Joint Optimization of System and User oriented Task Assignment in Mobile Crowdsensing

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

***Abstract*—One of the fundamental challenges in mobile crowdsensing (MCS) systems is efficient task assignment. Existing solutions consider the problem from system's point of view and try to maximize the system utility or minimize the cost of sensing. However, such a task assignment process does not consider the user (i.e., workers and task requesters) preferences and can yield unhappy users with their assignments, impairing the participation of users in the future. To incentivize the user participation, stable matching based solutions can be utilized to result in satisfactory assignments that will make the users happy based on their preferences. However, this may adversely affect the system utility especially when the set of eligible number of workers for each task is limited. To address this issue, we study the task assignment problem in MCS systems that maximizes the main system utility (i.e., number of workers and tasks assigned) as a system oriented goal while generating as happy users as possible with their assignment. As the problem is NP-complete, we first solve the problem optimally using Integer Linear Programming (ILP) and then we propose a heuristic based polynomial solution that runs very fast. Through simulations, we show that the proposed approach achieves the maximum possible system utility while generating small and close to optimal user unhappiness as in ILP results.**

***Index Terms*—Mobile crowdsensing, task assignment, stable matching.**

I. Introduction

Mobile crowdsensing (MCS) offers a promising and cost effective solution for complex computation and sensing tasks by exploiting the power of crowd [1]. With far-reaching proliferation of mobile devices that are equipped with various sensors (e.g., GPS, microphone, camera) and pervasive network accessibility, users wearing these devices are recruited to complete these tasks in parallel, hence achieving a shorter completion time. Examples of MCS applications include traffic monitoring [2], transit station labeling [3] and air quality detection [4].

In a typical MCS, there is 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 select the tasks based on their preferences or let the task requesters hire the workers as they want. 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.

[1]They are also called mobile users in some studies but we utilize the term ”user” to refer to both task requesters and workers.

MCS applications can also be classified into two based on the way the data is collected, namely, opportunistic or participatory MCS. In the former, the workers are not involved actively and sensing is performed opportunistically without changing their mobility. On the contrary, in the latter, workers are actively involved and a centralized platform recruits them and dispatches the sensing tasks. Depending on the MCS application and its requirements, one may have advantages over the other. However, in both types, the main challenging problem is the assignment of tasks to users under some optimization goal. In the literature, many studies have looked at this problem from different perspectives and proposed various solutions considering several parameters such as traveling distances and sociability [5], privacy [6] and truthfulness of users [7], location-awareness [8] and QoS sensitivity [9].

Despite the variety of the literature on the task assignment problem in MCS systems, the goal is mostly defined from overall system's point of view without considering the user preferences. Hence, the resulting task assignment could be dissatisfying and unappealing for both task requesters and workers. However, in practice, users would not like to sacrifice their individual convenience for the overall utility of the system. To avoid this problem, recently, stable matching has been used in the task assignment process [10], [11] to make the users happy with their assignments. However, such solutions can reduce the overall system utility by leaving some tasks and workers unassigned. Such cases usually occur when the eligible set of workers to do a task are not exhaustive (i.e., not covering the entire list of workers as they may not have required skills/trust/quality). For example, consider the example in Fig.1 with two workers and tasks, where worker 1 prefers task 1 over task 2, while worker 2 can only perform task 1 as he cannot travel longer distances. In a stable matching considering user preferences, worker 1 is assigned to task 1 and worker 2 and task 2 are left unmatched as worker 2 was not eligible to perform task 2. On the contrary, in a system oriented matching that aims to maximize the system utility by assigning as many workers and tasks, worker 1 could be

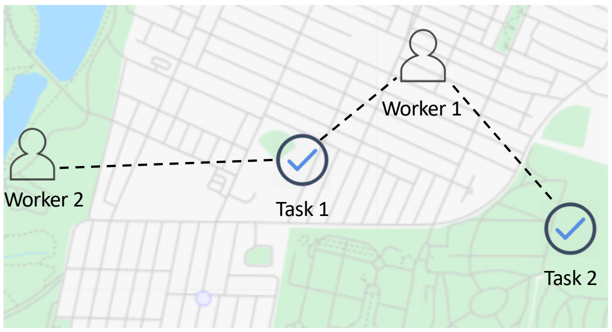


Fig. 1. Eligible workers for each task is shown by dashed edges. A system oriented matching that aims to maximize the assigned workers and tasks will assign worker 1 to task 2 and worker 2 to task 1. On the contrary, a user oriented task assignment considering user preferences (e.g., closer worker/task if cost of assignment is location dependent) will assign worker 1 to task 1 and leave the worker 2 and task 2 unassigned.

assigned to task 2 and worker 2 could be assigned to task 1.

In this paper, our goal is to jointly optimize the task assignment process by considering both the system level goals (e.g., maximizing the number of assigned workers/tasks) within given assignment constraints (e.g., budget of workers, rewards of tasks) and the user preferences. However, achieving both of them may not be possible at the same time. Thus, we aim to reach the system goals while generating as few unhappy users as possible. As this problem is NP-complete (since it can be reduced to max cardinality with min blocking pairs problem [12]), we first define an Integer Linear Programming (ILP) based solution to find the optimal result. Then, we provide a heuristic based cost efficient solution and show that it provides close to optimal results through simulations.

The rest of the paper is organized as follows. In Section II, we provide the system model and our assumptions. In Section III, we discuss the details of the ILP solution and heuristic approach. In Section IV, we present an evaluation of the proposed approach through simulations. Finally, we end up with conclusion in Section V.

II. System Model

A. 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. Let c_{ij} denote the cost of assigning worker w_i to task t_j. Let also r_j denote the reward of completing the task t_j. We assume that each worker will not perform a task if its cost is higher than the reward of the task. Then, the set of tasks that are eligible for worker w_i to perform are:

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

Moreover, in order to increase the profit from the task, the worker prefers the ones that have 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 $r_j - c_{ij} > r_{j'} - c_{ij'}$.

Similarly, from task requesters' side, they cannot hire a worker if the reward that the task requester can provide (which could be considered as the budget of the requester as well) is more than the cost of hiring a worker. The set of eligible workers that can perform the task t_j is then defined as:

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

Moreover, even though the task requester pays the same reward to any worker who completes the task, 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 [5], [8], the requester of the task 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. Otherwise it could 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 work the worker can provide [9]. We use $w_i \succ_{t_j} w_{i'}$ notation to express that t_j prefers w_i to $w_{i'}$.

B. System vs. User oriented Task Assignment

Once the set of eligible workers for each task and eligible tasks for each worker are determined, the platform can then assign the tasks to workers with some optimization goal. From the system's perspective, this goal could be to match as many workers with a task in order to increase the activity in the platform (and potentially get more commission from workers and requesters etc.) and attract more participants (more complex utility based models can be considered similarly). Such a matching can indeed be obtained by forming a bipartite matching graph between workers and tasks and solving it through the well-known Hungarian algorithm [13]. However, from user's perspective, this may result in unhappy workers and task requesters due to their preferences. Thus, a stable matching (also known as stable marriage [14]) based solutions could be utilized to find a task assignment that will make both the workers and task requesters happy with the assignment. Here, we define that a worker is happy with the assignment, if there is no other preferred task that would also prefer this worker compared to the worker its assigned. Similarly, a task requester or the task is considered happy with the assignment, if there is no other preferred worker that would also prefer this task compared to its assigned task. Such a stable task assignment can be found by using the deferred acceptance mechanism proposed in the Gale-Shapley algorithm [14].

Let $\mathcal{M}$ = $\{(w_{i_1}, t_{j_1}) \dots (w_{i_k}, t_{j_k})\}$, $k \leq \min\{m, n\}$ 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 an assignment $\mathcal{M}$ to be stable (i.e., no unhappy users) there should not exist a $\langle w_i, t_j \rangle$ pair such that $t_j \in \mathcal{E}(w_i)$, $w_i \in \mathcal{E}(t_j)$, and

- $t_j \succ_{w_i} \mathcal{M}(w_i)$ and $w_i \succ_{t_j} \mathcal{M}(t_j)$, or
- $t_j \succ_{w_i} \mathcal{M}(w_i)$ and $\mathcal{M}(t_j) = \emptyset$, or
- $w_i \succ_{t_j} \mathcal{M}(t_j)$ and $\mathcal{M}(w_i) = \emptyset$, or
- $\mathcal{M}(w_i) = \emptyset$ and $\mathcal{M}(t_j) = \emptyset$.

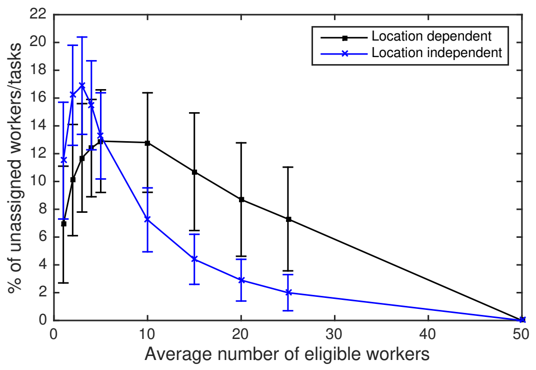


Fig. 2. Percentage of unassigned workers and tasks due to user oriented task assignment compared to the maximum possible worker task assignment in the system oriented task assignment. In the location-dependent case, eligible workers are determined based on geographical restrictions (i.e., travel distances) while in the location-independent case they are decided based on other characteristics.

In system oriented task assignment process, there will be such unhappy pairs as the user preferences are not considered and solely the maximum system utility (i.e., count of assigned pairs) is targeted. On the other hand, in user oriented stable matching process, there can be unmatched workers or tasks which may later impair the participation of them to the system. Fig. 1 shows an example of that scenario but to quantify unmatched user counts in general, we generated different scenarios. To this end, we deployed 50 workers and 50 tasks in a region of size 1 km by 1 km. We then defined eligible workers for each task and eligible tasks for each worker together with the preference orders. We have used two ways for the cost function. In the location-dependent case, we assume that each worker cannot travel more than the distance which will have the cost of travel more than the reward of the task and a worker prefers the task closer to the worker's location and vice versa. In the location-independent case, since each user may have a distinct and unique set of criteria to determine the eligibility, we randomly decide different number of eligible worker/task sets for each task/worker, respectively.

In Fig. 2, we show the percentage of decrease in the number of assigned workers/tasks in a user oriented stable task assignment compared to the assigned workers/tasks in a system oriented maximum utility assignment. For all results in this and the following sections, we take the average of 100 different runs for statistical significance. The error bars in graphs show the variance in results. As shown in Fig. 2, when the average number of eligible workers/tasks is small, the decrease in assigned workers/tasks approaches to 20% for randomly formed eligibility lists, meaning that 1 of every 5 tasks/workers is left unassigned. Although this decreases as the eligible worker/task set sizes increase, it should be noted that in practice there could only be a limited number of eligible workers for each task and vice versa.

III. Task Assignment with Joint Optimization

In this section, we first model the problem using Integer Linear Programming (ILP) to find the optimal solution for a given task and worker list with their restrictions and eligibility. Then, we propose a heuristic based cost efficient solution.

A. 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 follows:

$$\max \sum_{\forall i,j} \left(mn\mathcal{X}_{ij} - \mathcal{U}_{ij} \right) \tag{3}$$

with the constraints:

$$\begin{aligned} \sum_{\forall i} \mathcal{X}_{ij} &\leq 1 & \forall j \\ \sum_{\forall j} \mathcal{X}_{ij} &\leq 1 & \forall i \\ \mathcal{X}_{ij} &\leq e_{ij} & \forall i,j \end{aligned}$$

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 (which we also call as *unhappiness index (UI)*) can be at most mn. Incrementing the number of assigned pairs will increase the objective function value in (3) more than removing all unhappy pairs. Thus, it first tries to reach an assignment with maximum system utility (defined as the number of assigned worker and task pairs), then reduces *unhappiness index* as much as possible.

B. Heuristic based Approach

As the ILP solution is costly, we propose a two stage heuristic based algorithm that runs in polynomial time. 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* zero). 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 a bipartite graph G, a path $p = \{p_1, p_2, \ldots p_{2j+2}\}$ 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,

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

```
Algorithm 1: Heuristic Approach ($\mathcal{W}$, $\mathcal{T}$)
   Input: $\mathcal{W}$: The set of workers
          $\mathcal{T}$: The set of tasks
 1 $\mathcal{M}\leftarrow$ Find a stable matching via Gale-Shapley algorithm
   between $\mathcal{W}$ and $\mathcal{T}$.
 2 while true do
 3     set all $t \in \mathcal{T}$ as unvisited
 4     foreach unmatched $w \in \mathcal{W}$ do
 5         $p = \{w\}$
 6         $p \leftarrow$ FindBeneficialPath(p)
 7         if p.isBeneficialPath then
 8             break
 9         end
10     end
11     if a beneficial path $p = \{p_1, p_2, .., p_{2j+2}\}$ is found
        then
12         for $i \leftarrow 1$ to $j+1$ do
13             $\mathcal{M}(p_{2i-1}) \leftarrow p_{2i}$
14             $\mathcal{M}(p_{2i}) \leftarrow p_{2i-1}$
15         end
16     else
17         break
18     end
19 end
20 return $\mathcal{M}$
```

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 1. We first find a stable matching $\mathcal{M}$ between the given workers and tasks using deferred acceptance mechanism in Gale-Shapley algorithm [14]. 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 12-15. However, if we cannot find a beneficial path, it means $\mathcal{M}$ has reached the maximum possible assignment [15] and will be returned as the final matching.

The procedure of finding a beneficial path is shown in Algorithm 2. Starting from each worker w not matched currently in $\mathcal{M}$, we attempt to find a beneficial path (lines 4-10 in Algorithm 1). If w can be matched directly with an unmatched task in $\mathcal{E}(w)$, a beneficial path of length 1 is obtained immediately (lines 2-8 in Algorithm 2). Otherwise, the tasks that are currently matched in $\mathcal{M}$ are processed in their preference order. For each such task, a new potential path

```
Algorithm 2: FindBeneficialPath(p)
   Input: $p$: Current path
 1 $w \leftarrow p.last()$ ;              ▷ last node on current path
 2 foreach $t \in \mathcal{E}(w)$ do
 3     if $\mathcal{M}(t) = \emptyset$ then
 4         $p \leftarrow p \cup \{t\}$
 5         p.isBeneficialPath $\leftarrow$ true
 6         return $p$
 7     end
 8 end
 9 foreach $t \in \mathcal{E}(w)$ in the preference order do
10     if $t$ is unvisited then
11         set $t$ as visited
12         $p' \leftarrow$ FindBeneficialPath($p \cup \{t, \mathcal{M}(t)\}$)
13         if $p'$.FindBeneficialPath then
14             return $p'$
15         end
16     end
17 end
```

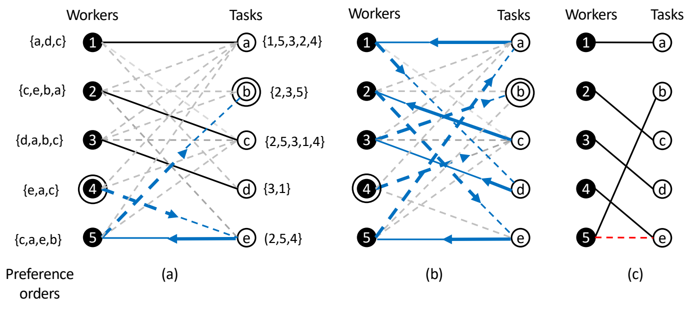
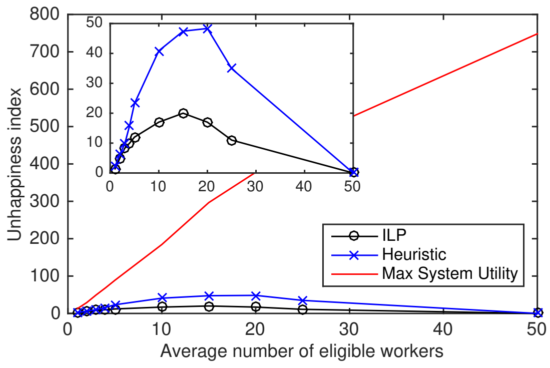


Fig. 3. Dashed lines show eligibility (in a and b) and unhappy pairs (in c), while solid lines show the actual assignment. Beneficial path examples on the current assignment, $\mathcal{M}$, between 4 and b: (a) $4 \rightarrow e \rightarrow 5 \rightarrow b$ with length 3; (b) $4 \rightarrow c \rightarrow 2 \rightarrow e \rightarrow 5 \rightarrow a \rightarrow 1 \rightarrow d \rightarrow 3 \rightarrow b$ with length 9.

is created by extending the current path with this task and its matched partner, and the same process is repeated recursively (lines 9-17 in Algorithm 2).

We run Algorithm 1 on a toy example given in Fig. 3. We first obtain the stable matching given in Fig. 3a. Then, we look for a beneficial path in this matching. The process in Algorithm 2 finds the beneficial path $4 \rightarrow e \rightarrow 5 \rightarrow b$ (of length 3). Executing the lines 12-15 in Algorithm 1 will result in the optimal solution (with *unhappiness index* 1 caused by (5,e)) shown in Fig. 3c. Since this matching is maximum, Algorithm 1 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. 3, 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* 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*.



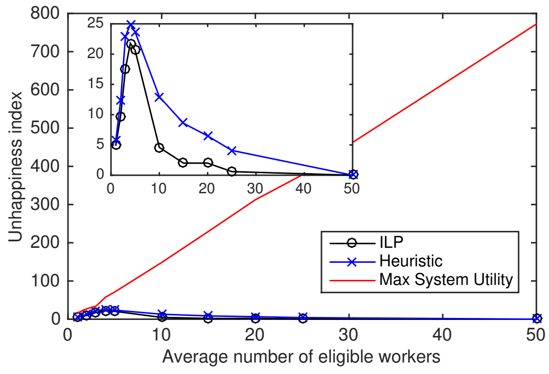


Fig. 4. Reduction in number of unhappiness index in location-dependent (upper) and location-independent (lower) settings, respectively.

As for the complexity of Algorithm 1, 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 the matching accordingly, has $\mathcal{O}(N^2)$ complexity (as we need to visit every edge once at most), the total running time of Algorithm 1 becomes $\mathcal{O}(N^3)$.

IV. Simulation Results

A. Settings

In this section, we evaluate the performance of the proposed algorithm. For main simulations, we use 50 workers and 50 tasks as the largest *unhappiness index* in maximum system utility based assignment happens when $\mathcal{W}/\mathcal{T}$=1. Thus, the reduction of it is much important and harder when the set sizes are equal. But we also provide results with different $\mathcal{W}/\mathcal{T}$ ratios and up to 1000 workers/tasks.

We study two different cases, where the eligibility is defined based on geographical restrictions (i.e., location dependent) and it is defined based on location-independent characteristics of workers. For the former, we assign location distances r_i that defines the serving region of each worker w_i and find the eligibility accordingly. In the latter, we use random eligibility and preferences, as described in Section II.

B. Results

We first look at the effectiveness of the heuristic based approach by comparing it with ILP results in terms of *unhappiness index* (as their system utility is always the same). Fig. 4a-b show the results in location-dependent and random settings, respectively. First of all, note that as expected, the *unhappiness index* in the initial maximum system utility based assignment grows linearly with increasing number of average eligible workers for the tasks. The heuristic algorithm can decrease *unhappiness index* remarkably while achieving maximum system utility and gives very close results to ILP in general. On the other hand, it behaves differently when the eligible workers are determined based on location restrictions or not. In location independent case, it can find very close results to ILP while the gap with ILP is higher in the location dependent case. Note that it is still very small compared to the *unhappiness index* in the initial assignment. Another point is with larger eligible worker counts, it always finds the assignment where everybody is happy and the maximum system utility is obtained. The maximum gap between the heuristic based algorithm and the ILP results occurs with around 10-20 eligible workers and gets smaller as the eligible worker count increases or decreases.

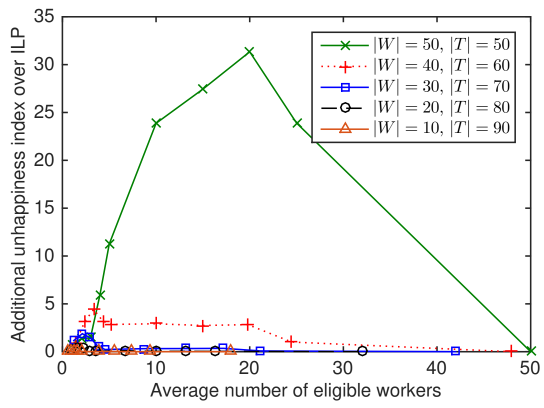


Fig. 5. Additional unhappiness index produced over ILP results with different ratios of worker counts and task counts.

TABLE I
Running time comparison.

Eligible worker count	ILP	Heuristic
10	44 s	0.24 s
20	5.25 min	0.43 s
50	60 min	1.05s

In Table I, we compare the running time of the heuristic based algorithm in location dependent setting (the results are similar in location independent case). As expected, ILP has an excessively long running time, which makes it impractical to try to find the optimal solution for applications that demand timely response. In contrary, the proposed algorithm runs very fast while providing very close results to the ILP.

In Fig. 5, we analyze the performance of the proposed algorithm when there are unequal number of workers and tasks in the system. Specifically, we calculate the difference of the *unhappiness index* between the ILP and heuristic algorithm results. We observe that the difference gets smaller as the ratio between the number of workers and tasks fades away from 1. The heuristic based algorithm can achieve better results in such cases as well. This is because when the number of workers and

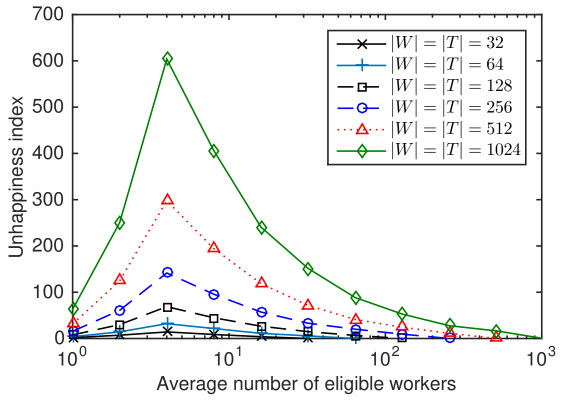


Fig. 6. The unhappiness index obtained with the heuristic based algorithm for different number of workers and tasks (in location-independent setting).

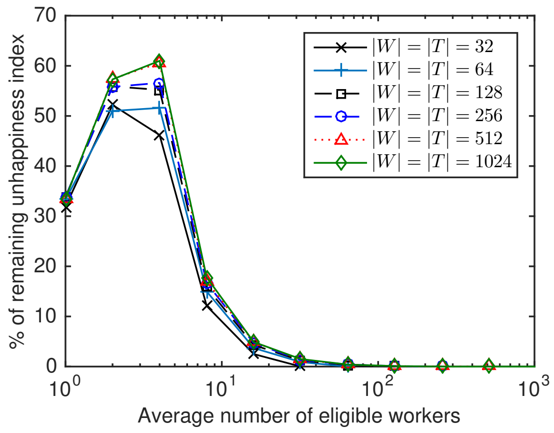


Fig. 7. The percentage of the unhappiness index in the heuristic based algorithm to the unhappiness index in the maximum system utility based assignment for different number of workers and tasks.

tasks are not equal, the *unhappiness index* is determined by the smaller set size, thus there will be a smaller *unhappiness index* in the maximum system utility based task assignment.

Finally, we look at the scalability results in location-independent case (location-dependent case provides similar outcomes). Fig. 6 shows the *unhappiness index* in the heuristic based algorithm results with different worker and task counts. The results show that the *unhappiness index* is more with more workers and tasks, as expected, but the peak is always obtained when the average number of eligible workers are around 4-5. Moreover, as shown in Fig. 7 when we calculate the ratio of this *unhappiness index* to the *unhappiness index* in the maximum system utility based task assignment, we obtain a similar percentage regardless of the number of workers and tasks. Thus, the performance scales well in percentage. It is also worth noting that as the average number of eligible workers increases, we achieve a better performance.

V. Conclusion

In this paper, we study the problem of joint optimization of system and user oriented task assignment process in mobile crowdsensing (MCS) systems. We show that it may not be possible to achieve a task assignment with maximum system utility and all happy users. To this end, we introduce a new metric called *unhappiness index* to define the level of unhappiness of users and aim a joint optimization that achieves the maximum system utility with minimum *unhappiness index*. We provide both ILP and a heuristic based fast algorithm, and show that the heuristic based algorithm can achieve close to optimal results. In our future work, we will develop other heuristic based approaches and perform extensive simulations with different system utility definitions.

Acknowledgement

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

References

[1] R. K. Ganti, F. Ye, and H. Lei, "Mobile crowdsensing: current state and future challenges," *IEEE Communications Magazine*, vol. 49, no. 11, pp. 32–39, 2011.

[2] S. Hu, L. Su, H. Liu, H. Wang, and T. F. Abdelzaher, "Smartroad: Smartphone-based crowd sensing for traffic regulator detection and identification," *ACM Transactions on Sensor Networks (TOSN)*, vol. 11, no. 4, p. 55, 2015.

[3] M. Elhamshary, M. Youssef, A. Uchiyama, H. Yamaguchi, and T. Higashino, "Transitlabel: A crowd-sensing system for automatic labeling of transit stations semantics," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2016, pp. 193–206.

[4] Y. Cheng, X. Li, Z. Li, S. Jiang, Y. Li, J. Jia, and 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.

[5] C. Fiandrino, B. Kantarci, F. Anjomshoa, D. Kliazovich, P. Bouvry, and 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.

[6] L. Wang, D. Yang, X. Han, T. Wang, D. Zhang, and 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.

[7] J. Li, Z. Cai, J. Wang, M. Han, and Y. Li, "Truthful incentive mechanisms for geographical position conflicting mobile crowdsensing systems," *IEEE Transactions on Computational Social Systems*, vol. 5, no. 2, pp. 324–334, 2018.

[8] X. Wang, R. Jia, X. Tian, and X. Gan, "Dynamic task assignment in crowdsensing with location awareness and location diversity," in *Proc. of IEEE INFOCOM*, 2018, pp. 2420–2428.

[9] T. Hu, M. Xiao, C. Hu, G. Gao, and B. Wang, "A qos-sensitive task assignment algorithm for mobile crowdsensing," *Pervasive and Mobile Computing*, vol. 41, pp. 333–342, 2017.

[10] W. Li, L. Wang, Y. Gu, R. Li, M. Song, and Z. Han, "Stable multiple activity matching based content sharing for mobile crowd sensing," in *IEEE International Conference on Comm. (ICC)*, 2018, pp. 1–6.

[11] Y. Chen and X. Yin, "Stable job assignment for crowdsourcing," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–6.

[12] P. Biró, D. Manlove, and S. Mittal, "Size versus stability in the marriage problem," *Theor. Comput. Sci.*, vol. 411, no. 16-18, pp. 1828–1841, 2010. [Online]. Available: https://doi.org/10.1016/j.tcs.2010.02.003

[13] 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. [Online]. Available: https://doi.org/10.1007/978-3-540-68279-0_2

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

[15] R. L. R. Thomas H. Cormen, Charles E. Leiserson and C. Stein, "Introduction to algorithms, third edition." MIT Press, 2009.