An Efficient Algorithm to Prevent Procrastination in Spatial Crowdsourcing

Naren Debnath, Sajal Mukhopadhyay and Fatos Xhafa

Abstract Spatial crowdsourcing entails many research works that address the issue of meeting deadlines by task executors (TE) to submit tasks where they can submit the tasks at any point of time within the given deadline. In accordance with the deadline, TEs may hold up their task submission until the penultimate day. Such a behavioural phenomenon is known as procrastination, when exercised by TEs, may create a problematic situation for the task provider (TP) who is also bounded by some deadline. Though procrastination has not at all been addressed in spatial crowdsourcing scenarios, one work addresses this issue by proposing a procrastination-aware scheduling algorithm in a bipartite graph environment. But the balancing effect of task distribution in different schedules (slots) has not been taken care of. In this paper, we have proposed a mechanism to prevent procrastination in spatial crowdsourcing scenarios considering the balancing effect. Extensive simulation is done, and in that our main finding is that the proposed mechanism performs significantly better than the existing method in terms of balancing effect.

1 Introduction

When a large set of tasks is brought to an open forum and an independent crowd is asked to execute them is called Crowdsourcing [9, 15]. Some examples of crowd-

Naren Debnath
Department of Computer Science and Engineering, National Institute of Technology Durgapur, Durgapur - 713209, West Bengal, India, e-mail: nd.20cs1104@phd.nitdgp.ac.in

Sajal Mukhopadhyay
Department of Computer Science and Engineering, National Institute of Technology Durgapur - 713209, Durgapur, West Bengal, India, e-mail: smukhopadhyay.cse@nitdgp.ac.in

Fatos Xhafa
Department of Computer Science, Universitat Politècnica de Catalunya, Barcelona - 08034, Catalonia, Spain, e-mail: fatos@cs.upc.edu

sourcing (cs) applications are agriculture [17], IoT [13], etc. A kind of crowdsourcing that specifically exploits sensors and mobile devices to acquire data is called mobile crowdsourcing [25, 8, 22]. A few applications of MCS are smart tourism and smart cities [12], IoT [13], crime watch [18].

In MCS, tasks at some locations might not attain adequate agents or TEs [7], which leads to the emergence of location-based crowdsourcing or spatial crowdsourcing (SC) where the task is allocated to the agents with spatiotemporal constraints. In SC, tasks are assigned to a TE with its location information and the deadline, and she has to be physically present at that location to execute the task within the given deadline. Examples of SC are food delivery service [14], on-demand vehicle service [7], collecting geo-spatial data [23], journalism [7] etc.

The literature of SC [14, 7, 23] concentrate on allocating tasks floated by TP to the TEs. Here TP and TEs both place their demands to a common platform. These literature of SC did not consider the fact that the submission of tasks by the TEs is getting delayed *i.e.,* the submission of tasks happening at the penultimate day or at the final day of the deadline. Such a behavioural phenomena is known as procrastination [20, 10, 11], when exercised by TEs, may create a problematic situation for the task provider (TP) who is also bound by some deadline and as the TEs are in haste, the submission quality may be poor as well. So, prevention of procrastination is necessary in SC environment and it is noticed that there is dearth of literature in SC to address this issue. One research work (though not in SC environment) by *Wang et al.* in a bipartite graph setting has proposed a procrastination-aware scheduling in [21]. But they have not considered the balancing of tasks to avoid procrastination in their paper.

Our proposed mechanism begins with the fact that the platform has already allocated tasks to the TEs. However, the TEs may procrastination within the given deadline. In this mechanism the platform (a cognitive agent) perceives that the TEs may procrastinate and efficiently distributes tasks into different schedules (slots with equal time frames) in a balanced manner to avoid procrastination. We have defined *schedules* in Section 3. The idea of distributing the tasks in a balanced manner will help us prevent procrastination in a systematic manner.

The rest of our paper is organised as follows. Section 2 presents the most relevant literature review. Section 3 provides the system model and objective function. Section 4 elaborates the proposed mechanism and provides the time complexity analysis. In Section 5, we have furnished the simulation. We have ended our paper with some concluding remarks and future directives in Section 6.

2 Literature Review

Procrastination could be comprehended from the story of the famous economist G. Akerlof mentioned in [1, 10, 20]. In brief, his story narrates that he procrastinated while returning his friend's luggage for months due to the long postal process. This story brings out the fact that procrastination is a common trait of human behaviour based on which systematic mechanisms could be designed. Our paper is motivated in this direction by the above-mentioned fact.

There are a salient number of research works have been carried out on *task scheduling* in crowdsourcing environments with load balancing and deadline [19, 2, 3, 16, 6, 24, 26]. In [19],[2], and [3], spatial task scheduling mechanisms were proposed where tasks were scheduled, maintaining the load balance among the TEs. [16] have addressed task scheduling in IoT/Mobile Crowdsourcing environments where they have ensured the balanced scheduling of tasks, which is deadline and budget-sensitive. [6], [24], and [26] have proposed task assignment scheduling mechanisms where the task is constrained by the deadline. The above review addresses load balancing and time constraints in spatial crowdsourcing, but procrastination has not at all been addressed.

A task-scheduling mechanism that explicitly addresses the issue of procrastination but not specifically in crowdsourcing is found in [21]. They have proposed a Procrastination-aware-Scheduling algorithm (*OFFPSP*) in which all the tasks or jobs are allocated into a set of schedules in some order where each schedule has a threshold. This algorithm produces an unbalanced set of schedules, which can cause irregular completion and submission of tasks. This irregular submission of tasks might be problematic for the task requester (or TP) as she will not be able to get ample time to verify all the submitted tasks. We have addressed the issue of balancing of tasks so that all the tasks are completed and submitted uniformly to the TP. In addition to that, we have pruned the choices of TEs to prevent procrastination.

3 System Models

As stated earlier, our model begins with the fact that the tasks are already allocated to the TEs and how we can prevent the procrastination of TEs within the given deadline $\mathbb{R}$. For example, suppose a TE is already assigned a set of 10 tasks and it was given a deadline $\mathbb{R} = 21$ days. It may be the case that the TE may submit the task on the last day ($\mathbb{R}^{th}$ day) or as late as possible. This procrastination issue of TE is not addressed in earlier literature on spatial crowdsourcing (SC). In our model, we have proposed a mechanism to prevent the procrastination of an arbitrary TE i and thereby, it is applicable to all the TEs in the same way. In our model, TEs are heterogeneous in nature, *i.e.,* they can perform different types of jobs and the capacity of performing of jobs is also different. The key constituents here are a set of schedules (Λ), a set of Tasks ($\Im$), and a set of locations ($\mathbb{L}$), discussed below.

1. Notations and their Usage

Schedules and Deadline: A set of schedules (or slots) $\Lambda = \{\alpha_1, \alpha_2, \cdots, \alpha_l\}$ is created based on the overall deadline $\mathbb{R}$. Each of these schedules or slots contains a subset of tasks (defined next). As the schedules (slots) are created based on $\mathbb{R}$, each schedule has a deadline also. The deadline for each schedule is created by dividing $\mathbb{R}$ into l sub-deadlines of equal length, represented as $\mathbb{R} = \{\theta_1, \theta_2, \cdots, \theta_l\}$, where $\theta_1 = \theta_2 = \cdots = \theta_{|\Lambda|}$ and $l = |\Lambda|$. $|\Lambda|$ is defined as $|\Lambda| = \frac{\mathbb{R}}{\theta_i}$ where $\theta_i \in \mathbb{R}, \theta_i \neq 0$, and

$\theta_i \leq \mathbb{R}$. The duration of θ_i is application-dependent, and accordingly, our proposed mechanism is pliable enough for any such application.
Set of Tasks: The set of tasks is defined as $\Im = \{\iota_1, \iota_2, \cdots, \iota_n\}$ where $n = |\Im|$ and the costs of these tasks is represented as $\kappa = \{\varkappa_1, \varkappa_2, \cdots, \varkappa_n\}$ where $\varkappa_i \in \kappa$ corresponds to the task $\iota_i \in \Im$. The cost could be the power consumption of the mobile device(s), time to complete the task, internet charge, etc. Each of these tasks will be allocated to a schedule α_i. The total cost of any schedule α_i will be denoted as α_i^c. Note that for any two schedules α_i and $\alpha_j, i \neq j$, will not be allocated with the same task, *i.e.,* $\alpha_i \cap \alpha_j = \phi, \forall_{(\alpha_i, \alpha_j) \in \Lambda}$.
Location: A set of locations defined as $\mathbb{L} = \{\lambda_1, \lambda_2, \cdots, \lambda_m\}$ constitutes the SC area where in each location a set of tasks is cluttered. That means the location $\lambda_e \in \mathbb{L}$ is populated with a set of tasks $\Im$. Each task $\iota_i \in \Im$ is associated with a cost $\varkappa_i$ and a location λ_e. So, the set of tasks $\Im$ could be redefined as $\Im = \{< \iota_1, \varkappa_1, \lambda_e >, < \iota_2, \varkappa_2, \lambda_e >, \cdots, < \iota_n, \varkappa_n, \lambda_e >\}$. Our proposed mechanism is based on an arbitrary location λ_i and could be applied to all such m locations.

2. Objective

Given a set of tasks $\Im$ and a set of schedules (slots) Λ, at the location $\lambda_e \in \mathbb{L}$ our objective is to distribute $\Im$ tasks into different schedules uniformly so that all the schedules become balanced in terms of total cost and number of tasks. In other words, our objective is to minimize the standard deviation of each schedule in terms of total cost and number of tasks. Such uniformity ensures the distribution of higher cost tasks into all the schedules which prohibits the accumulation of such higher cost tasks into one schedule. This will prevent delay in the execution of those higher cost tasks along with the other tasks of that schedule.

As our objective relies on uniform (or balanced in terms of the objective function defined below) distribution of tasks, the deadline for each schedule must also be uniform so that each schedule gets equal amount of time to complete the tasks assigned to it. To achieve that, we have divided $\mathbb{R}$ into $|\Lambda|$ equal sub-deadlines for each schedule and each sub-deadline is represented by θ_i. We have summarised our objective in Eq. (1) where, σ_c and σ_d represent the *standard deviations* (given in Eq. (2) and Eq. (3)) in terms of cost and number of tasks of each schedule, respectively.

$$\textit{Minimize } \sigma_c\{\Lambda\} \textit{ and } \sigma_d\{|\alpha_1|, |\alpha_2|, \cdots, |\alpha_l|\}, \textit{ Subject to } \theta_i = \frac{\mathbb{R}}{|\Lambda|} \tag{1}$$

where,

$$\sigma_c = \sqrt{\frac{\Sigma_{i=1}^{|\Lambda|}(\alpha_i^c - \frac{\Sigma_{k=1}^{|\Lambda|}\alpha_k^c}{|\Lambda|})^2}{|\Lambda|}} \quad (2) \qquad \sigma_d = \sqrt{\frac{\Sigma_{i=1}^{|\Lambda|}(|\alpha_i| - \frac{\Sigma_{k=1}^{|\Lambda|}|\alpha_k|}{|\Lambda|})^2}{|\Lambda|}} \quad (3)$$

4 Proposed Mechanism

This section discusses the proposed mechanism Uniform Cost Scheduling Mechanism for Prevention of Procrastination(UCSMPP) in two ways. First, a schematic diagram of the proposed mechanism is provided. Second, the algorithms are discussed. Finally, an example with an arbitrary dataset is furnished to illustrate our proposed mechanism.

4.1 Schematic Diagram of the Proposed Mechanism

Fig. 1 shows the flow of our proposed mechanism UCSMPP. In Block 1, the agent submits n tasks to the platform, where they are arranged in ascending order according to their costs (Block 2). In Block 3, the sorted tasks are allocated into l schedules in the following manner - $\iota_1, \iota_n \rightarrow \alpha_1$, $\iota_2, \iota_{n-1} \rightarrow \alpha_2$, and so on upto $\iota_{n-l-1}, \iota_l \rightarrow \alpha_l$. This process is repeated for all the tasks. Block 4 generates the schedules with allocated tasks. Finally, Block 5 publishes schedules with heaviest schedule first manner.

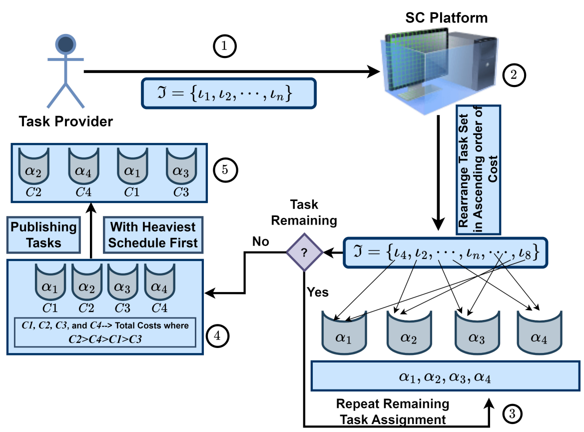


Fig. 1: Flow Diagram of the Proposed Mechanism

4.2 Uniform Cost Scheduling Mechanism for Prevention of Procrastination

4.2.1 Uniform Task Distribution as per Costs (UTDC)

This algorithm distributes the set of tasks into *units* number of schedules after reordering the set of tasks in ascending order (Algorithm 1). The input to this algorithm is $\Im$ and $\mathbb{R}$, and the output is Λ. Each $\lambda_i \in \Lambda$ is allocated with some tasks where each task has a cost. So, the total cost of each schedule is the sum of the costs of all tasks allocated to it. At the end of the Algorithm 1, the schedules are published to the TEs in reverse order of their total cost.

The proposed algorithm divides $\mathbb{R}$ into several schedules with the same deadline (defined in Section 3) (choice reduction) and distributes higher cost tasks along with other tasks into those schedules uniformly (balancing effect). By choice reduction, we mean that $\mathbb{R}$ is divided into smaller and equal time frames (schedules), and allocated tasks in each schedule are bound to be completed within the time frame of that schedule. Thus the choices of TEs for submitting tasks are reduced from the larger time frame $\mathbb{R}$ to smaller time frames so that they can periodically submit the tasks. This is one of the key ideas of our proposed algorithm.

So, we can see that the balancing effect and choice reduction in our proposed algorithm prevent procrastination. Also, the reverse order of publishing schedules will benefit TP in the sense that when TEs submit the tasks, TP will have enough time to verify the heaviest schedule first and so on.

Algorithm 1 Uniform Task Distribution as per Costs (UTDC)

Require: $\Im, \mathbb{R}$
Ensure: Λ *with allocated tasks*
$S\_list \leftarrow call\ TSRAO(\Im, 1, |\Im|)$ ▷*sorted list as per cost*
$l \leftarrow \frac{\mathbb{R}}{\theta},\ i \leftarrow 0,\ j \leftarrow |\Im| - 1$ ▷$\theta:$ *deadline of one schedule*
Initialise $\alpha^k \leftarrow \{\}$ for $k = 1\ to\ l$ ▷$l:$ *number of schedules*
while $i \leq \lfloor \frac{|\Im|}{2} \rfloor$ **do**
    $k \leftarrow 0$
    while $k < l$ **do**
        if $i == j$ **then** ▷ $: if\ |\Lambda| \mid |\Im|$
            $\alpha^k \cup S\_list_i$, *break*
        else if $i > j$ **then** ▷ $: if\ |\Lambda| \nmid |\Im|$
            break
        else
            $\alpha^k \cup S\_list_i,\ \alpha^k \cup S\_list_j$
            $k += 1,\ i += 1,\ j -= 1$
        end if
    end while
end while
Λ *is published in reverse order of the cost of each* $\alpha_k \in \Lambda$

Algorithm 2 Subroutine TSRAO$(A, 1, |A|)$

if $|\Im| > 1$ **then**
    $mid \leftarrow \frac{|A|}{2}$
    $TSRAO(A, 1, mid)$
    $TSRAO(A, mid + 1, |A|)$
    $merge(A, 1, mid, |A|)$ ▷*Merge sort is used here*
end if
return A

4.2.2 Task Set Rearrangement in Ascending Order (TSRAO)

Algorithm 1 calls the subroutine $TSRAO()$ ($Algorithm$ 2) to rearrange the tasks in $\Im$ in ascending order of their costs by implementing merge sort algorithm.

Example 1. Let $\kappa = \{89,57,6,80,85,53,20,98,10,32,34,11,23,50,60,14,18\}$ and $no_{sch} = 4$. Rearranging $\Im$ in ascending order of the cost, we get $S_{list} = \{6,10,11,14,18,20,23,32,34,50,53,57,60,80,85,89,98\}$. Now, from the new list S_{list}, 1^{st} and 17^{th} tasks are assigned to α_1, 2^{nd} and 16^{th} tasks are assigned to α_2, and so on up to α_4. It is repeated until all the tasks are assigned to the respective schedules. The final task allocation is, $\alpha_1 = \{98,6,60,18,34\}$, $\alpha_2 = \{89,10,57,20\}$, $\alpha_3 = \{85,11,53,23\}$, $\alpha_4 = \{80,14,50,32\}$ where the total cost for $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ are $216,176,172,176$ respectively. After task allocation, each schedule is published to the TEs according to the decreasing order of their costs (see Section 4.1). Each schedule will get $\frac{\mathbb{R}}{4} = 1$ week. In 1^{st} week, tasks in α_1, in 2^{nd} week tasks in α_2, in 3^{rd} week tasks in α_4 and in 4^{th} week tasks in α_3 will be executed.

4.3 Time Complexity Analysis

Lemma 1 *The time complexity by UCSMPP is* $O(|\Im| \log |\Im|)$.

Proof. Algorithm 1 will iterate atmost $\lceil \frac{|\Im|}{2} \rceil$ times including the inside *while* loop. As each pair of elements is assigned exactly once, so over $\lceil \frac{|\Im|}{2} \rceil$ number of iterations, atmost $|\Im|$ elements will be assigned. As each assignment takes $O(1)$ time, the algorithm takes $O(|\Im|)$ time. This analysis is motivated by one of the methods of amortised analysis, termed as aggregated analysis as mentioned in [5]. In our algorithm, the preprocessing (*merge sort*) takes $O(|\Im| \log |\Im|)$ and the main segment takes $O(|\Im|)$ time. □

5 Experiments and Results

5.1 Setup

Overall Deadline: $\mathbb{R}$ for all the tasks is considered 4 weeks, *i.e.,* $\mathbb{R} = 4$ *weeks*.
Schedules and Their Deadlines: Here, *i.e.,* $\Lambda = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}, |\Lambda| = 4, \theta_i = 1$ *week*. Therefore, deadline of each α_i is $\theta_i = \frac{\mathbb{R}}{|\Lambda|} = 1$ week.
System Specification: The simulation platform has 5^{th} generation microprocessor, 8 GB of RAM, Ubuntu as the operating system and *Google Colab* platform (with Python).

5.2 Dataset

Synthetic Dataset: The synthetic dataset, shown in Table 1, has been generated randomly with 200 jobs using normal distribution (given in Eq. (4)),

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\alpha})^2} \; with \; (\mu) = 75, \; (\sigma) = 80 \; and \; x = 200. \tag{4}$$

Table 1: Sample of Synthetic Dataset

Sl #	Job ($\Im$)	Job Cost ($\varkappa$)
1	ι_1	10
2	ι_2	40
3	ι_3	25
4	ι_4	100
5	ι_5	89

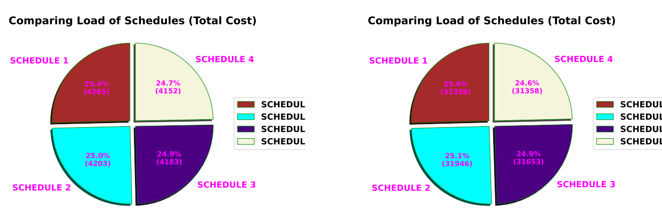
Table 2: Sample of BDD Dataset

Sl#	Bus_ID	Tot_Dur
1	1	1145
2	2	1110
3	3	850
4	4	1090
5	5	840

Bus Driver Scheduling Dataset (BDD): In BDD [4], *Bus_Line_ID* and *Duration* columns are extracted. Then for each *Bus_Line_ID* (each bus), total running cost is calculated for the entire day by summing up all the values in *Duration* column. For instance, for *Bus_Line_ID*= 1, the total running time is 1145 minutes. In such a way, a total of 359 bus records have been collected, some are shown in Table 2.

5.3 Simulation Study of UCSMPP

For our simulation, we have used the following parameters.
1. Total Cost of Each Schedule: With this simulation, we have shown that the total cost of each schedule is almost balanced. Each segment of the chart in Fig. 2 shows that each schedule is almost balanced in terms of the total cost of jobs.



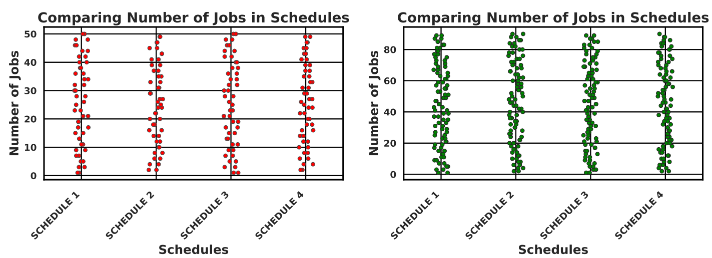
(a) Synthetic Dataset (b) Bus Driver Scheduling Dataset

Fig. 2: Total Cost of Each Schedule with Synthetic and BDD Datasets

2. Number of Tasks: This simulation shows that the number of tasks in each schedule is also uniform (Fig. 3). We have determined the number of tasks in each schedule in the following way. Let N denote the number of pairs of tasks where $N = \lceil \frac{|\Im|}{2} \rceil$ and let $M = |\Lambda|$. 1^{st} pair contains 1^{st} and $|\Im|^{th}$ tasks, 2^{nd} pair contains 2^{nd} and $(|\Im|-1)^{th}$ tasks and so on upto N^{th} pair which contains $\frac{|\Im|}{2}^{th}$ task, if $2 \nmid |\Im|$, else it contains $\frac{|\Im|}{2}^{th}$ and $(\frac{|\Im|}{2}+1)^{th}$ tasks. Such pairing is only considered when $\Im$ is sorted in ascending order.

$$\alpha_i = \begin{cases} \frac{N}{M} & if\ M \mid N,\ for\ i = 1\ to\ M \\ \frac{N}{M}+1 & if\ M \nmid N,\ for\ i = 1\ to\ N\ mod\ M \\ \frac{N}{M} & if\ M \nmid N,\ for\ i = (N\ mod\ M)+1\ to\ M \end{cases} \tag{5}$$

With such setting, Algorithm 1 will allocate 1^{st} pair to 1^{st} schedule, 2^{nd} pair to 2^{nd} schedule and so on upto M^{th} schedule, repeatedly for all pairs. So, the number of pairs in each schedule is atleast $\lfloor \frac{N}{M} \rfloor$. The number of tasks (pair-wise) allocated to each schedule is given in Eq. (5) where *mod* finds the remainder.



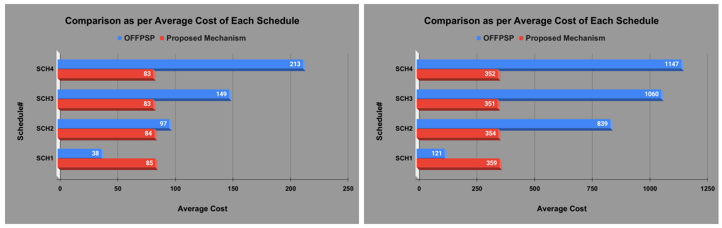
(a) Synthetic Dataset (b) Bus Driver Scheduling Dataset

Fig. 3: Number of Tasks in Each Schedule with Synthetic and BDD Datasets

5.4 Comparative Study: UCSMPP vs OFFPSP

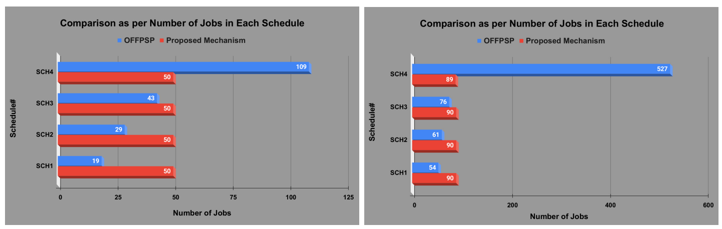
Our proposed mechanism, UCSMPP, is compared with an existing mechanism called OFFPSP on the basis of **Average Cost** and **Number of Tasks** in each schedule.

When compared with the OFFPSP, UCSMPP shows better balancing effect in terms of both average cost (Fig. 4) and number of tasks (Fig. 5) of each schedule. In each of the Fig. 4 and Fig. 5 , we can see that in UCSMPP, the average cost and number of tasks of each schedule is almost similar. Whereas in OFFPSP, the average cost and number of tasks of each schedule varies significantly.



(a) Synthetic Dataset (b) Bus Driver Scheduling Dataset

Fig. 4: Comparison Between UCSMPP and OFFPSP on Average Cost of Each Schedule using Synthetic and BDD Datasets



(a) Synthetic Dataset (b) Bus Driver Scheduling Dataset

Fig. 5: Comparison Between UCSMPP and OFFPSP on Number of Tasks in Each Schedule using Synthetic and BDD Datasets

6 Conclusion and Future Works

We claim that our proposed mechanism produces balanced schedules in terms of: *i*) total cost, *ii*) average cost, and *iii*) number of tasks in each schedule. The deadline of the tasks in each schedule is obtained by equally dividing $\mathbb{R}$. The schedules are published to the TEs in *heaviest schedule first* order so that the TEs will be able to perform with less load as they proceed with the schedules and the task provider will get enough time for verification of submitted tasks of heavier schedule. Through our proposed mechanism we have achieved, *i*) **Choice Reduction:** the choices for submitting tasks are reduced for TEs by equally dividing $\mathbb{R}$ into schedules, *ii*) **Balancing Effect:** all the schedules are balanced in terms of total cost and number of tasks, and *iii*) **Benefit of Task Provider:** task provider is benefited as all the schedules are published in reverse order of their total cost. Through rigorous simulation with synthetic and real datasets we have shown that our proposed mechanism performs better than the existing one in terms of balancing effect.

The possible future directive could be to use randomised mechanism to allocate tasks into schedules. Giving incentives to the TEs along with choice reduction could be another direction to prevent procrastination.

References

[1] George A Akerlof. "Procrastination and obedience". In: *The american economic review* 81.2 (1991), pp. 1–19.

[2] Afra A Alabbadi and Maysoon F Abulkhair. "Task-Scheduling Based on Multi-Objective Particle Swarm Optimization in Spatial Crowdsourcing". In: *J. King Abdulaziz Univ. Comput. Inf. Technol. Sci.* 8 (2019), pp. 45–57.

[3] Afra A Alabbadi and Maysoon F Abulkhair. "Multi-objective task scheduling optimization in spatial crowdsourcing". In: *Algorithms* 14.3 (2021), p. 77.

[4] Ademir Aparecido Constantino et al. "Solving a large real-world bus driver scheduling problem with a multi-assignment based heuristic algorithm". In: *Journal of Universal Computer Science* 23.5 (2017).

[5] Thomas H Cormen et al. *Introduction to algorithms*. PHI Learning (Originally MIT press), 2010.

[6] Dingxiong Deng, Cyrus Shahabi, and Linhong Zhu. "Task matching and scheduling for multiple workers in spatial crowdsourcing". In: *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 2015, pp. 1–10.

[7] Srinivasa Raghavendra Bhuvan Gummidi, Xike Xie, and Torben Bach Pedersen. "A survey of spatial crowdsourcing". In: *ACM Transactions on Database Systems (TODS)* 44.2 (2019), pp. 1–46.

[8] Shugang Hao and Lingjie Duan. "To Save Mobile Crowdsourcing from Cheap-talk: A Game Theoretic Learning Approach". In: *IEEE Transactions on Mobile Computing* 23.8 (2024), pp. 8418–8430.

[9] Jeff Howe. "The rise of crowdsourcing". In: *Wired magazine* 14.6 (2006), pp. 1–4.

[10] Jon Kleinberg and Sigal Oren. "Time-inconsistent planning: a computational problem in behavioral economics". In: *Proceedings of the fifteenth ACM conference on Economics and computation*. 2014, pp. 547–564.

[11] Jon Kleinberg, Sigal Oren, and Manish Raghavan. "Planning with multiple biases". In: *Proceedings of the 2017 ACM Conference on Economics and Computation*. 2017, pp. 567–584.

[12] Aristea Kontogianni et al. "Mobile Applications in Smart Tourism and Smart Cities Based on Crowdsourcing". In: *Smart Tourism–The Impact of Artificial Intelligence and Blockchain*. Springer, 2024, pp. 33–52.

[13] Bahareh Lashkari et al. "Crowdsourcing and sensing for indoor localization in IoT: A review". In: *IEEE Sensors Journal* 19.7 (2018), pp. 2408–2434.

[14] Yan Liu et al. "FooDNet: Toward an optimized food delivery network based on spatial crowdsourcing". In: *IEEE Transactions on Mobile Computing* 18.6 (2018), pp. 1288–1301.

[15] Jaya Mukhopadhyay et al. "A Balanced Dissemination of Time Constraint Tasks in Mobile Crowdsourcing: A Double Auction Perspective". In: *Advances on P2P, Parallel, Grid, Cloud and Internet Computing: Proceedings of the 15th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC-2020) 15*. Springer. 2021, pp. 74–85.

[16] Abbas M Ali Al-muqarm and Naseer Ali Hussien. “Dynamic cost-optimized resources management and task scheduling with deadline constraint for mobile crowd sensing environment”. In: *Int. J. Intell. Eng. Syst* 16.3 (2023). DOI: 10.22266/ijies2023.0630.16.
[17] Mahbubur Rahman et al. “Smartphone-based hierarchical crowdsourcing for weed identification”. In: *Computers and Electronics in Agriculture* 113 (2015), pp. 14–23.
[18] Sumit Shah et al. “Crowdsafe: crowd sourcing of crime incidents and safe routing on mobile devices”. In: *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 2011, pp. 521–524.
[19] Dezhi Sun, Yong Gao, and Dan Yu. “Efficient and Load Balancing Strategy for Task Scheduling in Spatial Crowdsourcing”. In: *Web-Age Information Management: WAIM 2016 International Workshops, MWDA, SDMMW, and SemiBDMA, Nanchang, China, June 3-5, 2016, Revised Selected Papers 17*. Springer. 2016, pp. 161–173.
[20] Roughgarden Tim. *CS269I: Incentives in Computer Science Lecture# 19: Time-Inconsistent Planning*. Stanford Course, 2016.
[21] Libin Wang et al. “Procrastination-aware Scheduling: A Bipartite Graph Perspective”. In: *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE. 2019, pp. 1650–1653.
[22] Xiumin Wang et al. “Promoting users’ participation in mobile crowdsourcing: A distributed truthful incentive mechanism (dtim) approach”. In: *IEEE Transactions on Vehicular Technology* 69.5 (2020), pp. 5570–5582.
[23] Zhi Wang et al. “Worker Collaborative group estimation in spatial crowdsourcing”. In: *Neurocomputing* 428 (2021), pp. 385–391.
[24] Yan Zhao et al. “Destination-aware task assignment in spatial crowdsourcing: A worker decomposition approach”. In: *IEEE Transactions on Knowledge and Data Engineering* 32.12 (2019), pp. 2336–2350.
[25] Ying Zhao et al. “Mobile Crowdsourcing Quality Control Method Based on Four-Party Evolutionary Game in Edge Cloud Environment”. In: *IEEE Transactions on Computational Social Systems* 11.3 (2024), pp. 3652–3666.
[26] Jingya Zhou, Jianxi Fan, and Jin Wang. “Task scheduling for mobile edge computing enabled crowd sensing applications”. In: *International Journal of Sensor Networks* 35.2 (2021), pp. 88–98.