based approaches. In Section 5, we present an extensive evaluation of the proposed approaches through real data based simulations. Finally, we end up with conclusion in Section 6.

2. Related work

2.1. Mobile crowdsensing

Mobile crowdsensing has received a great attention in recent years and several aspects have been studied by many researchers. A primary component of MCS systems is *incentive mechanisms*, which determine how the participants of the system should be incentivized to perform assigned tasks. In some studies, participants are simply assumed to be self-incentivized due to the entertaining [23] or mutually beneficial [24] nature of the tasks tackled in the system, or by the fact that the objective of the task coincides with their political, cultural, environmental or religious views. However, for the majority of the MCS systems that involve sensing tasks that require the workers to spend their own resources (e.g., time, travel costs) to achieve the assigned tasks, it may not be feasible to assume self-incentivized workers. In such systems, the workers should be compensated financially by the crowdsourcer for their efforts and disbursements. In order to achieve this, various incentive mechanisms (e.g., auctions, lottery contests, reputation based models) have been proposed [25], among which auction models are the most popular and commonly used [26–28]. In these models, generally, workers estimate and announce the level of effort they will have to put in to complete each task, and the platform strategically selects the workers and determines how much they will be rewarded while ensuring the quality of the sensed data.

Another challenge in MCS systems that is as crucial as incentive mechanisms is how to assign the requested tasks to the workers given the incentives that the task requesters are willing to provide and the cost of performing these tasks for each worker. This problem is referred to as *the task assignment/allocation or user recruitment problem* in the MCS literature, and has been studied quite extensively [4–16]. In these works, different objectives have been considered such as maximizing the number of completed tasks, minimizing the completion times of tasks [11], minimizing the incentives provided to the users [12,13], assuring the task or sensing quality [14–16] within some limits such as budget [29], energy consumption [15] and traveling distance [14]. Beyond these works, security [30], privacy [5,31,32], and trustfulness [6,33] of workers have also been considered during the recruitment process.

Despite the variety of the literature on the task assignment and incentive models in MCS systems, the goal is mostly defined from overall system's point of view without considering the user preferences. Most of these studies aim to maximize the overall system utility according to a specific performance metric, but fail to take the user preferences into account. Consequently, in the proposed models, some worker and task requester pairs end up preferring each other to their assignments, which leads to unstable and unfair results, discouraging users from future participation.

2.2. Stable matching

Stable Matching (SM) problem is introduced in the seminal paper of Gale and Shapley [21] 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 Gale and Shapley [21], the concept of stability has been utilized in different problems including hospital resident assignment [34], resource allocation in device-to-device communications [35], SDN controller assignment in data center networks [36], and electric vehicle charging [37].

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, Gale and Sotomayor [38] proves that the set of matched nodes is identical in all stable matchings, therefore all stable matchings are of the same size. Iwama et al. [39] 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. Irving et al. [40] 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, Gusfield [41] 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 problem of size $N = 32$ is larger than 10^{11}) and grows exponentially with increasing problem size [42], 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.

Recently, the concept of stability has also been utilized [17–20] in mobile crowdsensing (and crowdsourcing) systems. In terms of considering user preferences in the task assignment, the closest studies to this paper are [18,20]. In [18], the authors study the many-to-one stable job assignment problem under budget constraints. They define the unique stability conditions for many-to-one matching scenarios where a crowdsourcer would like to recruit as many workers as his budget permits, and propose an ILP-based algorithm to find pairwise stable matchings. On the other hand, Abououf et al. [20] focuses on the problem of finding many-to-many stable task assignments under capacity constraints and provides a polynomial time heuristic algorithm that aims to maximize user satisfaction in the resulting matching according to user preferences. However, neither of these studies addresses the issue of flawed system utility (or matching size) that comes with stable matchings. This issue is firstly addressed in Biró et al. [43] 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 [44,45] have mostly focused on reducing the running time by sacrificing from the stability, for which they propose truncated versions of the deferred acceptance procedure.

Despite the extensive studies in stable matching literature, to the best of our knowledge, there is no study that provides an experimental analysis that shows the severity of system utility loss in stable matchings, or proposes a heuristic algorithm to solve the problem of finding a maximum system utility matching with as few unhappy pairs (i.e., instability) as possible by trading the optimality for speed. In order to address these, this paper first investigates the trade-off between the system utility and the stability (user satisfaction), and then proposes two polynomial time heuristic algorithms that achieve very close to optimal results in different scenarios.

3. System model

3.1. Assumptions

Let $\mathcal{W} = \{w_1, w_2, \dots, w_n\}$ denote the set of $|\mathcal{W}| = n$ workers and $\mathcal{T} = \{t_1, t_2, \dots, t_m\}$ denote the set of $|\mathcal{T}| = m$ tasks in the system. Also, let c_{ij} denote the cost[2] 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:

$$\mathcal{E}(w_i) = \{t_j | r_j \geq c_{ij},\ \forall j \in [1, \dots, m]\} \tag{1}$$

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 $r_j - c_{ij} > r_{j'} - c_{ij'}$.

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:

$$\mathcal{E}(t_j) = \{w_i | r_j \geq c_{ij},\ \forall i \in [1, \dots, n]\} \tag{2}$$

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

[2] Note that cost can be defined with a complicated function that considers the worker's traveling and task completion duration due to spatio-temporal 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.

Table 1
Notations

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 $\mathcal{M}$.
$\|U(\mathcal{M})\|$	Unhappiness Index (UI).
$\mathcal{E}(u)$	Eligible tasks/workers for worker/task u.
$\|\mathcal{E}\|$	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.

dependent on the traveling distance from the worker location to the task location [7,46], 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 workers can provide. Given the eligibility relations, the corresponding undirected bipartite graph $G = (V, E)$ can be defined as

$$\begin{aligned} G.V &= \mathcal{W} \cup \mathcal{T} \\ G.E &= \{(u, v) | u, v \in G.V, u \in \mathcal{E}(v), v \in \mathcal{E}(u)\} \end{aligned} \tag{3}$$

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 the paper are summarized in Table 1.

3.2. Trade-off analysis

Having the set of eligible workers for each task and eligible tasks for each worker, the platform then assigns the tasks to workers with some optimization goal. An assignment aiming to maximize the system utility[3] can be obtained by constructing the corresponding maximum bipartite matching instance between workers and tasks and solving it via the well-known Hungarian algorithm [47]. Similarly, an assignment aiming to satisfy users with their assignments can be obtained using the deferred acceptance mechanism in the Gale–Shapley algorithm [21]. However, achieving both may not be possible at the same time and there is a trade-off 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 $\mathcal{M}(w) = \emptyset$, 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 $\mathcal{M}(t) = \emptyset$, or
- $w \succ_t \mathcal{M}(t)$ and $\mathcal{M}(w) = \emptyset$, or
- $\mathcal{M}(w) = \emptyset$ and $\mathcal{M}(t) = \emptyset$.

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 [43].

In Section 1, the instance in Fig. 1 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 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. 2 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

$$100 \times \frac{|\mathcal{M}_{MM}| - |\mathcal{M}_{SM}|}{|\mathcal{M}_{MM}|}.$$

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 massively larger unhappiness index (i.e., by definition, 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. 3, 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 (i.e., the set of workers and tasks have equal size), where we see the minimum but comparable unhappiness index obtained by maximum system utility matching.

In this paper, 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 matchings grows exponentially with the number of nodes. Moreover, this problem can be reduced to max cardinality with min blocking pairs problem [43], which is proven to be NP-complete, even when the size of eligible worker/task sets is 3.

4. User satisfaction aware maximum utility task assignment

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 provide the details of two different heuristic based cost efficient solutions.

4.1. ILP design

Our goal is to assign as many workers and tasks as possible with the minimum number of unhappy pairs, which can be formally defined as

[3] As 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. More complicated utility models can also be defined similarly and proposed algorithms could be updated accordingly.

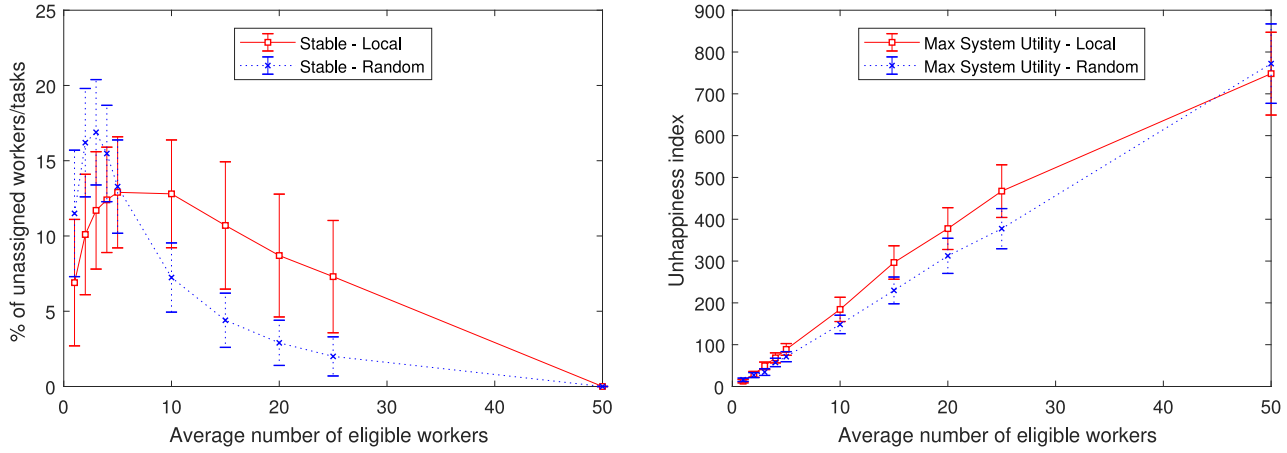

Fig. 2. Percentage of decrease in the number of assigned workers/tasks in stable matching compared to maximum system utility matching (left) and unhappiness index in maximum system utility matching (right) with varying size of eligible worker/task sets in *local* and *random* settings.

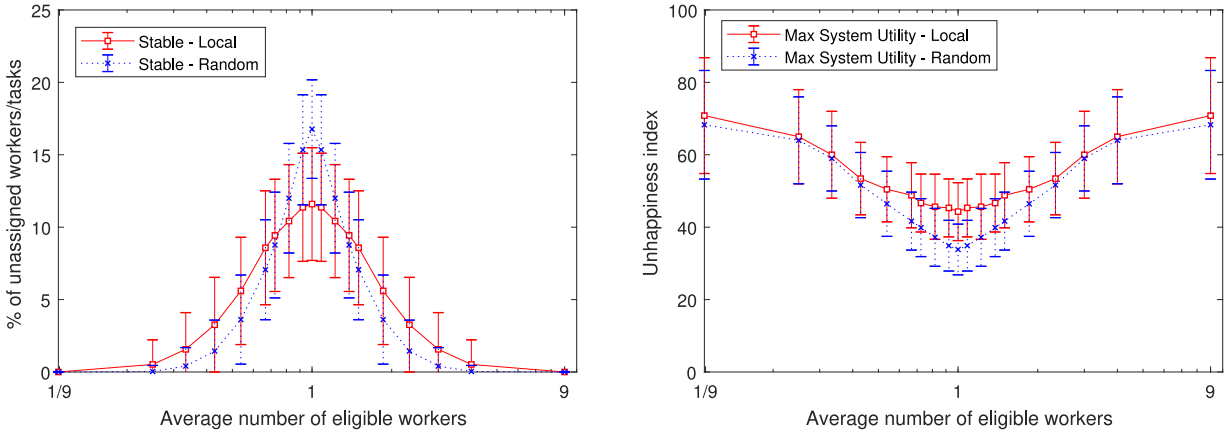

Fig. 3. Percentage of decrease in the number of assigned workers/tasks in stable matching compared to maximum system utility matching (left) and unhappiness index in maximum system utility matching (right) with different ratios of worker and task set sizes. We use an average eligible worker/task set size of 3 with the total number of tasks and workers fixed at 100.

follows:

$$\max \sum_{\forall i,j} \left(mn\mathcal{X}_{ij} - \mathcal{U}_{ij} \right) \tag{4}$$

with the constraints:

$$\sum_{\forall i} \mathcal{X}_{ij} \leq 1 \qquad \forall j$$

$$\sum_{\forall j} \mathcal{X}_{ij} \leq 1 \qquad \forall i$$

$$\mathcal{X}_{ij} \leq e_{ij} \qquad \forall i, j$$

where,

$$e_{ij} = \begin{cases} 1, & \text{if } w_i \text{ is eligible to perform } t_j \\ 0, & \text{otherwise} \end{cases}$$

$$\mathcal{X}_{ij} = \begin{cases} 1, & \text{if } w_i \text{ is assigned to } t_j \\ 0, & \text{otherwise} \end{cases}$$

$$\mathcal{U}_{ij} = \begin{cases} 1, & \text{if } (w_i, t_j) \text{ is an unhappy pair} \\ 0, & \text{otherwise} \end{cases}$$

Note that the number of unhappy pairs can be at most mn. Incrementing the assigned pair count will increase the objective function value in (4) more than removing all unhappy pairs. Thus, it first tries to reach an assignment with maximum system utility, then reduces *unhappiness index* as much as possible.

4.2. Maximum to stable reduction algorithm

The first proposed algorithm works based on a greedy heuristic which aims to first find the maximum utility matching and then try to decrease the number of unhappy pairs in it one by one without altering the total utility of the matching. Before elaborating the algorithm steps, we first describe *happify* procedure, which constitutes the core part of the algorithm.

4.2.1. 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. 4a, 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, 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

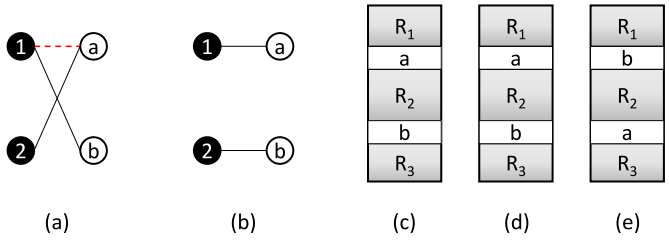

Fig. 4. An instance of happify procedure. (a) the matching $\mathcal{M}$ before happify; (b) the matching $\mathcal{M}'$ after happifying the unhappy pair $\langle 1, a\rangle$ in $\mathcal{M}$; (c) 1's preference list, P_1; (d) 2's possible preference list, P_2'; (e) 2's alternative preference list, P_2''. $\mathcal{R}_i$'s are defined in (5).

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. 4a, 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

$$(\forall x \in \mathcal{R}_1) \succ_1 a \succ_1 (\forall x \in \mathcal{R}_2) \succ_1 b \succ_1 (\forall x \in \mathcal{R}_3) \tag{5}$$

as illustrated in Fig. 4c. 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

$$\{\langle 1, x\rangle \mid x \in \mathcal{R}_2, \langle 1, x\rangle \in U(\mathcal{M})\},$$

if any, because for all $x \in \mathcal{R}_2$, $a \succ_1 x$. The set of other unhappy pairs formed as

$$\{\langle 1, x\rangle \mid x \in \mathcal{R}_1, \langle 1, x\rangle \in U(\mathcal{M})\}$$

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

$$\{\langle 1, x\rangle \mid x \in \mathcal{R}_3\}$$

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. 4d and e. Note that, regardless of P_2' or P_2'', happifying $\langle 1, a\rangle$ will not affect the unhappy pairs in

$$\{\langle 2, x\rangle \mid x \in \mathcal{R}_1, \langle 2, x\rangle \in U(\mathcal{M})\},$$

so that they will still be present in $\mathcal{M}'$, and

$$\{\langle 2, x\rangle \mid x \in \mathcal{R}_3, \langle 2, x\rangle \in U(\mathcal{M}) \cup U(\mathcal{M}')\} = \emptyset,$$

due to 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

$$\{\langle 2, x\rangle \mid x \in \mathcal{R}_2, 2 \succ_x \mathcal{M}'(x)\}$$

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

$$\{\langle 2, x\rangle \mid x \in \mathcal{R}_2, \langle 2, x\rangle \in U(\mathcal{M})\}.$$

The idea behind the happify procedure could also be applied to a group of unhappy pairs simultaneously. However, it would dramatically increase the number of permutations for the matching of former partners of nodes comprising the unhappy pairs. To address that, as it will be explained in the next section, we introduce a *phased approach*. Before going to its details, 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

$$U_N = \left\{ \langle x, y\rangle \notin U(\mathcal{M}) \mid y \succ_x \mathcal{M}'(x), x \succ_y \mathcal{M}'(y) \right\}, \tag{6}$$

and the set of unhappy pairs that were found in $\mathcal{M}$, but will disappear in $\mathcal{M}'$ is

$$U_O = \left\{ \langle x, y\rangle \in U(\mathcal{M}) \mid \mathcal{M}'(x) \succ_x y \text{ or } \mathcal{M}'(y) \succ_y x \right\}. \tag{7}$$

Then, the set of unhappy pairs in the new matching becomes

$$U(\mathcal{M}') = \left(U(\mathcal{M}) \cup U_N \right) \setminus U_O$$

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 $\mathcal{O}(N)$ complexity, where $N = \max\{m, n\}$.

4.2.2. The algorithm

The algorithm aims to reduce the number of unhappy pairs greedily through consecutive happify operations. To this end, we find the unhappy pair which, when happified, reduces the total number of unhappy pairs the most and proceed with 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.

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. 5 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 1 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 2.

Maximum to Stable Reduction algorithm, shown in Algorithm 1, begins with finding a maximum utility (i.e., cardinality) matching, $\mathcal{M}$, via Hungarian Algorithm [47]. For the first phase, when $i = 1$, we find the

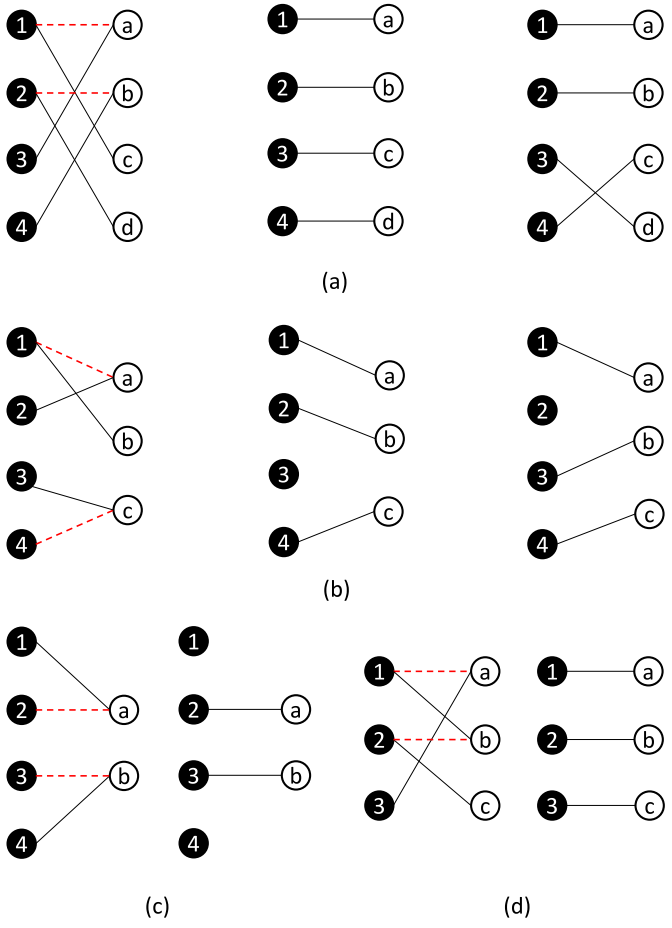

Fig. 5. Some possible happify attempts that can occur in Phase 2. Unhappy pairs are shown with red dotted lines. Once the two unhappy pairs are matched with each other in each case, remaining workers and tasks are matched with all possible permutations.

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 of our algorithm (i.e., $i = 2$), it tries to relax two unhappy pairs simultaneously (*happify* in line 13 returns all possible variations). If no improvement achieved in unhappy pair count by k hops, the algorithm ends.

4.2.3. A toy example

We provide a sample run of Algorithm 1 on the instance in Fig. 1 to demonstrate how it gradually decreases the number of unhappy pairs while preserving the maximum system utility. Firstly, a maximum matching is found via Hungarian algorithm, which, as shown in Fig. 6a, 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 2.

Algorithm 1: Maximum to stable reduction ($\mathcal{W}$, $\mathcal{T}$, k).

Input: $\mathcal{W}$: The set of workers
$\mathcal{T}$: The set of tasks
k: The number of hops

1 $\mathcal{M} \leftarrow$ Find a maximum cardinality matching via Hungarian algorithm between $\mathcal{W}$ and $\mathcal{T}$.
2 $U(\mathcal{M}) \leftarrow$ Identify the unhappy pairs in $\mathcal{M}$.
3 $\mathcal{M}_{best} \leftarrow \mathcal{M}$
4 **for** $i \leftarrow 1$ *to* 2 **do**
5 **if** $i == 2$ **then**
6 $j \leftarrow k$
7 **else**
8 $j \leftarrow 1$
9 **while** $j > 0$ **do**
10 $\mathcal{M}' \leftarrow NIL$ ▷ $|U(\mathcal{M}')| = \infty$
11 $\mathcal{S} \leftarrow \{\mathcal{A} \subseteq U(\mathcal{M}) : |\mathcal{A}| = i\}$
12 **for** $U \in \mathcal{S}$ **do**
13 **for** $\mathcal{M}_{new} \in$ *Happify*$(\mathcal{M}, U)$ **do**
14 **if** $|U(\mathcal{M}_{new})| < |U(\mathcal{M}')|$ **then**
15 $\mathcal{M}' \leftarrow \mathcal{M}_{new}$
16 **if** $|U(\mathcal{M}')| < |U(\mathcal{M}_{best})|$ **then**
17 $j \leftarrow k$
18 $\mathcal{M}_{best} \leftarrow \mathcal{M}'$
19 **else**
20 $j \leftarrow j - 1$
21 $\mathcal{M} \leftarrow \mathcal{M}'$
22 $\mathcal{M} \leftarrow \mathcal{M}_{best}$
23 **return** $\mathcal{M}_{best}$

Algorithm 2: 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 Figs. 4 and 5).
2 **foreach** $\mu \in \mathcal{M}_S$ **do**
3 Find U_N and U_O by Eqs. (6) and (7).
4 $|U(\mu)| = |U(\mathcal{M})| + |U_N| - |U_O|$
5 **return** $\mathcal{M}_S$

Table 2
Matchings to be obtained by happifying each unhappy pair in Fig. 6a.

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
$\langle 5, c\rangle$	$\langle 1, a\rangle$, $\langle 2, c\rangle$, $\langle 3, a\rangle$, $\langle 3, d\rangle$

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 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. 6b) to the second phase (any further try 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,

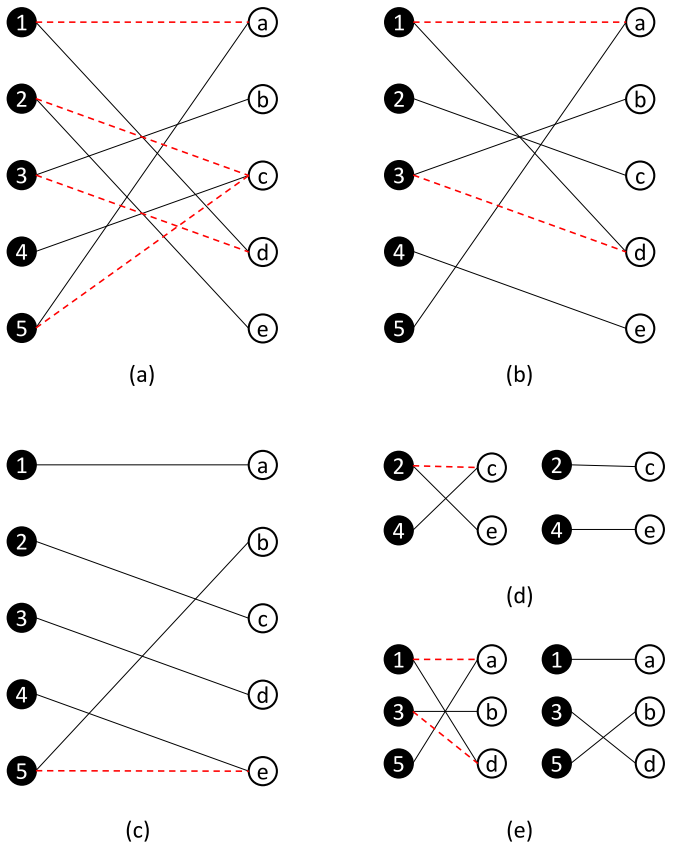

Fig. 6. Steps of running Algorithm 1 on the instance given in Fig. 1. (a) the initial maximum matching found via Hungarian algorithm; (b) the best matching reached by the end of the first phase; (c) the best matching ever found by the algorithm, also an optimal solution; (d) happifying $\langle 2, c\rangle$ on the matching in Fig. 6a; (e) happifying $\{\langle 1, a\rangle, \langle 3, d\rangle\}$ on the matching in Fig. 6b.

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. 6e. Besides, this yields a matching with just one unhappy pair, $\langle 5, e\rangle$, as shown in Fig. 6c, 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.

As for the complexity of Algorithm 1, since the loop starting at line 12 may run as many as the number of unhappy pairs in the current matching, which could be at most mn, and the computation of the benefit that can be obtained by happifying each unhappy pair takes $\mathcal{O}(N)$, where $N = \max\{m, n\}$ (see Section 4.2.1), the worst-case complexity of the first phase of our algorithm is $\mathcal{O}(N^5)$. Luckily, this can be improved as the calculation of the benefit that will come from happifying each set of unhappy pairs, U, is independent from the others, hence can be run in parallel. Thus, a time complexity of $\mathcal{O}(N^3)$ is theoretically achievable. In simulations, we obtain results showing this improvement. Similarly, the time complexity of the second phase is $\mathcal{O}(B^3N)$, where B is the number of unhappy pairs in the final matching produced by the first phase. The same parallelization approach could also be applied here to further reduce the complexity to $\mathcal{O}(B^2 + BN)$, however, as B is usually very small (as will be shown in simulations) compared to the initial number of unhappy pairs that the first phase deals with, the complexity is determined by the complexity of the first phase.

4.3. Stable to maximum convergence algorithm

In the second algorithm, we propose a reversed approach. That is, we first aim to obtain an assignment based on the preferences of both task requesters and workers so that every user is happy (i.e., *unhappiness index* is 0). Then, we update it iteratively to obtain the assignment with maximum system utility. Note that this can increase *unhappiness index* but we aim to minimize it as much as possible.

The iterative process goes through finding special paths between workers and tasks at every step that will increase the number of assigned pairs with respect to the current assignment. We simply call these paths as *beneficial* paths. Given a matching $\mathcal{M}$ defined on the bipartite graph G (defined in (3)), 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,

$$\begin{aligned}
&\mathcal{M}(p_1) = \emptyset, \mathcal{M}(p_{2j+2}) = \emptyset \\
&\mathcal{M}(p_{2i}) = p_{2i+1}, \text{ and } (p_{2i}, p_{2i+1}) \in \mathcal{M} && \forall i \in [1, \dots, j] \\
&\mathcal{M}(p_{2i-1}) \neq p_{2i}, \text{ but } (p_{2i-1}, p_{2i}) \in G.E \setminus \mathcal{M} && \forall i \in [1, \dots, j]
\end{aligned}$$

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 3. We first find a stable matching $\mathcal{M}$ between the given workers and tasks using the deferred acceptance mechanism in Gale–Shapley algorithm [21]. Then, in each iteration of the while loop in line 2, we try to find a beneficial path p in $\mathcal{M}$. If we find one, we update $\mathcal{M}$ as follows

$$\mathcal{M} \leftarrow (\mathcal{M} \setminus E(p)) \cup (E(p) \setminus \mathcal{M}),$$

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 [48] and will be returned as the final matching.

The procedure of finding a beneficial path is shown in Algorithm 4. Starting from each worker w not matched currently in $\mathcal{M}$, we attempt to find a beneficial path (lines 4–8 in Algorithm 3). 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 4). 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 4).

We run Algorithm 3 on the same toy example given in Fig. 1. We first obtain the stable matching given in Fig. 7a. Then, we look for a beneficial path in this matching. The process in Algorithm 4 finds the beneficial path $4 \rightarrow e \rightarrow 5 \rightarrow b$ (of length 3). Executing the lines 10–12 in Algorithm 3 will result in the optimal solution (with *unhappiness index* of 1 caused by (5,e)) shown in Fig. 7c. Since this matching is maximum, Algorithm 3 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. 7, 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 *unhappiness index* of 4 and hence is not an optimal solution. In our implementation, we visit the neighbors greedily in their preference orders to find a beneficial path with the expectation that it will generate smaller *unhappiness index*.

As for the complexity of Algorithm 3, the Gale–Shapley algorithm takes $\mathcal{O}(N^2)$, where $N = \max\{m, n\}$. There can be at most $\mathcal{O}(N)$ cardinality difference between a stable matching and a maximum matching in a bipartite graph. Since finding a beneficial path, as well as updating

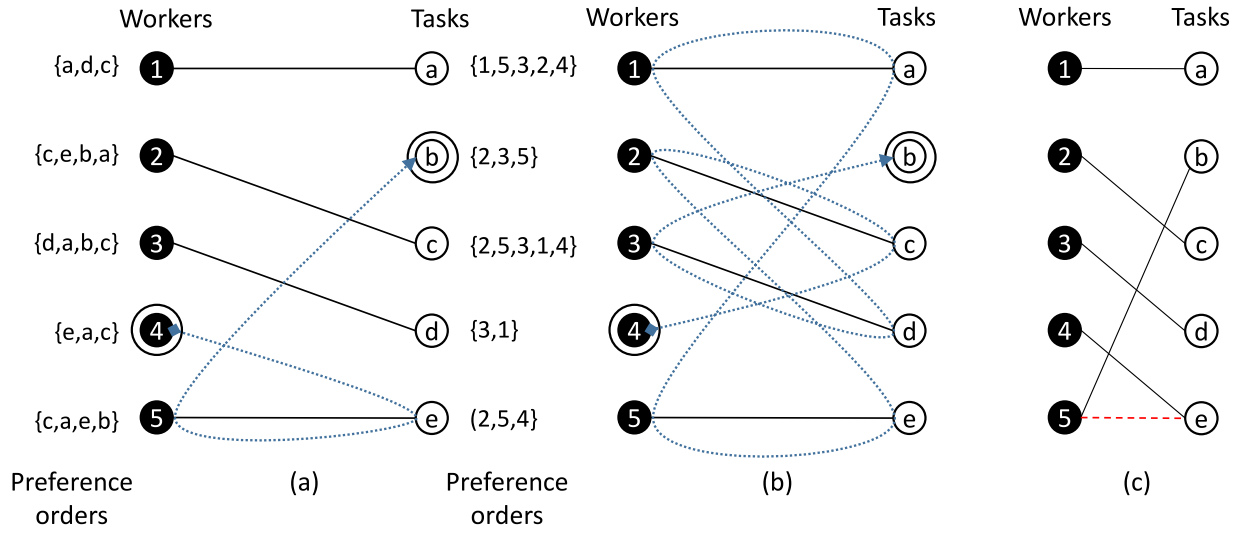

Fig. 7. Two example beneficial paths (shown with dotted lines) in the initial stable matching (shown with solid lines) generated in the first line of Algorithm 3 when it is run on the example illustrated in Fig. 1. (a) Beneficial path ($4 \rightarrow e \rightarrow 5 \rightarrow b$) found by Algorithms 3 and 4; (b) An alternative beneficial path ($4 \rightarrow c \rightarrow 2 \rightarrow e \rightarrow 5 \rightarrow a \rightarrow 1 \rightarrow d \rightarrow 3 \rightarrow b$) that could be found if the search was done without considering preference orders; (c) Matching obtained by Algorithm 3 using the beneficial path shown in (a). The only remaining unhappy pair is shown with a dashed line.

Algorithm 3: Stable to maximum convergence ($\mathcal{W}$, $\mathcal{T}$).

Input: $\mathcal{W}$: The set of workers
$\mathcal{T}$: The set of tasks
$\mathcal{M} \leftarrow$ Find a stable matching via Gale–Shapley algorithm between $\mathcal{W}$ and $\mathcal{T}$.
while *true* **do**
  set all $t \in \mathcal{T}$ as unvisited
  foreach *unmatched* $w \in \mathcal{W}$ **do**
    $p = \{w\}$
    $p \leftarrow$ *FindBeneficialPath(p)*
    if *p.isBeneficialPath* **then**
      break
  if *a beneficial path* $p = \{p_1, p_2, .., p_{2j+2}\}$ *is found* **then**
    for $i \leftarrow 1$ *to* $j+1$ **do**
      $\mathcal{M}(p_{2i-1}) \leftarrow p_{2i}$
      $\mathcal{M}(p_{2i}) \leftarrow p_{2i-1}$
  else
    break
return $\mathcal{M}$

Algorithm 4: FindBeneficialPath(p).

Input: p: Current path
$w \leftarrow p.last()$ ▷ last node on current path
foreach $t \in P_w$ **do**
  if $\mathcal{M}(t) = \emptyset$ **then**
    $p \leftarrow p \cup \{t\}$
    p.isBeneficialPath $\leftarrow$ *true*
    return p
foreach $t \in P_w$ *in the preference order* **do**
  if t *is unvisited* **then**
    set t as visited
    $p' \leftarrow$ *FindBeneficialPath*$(p \cup \{t, \mathcal{M}(t)\})$
    if p'*.FindBeneficialPath* **then**
      return p'

the matching accordingly, has $\mathcal{O}(N^2)$ complexity (as an edge is visited at most once), the total running time of Algorithm 3 becomes $\mathcal{O}(N^3)$.

5. Simulation results

In this section, we evaluate the performance of the proposed algorithms using a real world dataset.

5.1. Settings

In order to have a realistic set of user locations, we have used a taxi trip dataset [49] in a city (i.e., New York City (NYC)) similar to previous work [50–52]. 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. Fig. 8 shows a distribution of different number of workers and tasks on the NYC map.

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. 3, 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 $\mathcal{W}/\mathcal{T}$ values. 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 3.

5.2. Results

We first look at the effectiveness of the proposed approaches by comparing them with ILP results in terms of unhappiness index. Throughout

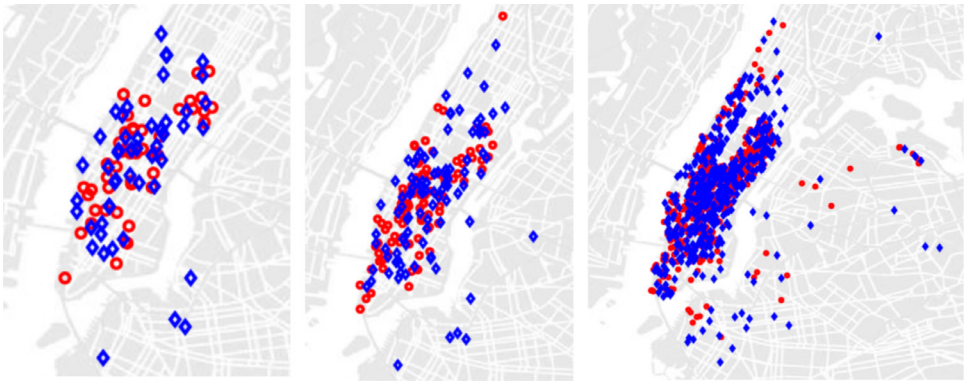

Fig. 8. The distribution of workers (circles) and tasks (diamonds) on the NYC map: (i) 50 (ii) 100 and (iii) 1000 workers/tasks.

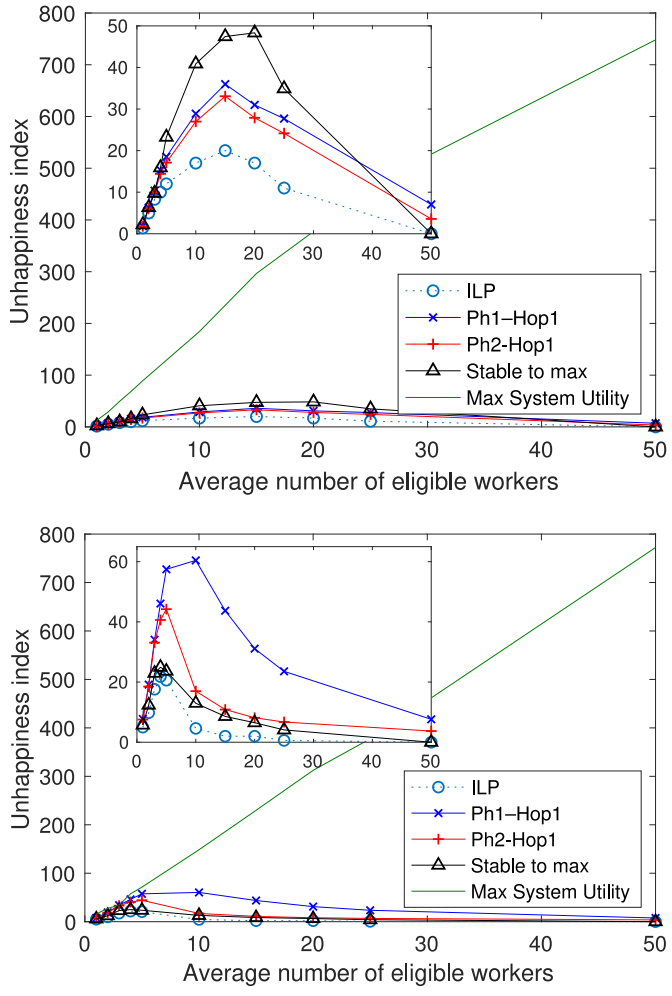

Fig. 9. Performance comparison of the heuristic algorithms with optimal results (ILP) and maximum system utility matching in terms of unhappiness index (UI) in local (upper) and random (lower) settings, respectively. Both heuristics achieve UI values that are very close to optimal UI and significantly smaller than that of the maximum system utility matching. It is also noteworthy that the best-performing heuristic is different in local and random settings.

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. 9 shows the performance comparison of *Ph***1**-*Hop***1**, *Ph***2**-*Hop***1**, *Stable to Max* and *Max System Utility*[4] algorithms with ILP results in 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 random setting. *Stable to Max* algorithm also performs differently. It performs better (i.e., fewer unhappy pairs) than other algorithms in random setting, while it results in more unhappiness index in local setting. With larger $|\mathcal{E}|$, it also performs better in local setting and always reaches complete perfect user satisfaction and stability with unhappiness index zero. 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. 10, 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 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 random setting. Besides, a sharper decrease is observed after the peak in random setting compared to local setting. The results of 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 benefit in random setting compared to local setting. However, there is not much benefit in running Phase 2 (or Phase 3) before the peak in neither settings. 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 (as it is shown in Fig. 12). Another point is that the difference between the performance of same phases with different number of hops is more profound in random setting than it is in local setting. In fact, as it is shown in Fig. 11, running the algorithm with higher number of hops reduces the unhappiness index by 1.12 per hop in random setting and by only 0.52 per hop in local setting, on average. Nonetheless, increasing the number of hops does not seem to be very beneficial after a certain point (around 5–10 hops) in either setting.

In Fig. 12, we compare the running time of the proposed algorithms in local setting (since the results are almost identical in 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 ap-

[4] It refers to the solution found by Hungarian algorithm [47] without taking into account the preferences of the users.

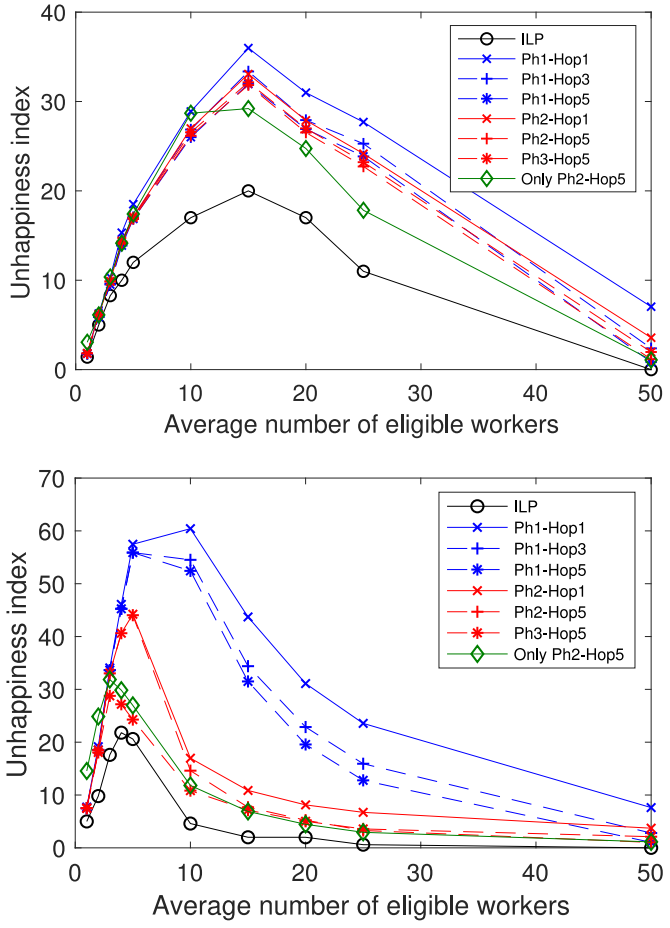

Fig. 10. The impact of number of hops and different phases on the performance of the Maximum to Stable Reduction algorithm in local (upper) and random (lower) settings, respectively. In both settings, we observe an improvement in the performance with increasing number of hops and additional phases.

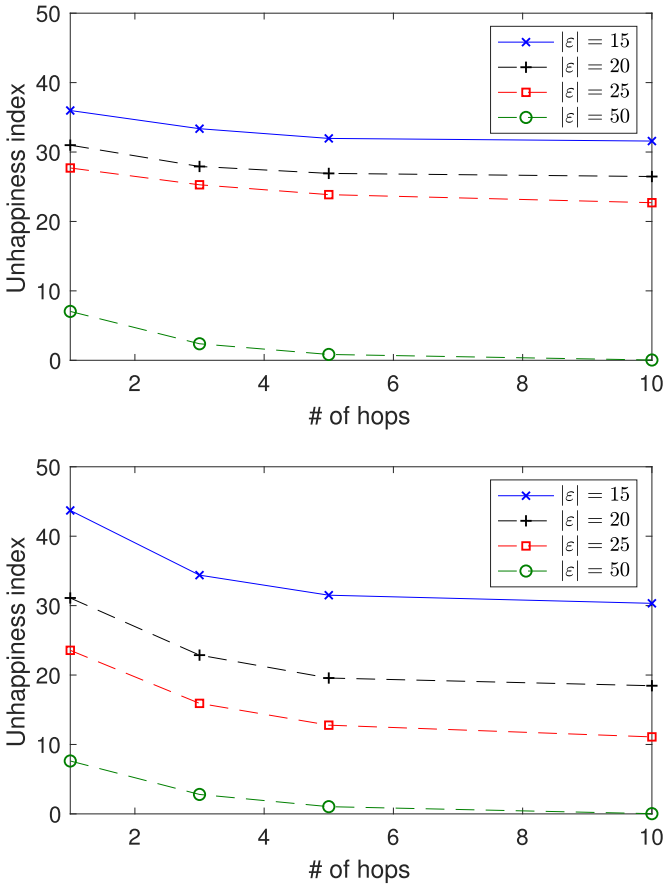

Fig. 11. The change in the unhappiness index (in Ph**1**-Hop#) with different number of hops in local (upper) and random (lower) setting, respectively, for different eligible worker/task sizes ($|\mathcal{E}|$). Increasing the number of hops notably improves the performance of the algorithm, and the improvement is usually more profound in random setting.

plications 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 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. 9, when the average number of eligible workers is 10 in 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. 13, 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.[5] This is also consistent with the results

[5] 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.

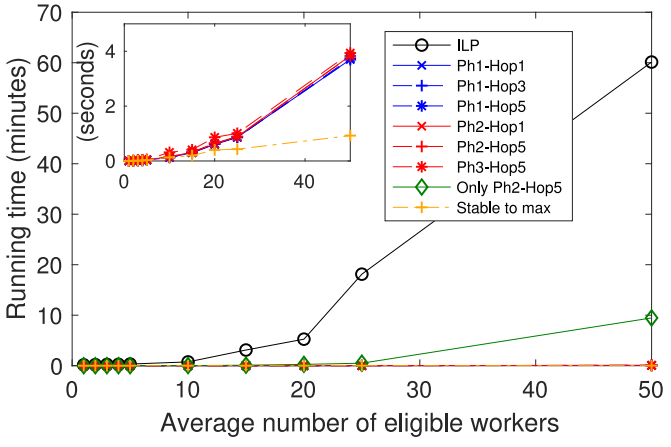

Fig. 12. Comparison of running times of proposed algorithms. Regardless of the average number of eligible workers, ILP has the worst running time, and Stable to Max algorithm has the best running time (which are in accordance with the theoretical time complexities of the algorithms).

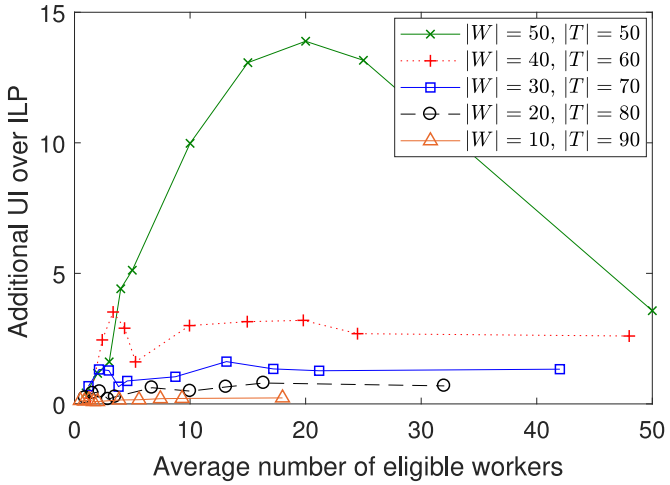

Fig. 13. The difference in the unhappiness index between ILP and *Ph***2**-*Hop***1** for different number of workers/tasks ratios. Note that when $|\mathcal{W}| \approx |\mathcal{T}|$, the average loss in system utility with stable matchings is larger as shown in Fig. 3, implying a harder scenario for trade-off between system utility and user satisfaction.

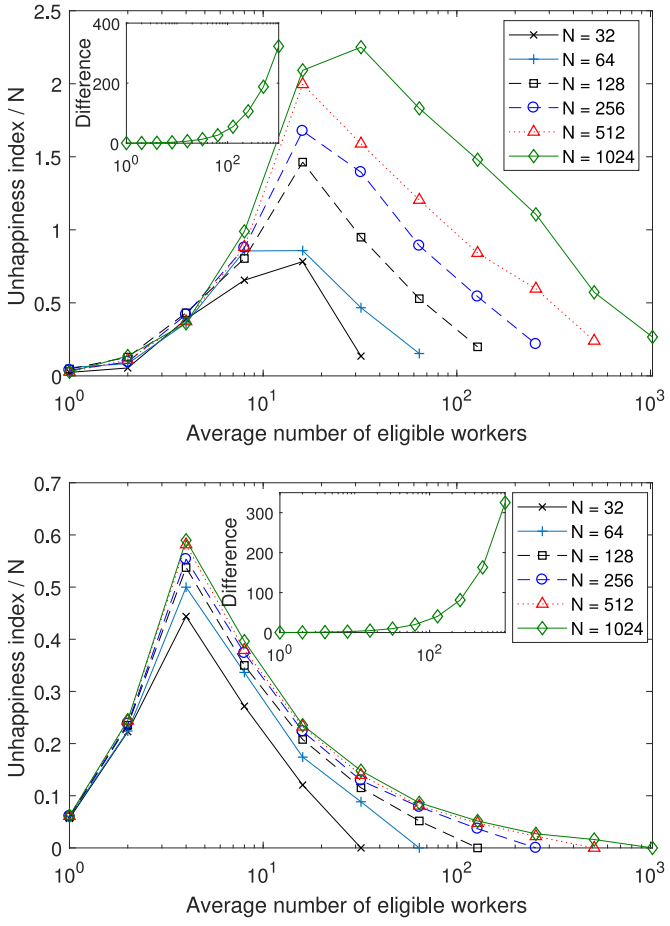

Fig. 14. The ratio of unhappiness index (*UI*) to $N = |\mathcal{W}| = |\mathcal{T}|$ in *Ph***1**-*Hop***1** and *Stable to Max* algorithms, respectively, for different number of workers and tasks (in local and random setting, respectively). The inner graphs show the difference of *UI*/*N* in Max System Utility matching from the compared algorithms when $N = 1024$.

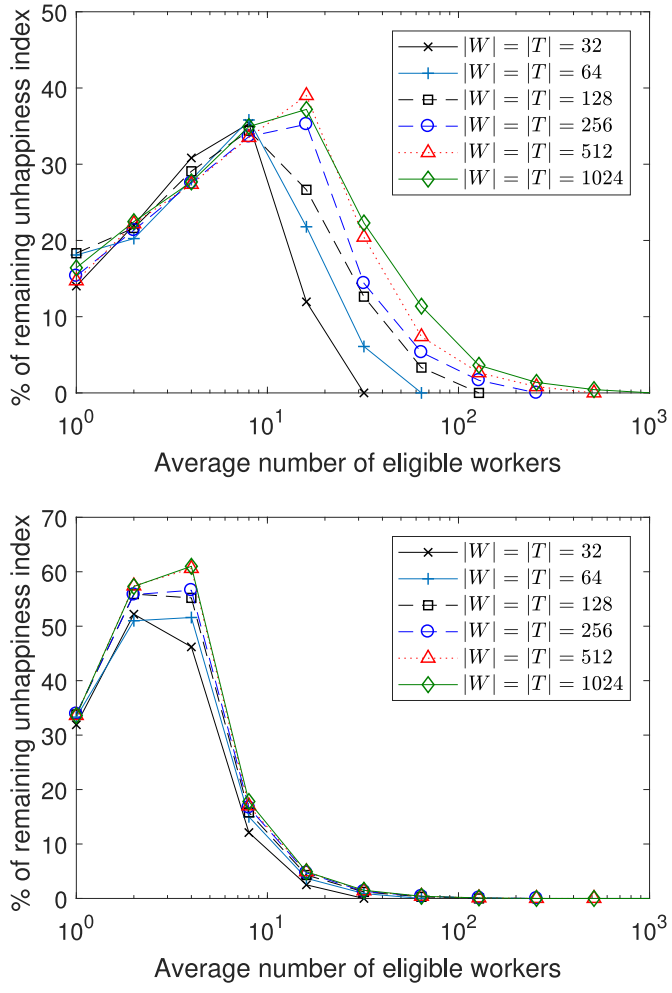

Fig. 15. The percentage of the unhappiness index in the *Ph***1**-*Hop***1** and *Stable to Max* algorithms, respectively, to the unhappiness index in the maximum system utility based assignment for different number of workers and tasks (in local and random setting, respectively).

in Fig. 3, 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. 9) and *Stable to Maximum* algorithm in random case. Fig. 14 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. 15 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. 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.

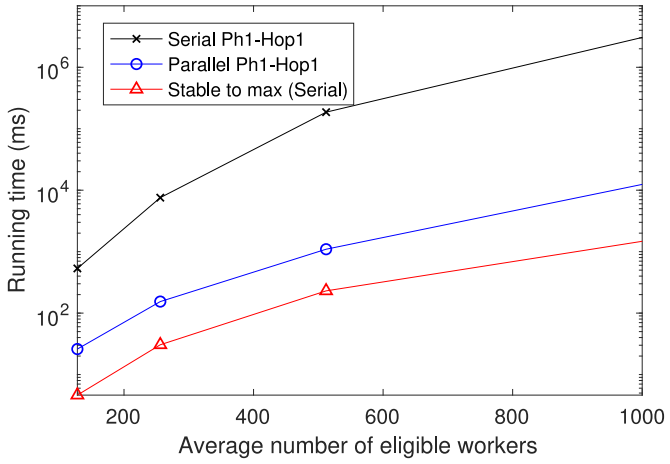

Fig. 16. The running time comparison of serial and parallel *Ph1-Hop1* algorithms and the *Stable to Max* algorithm. The running time of the parallel Ph1-Hop1 is a few orders of magnitude shorter than its serial counterpart.

In order to show the impact of parallelizing the first phase of *Maximum to Stable* reduction algorithm (which is expected to reduce the worst case complexity from $\mathcal{O}(N^5)$ to $\mathcal{O}(N^3)$), we obtain results with different number of workers and tasks in a GPU server with NVIDIA Tesla V100 PCIe 32GB. Fig. 16 shows the comparison of running times of *Ph1-Hop1* algorithm in both serial and parallel as well as the running time of *Stable to Maximum* algorithm (when all workers are eligible for all tasks and in random setting as it takes the longest). The results show that we can obtain two orders of magnitude saving with parallelization. Parallelized *Ph1-Hop1* algorithm scales similar to *Stable to Max* algorithm (slightly higher due to the larger constant factor), however *Maximum to Stable* algorithms in general perform better in local setting.

6. Conclusion

In this paper, we study the problem of maximizing the system utility of task assignment between workers and tasks in MCS while considering the satisfaction and preferences of workers and task requesters as much as possible. We propose two different heuristic based algorithms. In the former, we first obtain the maximum system utility matching and try to reduce the unhappy pairs iteratively through a process we call happify until it cannot do more. In the latter, we first obtain a stable matching and converge the graph to maximum system utility matching by finding beneficial paths and reassigning the workers and tasks on this path accordingly. The results show that the proposed algorithms run very fast compared to ILP solution and can achieve close to optimal results. Moreover, while *Maximum to Stable* reduction algorithm performs better in local setting, the *Stable to Maximum* convergence algorithm performs better in random setting, complementing each other. Note that the findings in this paper can be applied to any matching problem in which the overall system utility has a higher priority than the happiness of the users but both are targeted. In our future work, we will study the many-to-one assignment scenario, in which multiple workers could be assigned to a single task while staying in the budget constraints of the task. Note that this will require redefinition of stability and the happy pairs; thus, it will require new solutions.

Declaration of Competing Interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Fatih Yucel: Conceptualization, Methodology, Software, Validation, Investigation, Data curation, Writing - original draft, Writing - review & editing, Visualization. **Eyuphan Bulut:** Conceptualization, Methodology, Investigation, Writing - original draft, Writing - review & editing, Supervision, Funding acquisition.

Acknowledgment

This material is based upon work supported by the U.S. National Science Foundation (NSF) under Grant CNS1647217.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.comnet.2020.107156.

References

[1] F. Khan, A.U. Rehman, J. Zheng, M.A. Jan, M. Alam, Mobile crowdsensing: a survey on privacy-preservation, task management, assignment models, and incentives mechanisms, Future Gener. Comput. Syst. 100 (2019) 456–472.
[2] S. Hu, L. Su, H. Liu, H. Wang, T.F. Abdelzaher, Smartroad: smartphone-based crowd sensing for traffic regulator detection and identification, ACM Trans. Sensor Netw. (TOSN) 11 (4) (2015) 55.
[3] Y. Cheng, X. Li, Z. Li, S. Jiang, Y. Li, J. Jia, X. Jiang, Aircloud: a cloud-based air-quality monitoring system for everyone, in: Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems, ACM, 2014, pp. 251–265.
[4] M. Xiao, J. Wu, L. Huang, Y. Wang, C. Liu, Multi-task assignment for crowdsensing in mobile social networks, in: IEEE INFOCOM, IEEE, 2015, pp. 2227–2235.
[5] L. Wang, D. Yang, X. Han, T. Wang, D. Zhang, X. Ma, Location privacy-preserving task allocation for mobile crowdsensing with differential geo-obfuscation, in: Proceedings of the 26th International Conference on World Wide Web, International World Wide Web Conferences Steering Committee, 2017, pp. 627–636.
[6] J. Li, Z. Cai, J. Wang, M. Han, Y. Li, Truthful incentive mechanisms for geographical position conflicting mobile crowdsensing systems, IEEE Trans. Comput. Soc. Syst. 5 (2) (2018) 324–334.
[7] X. Wang, R. Jia, X. Tian, X. Gan, Dynamic task assignment in crowdsensing with location awareness and location diversity, in: Proc. of IEEE INFOCOM, 2018, pp. 2420–2428.
[8] S. He, D. Shin, J. Zhang, J. Chen, Near-optimal allocation algorithms for location-dependent tasks in crowdsensing, IEEE Trans. Veh. Technol. 66 (4) (2017) 3392–3405.
[9] Y. Liu, B. Guo, Y. Wang, W. Wu, Z. Yu, D. Zhang, Taskme: multi-task allocation in mobile crowd sensing, in: Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing, ACM, 2016, pp. 403–414.
[10] J. Wang, Y. Wang, D. Zhang, F. Wang, H. Xiong, C. Chen, Q. Lv, Z. Qiu, Multi-task allocation in mobile crowd sensing with individual task quality assurance, IEEE Trans. Mob. Comput. 17 (9) (2018) 2101–2113.
[11] M. Abououf, R. Mizouni, S. Singh, H. Otrok, A. Ouali, Multi-worker multi-task selection framework in mobile crowd sourcing, J. Netw. Comput. Appl. 130 (2019) 52–62.
[12] T. Liu, Y. Zhu, L. Huang, Tgba: a two-phase group buying based auction mechanism for recruiting workers in mobile crowd sensing, Comput. Netw. 149 (2019) 56–75.
[13] J. Nie, J. Luo, Z. Xiong, D. Niyato, P. Wang, A stackelberg game approach toward socially-aware incentive mechanisms for mobile crowdsensing, IEEE Trans. Wirel. Commun. 18 (1) (2018) 724–738.
[14] W. Gong, B. Zhang, C. Li, Location-based online task assignment and path planning for mobile crowdsensing, IEEE Trans. Veh. Technol. 68 (2) (2018) 1772–1783.
[15] J. Wang, J. Tang, G. Xue, D. Yang, Towards energy-efficient task scheduling on smartphones in mobile crowd sensing systems, Comput. Netw. 115 (2017) 100–109.
[16] T. Hu, M. Xiao, C. Hu, G. Gao, B. Wang, A QoS-sensitive task assignment algorithm for mobile crowdsensing, Pervasive Mob. Comput. 41 (2017) 333–342.
[17] W. Li, L. Wang, Y. Gu, R. Li, M. Song, Z. Han, Stable multiple activity matching based content sharing for mobile crowd sensing, in: IEEE International Conference on Comm. (ICC), 2018, pp. 1–6.
[18] Y. Chen, X. Yin, Stable job assignment for crowdsourcing, in: GLOBECOM 2017-2017 IEEE Global Communications Conference, IEEE, 2017, pp. 1–6.
[19] X. Yin, Y. Chen, B. Li, Task assignment with guaranteed quality for crowdsourcing platforms, in: Quality of Service (IWQoS), 2017 IEEE/ACM 25th International Symposium on, IEEE, 2017, pp. 1–10.
[20] M. Abououf, S. Singh, H. Otrok, R. Mizouni, A. Ouali, Gale–Shapley matching game selection—A framework for user satisfaction, IEEE Access 7 (2019) 3694–3703.
[21] D. Gale, L. Shapley, College admissions and stability of marriage, Am. Math. Mon. 69 (1962) 9–15.
[22] F. Yucel, E. Bulut, Joint optimization of system and user oriented task assignment in mobile crowdsensing, 2019, (in Global Telecommunications Conference (Globecom), 2019. [Online] Available: http://www.people.vcu.edu/~ebulut/Globecom2019-matching.pdf).

[23] K. Tuite, N. Snavely, D. Hsiao, N. Tabing, Z. Popovic, PhotoCity: training experts at large-scale image acquisition through a competitive game, in: Proceedings of the International Conference on Human Factors in Computing Systems, CHI 2011, Vancouver, BC, Canada, May 7–12, 2011, 2011, pp. 1383–1392.
[24] P. Mohan, V.N. Padmanabhan, R. Ramjee, Nericell: rich monitoring of road and traffic conditions using mobile smartphones, in: Proceedings of the 6th International Conference on Embedded Networked Sensor Systems, SenSys 2008, Raleigh, NC, USA, November 5–7, 2008, 2008, pp. 323–336.
[25] T. Luo, S.S. Kanhere, J. Huang, S.K. Das, F. Wu, Sustainable incentives for mobile crowdsensing: auctions, lotteries, and trust and reputation systems, IEEE Commun. Mag. 55 (3) (2017) 68–74.
[26] Z. Duan, W. Li, Z. Cai, Distributed auctions for task assignment and scheduling in mobile crowdsensing systems, in: 37th IEEE International Conference on Distributed Computing Systems, ICDCS 2017, Atlanta, GA, USA, June 5–8, 2017, 2017, pp. 635–644.
[27] Y. Wen, J. Shi, Q. Zhang, X. Tian, Z. Huang, H. Yu, Y. Cheng, X. Shen, Quality–driven auction-based incentive mechanism for mobile crowd sensing, IEEE Trans. Veh. Technol. 64 (9) (2015) 4203–4214.
[28] S. Chen, M. Liu, X. Chen, A truthful double auction for two-sided heterogeneous mobile crowdsensing markets, Comput. Commun. 81 (2016) 31–42.
[29] Z. Zheng, F. Wu, X. Gao, H. Zhu, S. Tang, G. Chen, A budget feasible incentive mechanism for weighted coverage maximization in mobile crowdsensing, IEEE Trans. Mob. Comput. 16 (9) (2016) 2392–2407.
[30] M. Xiao, J. Wu, S. Zhang, J. Yu, Secret-sharing-based secure user recruitment protocol for mobile crowdsensing, in: IEEE INFOCOM 2017-IEEE Conference on Computer Communications, 2017, pp. 1–9.
[31] L. Wang, D. Yang, X. Han, T. Wang, D. Zhang, X. Ma, Location privacy-preserving task allocation for mobile crowdsensing with differential geo-obfuscation, in: Proceedings of the 26th International Conference on World Wide Web, International World Wide Web Conferences Steering Committee, 2017, pp. 627–636.
[32] Z. Wang, J. Hu, R. Lv, J. Wei, Q. Wang, D. Yang, H. Qi, Personalized privacy-preserving task allocation for mobile crowdsensing, IEEE Trans. Mob. Comput. 18 (6) (2018) 1330–1341.
[33] H. Cai, Y. Zhu, Z. Feng, H. Zhu, J. Yu, J. Cao, Truthful incentive mechanisms for mobile crowd sensing with dynamic smartphones, Comput. Netw. 141 (2018) 1–16.
[34] National resident matching program, 2018, http://www.nrmp.org.
[35] Z. Zhou, C. Gao, C. Xu, T. Chen, D. Zhang, S. Mumtaz, Energy-efficient stable matching for resource allocation in energy harvesting-based device-to-device communications, IEEE Access 5 (2017) 15184–15196.
[36] T. Wang, F. Liu, J. Guo, H. Xu, Dynamic SDN controller assignment in data center networks: stable matching with transfers, in: INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, 2016, pp. 1–9.
[37] R. Zhang, X. Cheng, L. Yang, Flexible energy management protocol for cooperative EV-to-EV charging, IEEE Trans. Intell. Transp. Syst. 20 (1) (2019) 172–184, doi:10.1109/TITS.2018.2807184.
[38] D. Gale, M. Sotomayor, Some remarks on the stable matching problem, Discrete Appl. Math. 11 (3) (1985) 223–232.
[39] K. Iwama, S. Miyazaki, H. Yanagisawa, Approximation algorithms for the sex-equal stable marriage problem, ACM Trans. Algorithms 7 (1) (2010) 2:1–2:17.
[40] R.W. Irving, P. Leather, D. Gusfield, An efficient algorithm for the “optimal” stable marriage, J. ACM 34 (3) (1987) 532–543.
[41] D. Gusfield, Three fast algorithms for four problems in stable marriage, SIAM J. Comput. 16 (1) (1987) 111–128.
[42] R.W. Irving, P. Leather, The Complexity of Counting Stable Marriages, SIAM J. Comput. 15 (3) (1986) 655–667.
[43] P. Biró, D. Manlove, S. Mittal, Size versus stability in the marriage problem, Theor. Comput. Sci. 411 (16-18) (2010) 1828–1841, doi:10.1016/j.tcs.2010.02.003.
[44] P. Floréen, P. Kaski, V. Polishchuk, J. Suomela, Almost stable matchings by truncating the Gale-Shapley algorithm, Algorithmica 58 (1) (2010) 102–118, doi:10.1007/s00453-009-9353-9.
[45] R. Ostrovsky, W. Rosenbaum, Fast distributed almost stable matchings, in: Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21–23, 2015, 2015, pp. 101–108, doi:10.1145/2767386.2767424.
[46] C. Fiandrino, B. Kantarci, F. Anjomshoa, D. Kliazovich, P. Bouvry, J. Matthews, Sociability-driven user recruitment in mobile crowdsensing internet of things platforms, in: 2016 IEEE Global Communications Conference (GLOBECOM), IEEE, 2016, pp. 1–6.
[47] H.W. Kuhn, The hungarian method for the assignment problem, in: 50 Years of Integer Programming 1958–2008 - From the Early Years to the State-of-the-Art, 2010, pp. 29–47, doi:10.1007/978-3-540-68279-0_2.
[48] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, third ed., MIT Press, 2009.
[49] Taxi and limousine commission (tlc) trip record data., 2019, (NYC Taxi Limousine Commission), https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.
[50] E. Wang, Y. Yang, J. Wu, W. Liu, X. Wang, An efficient prediction-based user recruitment for mobile crowdsensing, IEEE Trans. Mob. Comput. 17 (1) (2017) 16–28.
[51] Y. Yang, W. Liu, E. Wang, J. Wu, A prediction-based user selection framework for heterogeneous mobile crowdsensing, IEEE Trans. Mob. Comput. 18 (11) (2019) 2460–2473, doi:10.1109/TMC.2018.2879098.
[52] G. Gao, M. Xiao, J. Wu, L. Huang, C. Hu, Truthful incentive mechanism for nondeterministic crowdsensing with vehicles, IEEE Trans. Mob. Comput. 17 (12) (2018) 2982–2997.

Fatih Yucel (M’17) received B.S. degree in Gazi University in Turkey in 2017. He is now doing Ph.D. in the Computer Science Department of Virginia Commonwealth University under the supervision of Dr. Eyuphan Bulut. He joined MoWiNG lab in Fall 2017. He is working on the development of efficient algorithms for Internet of Things (IoT) and mobile crowdsensing. He is a student member of IEEE.

Eyuphan Bulut (M’08) received the Ph.D. degree in the Computer Science department of Rensselaer Polytechnic Institute (RPI), Troy, NY, in 2011. He then worked as a senior engineer in Mobile Internet Technology Group (MITG) group of Cisco Systems in Richardson, TX for 4.5 years. He is now an Assistant Professor with the Department of Computer Science, Virginia Commonwealth University (VCU), Richmond, VA. His research interests include mobile and wireless computing, network security and privacy, mobile social networks and crowdsensing. Dr. Bulut is an Associate Editor in IEEE Access. He has been also serving in the organizing committee of the LCN and in the technical program committee of several conferences. He is a member of IEEE and ACM.