01 Jan 2022

Volunteer Selection in Collaborative Crowdsourcing with Adaptive Common Working Time Slots

Riya Samanta

Vaibhav Saxena

Soumya K. Ghosh

Sajal K. Das
Missouri University of Science and Technology, sdas@mst.edu

Recommended Citation

R. Samanta et al., "Volunteer Selection in Collaborative Crowdsourcing with Adaptive Common Working Time Slots," *2022 IEEE Global Communications Conference, GLOBECOM 2022 - Proceedings*, pp. 4643 - 4648, Institute of Electrical and Electronics Engineers, Jan 2022.
The definitive version is available at https://doi.org/10.1109/GLOBECOM48099.2022.10001191

Volunteer Selection in Collaborative Crowdsourcing with Adaptive Common Working Time Slots

Riya Samanta[†], Vaibhav Saxena[†], Soumya K. Ghosh[†], and Sajal K. Das[‡]
[†]Indian Institute of Technology Kharagpur, India
[‡]Missouri University of Science and Technology, Rolla, USA
riya.samanta@iitkgp.ac.in, saxenavaibhav@kgpian.iitkgp.ac.in, skg@cse.iitkgp.ac.in, sdas@mst.edu

***Abstract*—Skill-based volunteering is an expanding branch of crowdsourcing where one may acquire sustainable services, solutions, and ideas from the crowd by connecting with them online. The optimal mapping between volunteers and tasks with collaboration becomes challenging for complex tasks demanding greater skills and cognitive ability. Unlike traditional crowdsourcing, volunteers like to work on their own schedule and locations. To address this problem, we propose a novel two-phase framework consisting of Initial Volunteer-Task Mapping (i-VTM) and Adaptive Common Slot Finding (a-CSF) algorithms. The i-VTM algorithm assigns volunteers to the tasks based on their skills and spatial proximity, whereas the a-CSF algorithm recommends appropriate common working time slots for successful volunteer collaboration. Both the algorithms aim to maximise the overall utility of the crowdsourcing platform. Experimenting with the UpWork dataset demonstrates the efficacy of our framework over existing state-of-the-art methods.**

***Index Terms*—Crowdsourcing, Skill-based volunteering, Spatial task allocation, Collaboration, Common working time**

I. Introduction

Crowdsourcing, be it spatial or non-spatial, has even shifted from traditionally professional areas to involving a large number of people. This happened due to the significant technological advancements over the last decade. The capacity to quickly create content online via Web 2.0 and the proliferation of smart mobile devices that can timely record the location of assets have always had a positive technological impact. Now, we use this technology to create a skill-based volunteering crowdsourcing system (VCS) that finds its role in different domains like healthcare, emergency, sustainable development, social awareness, etc.

VCS is an expanding branch of crowdsourcing where one may acquire sustainable services, solutions, and ideas from the crowd by connecting with them online. This is different from commercial platforms like Mechanical Turk [1], UpWork [2], Freelancer.com, and Topcoder.com that allow enterprises and organizations to hire crowd-workers to do various tasks, saving money and resources. In contrast, volunteer crowd-workers can be neophytes, expert amateurs, professionals, or a combination of all contributing to the socio-economic upliftment and sustainable development goals of shared interests.

One of the most important research issues of any crowdsourcing infrastructure is task assignment to best-fit workers. The mapping between tasks and ideal crowd-workers becomes particularly challenging for complex tasks that demand high skill needs and cognitive capacities; but no single worker may satisfy diverse requirements of such a complex task [3]–[5]. Dealing with volunteers raises additional challenges that are also common in traditional enterprise crowdsourcing. One factor we address in this research which is more prevalent to VCS is that volunteers prefer to work in their physical vicinity, at home (for non-spatial tasks) or in local neighborhood (for spatial tasks), and also at their own schedule. They presumably prioritise tasks that best fit their skills, abilities, and interests. Although they are not paid directly, they are reimbursed for any expenses related to the execution of assigned activities, such as travel expenses, instrument costs, medical aid, etc. One of the primary critiques of volunteering is that inexperienced volunteers do tasks that normally need qualified personnel. Volunteered instructors with little or no experience are prevalent. Volunteering work abroad allows medical students to undertake unskilled procedures tests. In any women's empowerment projects, micro-scale rural commerce units, etc.,there is a need of highly skilled professionals to train novice workers. Thus, in order to ensure that volunteers' skills fit organisational needs while also accommodating their schedules, a thorough volunteer-task mapping method is necessary.

Our Contributions: This paper proposes a novel two-phase framework to select volunteers for spatially scattered tasks (Phase-1) and then recommend an adaptive common working slot for effective volunteer collaboration (Phase-2). Our main contributions are:

(1) We formulate the volunteer-task mapping (VTM) as a constraint optimization problem. The tasks into consideration have a location attribute along with certain skill requirements.

(2) We propose an *Initial Volunteer-Task Mapping Algorithm* (**i-VTM**) for assigning volunteers to the tasks based on their skills and location proximity. Next, we propose an *Adaptive Common Slot Finding Algorithm* (**a-CSF**) to suggest a common working slot for effective volunteer collaboration.

(3) We conduct experiments on a real dataset, **UpWork** [2] (a marketplace for freelancing and crowdsourcing) consisting of 97 tasks and 1,575 candidates in the sample. Results show that the proposed strategy is effective in comparison to the other state-of-the-art techniques.

The rest of the paper is organized as follows. Section II briefly reviews the related work. Section III formulates the problem while Section IV describes the proposed framework. Section V presents experimental results to evaluate the performance of the framework. Section VI concludes the paper.

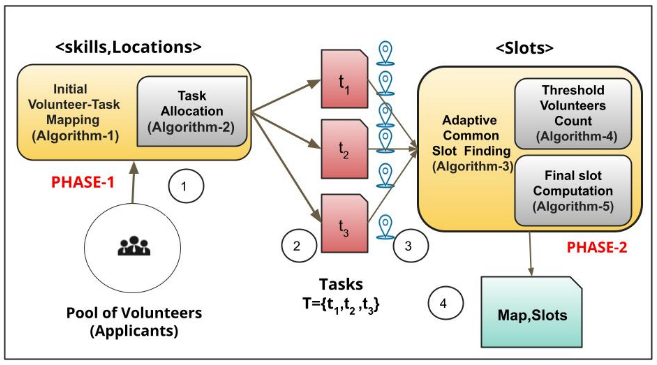

<skills,Locations>
Initial Volunteer-Task Mapping (Algorithm-1)
Task Allocation (Algorithm-2)
PHASE-1
Pool of Volunteers (Applicants)
Tasks T={t1,t2,t3}
<Slots>
Adaptive Common Slot Finding (Algorithm-3)
Threshold Volunteers Count (Algorithm-4)
Final slot Computation (Algorithm-5)
PHASE-2
Map,Slots

Fig. 1: Proposed Framework

II. Related Work

Regardless of the form of crowdsourcing, task allocation is one of the most important issues. Because of the emergence of GPS and mobile sensing technology, spatial crowdsourcing [6], [7] is gaining popularity. Since participants in spatial crowdsourcing must physically progress to the task's location, the task allocation procedure should also include their ability to reach the target place. The conceptualization and management of complex tasks is extensively studied [3], [8].

In [9], the authors proposed the GMA approach, which was influenced by group strategy, and subsequently fine-tuned it to produce the RGMA method. Other works based on diverse skill based complex task allocation are [3], [4]. The authors [8] of paper stresses on decomposing a complex tasks into their atomic counterparts for improved completion rate and reliable solutions. On the same ground, the [8] proposes combining related subtasks into a batch that will be assigned to the same participants. Bounded budget and non-homogeneous tasks, each demanding certain skills, are considered in [10], which uses bipartite matching to design an incentive-compatible method; but it does not consider the dynamic setting of the entity's arrival trend. The paper [11] discusses a participant selection scenario of crowdsourcing in disaster management area. In [12] the authors presented a team recruiting technique for collaborative crowdsourcing based on team members' social network neighbours' familiarity. Collaborative crowdsouring is popular in software development marketplace [13], [14]. Yet another popular idea is to use incentives mechanisms [15], [16].

To the best of our knowledge, none of the above works explored skill-based volunteer allocation for spatially distributed complex tasks, which require a variety of skills that may not be met by a single volunteer. Though the paper [3] deals with the complex spatial task allocation, their greedy approach runs in $O(|C|+|T|)^2$. We also advocate a shared working slot for all volunteers to help in collaboration. Volunteers are preferably assigned tasks that are nearest to their proximity. The authors in [14] do consider workers' active time preferences, but the tasks are neither spatial nor they recommended any common time slot. Their allocation is also one-to-one.

III. Problem Formulation

Given a set of complex spatial tasks T. Any task $t \in T$ has a list of skills Q_t required by it and a set of locations L_t. Each replica r_{ti} of t may be will be located at P_{ti} from the set L_t. It is assumed that when t is posted on a VCS platform, the interested volunteers would apply. Thus, a cluster of applicants A is formed. Any volunteer $c \in A$ has a list of skills Q_c and is linked to a current location P_c. Along with it, c also has a time slot $Slot_c$ with starting time S_c and ending time E_c. This slot is the preferable active online time for c. It is obvious that the interested volunteer should have certain skills/experience/abilities/knowledge required for task t. These matching competencies actually encourage them to apply. The cost Z_c incurred by c is based on the distance $Dist(P_c, P_{ti})$. This $Dist$ is the distance between c and any i^{th} the replica of the task t for which c will be selected and the expense is charged per unit distance, denoted by δ. For simplicity, we considered the Euclidean distance formula, and the locations for both the entities are assumed to be in (x,y) point format in a 2D Cartesian Coordinate system.

The working time of the volunteers A is utilized as an important driving factor for initiating collaboration among them. We need to determine a common working time slot applicable to most of the selected volunteers for a given t's i^{th} replica. It is to be noted that all replicas of any task t shall have the same skill requirements but vary with respect to location attributes. Thus, skill requirements Q_t for all replicas of each task t, need to be satisfied for successful allocation and every c would be assigned to his/her near-most located replica.

Problem Statement: We propose a two-phase framework consisting of: (1) *Volunteer-Task Mapping* (VTM): Given a set of spatial tasks T (including their corresponding replicas) and a set of interested volunteers A, assign volunteers to the best-fit tasks' replica based on the former's skills and spatial proximity such that skills requirements of the tasks are covered and the net utility (refer Formula-1) is maximized. (2) *Common working time slot* finding for productive collaboration among the volunteers C_{ti} selected for r_{ti} replica of $t \in T$, while assuring volunteers' feasibility. Thus, when a candidate c is assigned to a task t, the emphasising utility (c,t) is given by:

$$U(c,t) = |S_t'|/Z_c \tag{1}$$

where S_t' represents the candidate's matched skills, Z_c is calculated as $Dist(P_c, P_{ti}) * \delta$ for each $c \in C_{ti}$ towards the i^{th} replica of t and δ is the trip expense per unit distance. The net utility is the overall utility supplied by all selected (task, volunteer) pairings at the conclusion of assignment cycle. The goal is to select a set of volunteers C_{ti} that reduces the incurred cost while covering Q_t so as to maximise the utility gained by including c. This is followed for every task,volunteer pair, hence optimising the overall utility of the VCS platform. This leads us to construct the VTM problem as constraint optimization problem.

Theorem 1. *VTM can be reduced to Weighted Set Cover (WSC) problem and hence is NP-hard.*

Proof. In an instance of WSC there is a m size universal set $U = \{a_1, a_2, ...a_m\}$ and its n subsets $A_1, A_2, ...A_n$. Each A_i has a weight w_i.The aim is to find $A^* \subseteq A = \{A_1, A_2, ...A_n\}$ such that union of A is U and $\sum_{A_i \in A^*} w_i$ is minimized.

In case of VTM, suppose there is a single task $T = \{t\}$ with only one replica, whose skill requirements is $Q_t = U$. There set of applicants is A of cardinality n. The cost incurred by each applicant from A is Z_i. Our goal is to find a set of volunteers $C_t \subseteq A$ to minimize the cost $\sum_{i \in C_t} Z_i$ in order to maximise the utility (Equation-1). Thus, VTM can be successfully reduced to WSC which is a popular example of NP-hard problems [17]. Hence, VTM is also NP-hard. □

IV. Proposed Framework

The framework has two distinguished phases. The first one is the *i-VTM* phase, and the second one is the *a-CSF* phase. The illustration of our proposed framework is depicted in Figure-1. The primary objective of this work is optimizing task allocation, whereas proposing common work slots to chosen applicants is an added functionality. The phase-1's action is contingent upon the parameters *skill set* and *spatial location* being known. The phase-2's functionality is determined by the information given on the parameters *slot timings* and the output *allocation map* from phase-1.

The concept of *bipartite graph* is utilized in scheming three main data structures, namely Allocation Map (*Map* or G_{map}), Skill-Task Mapper (G_{ST}) and Skill-Volunteer Mapper (G_{SM}).

A. Initial Volunteer-Task Mapping Algorithm

This section describes in detail the i-VTM algorithm. To begin, the volunteer having the highest skill is selected at step-4. In step-5, the *Task Allocation* subroutine is invoked which returns the ideal replica of the best-fit task suitable for that volunteer. This holds true for all volunteers in A. If any t is still incomplete, the client or the task requester is notified, and searching outside set A is initiated.

Algorithm 1 Initial Volunteer-Task Mapping (i-VTM)

```
Input: Set of interested applicants $A$, set of tasks $T = \{t_1, t_2, ...t_n\}$
Output: Allotment dictionary $Map$
1:  Start
2:  Generate Skill-Task Mapper matrix $G_{ST}$
3:  for $c \in A$ do
4:      $\omega \leftarrow argmax\ (|Q_c|)$ {Select the applicant volunteer who has the highest
        number of skills}
5:      $t'$=Task Allocation($\omega, T$)
6:      Update $Map, G_{ST}$
7:  end for
8:  if any $t \in T$ has some existing skill requirements or is incomplete then
9:      Notify the client
10:     Initiate search outside $A$
11: end if
12: return $Map$
13: End
```

G_{ST} and G_{SM} are constructed inside the Task Allocation subroutine (refer Algorithm-2). G_{ST} or *Skill-Task Mapper* is used for storing skills per task requirement and is implemented using a 2D matrix. For constructing the matrix, all replicas of all posted tasks are taken together to represent the columns and are considered as general tasks (denoted as T'). Thus, if combining all the n tasks in T', there are in total m distinct skills, then the size of G_{ST} will be $m \times n$. The entry (x, y) is set to one if and only if, a skill x is required by any task y;

Algorithm 2 Task Allocation

```
Input: volunteer $\omega$, Set of Tasks $T = \{t_1, t_2, .., t_n\}$
Output: Allocated Task $t'$
    Initialization: Initialize list of recommended tasks $P \leftarrow \varnothing$, $Reco \leftarrow \varnothing$
1:  Start
2:  $G_{SM} \leftarrow$ Generate Skill-Volunteer Mapper for $\omega$
3:  $col\_sum \leftarrow$ Column-wise sum of $G_{SM}$
4:  $col\_max\_sum \leftarrow$ Max($col\_sum$)
5:  for $i \in |col\_sum|$ do
6:      if $col\_sum[i] == col\_max\_sum$ then
7:          Append $i$ to $P$ {Recommend all tasks in T who match the condition}
8:      end if
9:  end for
10: for each i $\in$ P do
11:     $r = argmin\ Dist(\omega, replica\ of\ i)$ {Using any distance formulation metric}
12:     Append $r$ to $Reco$
13: end for
14: $t' \leftarrow argmin\ Reco$
15: return $t'$
16: End
```

else zero. G_{ST} is necessary for maintaining current knowledge of the status of a task's skill requirement coverage, and it is dynamically modified after each allocation. The first step of Algorithm-1 is to generate this G_{ST}. To clarify, an example is presented through Table-I and Table-II.

TABLE I: Details of the Tasks and their Replicas

Tasks	Skill requirements	Replicas	Locations
t_1	Photography, Cartographer, Planning knowledge	t_{11}, t_{12}, t_{13}	(1,2), (3,5), (5,5)
t_2	Food critic, Nutritionist, Chef	t_{21}, t_{22}	(5,8), (3,1)
t_3	Photography, Cartographer, Sport enthusiast	t_{31}	(6,10)

TABLE II: Details of the volunteers

Volunteers	Skill-set	Current Location	Preferred Slot
c_1	Cartographer, Guitarist, Photographer, Chef	(3,7)	12-15 hr
c_2	Food critic, Nutritionist	(1,1)	12-14 hr
c_3	Photography, Cartographer, Sport enthusiast	(6,4)	11-15 hr
c_4	Planning knowledge, Cartographer	(3,3)	13-17 hr
c_5	Chef, Cartographer, Driving	(1,5)	9-11 hr
c_6	Driving, Chef	(7,7)	14-16 hr

G_{SM} or *Skill-Volunteer Mapper* is for matching each volunteer's skills to that of the requirements of tasks. G_{SM} is also implemented by a 2D matrix called. If a volunteer has m skills, the G_{SM} for n tasks will be $m \times n$ in size. If and only if skill x is required by task y, the entry (x, y) is set to one; otherwise, zero. Any skill a volunteer has that is not in any of the posted tasks lists is not taken into account. Moreover, construction of G_{SM} is straight forward. In fact, G_{SM} is a sub-graph of G_{ST}. The details of volunteers exemplified in the Table-II are used throughout the paper. The corresponding G_{SM} of c_1 is:

$$\begin{array}{cc} \begin{array}{cccccc} t_{11} & t_{12} & t_{13} & t_{21} & t_{22} & t_{31} \end{array} & \\ \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} & \begin{array}{c} \textit{Photography} \\ \textit{Cartography} \\ \textit{Chef} \end{array} \end{array}$$

According to the Algorithm-2, the first volunteer to be chosen is c_1. Then, G_{SM} of c_1 is generated and *column-wise sum* of G_{SM} is manipulated. There is no need to update G_{SM}

because once an applicant is assigned to any replica, he or she is removed from the decision process for the particular assignment cycle. In steps 11-13, for every recommended tasks (here both t_1 and t_3), its corresponding nearest replica is selected and is appended to a temporary list *Reco*. The decision is made depending on distance between C_1's current location and that of the replicas. After this, the closest among nominated replicas (t_{12}) in *Reco* is returned. The t_{12} is a replica of t_1 and is located at position $(3,5)$ and is at 2 unit distance away from C_1, following *Euclidean distance*. Then, *Map* is updated in i-VTM. The contributed skills of c_1 are *Photography and Cartography*. This does not necessarily imply that the requirements of t_1 are entirely satisfied. It signifies that the requirements of t_1's replica t_{12} for only these two skills are covered. The respective cell entries of G_{SM} are made zero. The resultant modified G_{ST} is:

$$\begin{array}{cccccccc}
 & t_{11} & t_{12} & t_{13} & t_{21} & t_{22} & t_{31} & \\
\left(\right. & 1 & \overset{0}{\cancel{1}} & 1 & 0 & 0 & 1 & \left.\right) \quad Photography \\
 & 1 & \overset{0}{\cancel{1}} & 1 & 0 & 0 & 1 & Cartography \\
 & 1 & 1 & 1 & 0 & 0 & 0 & Planning\,Know \\
 & 0 & 0 & 0 & 1 & 1 & 0 & Food\,Critic \\
 & 0 & 0 & 0 & 1 & 1 & 0 & Nutritionist \\
 & 0 & 0 & 0 & 1 & 1 & 0 & Chef \\
 & 0 & 0 & 0 & 0 & 0 & 1 & Sport\,Enthusiast
\end{array}$$

Theorem 2. *The proposed solution for VTM problem has a soft lower-bound*

Proof. A task t is considered satisfied if and only if all its skills requirements are met by the volunteers (C_t) assigned to t that is $Q_t \subseteq \bigcup_{c \in C_t} Q_c$. The soft lower bound is set on the task assignment because it is impossible to determine whether there is a 100% success rate due to the greedy nature of the solution. However, we assure that in order to declare our given assignment decision successful for task t, it should at least as good as covering the skill requirements of t in order. The idea has been derived from the papers [18], [19]. □

B. *Adaptive Common Slot Finding Algorithm*

This algorithm (refer Algorithm-3) defines the second phase of our framework. The *Map* from the previous phase is used as the input here. The final result is a list of common slots, *cs*. The classic *prefix sum calculation* is combined with a novel self-adjusting satisfaction threshold strategy to determine the most advantageous common working time slot for the volunteers who cooperated for any given task's replica. Note that only the successfully assigned tasks and volunteers (presented in Map) in i-VTM phase are passed to the second phase.

First, the preferred slots' HH:MM time format is transformed to MM:MM for precise manipulation and saved in *pref_slots_minutes*. Next, an array *time_slot_minutes* is initialized to contain the total number of minutes in a day beginning at 12 AM (0000) and ending at 11:59 PM (1439). The |*time_slot_minutes*| array shall always be bounded and equal to 1,440. Then, the start (+1) and end (-1) time boundaries of each *slot* in *pref_slots_minutes* are marked. Doing the prefix sum calculation on *time_slot_minutes* array, the count of slots in which the minutes appeared is computed.

Algorithm 3 Adaptive Common Slot Finding (a-CSF)

```
Input: Allotment dictionary Map
Output: List of recommended common slots cs with respective volunteers count
    Initialization: Initialize self-adjusting satisfaction threshold $\theta = 0.5$, List n_thresh ← $\varnothing$
1: Start
2: pref_slots_hours=Lists the slots of all volunteers in Map, each in the form [start, end]
3: pref_slots_minutes= Convert each slot in pref_slot_hours in minutes {Convert HH:MM to MMMM format}
4: for i=0 to (24*60+1) do
5:     Initialize time_slot_minutes[i] = 0 {Account for all the minutes present in 24 hour}
6: end for
7: for slot in pref_slots_minutes do
8:     time_slot_minutes[slot[0]]+ = 1 {For each slot just mark the start marker i.e +=1 and the end marker i.e -1}
9:     if slot[1] + 1 < length(time slot minutes) then
10:        time_slot_minutes[slot[0]]− = 1
11:    end if
12: end for
13: Find the prefix sum of every element in time_slot_minutes array
14: n_thresh= Threshold Volunteers Count (length(pref_slots_minutes)) {Number of threshold volunteers to look for}
15: for $\theta$ in n_thresh do
16:    cs = Final Slot Computation Algorithm(time_slot_minutes, $\theta$) {Get the recommended common slots for every given threshold count}
17:    if $cs_{end} - cs_{start} + 1 < 60$_minutes then
18:        Stretch $cs_{start}$ and $cs_{end}$ to 60 minutes each.
19:    end if
20: end for
21: return cs
```

Algorithm 4 Threshold Volunteers Counting

```
Input: length of preferred slots l
Output: Set of the number of volunteers of every threshold generated from 1 to 0.5, n_thresh
1: start
2: $\theta = 1$
3: while $\theta >= 0.5$ do
4:     Add ceil($l * \theta$) to n_thresh
5:     $\theta - = 0.1$
6: end while
7: return n_thresh
8: End
```

In step 14, the *Threshold Volunteers Count* method is used to set the lower bound of *n_thresh*. The element $\theta \in n\_thresh$ is called *self-adjusting satisfaction threshold*. The *n_thresh* is a non-increasing array whose 0^{th} index is always 1 and the last index is 0.5 (the lower bound in our setting). This means that a-CSF should first try to search for a slot to satisfy 100% of the volunteers and gradually lower the bar till its threshold meets the pre-defined lower bound of 50%. This accounts for adaptability of a-CSF. The *Final Slot Computation* method invoked at line 16 compute the slots for every threshold counts. Given, A is the number of applicants and T' is the number of all the replicas of all available tasks in the VCS platform, the i-VTM runs in $O(|A| \times |T'|)$ time and the a-CSF runs in $O(|A'|)$, where $|A'|$ is the number of volunteers qualified in phase one for a specific tasks' replica.

Algorithm 5 Final Slot Computation

```
Input: List of time slots in minutes $timeslot\,minutes$, List of calculated thresholds $thresh$
Output: Common slot $slot$
1: start
2: $start = -1, end = -1$ {Stores the final common slot having maximum duration}
3: $i = -1, j = -1$ {Pointers to mark the current maximum length slot}
4: $curMaxDur = 0$ {To compare the next valid slot with previous slot of maximum duration}
5: while $i < length(timeslot\,minutes)$ do
6:     if $timeslot\,minutes[i] \geq thresh$ then
7:         $j = i + 1$
8:         while $j < length(timeslot\,minutes)$ and $timeslot\,minutes[j] \geq thresh$ do
9:             $j = j + 1$
10:        end while
11:        if $j - i > curMaxDur$ then
12:            $start = i$, $end = j - 1$, $curMaxDur = j - i$
13:        end if
14:        if $j < curMaxDur$ then
15:            $i = j$
16:        else
17:            break
18:        end if
19:    else
20:        $i = i + 1$
21:    end if
22: end while
23: $slot = [start, end]$
24: return $slot, thresh$
25: End
```

V. Performance Evaluation

We conducted simulations using an Intel i3 dualcore CPU running at 2GHz, 4GB of RAM, and Windows 10. The code was written and compiled in Python.

A. Dataset

On Feb 26, 2021, data was crawled from UpWork [2] for experimental purposes. The locations are produced synthetically using a random distribution, such that each entity is spatially constrained within a $[50 \times 50]$ 2D Cartesian plane. The volunteers' chosen time slots are generated randomly. All starting times lies within $[00:00, 20:59]$ in accordance with the HH:MM norm. All volunteer slots are assumed to be 3 hours long,. The end time should not exceed 11:59 p.m. The task's category contains the task's name, ID, skill requirements, and XY positions. The features of candidates include their names, ID, preferred time slots, and XY locations.

TABLE III: Parameter Settings

Parameters	Settings
Number of tasks	97
Number of volunteers	250 to 1575
Average number of replicas per task	3
Average Skill count per volunteer	7
Average skills demanded per task	10
Average completion time of task	7.5 hours

B. Performance Comparison

The following three works serve as benchmarks for evaluating our framework's performance: (1) *Online Greedy (**OG**) [3]:* It is also greedy and allocates spatial tasks online based on multi-skill needs. The time complexity of OG is $O(|C| + |T|)^2$. (2) *Group Matching Algorithm (**GMA**) [9]:* It is also greedy based, considers distance factor between any task and candidate into account and follows one-to-many task allocation in offline mode. (3) ***TM-Uniform* [10]:** It allocates tasks to applicants maintaining budget sustainability. It also takes into account the skill needs of tasks, but in a different way than our more common greedy method. We integrated the proposed *a-CSF* phase into the baselines to show the performance comparisons of the proposed *i-VTM* method. To our knowledge, this is the first study to recommend a common working slot in the context of crowdsourcing that assures at least θ participants' preferences are met.

Performance metrics: The performance metrics for evaluating the algorithms are:

(1) *Net utility*: For a single pair, utility is computed using the Formula-1. The net utility is supplied by all selected (task, volunteer) pairs at the conclusion of a given assignment cycle.

(2) *Success rate*: It quantifies the mean proportion of allotted volunteers whose involvement results in the completion of all available tasks. Assuming, the present tasks in the system is $T = \{t_1, t_2, ..t_{|T|}\}$ and their corresponding number of replicas is $R = \{r_1, r_2, .., r_{|T|}\}$ such that t_i has total r_i replicas distributed at different locations. The number of successful allocation count for the replicas are given as $S = \{s_1, s_2, ...s_{|R|}\}$. This means out of r_i replicas of task t_i, s_i are allocated. Thus, success rate is calculated using the following formula: $\frac{\sum_{i=1}^{|R|} s_i}{\sum_{i=1}^{|T|} r_i} \times 100$.

(3) *Average waiting Time*: It specifies the average time spent by the tasks waiting to be assigned in an assignment cycle and is expressed in seconds.

(4) *Overall task completion rate*: It is computed for each task as a whole considering all the replicas and is calculated using the formula: $\frac{\sum_{i=1}^{|R|} (\frac{s_i}{r_i})}{\sum_{i=1}^{|T|} t_i} \times 100$.

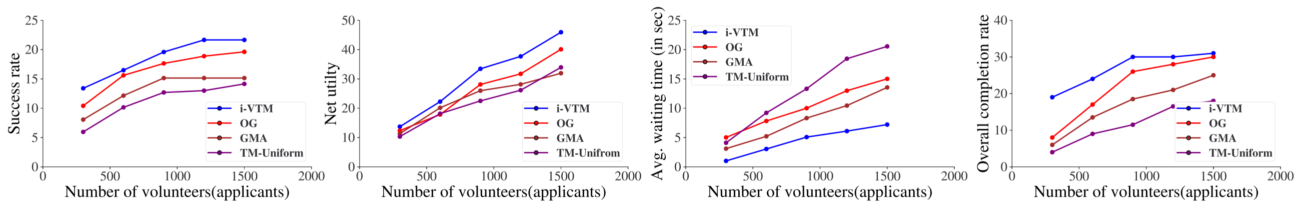
(5) *Total satisfactory rate*: It is determined as per the final computation of common slot and comparing the same with the actual preferred slot given by the applicants. It is calculated by using the formula: $\frac{\sum_{j=1}^{|R|} satisfaction\_score_j \times |A'|}{|A|} \times 100$, where $satisfaction\_score$ of all selected volunteers V assigned to any replica r is determined as $satisfaction\_score_r = \sum_{v=1}^{|V|} (length(slot_v) \cap length(cs_r))$. This metric is used to test the performance of the a-CSF algorithm by capturing how each recommended common working time slot for a replica is covering the selected volunteers' given time slot preferences.

The first four metrics (net utility, success rate, average waiting time, and overall task completion rate) are used to observe the performance between the proposed framework's i-VTM (phase-1) and the three baselines, based on the respective task assignment strategy. The *Total satisfactory rate* is utilised to compare the effectiveness of our framework (phases 1 and 2) to that of the baselines integrated with a-CSF module.

C. Experimental Results

***Effect on success rate*:** Our proposed i-VTM approach has average success rate is 18.6%, the second-best provider is OG which gives average of 16% (Figure-2(a)). The respective values for the rest are 13% (GMA) and 11%(TM-Uniform). Our method establishes a preliminary match based on task skill requirements and volunteer skill availability. From the filtered tasks and replicas, a volunteer gets the spatially nearest replica.

***Effect on net utility*:** The i-VTM, OG, GMA, and TM-Uniform have average net utilities of 31, 26, 23 and 22 units respectively (Figure-2(b)). They all try to map volunteers by



(a) (b) (c) (d)

Fig. 2: Illustration of (a) success rate, (b) net utility, (c) average waiting times and (d) overall completion rate

skills and vicinity. Our method looks for volunteers who can not only provide the highest skill coverage at a given time but also have the highest skill set among all compatible volunteers, so they can be provisionally matched to more tasks.

***Effect on average waiting time*:** Figure-2(c) shows that our i-VTM task allocation algorithm (phase-1) is extremely fast, taking only 5 seconds on average, while others take 10 (OG), 8 (GMA) and 13 (TM-Uniform) seconds.

***Effect on overall task completion rate*:** If a task has x replicas and all of them have been successfully assigned to ensure that all of their skill criteria are satisfied, the task's completion rate is 100%. From Figure-2(d), overall completion rate of i-VTM is high which is on average is 27%. OG is the second lead 22%. Others are 16% (GMA) and 12% (TM-Uniform).

***Effect on total satisfactory rate*:** From Figure-3, it is noted that our proposed framework (i-VTM with a-CSF) gives an average satisfactory rate of 86%. It is noted that a-CSF does have a positive impact on increasing the satisfactory gain of the assigned volunteers from the task allocation phase.

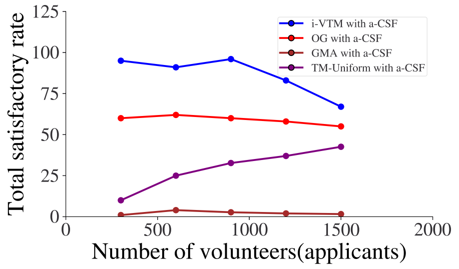


Fig. 3: Total satisfactory rate of the assigned volunteers

VI. Conclusion

In this work, we proposed a two-phase framework consisting of Initial Volunteer-Task Mapping (i-VTM) and Adaptive Common Slot Finding (a-CSF) algorithms. The *i-VTM* algorithm is for assigning volunteers to tasks based on their skills and spatial proximity and *a-CSF* algorithm is for recommending appropriate common working time slots to aid in successful volunteer collaboration. The UpWork dataset proves the efficiency of our strategy with respect to the existing state-of-the-art methods. We intend to further explore this work by experimenting in an both-way online setting. We have also planned to look into the effects of the participating volunteers' latent potential levels and willingness on task assignment.

References

[1] "Amazon Mechanical Turk," accessed: 2021-04-29. [Online]. Available: https://www.mturk.com/

[2] "In-demand talent on demand.™ Upwork is how.™," accessed: 2021-03-26. [Online]. Available: https://www.upwork.com/

[3] T. Song, K. Xu, J. Li, Y. Li, and Y. Tong, "Multi-skill aware task assignment in real-time spatial crowdsourcing," *GeoInformatica*, vol. 24, no. 1, pp. 153–173, 2020.

[4] P. Cheng, X. Lian, L. Chen, J. Han, and J. Zhao, "Task assignment on multi-skill oriented spatial crowdsourcing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 8, pp. 2201–2215, 2016.

[5] R. Samanta, S. Ghosh, and S. K. Das, "SWill-TAC: skill-oriented dynamic task allocation with willingness for complex job in crowdsourcing," in *2021 IEEE Global Communications Conference: Selected Areas in Communications: Social Networks (Globecom2021 SAC SN)*, Madrid, Spain, Dec. 2021.

[6] Y. Tong, Z. Zhou, Y. Zeng, L. Chen, and C. Shahabi, "Spatial crowdsourcing: a survey," *The VLDB Journal*, vol. 29, no. 1, pp. 217–250, 2020.

[7] S. Wu, X. Gao, F. Wu, and G. Chen, "A Constant-Factor Approximation for Bounded Task Allocation Problem in Crowdsourcing," *2017 IEEE Global Communications Conference, GLOBECOM 2017 - Proceedings*, vol. 2018-January, pp. 1–6, 2017.

[8] J. Jiang, Y. Zhou, Y. Jiang, Z. Bu, and J. Cao, "Batch allocation for decomposition-based complex task crowdsourcing e-markets in social networks," *Knowledge-Based Systems*, vol. 194, p. 105522, 2020. [Online]. Available: https://doi.org/10.1016/j.knosys.2020.105522

[9] J. Liu, H. Zhu, and X. Chen, "Complicated-skills-based task assignment in spatial crowdsourcing," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9998 LNCS, pp. 211–223, 2016.

[10] G. Goel, A. Nikzad, and A. Singla, "Matching Workers Expertise with Tasks: Incentives in Heterogeneous Crowdsourcing Markets," *Proceedings of NIPS '13 Workshop on Crowdsourcing*, pp. 1–15, 2014.

[11] R. Samanta and S. K. Ghosh, "FogiRecruiter : A fog-enabled selection mechanism of crowdsourcing for disaster management," *Concurrency and Computation: Practice and Experience*, pp. 1–9. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.7207

[12] A. Hamrouni, H. Ghazzai, T. Alelyani, and Y. Massoud, "Optimal team recruitment strategies for collaborative mobile crowdsourcing systems," in *2020 IEEE Technology & Engineering Management Conference (TEMSCON)*. IEEE, 2020, pp. 1–6.

[13] I. Illahi, H. Liu, Q. Umer, and S. A. H. Zaidi, "An Study on Competitive Crowdsource Software Development: Motivating and Inhibiting Factors," *IEEE Access*, vol. 7, pp. 62 042–62 057, 2019.

[14] D. Yu, Z. Zhou, and Y. I. Wang, "Crowdsourcing Software Task Assignment Method for Collaborative Development," *IEEE Access*, 2019. [Online]. Available: http://www.ieee.org/publications_standards/publications/rights/index.html

[15] T. Luo, S. K. Das, H. P. Tan, and L. Xia, "Incentive mechanism design for crowdsourcing: An all-pay auction approach," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 7, no. 3, pp. 1–26, 2016.

[16] F. Restuccia, P. Ferraro, S. Silvestri, S. K. Das, and G. L. Re, "Incentme: Effective mechanism design to stimulate crowdsensing participants with uncertain mobility," *IEEE Transactions on Mobile Computing*, vol. 18, no. 7, pp. 1571–1584, 2018.

[17] S. S. Skiena, *The algorithm design manual*. Springer, 1998, vol. 2.

[18] X. Yin, Y. Chen, C. Xu, S. Yu, and B. Li, "Matchmaker: Stable Task Assignment with Bounded Constraints for Crowdsourcing Platforms," *IEEE Internet of Things Journal*, vol. 8, no. 3, pp. 1599–1610, 2021.

[19] Y. Chen and X. Yin, "Stable Job Assignment for Crowdsourcing," *2017 IEEE Global Communications Conference, GLOBECOM 2017 - Proceedings*, vol. 2018-Janua, pp. 1–6, 2017.