Compact Scheduling for Task Graph Oriented Mobile Crowdsourcing

Liang Wang, Zhiwen Yu, Qi Han, Dingqi Yang, Shirui Pan, Yuan Yao, Daqing Zhang

Compact Scheduling for Task Graph Oriented Mobile Crowdsourcing

Liang Wang, Zhiwen Yu, *Senior Member, IEEE,* Qi Han, Dingqi Yang, Shirui Pan, Yuan Yao, and Daqing Zhang, *Fellow, IEEE,*

Abstract—With the proliferation of increasingly powerful mobile devices and wireless networks, mobile crowdsourcing has emerged as a novel service paradigm. It enables crowd workers to take over outsourced location-dependent tasks, and has attracted much attention from both research communities and industries. In this paper, we consider a mobile crowdsourcing scenario, where a mobile crowdsourcing task is too complex (e.g., post-earthquake recovery, citywide package delivery) but can be divided into a number of easier subtasks, which have interdependency between them. Under this scenario, we investigate an important problem, namely *task graph scheduling in mobile crowdsourcing* (TGS-MC), which seeks to optimize a compact scheduling, such that the task completion time (i.e., makespan) and overall idle time are simultaneously minimized with the consideration of worker reliability. We analyze the complexity and NP-complete of the TGS-MC problem, and propose two heuristic approaches, including BFS-based dynamic priority scheduling *BFSPriD* algorithm, and an evolutionary multitasking-based *EMTTSch* algorithm, to solve our problem from local and global optimization perspective, respectively. We conduct extensive evaluation using two real-world data sets, and demonstrate superiority of our proposed approaches.

Index Terms—Mobile Crowdsourcing, Task Schedule, Directed Acyclic Graph(DAG), Makespan.

1 Introduction

With the increasing popularity of smart mobile devices and wireless communication network, *Mobile Crowdsourcing* (MC) has emerged that utilizes the power of people crowds to complete traditionally time-consuming or costly tasks [1], [2]. Specifically, a typical mobile crowdsourcing platform schedules spatial tasks released by crowdsourcers to a group of participant workers, and workers are required to physically move to some specified spatial locations and conduct these tasks. Recently, mobile crowdsourcing has spurred a wide interest from both academia and industry [3], [4]. Many MC applications and systems have also been developed, e.g., Gigwalk, TaskRabbit, etc.

Nowadays, more and more complex MC tasks have been published and conducted on mobile crowdsourcing platforms, e.g., searching for a lost child, post-disaster recovery, citywide package delivery [5], [6], [7], [8]. Different from simple MC tasks such as taking a photo or checking street signs, it is impossible for just one worker to independently accomplish a complex task. In practice, one common practice is to decompose a complex task into a set of easier subtasks [9], [10], [11], and recommend these divided subtasks to a group of workers, by the coordination between the subtasks' requirements and the workers' skill/resource conditions. Subsequently, a complex MC task can be completed upon the completion of all these subtasks.

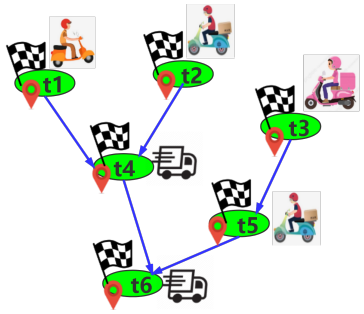


Fig. 1. Toy Example of A Task Graph in MC.

Interestingly, in many cases, the divided subtasks are not independent, but highly interdependent [12], [13]. For instance, in post-earthquake recovery [5], [6], temporary shelters should be built after safety and environment assessment; rescue facilities must be in place before the search-and-rescue activities. In crowdsourcing citywide delivery [7], [8], one package may be relayed to its destination by more than one worker, where the successive relays can be regarded as dependent subtasks. Fig. 1 illustrates a toy example, the delivery vehicle at t_4 will not drive away until both the packages from t_1 and t_2 have been collected. Inspired by the above examples, in this paper, we consider a complex MC task with precedence-constrained subtasks, where one subtask can not be started until one or more relevant subtasks have been completed.

It is obvious that dependencies among subtasks make MC task scheduling significantly different from related prior

- *L. Wang, Z. Yu and Y. Yao are with School of Computer Science, Northwestern Polytechnical University, China, 710129.*
- *Q. Han is with the Colorado School of Mines, Golden, CO, USA.*
- *D. Yang is with University of Macau, Macau, China.*
- *S. Pan is with Centre for Artificial Intelligence, University of Technology Sydney, Australia.*
- *D. Zhang is with SAMOVAR Lab, TELECOM SudPairs, Evry, France and Peking University, China.*
- **L. Wang is the corresponding author, Email: liangwang0123@gmail.com.*

Manuscript received April 19, 2005; revised August 26, 2015.

work [14], [15], [19]. Specifically, in addition to meeting the specific requirements of individual subtasks (spatial location, skill requirements, etc.), it is indispensable to consider the explicit precedence relationships among them [20]. Actually, precedence-constrained tasks are common in multi-core computing and distributed computing [21], [22], [23]. The interdependent subtasks are usually characterized with the tool of task graph, i.e., Directed Acyclic Graphs (DAGs), where nodes represent the involved computing subtasks, and edges represent dependency constraints. And many DAG task scheduling techniques have also been developed, including list-based, duplication-based scheduling approaches. However, these techniques cannot be directly adopted to our MC scheduling problem, as they fail to consider the spatial constraints in mobile crowdsourcing. More precisely, during the process of task execution, for each candidate worker, its relevant task implementation cost is not fixed, due to the fact that worker's spatial position may vary from location to location.

Until recently, many MC task scheduling techniques have been proposed for different scenarios, but few specifically address the precedence constraints as mentioned above. To this end, in the work, we investigate a scheduling problem for task graph oriented mobile crowdsourcing (**TGS-MC**), which assigns the interdependent subtasks to available workers, with the goal of simultaneously minimizing task completion time, i.e., makespan, and idle time. In other words, our goal is to find optimal subtask-worker assignments to achieve "***compact scheduling performance***", which has not only minimum task completion but also high throughput of MC systems with efficient worker resource utilization. What is more, to guarantee high quality results, the reliability of candidate workers is considered.

However, solving our TGS-MC problem faces following challenges. First, as a NP-complete problem, it is computationally intractable to obtain an optimal scheduling solution in polynomial time. Thus, we need to design a computationally efficient approaches to explore near-optimal results. Second, due to the precedence and spatial constraints, we need to systematically consider all involved subtasks and available workers, and more importantly, capture and track workers' varying locations during scheduling process. Third, in TGS-MC problem, we seek a compact task schedule with not only minimum makespan but also minimum idle time. As a result, it is necessary to explore a trade-off between these two objectives.

To tackle it, we formalize TGS-MC problem with a dual-objective optimization problem. By analyzing its complexity, we draw inspiration from the domain of DAG task scheduling, to find quasi-optimal assignments among all the involved subtasks and available workers. Two heuristic algorithms, including a breadth-first search based *BFSPriD* algorithm, and an evolutionary multitasking-based *EMTTSch* algorithm, are proposed to solve our TGS-MC problem from local and global optimization perspective, respectively. To be specific, driven by an integrated optimization objective, *BFSPriD* algorithm sequentially and locally construct scheduling solution layer by layer, by following a breadth-first-search pattern. By utilizing knowledge transfer between makespan and idle time objective, two independent evolutionary solvers are built in *EMTTSch* algorithm to discover promising solutions throughout their respective problem spaces globally. In other words, *BFSPriD* makes a trade off during optimization process; while *EMTTSch* individually optimizes these two objectives in optimization process, and makes trade off at the end. Specifically, we make the following contributions.

TABLE 1
Definitions of Notations

Symbol	Explanation
$\mathcal{W} = \{w_j\}$	Mobile Workers
$\mathcal{T} = \{t_i\}$	MC Complex Task
$G = (\mathcal{T}, E)$	$\mathcal{T}$'s Task Graph
$\mathcal{S}$	$\mathcal{T}$'s Required Skills/Resources Set
$Ly(t)$	Subtask t's Layer in G
$\varepsilon(t, w)$	Matching Degree between t and w
$P_r(t)$	Subtask t's Process Ready Time
$P_s(t, w)$	Subtask t's Startup Time for w
$P_f(t, w)$	Subtask t's Completion Time for w
$IT(t, w)$	Subtask t's Idle Time for w
$MS(\mathcal{T})$	$\mathcal{T}$'s Makespan

- To the best of our knowledge, this work is the first to propose and formalize an important problem in MC systems, namely task graph scheduling in MC, i.e., TGS-MC problem.
- From the perspective of local and global optimization, we propose two effective heuristic approaches, respectively, namely *BFSPriD* and *EMTTSch* algorithm, to solve our TGS-MC problem.
- We extensively evaluate our proposed approaches using two real-world datasets, and show the efficiency and effectiveness of our proposed approaches.

The rest of this paper is organized as follows. Section 2 presents the preliminary, formulates TGS-MC problem and analyze its complexity. Two problem-solving approaches are proposed in Section 3, including *BFSPriD* algorithm and *EMTTSch* algorithm. Section 4 provides the extensive simulation results based on two real-world datasets. Related work is summarized in Section 5. Finally, we conclude this paper in Section 6.

2 Preliminary and Problem Formulation

To clearly illustrate the research problem, in this section, we first introduce several important concepts of task graph in MC systems, and then formally define our **TGS-MC** problem and discuss its complexity. A list of notations and their definitions used in the paper is summarized in Table 1.

2.1 Preliminary

Definition 1 (Complex Task). Generally, a complex task $\mathcal{T}$ in MC systems can not be accomplished by just one worker on his own, but needs a group of workers to collaborate together. In practice, $\mathcal{T}$ is divided into a number of easier subtasks which usually require less workload (cognition, resource and so on), i.e., $\mathcal{T} = \{t_i | 1 \leqslant i \leqslant n\}$. Task $\mathcal{T}$

is regarded as successfully completed if and only if all subtasks are accomplished. □

Definition 2 (Subtask). Subtask $t \in \mathcal{T}$ is an easier work that can be handled autonomously by workers with less workload. Formally, t is represented as $t = (loc_t, S_t)$, where $loc_t \in \mathcal{L}$ denotes its specified location, and $S_t \in \mathcal{S}$ is the set of required skills/resources to accomplish it. In practice, the requirements of each subtask are diverse and heterogeneous, for instance, painting, photography, gyroscope, etc. □

In this work, the subtasks we considered are interdependent with each other. Concretely, there exist **precedence constraints among these subtasks**. For instance, subtask t_j can not be started until its predecessor subtask t_i has been finished. Here, we adopt task graph tool, i.e., Directed Acyclic Graph (DAG), to capture such precedence constraints.

Definition 3 (Task Graph). Given a complex task $\mathcal{T} = \{t_i | 1 \leqslant i \leqslant n\}$, one task graph $G = (\mathcal{T}, E)$ is used to characterize the precedence among its subtasks, where $\mathcal{T}$ represents the node set, and each edge in E represents precedence constraint associated with the two involved nodes, i.e., subtasks. For example, one edge $t_i \rightarrow t_j$ indicates that t_i is the immediate predecessor of t_j, i.e., t_j is t_i's successor subtask. □

Hereafter, we will use the terms "subtask" and "node" interchangeably. Fig. 2 gives an example of a task graph which contains 10 subtasks and 10 precedence links. For a task graph $G = (\mathcal{T}, E)$, the nodes without any predecessors is named as **Entry Node** $\mathcal{T}_{\texttt{Entry}}$, whereas a node without any successor nodes is called an **Exit Node** $\mathcal{T}_{\texttt{Exit}}$. Logically, task $\mathcal{T}$ can be claimed to be completed only when all its exit nodes have been accomplished. For one node t_i in G, its **Predecessor** and **Successor** nodes can be formalized as blow:

$$\begin{aligned} \mathcal{T}_{\texttt{pre}}(t_i) &= \{t_j | t_j \in \mathcal{T} \wedge (t_j \rightarrow t_i) \in E\} \\ \mathcal{T}_{\texttt{suc}}(t_i) &= \{t_j | t_j \in \mathcal{T} \wedge (t_i \rightarrow t_j) \in E\}, \end{aligned} \tag{1}$$

where $\mathcal{T}_{\texttt{pre}}(t_i)$ and $\mathcal{T}_{\texttt{suc}}(t_i)$ denote subtask t_i's predecessor and successor subtasks, respectively. For example, in Fig. 2, subtask t_2's predecessor and successor node sets are $\mathcal{T}_{\texttt{pre}}(t_2) = \{t_1\}$ and $\mathcal{T}_{\texttt{suc}}(t_2) = \{t_5, t_6\}$.

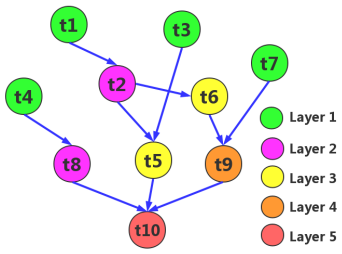


Fig. 2. A Toy Example of Task Graph.

Definition 4 (Layer in Task Graph). Due to precedence constraints in task graph G, subtask scheduling and execution should not be random. Instead, it should have an ordered sequence. By traversing task graph G, we analyze each node's predecessor nodes, and represent this sequential relationship with the layer of each subtask. Formally, the layer is given as below:

$$Ly(t_i) = \begin{cases} 1, & if\ \mathcal{T}_{\texttt{pre}}(t_i) = \varnothing, \\ \underset{t_j \in \mathcal{T}_{\texttt{pre}}(t_i)}{Max} \{Ly(t_j)\} + 1, & Otherwise. \end{cases} \tag{2}$$

Clearly, the smaller the layer value, the earlier it can be executed, as it has fewer associated precedence constraints. Fig. 2 lists the layer value of each subtask.

Definition 5 (Workers). There is a set of available workers $\mathcal{W} = \{w_1, w_2, ..., w_m\}$ registered on MC platform to receive and perform tasks/subtasks. Each worker is denoted as $w_i = (loc_{w_i}, v_i, S_{w_i})$, where (1) spatial location $loc_{w_i} \in \mathcal{L}$ is w_i's current position; (2) v_i denotes w_i's moving speed; (3) S_{w_i} is the skills/resources possessed by the worker. □

Considering a complete set of skills/resources, such as $\mathcal{S} = \{s_1, s_2, ..., s_q\}$, one worker w's skill/resource profile could be denoted as $S_{w_i} = \{s^1_{w_i}, s^2_{w_i}, ..., s^q_{w_i}\}$, where $s^k_{w_i}, 1 \leqslant k \leqslant q$, denotes the skill/resource level, and $s^k_{w_i} \in [0, 1]$. For instance, when $s^k_{w_i}$ equals to 0, it means that w_i does not have skill s_k; when it equals to 1, the worker is very good at s_k. In practice, such level information could be subscribed by the worker himself, or learned from his participant history. Identifying such information might be an interesting challenge, but it is out of the scope of this paper, and we will leave it for our future work.

By harnessing MC platform, each node contained in $\mathcal{T}$ can be dispatched to available workers. However, given required skills/resources in subtask t, not all the available workers are qualified to complete it. In other words, only the workers whose skills/resources can completely cover t's requirements are supposed to be valid candidates $\mathcal{W}_t$. Hence, we need firstly to identify the candidate workers $\mathcal{W}_t$, via a matching between t's resource/skill requirements and workers' profile. Similar to the representation of worker's skill/resource profile, the required skills/resources in subtask t are also represented as $S_t = \{s^1_t, s^2_t, ..., s^q_t\}$, where $s^k_t \in [0, 1], 1 \leqslant k \leqslant q$, indicating skill/resource s_k is necessary for completing t, and the minimum requirement level is s^k_t. Formally, the valid candidate workers for node t could be formalized as below:

$$\mathcal{W}_t = \{w \mid \forall s^k_t > 0, \exists s^k_w \geqslant s^k_t, 1 \leqslant k \leqslant q\}, \tag{3}$$

where $s^k_w \in S_w$ denotes worker w's skill/resource level, and $s^k_t \in S_t$ denotes subtask t's skill/resource requirement, $1 \leqslant k \leqslant q$.

Nevertheless, due to the diverse requirements in subtasks, the candidate workers derived from Eq. 3 are usually sparse, especially for large skill/resource dimensions. In that case, the candidate workers might be not enough for effective task scheduling. Given this, by applying a lower bound threshold ε_{thre}, we loose the matching condition above. Mathematically, for subtask $t = \{loc_t, S_t\}$ and candidate worker $w = \{loc_w, v, S_w\}$, the matching degree $\varepsilon(t, w)$ can be determined as follows:

$$\varepsilon(t, w) = \frac{1}{\|S_t\|_0} \sum_{s^k_t \in S_t \wedge s^k_t > 0} \Theta(s^k_t, s^k_w) \tag{4}$$

where

$$\Theta(s^k_t, s^k_w) = \begin{cases} 1, & s^k_t \leqslant s^k_w \ \wedge \ s^k_t > 0 \\ s^k_w, & s^k_t > s^k_w \ \wedge \ s^k_t > 0, \end{cases} \tag{5}$$

and L0 regularization $\|S_t\|_0$ denotes the number of required skill/resources in S_t, i.e., $s_t^k > 0, 1 \leqslant k \leqslant q$. Intuitively, if $s_t^k \leqslant s_w^k$, it means t's skill/resource requirement s_t^k could be completely satisfied, $1 \leqslant k \leqslant q$; otherwise, it could be partly satisfied. As a result, valid candidate workers for node t would be rewritten as below:

$$\mathcal{W}_t = \{w_j | \varepsilon(t, w_j) \geqslant \varepsilon_{thre}, \forall w_j \in \mathcal{W}\}. \quad (6)$$

In other words, given one subtask t, only the workers whose average skill/resource levels are no less than the specified threshold ε_{thre} will be regarded as candidate workers. And the threshold can be determined by the MC application's specific requirements and skill/resource profile distribution over available workers. To distinguish this matching from the one in Eq. 3, we name it as *weak matching*, while the one in Eq. 3 as *strong matching*.

Here, for one specific subtask t, the identified candidate workers $\mathcal{W}_t$ are qualified to execute it. However, in the condition of "weak matching", each candidate w's qualification, i.e., matching degree $\varepsilon(t, w)$, is different. In general, it seems clear that the greater the value of $\varepsilon(t, w)$, the higher the **expected reliability** of subtask t's result [14], [15]. Following this common practice, we also evaluate t's expected result reliability, using the matching degree between t's requirements and worker w's profile, i.e., $\varepsilon(t, w)$. And the above threshold ε_{thre} is regarded as reliability constraint accordingly.

2.2 Problem Formulation

For each subtask t, after obtaining its candidate worker set $\mathcal{W}_t$, we should optimally select suitable workers to schedule all these involved subtasks in $\mathcal{T}$. In this work, we formalize a compact task scheduling problem in MC systems, in which **Task Makespan** and **Idle Time** are optimized simultaneously. In the following, we formally define these two optimization objectives.

To fulfill each assigned subtask t, the selected worker $w_j \in \mathcal{W}_t$ needs to physically move to the specified location loc_t, and utilizes human cognition, device resource to conduct it. As a result, both the traveling time and execution time should be considered in t's completion time. Here, for simplicity, we adopt Euclidean distance to calculate the implicit traveling cost from w_j's current location loc_{w_j} to loc_t, i.e., $dist(loc_{w_j}, loc_t)$. Note that, our proposed approaches could be easily extend to other distance metrics, such as Manhattan distance, road network distance and so on. Assuming worker w_j adopts moving speed v_j, then the traveling time can be derived as follows: $\theta_m^{t,w_j} = dist(loc_{w_j}, loc_t)/v_j$.

After arriving at specified location, w_j needs to utilize his resource/skill to accomplish subtask t. Intuitively, the execution time is directly related to the worker's qualification with respect to t. As stated in [16], an experienced worker, who has a high skill level, needs less time to accomplish a skill requirement than a worker with a lower skill level. That is to say, the better w_j can cover subtask t's requirements, the shorter the execution time is. Following the previous research [16], [17], [18], in this paper, we adopt an exponential function to model the execution time as below:

$$\theta_e^{t,w_j} = \alpha * \sum_{1 \leqslant k \leqslant q} e^{-\beta * \Theta(s_t^k, s_w^k)}, \quad (7)$$

where α and β denote constant coefficients. Clearly, the role of β, i.e., decay ratio, is to control the decay of exponential function. As shown in Fig. 3, we present a set of decay curves with different values of parameter β. Intuitively, β

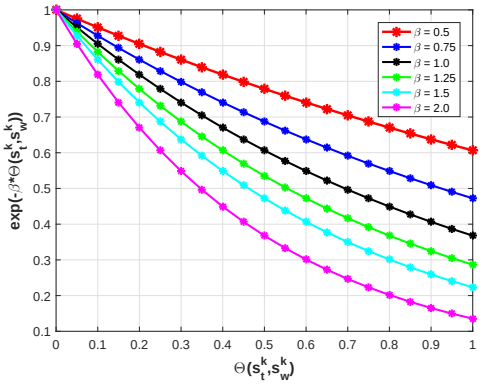


Fig. 3. Different Values of Parameter β.

should be determined appropriately which can distinguish different skill level well. On the other hand, we argue that there must exist a low bound to reflect the necessary execution cost. By considering these two factors simultaneously, we suggest a suitable range from 0.75 to 1.25 for parameter β. And in the experiments, we set the value of $\beta = 1$. While for parameter α, it can be regarded as an amplification factor. In practice, its value can be adjusted according to different application requirements. For simplification, we set it as 1 in experiments. Note that, the choices of these two parameters are orthogonal to our method proposed later.

Except for the cost of traveling time and execution time, the underlying dependency relationship, i.e., graph topology constraints, between subtasks should also be seriously considered. To be specific, one subtask's completion time can not be just determined by its performer's arrival time and the associated execution time, but need to coordinate all its predecessor subtasks' completion times. In practice, during the coordination process, it might incur the idle time in two cases: 1) when selected worker w_j has arrived at designated subtask t_i's location, he could not start the subtask, but has to wait for the accomplishment of t_i's all predecessor nodes; 2) when t_i's predecessor nodes all have been completed, i.e., subtask t_i is ready to be processed, but the selected worker w_j has not arrived in time. Evidently, both these cases would make against the implementation of task $\mathcal{T}$, and lower throughput and worker resource utilization. Thus, the idle time should be considered as one optimization goal in our work.

With respect to one subtask $t_i \in \mathcal{T}$, from the perspective of task graph topology, its **Process Ready Time** $P_r(t_i)$ is equal to the maximum completion time of all its predecessor nodes',

$$P_r(t_i) = Max\{P_f(t_x) | t_x \in \mathcal{T}_{\texttt{pre}}(t_i)\}, \quad (8)$$

where $P_f(t_x)$ represents the time instant at which one predecessor node t_x has just been completed. If the designated worker w_j can not arrive at t_i's location before its process ready time $P_r(t_i)$, t_i is unable to be carried out. Therefore, t_i's start time $P_s(t_i, w_j)$ is determined solely based on its process ready time $P_r(t_i)$ and the designated worker w_j's arrival time $P_v(t_i, w_j)$. If $P_r(t_i) < P_v(t_i, w_j)$, subtask t_i needs to wait until worker's arrival; otherwise, the selected worker will have to wait for the completion of t_i's all predecessor nodes, i.e., $P_r(t_i)$. Therefore, t_i's real start time can be calculated as follows:

$$P_s(t_i, w_j) = \begin{cases} P_v(t_i, w_j) & if\ P_r(t_i) \leqslant P_v(t_i, w_j), \\ P_r(t_i), & Otherwise. \end{cases} \quad (9)$$

Similarly, t_i's completion time can be derived by adding its corresponding execution time,

$$P_f(t_i, w_j) = \begin{cases} P_v(t_i, w_j) + \theta_e^{t_i, w_j} & if\ P_r(t_i) \leqslant P_v(t_i, w_j), \\ P_r(t_i) + \theta_e^{t_i, w_j}, & Otherwise. \end{cases} \quad (10)$$

Thus, we define **Idle Time** $IT(t_i, w_j)$ as the difference between t_i's process ready time $P_r(t_i)$ and its designated worker w_j's arrival time $P_v(t_i, w_j)$. Formally, it can be calculated as below:

$$IT(t_i, w_j) = |P_r(t_i) - P_v(t_i, w_j)|. \quad (11)$$

Fig. 4 illustrates a toy example of idle time for subtask t_4 with three predecessor subtasks: t_1, t_2 and t_3, i.e., $\mathcal{T}_{\texttt{pre}}(t_4) = \{t_1, t_2, t_3\}$. Assume there are two candidate workers, w_* and $w_\#$, the startup time and idle time of these two candidates are listed correspondingly, in which these two workers' arrival times $P_v(t_4, w_*)$ and $P_v(t_4, w_\#)$ are different.

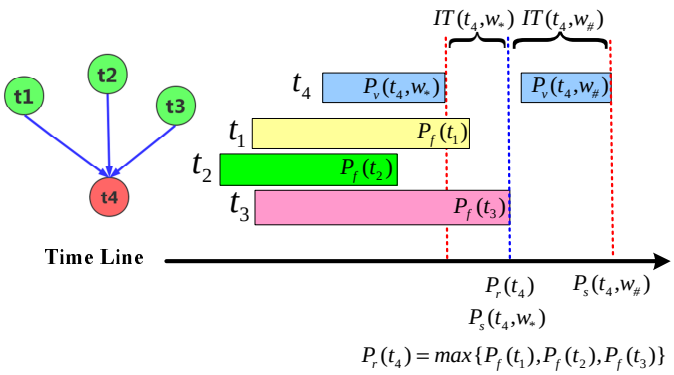


Fig. 4. Idle Time for One Subtask with Predecessors.

In order to avoid violating the precedence constraints in $\mathcal{T}$, it is most natural to schedule subtasks layer by layer. Obviously, as predecessor nodes are empty, subtasks located at entry nodes, i.e., $\mathcal{T}_{\texttt{Entry}}$, are not restricted by any precedence constraints, such that they can be performed at any time without waiting any other subtasks' accomplishment. Without loss of generality, we assume entry nodes in G start execution at time 0. To ensure task fulfillment, we assume that *once a worker starts participating into a MC campaign, he will not leave halfway through the execution process of task $\mathcal{T}$, unless all his designated subtasks have been completed*. Provided that task completion time would not be increased, workers can decide by himself at which time he will initiate his execution schedule. As a result, to obtain the overall makespan of task $\mathcal{T}$, it is a must to jointly simulate and act the whole scheduling scheme step by step. Concretely, starting from the "Entry Node" in $\mathcal{T}$, we need to iteratively simulate the execution process of all the involved subtasks, and continuously track all the involved workers' mobility process. And only if all the "Exit Nodes", i.e., subtasks in the maximum layer, have been accomplished, the overall makespan of $\mathcal{T}$ can be obtained. Formally, the **Makespan** MS of $\mathcal{T}$ can be formalized as below:

$$MS(\mathcal{T}) = Max\{P_f(t_{k_1}), P_f(t_{k_2}), ..., P_f(t_{k_p})\}, \quad (12)$$

where $t_{k_i} \in \mathcal{T}_{\texttt{Exit}}, 1 \leqslant i \leqslant p$, $P_f(t_{k_i})$ denotes the completion time of subtask t_{k_i}, and $\mathcal{T}_{\texttt{Exit}}$ denotes the set of exit nodes in G. As needs to traverse and simulate all the involved subtasks, the calculation of $MS(\mathcal{T})$ is computationally expensive.

For the ease of presentation, we adopt a matrix structure $\mathcal{X}$ to represent task schedule solution, i.e., a set of subtask-worker pairs. Specifically, one entry $\mathcal{X}_{i,j}, 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m$, denote whether subtask t_i has been assigned to worker w_j. If so, the value of $\mathcal{X}_{i,j}$ would be set as 1; otherwise, $\mathcal{X}_{i,j}$ equals 0.

According to the definitions of task graph and the related concepts, we define the **TGS-MC** problem as follows.

Definition 6 (Task Graph Scheduling in Mobile Crowdsourcing, TGS-MC). Consider a complex MC task $\mathcal{T} = \{t_i | 1 \leqslant i \leqslant n\}$, where each subtask t_i is specified with spatial location loc_t and skill/resource requirement S_t, i.e., $t = (loc_t, S_t)$. And dependency constraints are attached with these subtasks, which is represented as a task graph $G = (\mathcal{T}, E)$, where E denotes the associated dependency constraints between subtasks. For one set of available workers $\mathcal{W} = \{w_1, w_2, ..., w_m\}$, where $w_j = (loc_{w_j}, v_j, S_{w_j})$, we aim to coordinate all the involved subtasks and available workers, and build an optimal scheduling solution with the form of subtask-worker pairs $\mathcal{X}_{i,j}$, such that task $\mathcal{T}$'s makespan and the overall idle time are simultaneously minimized, subject to the worker reliability constraints. Formally, our TGS-MC problem is to optimize both objectives simultaneously as below:

$$\begin{cases} Min: & Max\{P_f(t_{k_1}), P_f(t_{k_2}), ..., P_f(t_{k_p})\}, \\ Min: & \sum_{t_i \in \mathcal{T}} |P_r(t_i) - P_v(t_i, w_j)|, \end{cases} \quad (13)$$

$$Subject\ to: \forall \mathcal{X}_{i,j} \in \mathcal{X}, \mathcal{X}_{i,j} = 1 \vee \varepsilon(t_i, w_j) \geqslant \varepsilon_{thre}, \quad (14)$$

where $t_i \in \mathcal{T}$, $w_j \in \mathcal{W}$, $t_{k_x} \in \mathcal{T}_{\texttt{Exit}}, 1 \leqslant x \leqslant p$, and $\mathcal{X}_{i,j}$ denotes subtask-worker pair (t_i, w_j). □

2.3 Problem Complexity Analysis

First of all, as having been shown in [21], [22], [24], DAG task scheduling is a NP-complete problem, in which there exists no technique that can achieve an optimal scheduling solution in polynomial time. Thus, our TGS-MC problem is also NP-complete. And it is computationally intractable to examine all the possible solutions, due to its enormous search space. For example, given task $\mathcal{T} = \{t_i | 1 \leqslant i \leqslant n\}$, assuming that the valid candidate workers for subtask t_i are

$\mathcal{W}_{t_i}$, the whole solution space is large as $\prod_{i=1}^{n} |\mathcal{W}_{t_i}|$, where $|\mathcal{W}_{t_i}|$ denotes the size of $\mathcal{W}_{t_i}$. In worst case, it could be expanded to $|\mathcal{W}|^n$. So, it calls for low-complexity efficient near-optimal algorithms.

Furthermore, different from the traditional DAG task scheduling problem, as the recruited workers need to physically move to the specified locations of outsourced subtasks, assignment in DAG oriented MC systems needs to additionally take into consideration of the geographical distances between tasks and workers, which have a great impact on factors of concern, including workers' traveling distance, tasks' makespan and the overall idle time. And since the workers' positions would dynamically change from time to time, the designed DAG-oriented task scheduling algorithms must track and update the workers' current positions timely. Therefore, considering the spatial distance restriction, the problem becomes more difficult and intractable.

Last but not least, our TGS-MC problem seeks to achieve a compact scheduling solution, i.e., minimizing not only task completion time (makespan), but also the idle time. However, these two goals are *neither coincident nor conflicting*. On the one hand, makespan focuses on the overall task execution efficiency, while idle time emphasizes the efficient utilization of worker resource. While on the other hand, improving one goal does not necessarily result in deterioration of the other goal. For example, if sufficient worker resources are available, it might result in less idle time, but it is not necessarily so for makespan. Thus, we need to carefully balance between these two objectives.

3 Proposed Approaches

In this section, based on a thorough and comprehensive analysis of our TGS-MC problem, we propose two problem-solving approaches from local and global optimization perspective, respectively. In the following, we will elaborate them in detail.

3.1 Local Optimization: Layered Dynamic Priority Scheduling

As the fact that subtasks in upper layer are preceded by the ones in lower layer, it is naturally to utilize a divide-and-conquer technique to sequentially construct task scheduling solution. Basically, we adopt a layered mode, i.e., from the lowest layer (entry nodes) in G to the highest layer (exit nodes), to tackle involved subtasks. In each layer, the contained subtasks are scheduled to appropriate workers with the goal of minimizing makespan and idle time simultaneously, according to workers' current conditions. Essentially, it strives to achieve optimality within each layer locally during the sequential scheduling process. Here, we name this qusi-Breadth-First-Search (BFS) based scheduling method as *BFSPriD* algorithm. Next, we will explain its workflow detailedly.

Firstly, all the subtasks contained in incumbent layer need to be traversed and extracted, such as $\mathcal{T}_{\#} = \{t_{i_1}, t_{i_2}, ..., t_{i_p}\}$. Since we follow a direction of increasing layer, subtasks in $\mathcal{T}_{\#}$ are all process ready tasks, and independent of each other with respect to precedence constraints. That is to say, there exist no predecessor nodes of subtasks in $\mathcal{T}_{\#}$ (Entry nodes $\mathcal{T}_{\texttt{Entry}}$), or all the predecessor nodes of subtasks in $\mathcal{T}_{\#}$ have already been finished. Indeed, the divided task scheduling sub-problem in current layer is also NP-complete, due to worker resource competition among these involved subtasks. As shown in Fig. 5(a), t_1 can be assigned to candidate workers w_1, w_2 and w_3. Once w_1 has been determined to implement t_1, w_2 and w_3 would be immediately excluded for t_1, and w_1 might lose chance to undertake the possible subtask t_3 if t_1 and t_3 are far away from each other. In Fig. 5(b), we maintain a list to record each worker's current position. When w_1 has finished t_1, his position should be updated as loc_1. Therefore, w_1's traveling cost to t_3 should be recalculated and updated accordingly.

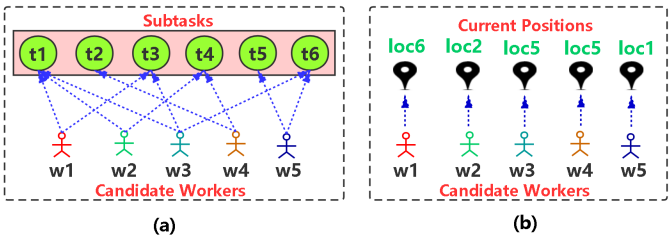


Fig. 5. Worker Resource competition Among Subtasks.

To avoid exhaustive search, we turn to sequential heuristics to construct sub-solution for $\mathcal{T}_{\#}$, by handling them one by one. Inspired by list heuristic methods in DAG scheduling [21], [22], we consider to prioritize subtasks in $\mathcal{T}_{\#}$ and schedule them accordingly. However, it is infeasible to assign priority to each subtask directly in terms of involved optimal objectives, i.e., makespan and idle time. The reason lies in: 1) subtasks usually compete with each other for available worker resources as above, thus the scheduling for each subtask needs to be coordinated properly; 2) each subtask's associated makespan and idle time are determined based on available workers' current conditions, i.e., constantly changing positions. As a result, a dynamical priority and scheduling mechanism is devised. Specifically, subtasks in $\mathcal{T}_{\#}$ are assigned with dynamic priorities based on all available workers' updated locations. And then, according to their priorities, the highest priority subtask will be firstly tackled, and removed from $\mathcal{T}_{\#}$ after being scheduled. This procedure continues until all the subtasks have been scheduled.

To dynamically prioritize subtasks in $\mathcal{T}_{\#}$, we employ a Max-Min travelling time cost metric as follows:

$$\begin{aligned} Arg: \underset{t_{i_k} \in \mathcal{T}_{\#}}{Max} \ \underset{w_j \in \mathcal{W}_{t_{i_k}}}{Min} \{dist(loc_{w_j}, loc_{t_{i_k}})/v_j\} \\ = Arg: \underset{t_{i_k} \in \mathcal{T}_{\#}}{Max} \ \underset{w_j \in \mathcal{W}_{t_{i_k}}}{Min} \{\theta_m^{t_{i_k}, w_j}\}. \end{aligned} \tag{15}$$

The basic idea is that if one subtask's nearest candidate worker is the farthest among all these subtasks in $\mathcal{T}_{\#}$, it should be scheduled preferentially. For one subtask t_{i_k} to be scheduled, we handle it as follows. Above all, its process ready time $P_r(t_{i_k})$ is calculated according to all its predecessor nodes' completion time $P_f(t_j)$, $t_j \in \mathcal{T}_{\texttt{pre}}(t_{t_{i_k}})$. Its valid candidate worker set $\mathcal{W}_{t_{i_k}}$ then needs to be identified. Note that, the reliability constraints have been used as a threshold to filter valid candidates. Thus, all the final selected workers

satisfy the reliability constraints: $\varepsilon(t_{i_k}, w_j) \geqslant \varepsilon_{thre}$. By computing each candidate worker $w_j \in \mathcal{W}_{t_{i_k}}$'s traveling cost and execution cost, we can directly derive the expected completion time $P_f(t_{i_k}, w_j)$ and idle time $IT(t_{i_k}, w_j)$ for w_j. Based on these two obtained metrics, one suitable worker could be recognized and selected to undertake subtask t_{i_k}. However, as analyzed previously, these two metrics are not coincident. So, it is indispensable to seeking a balance between makespan and idle time goals, e.g., constructing an integrated utility function. Here, we adopt the commonly used linear weighted sum method to construct an integrated utility function [15], [25]. To be specific, each candidate worker w_j's utility with respect to subtask t_{i_k} can be computed as follows:

$$\partial(t_{i_k}, w_j) = \varphi * P_f(t_{i_k}, w_j) + (1 - \varphi) * IT(t_{i_k}, w_j), \quad (16)$$

where the factor φ equals to 0.5 for balancing these two objectives. Clearly, the one that has maximum utility should be regarded as the most competitive candidate, and determined as the desired task performer. The workflow of *BFSPriD* will terminate when all the involved subtasks in $\mathcal{T}$ have been processed. The overall process of *BFSPriD* is outlined in **Algorithm 1**.

Next, we analyze the time complexity of *BFSPriD* Algorithm. For simplicity, the size of valid candidate workers for each subtask t, i.e., $\mathcal{W}_t$, is represented as a fixed average value $|\mathcal{W}_t|_{av}$. And the average number of subtasks over each layer is represented as $|\mathcal{T}|/Ly$. Within each layer of task graph G, the computational overhead of the subtask scheduling is dominated by dynamic prioritization, which is $O((|\mathcal{T}|/Ly) * |\mathcal{W}_t|_{av} * log_2 |\mathcal{W}_t|_{av})$. Summarizing the above analysis, the computational complexity of *BFSPriD* algorithm is $O(|\mathcal{T}|^2 * Ly^{-1} * |\mathcal{W}_t|_{av} * log_2 |\mathcal{W}_t|_{av})$. In nature, as a local optimization algorithm, *BFSPriD* achieves high efficiency, but sacrifices quality to some extent.

ALGORITHM 1: *BFSPriD* Algorithm

Input: Task graph: $G = \{V, E\}$, Reliability threshold: ε_{thre}, Workers: $\mathcal{W}$;
Output: Task schedule solution: $\mathcal{X}$;
for *each layer* Ly *in* G **do**
  Pick out all the nodes $\mathcal{T}_\#$ contained in Ly;
  while $\mathcal{T}_\# \neq \varnothing$ **do**
    Prioritize remainder subtasks in $\mathcal{T}_\#$ by Eq. 15;
    Pick out subtask t with the highest priority;
    Calculate $P_r(t)$ from its predecessors $\mathcal{T}_{\texttt{pre}}(t)$;
    Search candidate worker set $\mathcal{W}_t$;
    for *each candidate* w *in* $\mathcal{W}_t$ **do**
      Derive benefit by Eq. 16;
    end
    Determine worker w_* with maximum benefit;
    $\mathcal{X} = \mathcal{X} \bigcup (t, w_*)$ and $\mathcal{T}_\# = \mathcal{T}_\# - \{t\}$;
  end
end

3.2 Global Optimization: Evolutionary multi-tasking Algorithm

In this subsection, we devise a global optimization approach to solve our **TGS-MC** problem by harnessing population-based meta-heuristic evolutionary algorithm. Towards both optimization objectives simultaneously, we utilize a novel evolutionary search paradigm, namely Evolutionary MultiTasking (**EMT**) [26], [27], to address **TGS-MC** problem. Different from traditional evolutionary algorithms, evolutionary multitasking conducts evolutionary search concurrently on multiply search spaces corresponding to different optimization problems. By transferring implicit knowledge between the correlative optimization problems, the performances including solution quality and search efficiency can be enhanced markedly. In other words, the common knowledge shared by involved optimization tasks can be explored to collaboratively optimize the problems. Based on EMT, we propose a global optimization approach named *EMTTSch* algorithm, which will be elaborated subsequently.

3.2.1 Framework

The framework of our proposed *EMTTSch* algorithm is present in Fig. 6. For each optimized objective, i.e., makespan and idle time, we build two separate evolutionary solvers, namely makespan evolutionary solver $\mathbf{SV}_1$ and idle time evolutionary solver $\mathbf{SV}_2$, respectively. And then, these two solvers individually and iteratively tackle their respective optimization problems with independent populations. Here, we employ the basic workflow of generalized differential evolution method [28] to construct their evolutionary process. The reason is that differential evolution is capable of searching very large spaces of candidate solutions. Firstly, the initial population is initialized for each solver $\mathbf{SV}_1$ and $\mathbf{SV}_2$, respectively. And then, during the evolutionary process, the operations of reproduction and selection are conducted to generate offspring solutions, and achieving improvement on respective optimization objectives. Importantly, the knowledge contained in fittest solutions is transferred across these two solvers through learned mapping relationship. Besides, an external archive is introduced to dynamically record the found better solutions in each generation. The procedure continues until the termination criterion is met. Finally, we conduct a post-processing operation, i.e., elite solution selection, to determine the desired solution which can balance well on these two objectives. The overall process of *EMTTSch* is outlined in **Algorithm 2**. And the subcomponents are explained in details in the following.

3.2.2 Solution Representation and Initialization

According to the structure of task graph G, we adopt a dual-chromosome scheme to represent scheduling solution of **TGS-MC** problem, i.e., task and worker chromosomes. The task chromosome $\vec{\mathcal{T}}$ is a permutation of n subtasks in $\mathcal{T}$ to be scheduled, according to the order of layers in G, and the worker chromosome $\vec{\mathcal{X}}$ which is in numerical vector form is used to indicate the task scheduling result. To be specific, if i-th subtask in task chromosome $\vec{\mathcal{T}}$ is assigned to worker w_j, the corresponding entry in worker chromosome $\vec{\mathcal{X}}$, i.e., i-th element $\vec{\mathcal{X}}(i)$, is filled with w_j. During evolutionary process, $\vec{\mathcal{T}}$ is fixed and act as a reference, while $\vec{\mathcal{X}}$ evolves via modifying its contained entries. Actually, compared with the previously defined solution form $\mathcal{X}$, $\vec{\mathcal{X}}$ can be directly obtained from the indexes of non-zero elements in $\mathcal{X}$.

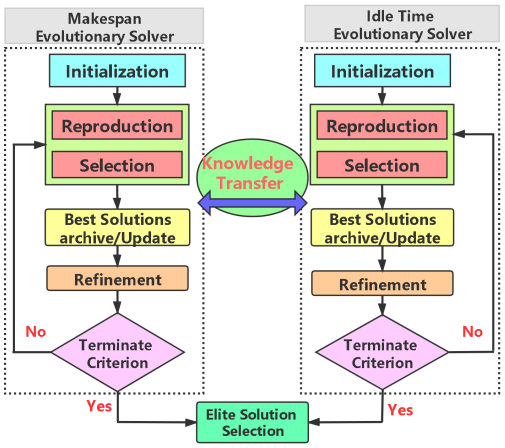


Fig. 6. Framework of *EMTTSch* Algorithm.

For population initialization, it might seem unnecessary to generate initial solutions from scratch. Particularly, problem-specific heuristics could be exploited to facilitate its convergence speed and solution quality significantly. To generate initial solutions of high quality, we devise a hybrid initialization mechanism, where the initial solutions are partially constructed using *BFSPriD* algorithm due to its high efficiency, and others are randomly generated from search space. To be specific, for the former mode, we change the value of balancing factor φ in Eq. 16 from 0 to 1, to generate candidate solutions with different preferences over these two optimization goals. For example, when φ equals to 0.4, it means that the weight of makespan goal is 0.4, while idle time goal is 0.6. In the latter one, initial solutions are randomly created from recognized candidate workers, e.g., selecting one candidate worker from $\mathcal{W}_t$ for subtask t at random. Clearly, the role of random initialization is to enhance the diversification of population. At the end, we achieve two sets of solution populations of size PS, i.e., $\mathbf{P}_1$ and $\mathbf{P}_2$, for solver $\mathbf{SV}_1$ and $\mathbf{SV}_2$, respectively, where $|\mathbf{P}_1| = |\mathbf{P}_2| = PS$.

ALGORITHM 2: *EMTTSch* Algorithm

Input: Task graph: $G = \{V, E\}$, Workers: $\mathcal{W}$, Population size: PS, Paramters: ε_{thre}, cp, DW;
Output: Task schedule solution: $\vec{\mathcal{X}}^*$;

```
Initialize population sets P1 and P2;
Calculate mapping relationship M1,2 and M2,1;
while termination criterion not satisfied do
    for each candidate solution X in P1/P2 do
        Generate offspring X_# by mutation operation;
        Update X_# using recombination operation;
        Greedily select one to survive from X_# and X;
    end
    Evaluate the fitness of population P1/P2;
    Bidirectional knowledge transfer by multiplication with M1,2/M2,1;
    Update archive with newly generated solutions;
    Refine population P1/P2 with tournament;
end
Elite solution selection to balance both objectives;
```

3.2.3 Population Reproduction and Selection

During the evolutionary process, for each candidate solution $\vec{\mathcal{X}}$ in $\mathbf{P}_1$ or $\mathbf{P}_2$, we choose three solutions $\vec{\mathcal{X}}_a$, $\vec{\mathcal{X}}_b$, and $\vec{\mathcal{X}}_c$ at random, and they must be different from each other as well as from incumbent solution $\vec{\mathcal{X}}$. Based on these selected solutions, it conducts mutation operation to generate new offspring $\vec{\mathcal{X}}_{\#}$. Mathematically, the new offspring $\vec{\mathcal{X}}_{\#}$ can be generated as below:

$$\vec{\mathcal{X}}_{\#}(i) = \vec{\mathcal{X}}_a(i) + DW * (\vec{\mathcal{X}}_b(i) - \vec{\mathcal{X}}_c(i)), \tag{17}$$

where $\vec{\mathcal{X}}_{\#}(i)$ denotes the i-th entry of $\vec{\mathcal{X}}_{\#}$, and the parameter DW is differential weight which is ranging from 0 to 2 practically.

Next, the recombination operation is implemented over the incumbent solution $\vec{\mathcal{X}}$ and the newly generated offspring $\vec{\mathcal{X}}_{\#}$. Concretely, given a recombination probability cp, we update each entry in $\vec{\mathcal{X}}_{\#}$ one by one. If one random number ranged from 0 to 1 is not more than probability cp, the entry $\vec{\mathcal{X}}_{\#}(i)$ remains unchanged; otherwise, it will be replaced by the corresponding entry in $\vec{\mathcal{X}}(i)$. Formally, it can be represented as follows:

$$\vec{\mathcal{X}}_{\#}(i) = \begin{cases} \vec{\mathcal{X}}_{\#}(i), & if\ rand \leqslant cp \\ \vec{\mathcal{X}}(i), & if\ rand > cp, \end{cases} \tag{18}$$

where $rand$ is a random number, i.e., $rand \sim U[0, 1]$.

Afterwards, for incumbent solution and the offspring one, we greedily select one to survive to the next generation. To be specific, the new offspring $\vec{\mathcal{X}}_{\#}$ is need to compare with the incumbent $\vec{\mathcal{X}}$, in terms of their corresponding optimized objective. If the objective evaluation (makespan or idle time) of $\vec{\mathcal{X}}_{\#}$ is not more than $\vec{\mathcal{X}}$, $\vec{\mathcal{X}}_{\#}$ will be survived to the next generation; otherwise, $\vec{\mathcal{X}}$ will survived. In other words, if the objective of new offspring is an improvement, it is accepted and entered into optimization population, otherwise it is directly discarded.

3.2.4 Solution Knowledge Transfer and Refinement

We adopt denoising autoencoder-based explicit genetic transfer [27], to exploit the underlying synergistic effect between these two correlated optimization solvers. First of all, the mapping relationship between these two optimization problems should be built. Concretely, two subsets of solutions are sampled uniformly and independently from the these two optimization problems' search spaces, respectively. Formally, we use the symbols $\mathbf{X}$ and $\mathbf{Y}$ to denote them, where $\mathbf{X} = \left\{\vec{\mathcal{X}}_1, \vec{\mathcal{X}}_2, ..., \vec{\mathcal{X}}_N\right\}$, $\mathbf{Y} = \left\{\vec{\mathcal{Y}}_1, \vec{\mathcal{Y}}_2, ..., \vec{\mathcal{Y}}_N\right\}$, $\mathbf{X} \subseteq \mathbf{P}_1$ and $\mathbf{Y} \subseteq \mathbf{P}_2$. Note that, the sampled solutions in $\mathbf{X}$ (or $\mathbf{Y}$) need to be sorted with respect to their respective optimization objectives, i.e., makespan and idle time. Under the denoising autoencoder model, the mapping $\mathbf{M}_{1,2}$ of size $|\mathcal{T}| \times |\mathcal{T}|$ from makespan solver $\mathbf{SV}_1$ to idle time solver $\mathbf{SV}_2$ can be derived by using $\mathbf{X}$ as input and $\mathbf{Y}$ as output. And the squared reconstruction loss can be calculated as follows:

$$\mathcal{L}_{sq} = \frac{1}{2N} \sum_{i=1}^{N} \left\| \vec{\mathcal{X}}_i - \mathbf{M}_{1,2} * \vec{\mathcal{Y}}_i \right\|. \tag{19}$$

Generally, the mapping relationship $\mathbf{M}_{1,2}$ can be computed in close form as below:

$$\mathbf{M}_{1,2} = (\mathbf{Y} * \mathbf{X}^T)(\mathbf{X} * \mathbf{X}^T)^{-1}. \quad (20)$$

Similarly, the mapping from $\mathbf{SV}_2$ to $\mathbf{SV}_1$, i.e., $\mathbf{M}_{2,1}$, could also be derived accordingly. Along the evolutionary processes of these two solvers, better solutions discovered by any solvers will be transferred and injected into the population of the counterpart solver, via the operation of matrix multiplication with $\mathbf{M}_{1,2}$ from $\mathbf{SV}_1$ to $\mathbf{SV}_2$ (or with $\mathbf{M}_{2,1}$ from $\mathbf{SV}_2$ to $\mathbf{SV}_1$). Note that, for efficiency, knowledge transfer is conducted across these two optimization problems, with a fixed generation interval along the evolutionary process, e.g., every ten generations.

After knowledge transferring, we need to evaluate all the candidate solutions, including pre-existing and injected ones. And the fittest solutions will be recorded into the external archive of size $\mathcal{K}$. If there is candidate solution whose fitness is better than the ones in archive, external archive will be updated by adding this newly discovered solution. Afterwards, it is necessary to implement refinement operation to maintain the size of each solver's population to a constant scale. Here, the tournament selection strategy is utilized to select the fittest individuals from the current generation. Via tournament selection, the determined individual solutions are passed on to the next generation.

3.2.5 Elite Selection Strategy

When the terminal criterion has been met, both evolutionary solvers stop, and provide the final solutions with respect to their corresponding optimization objectives. As our goal is to concurrently optimize these two objectives, we adopt an elite selection strategy to pick out the desired solution which could better balance these involved objectives, i.e., P_f and IT. Suppose the solutions recorded in solver $\mathbf{SV}_1$ and $\mathbf{SV}_2$'s external archives are represented as $\mathbf{X}_{bs}$ and $\mathbf{Y}_{bs}$. For each solution in $\mathbf{X}_{bs}$ or $\mathbf{Y}_{bs}$, we evaluate its fitness in terms of makespan and idle time, respectively. And then, the domination relationship is borrowed from the domain of multi-criteria decision making [29]. By pairwise comparison, a subset of non-dominated solutions is obtained which is denotes as $\mathbf{X}^*$. Finally, we pick out one solution $\vec{\mathcal{X}}^*$ which located at the median of the objective space, as it can better balance the makespan and idle time objectives.

3.2.6 Time Complexity Analysis

In *EMTTSch*, evaluating a candidate solution is to simulate the complete process of subtask implementation in turn. The complexity of the fitness evaluation is thus $O(|\mathcal{T}|)$. The random population initialization strategy costs $O(|\mathcal{T}|)$ to create initial candidate solution. The computational overhead of the mapping relationship construction is dominated by matrix inversion, which is $O(N^3)$, where N denotes the number of sampled candidate solutions in each solver. In the following, we discuss the time complexity of each major component of *EMTTSch* algorithm in every generation. The operations of reproduction and selection, i.e., differential evolution, cost $O(2 * PS * |\mathcal{T}|)$ to generate new offspring, and achieve improvement in the specific optimization goal (makespan/idle time). The knowledge transfer consumes $O(M * |\mathcal{T}|^2)$, where M denotes the scale of better solutions which will be injected into counterpart solver. It takes $O(PS * (\mathcal{K}+1) * log_2(\mathcal{K}+1))$ to update the external archive. And the population refinement averagely cost $O(PS)$ because of tournament operation. Finally, the elite selection operation takes $O(\mathcal{K} * (2\mathcal{K}+1))$ to achieve the pair-wise comparison amongst solutions recorded in these two external archives, where $\mathcal{K}$ is the archive size.

4 Evaluation and Discussion

In this section, we systematically evaluate the performance of our proposed techniques using two real-world datasets. Our experiments and latency observations are conducted on a standard server (Windows), with Intel(R) Core(TM) i7-6700HQ CPU, 2.60 GHz and 32 GB main memory.

4.1 Experimental Settings

Data Set: In our experiments, we choose two real-world data sets. The first small-scale data set is StudentLife [1] published by researchers at Dartmouth College. The StudentLife data set was collected from mobile phones of a class of 49 Dartmouth students over 10 weeks, including the readings of GPS, WiFi and so on. We use the WiFi readings as workers' initial positions, and 101 Point-of-Interests (POIs), i.e., location with semantic tags, across Dartmouth College are utilized to generate subtasks. The second large-scale data set we used is a GPS trajectory data set collected from about 1200 taxicabs in Chengdu city, China. The GPS data points are employed to simulate participant workers' initial positions. And 3000 POIs in Chengdu city are randomly extracted to generated subtasks. Using these two real-world data sets, the initial positions of workers, and their associated moving speeds are directly sampled from the existing historical trajectories. Concretely, the original locations of recorded trajectories in the data sets are utilized as the initial positions of the recruited MC workers. In addition, to generate the outsourced MC subtasks, the involved specified locations and required skills/resources are simulated based on the contained POIs' spatial locations and associated semantics, e.g., gymnasium, art museum, etc. For example, one subtask might require taking a photograph at art museum.

Baseline Algorithms: To the best of our knowledge, our work is the first to investigate task scheduling for mobile crowdsourcing that considers task dependency constraints. Existing mobile crowdsourcing task scheduling approaches cannot be tailored to address our TGS-MC problem. Consequently, we select three most related methods from DAG task scheduling for comparison, which are listed as follows:

1) *HEFTM Algorithm:* the method is developed by modifying one of the most frequently cited and used DAG task scheduling algorithm, i.e., *HEFT* [21], [30]. It consists of two phases, i.e., task prioritizing and scheduling. Specifically, all subtasks are firstly prioritized based on their average

1. http://studentlife.cs.dartmouth.edu/

execution time and traveling time, with an upward style as below.

$$rank(t) = \frac{1}{|\mathcal{W}_t|} \sum_{w_j \in \mathcal{W}_t} \theta_e^{t,w_j} + \underset{t_j \in \mathcal{T}_{\text{suc}}(t)}{Max} (\overline{ct_{t,t_j}} + rank(t_j)), \tag{21}$$

where $\overline{ct_{t,t_j}}$ denotes the average traveling cost from loc_t to loc_{t_j}, that $\overline{ct_{t,t_j}} = \frac{1}{|\mathcal{W}_t|} \sum_{w_j \in \mathcal{W}_t} dist(low_{w_j}, loc_t)/v_j$, and $loc_{w_j} = loc_{t_j}$. Obviously, different from our *BFSPriD* approach, *HEFTM* employ a static manner to prioritize subtasks. Afterwards, the scheduling procedure will handle subtasks in the already determined priority order.

2) *ClustS Algorithm:* this method adopts the clustering heuristics approaches in DAG scheduling. Each subtask's spatial location and skill/resource requirements are concatenated into an unified representation. By calculating pairwise similarity between subtasks in the representation space, all the subtasks are clustered into different groups. And the subtasks contained in one cluster are exclusively scheduled to the same worker, in order to reduce the traveling cost and meet reliability constraints.

3) *GloGa Algorithm:* this method adopts Genetic Algorithm to search optimal scheduling solutions. Based on the population-based evolutionary process, it globally search a desired solution throughout the whole problem space.

Task Construction and Parameter Setting: To construct task graph $\mathcal{T}$, first of all, we need to specify its involved subtasks, including spatial locations and required skills/resources. As explained previously, subtasks' locations can be directly casted from the POIs' spatial distribution. Furthermore, one set $\mathcal{S}$ sampled from the semantic tags associated with POIs is built to simulate subtasks' skill/resource requirements. In our experiments, $\mathcal{S}$ contains 10 different device resources and operation skills. Concretely, each subtask's required skill/resource is no more than 5. Especially for the task dependent relationship, we harness the mobility transition probabilities learned from the trajectories, i.e., the conditional probability of traveling to a successor subtask's location, given that a crowd of people has already arrived at a predecessor subtask's location. Afterwards, two parameters, i.e., maximum layer and maximum successor subtasks, are used to generate the task graph $\mathcal{T}$. Clearly, the role of the prior parameter is to control the "flatness" of task graph $\mathcal{T}$. While the latter one is utilized to specify the relevant precedence-constraints, i.e., edges in $\mathcal{T}$. To be specific, for one subtask, say t, a subset of nodes, such that no more than the maximum successor subtasks, are randomly selected as its successor ones. In the following experiments, we will investigate the impact of these two parameters on the average performance of our approaches.

The involved parameters are set as follow: both the parameter α and β are set to 1. The reliability parameter ε_{thre} is set to 0.85. In *EMTTSch* algorithm, differential weight DW is set to 1, and the recombination probability cp is set to 0.45. The size of population PS is 150, the scale of sampled solutions is 40, and the external archive contains $\mathcal{K} = 20$ fittest solutions. And the termination criterion is set as maximum generations: $MxGen = 120$. During population initialization, the size of solution generated based on heuristics equals to 10, i.e., φ varies from 0 to 1 with 0.1 increments. And the remainders are randomly initialized.

4.2 Experimental Results and Analysis

Scheduling Performance: In this part, we conduct extensive simulation experiments on small and large scale problem settings, respectively, to evaluate the performance of our proposed approaches. Firstly, we implement two small-scale experiments using the first StudentLife data set. In the prior one, we fix the number of subtasks with 50, and vary the scale of student workers from 33 to 49 with 4 increments. For the latter one, the scale of workers is set to 49, while the number of subtasks varies from 40 to 80 with 10 increments. In generated task graph, the maximum layer and maximum successor subtasks are set as 10 and 4, respectively. The corresponding results are reported in Fig. 7 (a)∼(d). In order to fix the performance of all involved scheduling algorithms, we report the average results over 10 repeated trails.

From the experiment results, it is obvious that the scheduling performance of *EMTTSch* is the best, followed by *BFSPriD*, *HEFTM*, *GloGA* and *ClustS* algorithms. Based on solution knowledge transfer and several enhancement strategies, *EMTTSch* obtains the best scheduling solution. What is more, compared with evolutionary *GloGA* algorithm, its convergence speed is improved prominently by incorporating heuristic initialization from *BFSPriD* algorithm. Because *BFSPriD* algorithm adopts a dynamic prioritization scheme, its performance outperforms the *HEFTM* algorithm steadily. This also verifies the difference between our problem and traditional DAG task scheduling problem. Finally, *ClustS* algorithm's performance is the worst among all these techniques, as it implement task scheduling on the group level. Furthermore, we also observe that the performance metrics, including makespan and idle time, will be improved with the growth of candidate workers. It is common sense that task implementation naturally benefits from more available worker resources.

Next, we simulate two large-scale experiments using the second Chengdu data set. In the prior one, we fix the number of the subtasks with 1,000, and vary the size of driver workers from 800 to 1,200 with 100 increments. For the latter one, the scale of workers is set to 1,000, and the number of subtasks varies from 800 to 1,200 with 100 increments. Here, the maximum layer and maximum successor subtasks are set as 150 and 3, respectively, in generated task graph. The relevant experimental results are present in Fig. 7 (e)∼(h). Obviously, the result is consistent with the findings from the experimental results above. What is more, compared with the results of small-scale experiments, the performance of *GloGA* has degenerated clearly. One possible reason is that it is difficult for *GloGA* to find promising solution region, with the increase of problem dimensionality. Note that, in large-scale experiments, the relevant metrics increase significantly as they are conducted in a citywide environment, thus the associated traveling time would grow sharply.

Runtime Efficiency: In this part, the runtime results of the above experiment are given in Fig. 8. We observe that the efficiency of non-evolutionary algorithms, i.e., *BFSPriD*, *HEFTM* and *ClustS* algorithms, is superior to evolutionary algorithms, such as *GloGA* and our proposed *EMTTSch*. Clearly, *GloGA* and *EMTTSch* require more evolution iterations which are time-consuming. What is more, our proposed *EMTTSch* algorithm costs more time than *GloGA*,

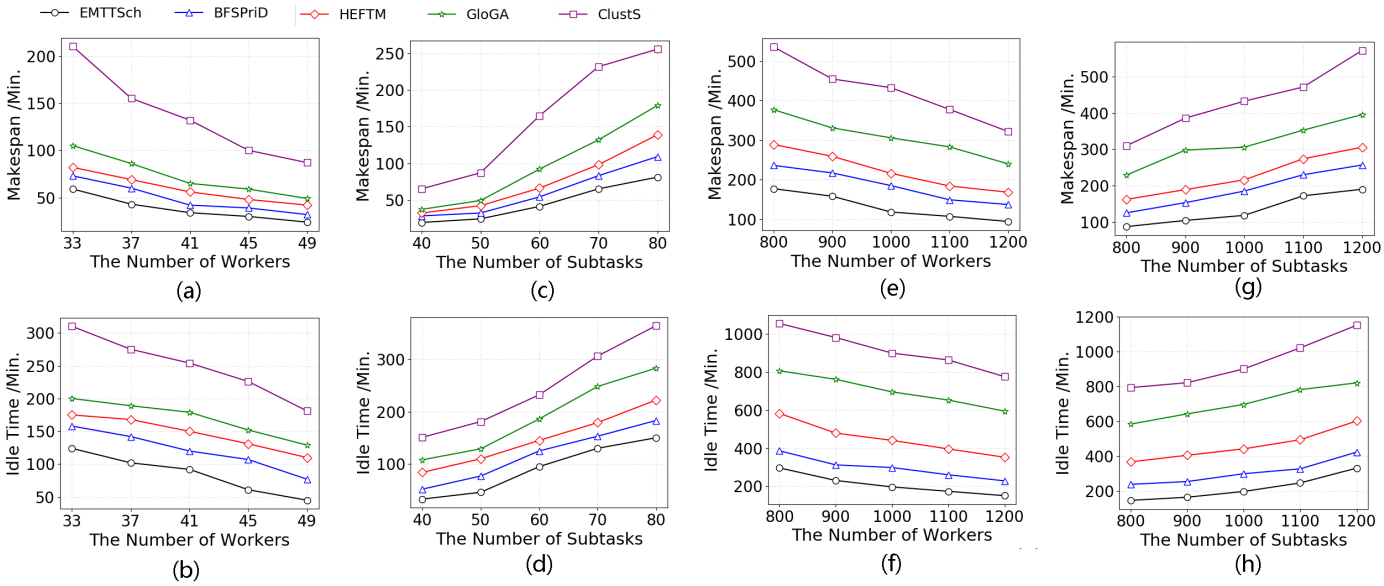


Fig. 7. Experimental Results of Scheduling Algorithms.

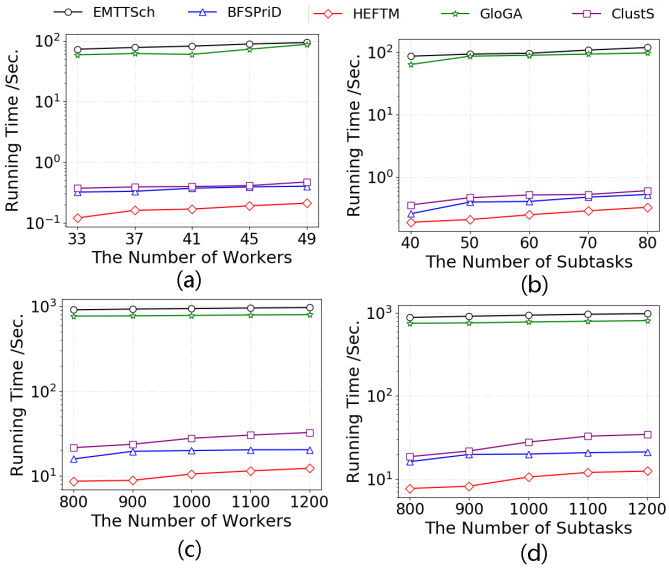


Fig. 8. Running Time of Different Approaches.

due to the operations of solution knowledge transfer, elite selection, etc. As our *BFSPriD* approach needs to repeatedly prioritize all the involved subtasks, it is more time consuming compared with the baseline *HEFTM* algorithm. With respect to *ClustS* algorithm, its computational overhead is dominated by calculating the similarity between each pair of subtasks. Thus, in large-scale problem settings, its running time increases markedly.

Next, we conduct the cost-effectiveness analysis of all these involved testing methods. In the small scale simulation experiments, although it requires extra running time of average about 83.0 sec than *HEFTM* and *ClustS* algorithms, the performance of our *EMTTSch* algorithm has been improved significantly. For instance, its makespan has been reduced by about 100 min and 21 min, and the idle time has been decreased by average about 165 min and 62 min, compared with *ClustS* and *HEFTM* algorithm, respectively. While for our *BFSPriD* algorithm, its metric of cost effectiveness is better than *EMTTSch*, as it just need less running time to return a scheduling solution. Among these testing methods, the *GloGA* algorithm's cost-effectiveness metric is worst, as it needs almost equal running time as *EMTTSch* algorithm, but achieves worse performance than *HEFTM* algorithm. And in the large scale experiments, we observe the similar trend as well, by comparison of performance measure, i.e., makespan and idle time, and search efficiency.

Parameter Sensitivity In addition, we also investigate other factors which may potentially affect the scheduling performance, such as reliability threshold parameter ε_{thre}, the maximum layer, and maximum successor subtasks. Using the large-scale data set, we simulate experiments with 1,000 subtasks and 1,000 candidate workers. And the other experimental settings remain unchanged as in the above experiments.

Firstly, we investigate the effect of parameter ε_{thre} by varying it from 75% to 95% with increments of 5%. The experimental results are reported in Fig. 9. Since the valid candidate worker set $\mathcal{W}_t$ for each subtask t is first identified before the task scheduling, and all scheduling solutions are generated among candidate workers $\mathcal{W}_t$, it thus ensures the reliability constraints. Furthermore, as shown in Fig. 9, both the metrics of makespan and idle time evidently grow as the increase of ε_{thre}. The reason lies in that, with the growth of ε_{thre}, the number of available workers would shrink accordingly, and task schedule would become tight.

Afterwards, we examine the effect of maximum layer in task graph, by varying it from 130 to 170 with increments of 10. The experimental results reported in Fig. 10 show that with the increase of task graph layers, the makespan and idle time grow accordingly. Intuitively, this parameter has greater impact on the metric of makespan than idle time. Generally, as more layers imply more dependency

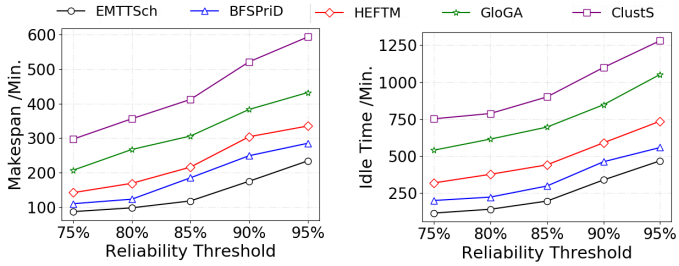


Fig. 9. Experimental Results of Different ε_{thre}.

constraints in task graph, it will lead to large awaiting time in task execution. Thus, the idle time will also increase.

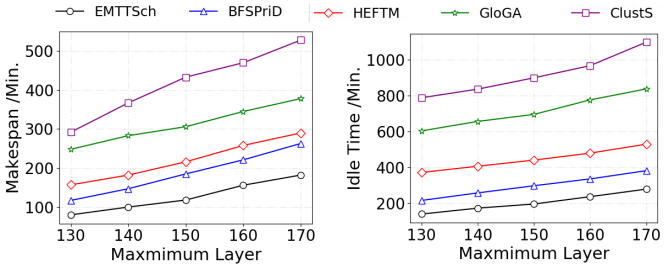


Fig. 10. Experiment Result of Different Maximum Layer.

Next, we also investigate the impact of maximum successor subtasks in task graph, by varying it from 2 to 6 with increments of 1. The experimental results are present in Fig. 11. Obviously, we observe that both the metrics of makespan and idle time increase, with the increase of maximum successor subtasks. One possible reason for this result is that, the larger value of maximum successor subtasks directly import more precedence constraints in task graph, so these two metrics would increase concurrently. What is more, compared with the parameter of maximum layer, the final scheduling performance is more affected by parameter maximum successor subtasks.

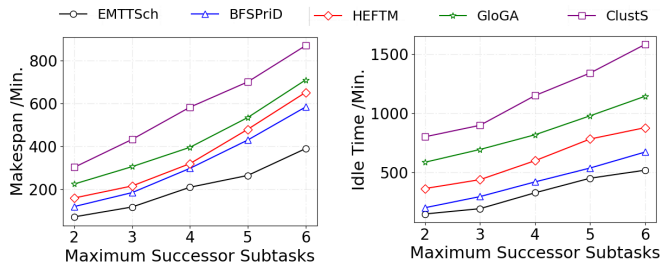


Fig. 11. Experiment Result of Different Maximum Successor Subtasks.

With respect to *EMTTSch* algorithm, in Fig. 12 we present its evolution process with the number of generation increases. To be specific, the involved subtasks are 1,200, and the candidate workers are 1,000. And all the relevant parameters remain unchanged as in Fig. 7 (g)∼(h). Clearly, both the metrics converge within their respective evolutionary solvers, respectively.

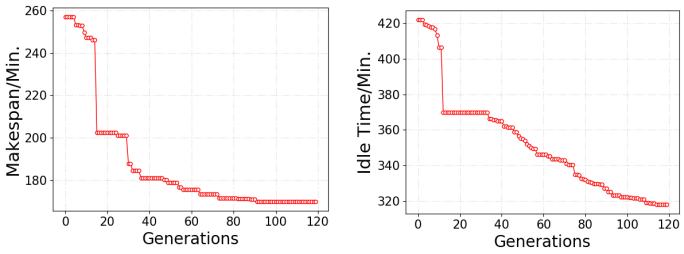


Fig. 12. The Evolution Process of *EMTTSch* Algorithm.

5 Related Work

Online Crowdsourcing. To facilitate crowdsourcing task solving, complex tasks are usually decomposed into many easier subtasks that can be accomplished either sequentially or in parallel by workers [9], [10], [11]. Kittur et al. presented a general purpose framework for complex and interdependent crowdsourcing tasks via micro-task markets [12]. Tran-Thanh et al. considered a task allocation problem in crowdsourcing systems with multiple complex workflows, where each of workflows consists of a group of inter-dependent micro-tasks [13]. By case studies of article writing, decision making, and science journalism, it demonstrates the benefits over a web-based prototype. Kim et al. studied fiction writing tasks on crowdsourcing by breaking a task down into interdependent subtasks, i.e., short fiction writing, on Amazon Mechanical Turk platform [31]. Chatterjee et al. develop a centralized and computationally-efficient scheme to allocate tasks to time-varying crowd agent resources, in a skill-based crowdsourcing platform [32]. Considering precedence and flexibility constraints, a greedy-based allocation algorithm is proposed to match vector-valued service requirements and agents' skills. Because online crowdsourcing does not consider spatial location constraints, thus the proposed approaches are not applicable to MC task scheduling.

Mobile Crowdsourcing Task Scheduling. In recent years, some location-based crowdsourcing tasks have emerged, and brought forth a new paradigm, namely mobile crowdsourcing. The spatial nature of mobile crowdsourcing differs from online crowdsourcing and raises many new and fundamental research questions. Under a limited budget, Miao et al. proposed a budget-aware task scheduling approach for mobile crowdsourcing to optimally schedule tasks to suitable workers, with a special consideration of workers' reputation and proximity to task locations [14]. Cheng et al. investigated a reliable diversity-based spatial crowdsourcing problem, in which spatial tasks are time-constrained, and crowd workers are moving dynamically within different directions [33]. Three approximation methods are proposed to identify worker-task pairs, such that the completion reliability and spatiotemporal diversities are maximized. A unified mobile crowdsourcing task scheduling framework, namely UniTask, is proposed to optimize the overall system performance, including coverage, latency and accuracy metrics [34]. Li et al. developed a complete mobile crowdsourcing task scheduling mechanism, in which it first clusters published tasks to form task groups, and then schedules workers to task groups via an optimal selection mechanism [35]. As dependency constraints be-

tween subtasks have not been considered in mobile crowdsourcing, the above-mentioned scheduling techniques can not be directly applied to our problem setting.

DAG Task Scheduling. For distributed heterogeneous computing systems, DAG task scheduling is in charge of assigning application tasks to processors. Because of its key importance and complexity, DAG scheduling has been studied extensively in the literature. Typically, DAG task scheduling techniques are classified into static and dynamic scheduling. In static scheduling, all the information of DAG task graph and processors are known in advance, and the scheduling decision is determined in an offline mode. While in dynamic scheduling, it schedules tasks on-line as tasks and workers arrive. Generally, static task scheduling algorithms mainly include: 1) List scheduling algorithm [21], [36]. Its basic workflow is to prioritize the scheduling tasks, and then to choose a suitable processor for each task from the ranked list. 2) Duplication based algorithms [23], [39]. This kind of techniques duplicate some of the scheduling tasks in different processors, in order to reduce the communication overhead in a data-intensive scenario. 3) Cluster based scheduling algorithms [22], [37], which groups DAG tasks to a number of clusters, and tasks contained in one cluster are assigned to the same processor. Considering task dependency and communication delay, a heuristic algorithm is proposed to offload task to edge hosts, mobile terminals and cloud serve, etc., with the goal of minimizing completion time [38]. In [40], an incentive management scheme is formalized to implement business tasks with dependencies in a cost-optimal manner. Like task allocation problems in theoretical computer science, in above works, each agent's implementation cost is fixed. Therefore, it calls for sophisticated solutions able to guarantee dynamic mobility of users. As a matter of fact, the most relevant problem scenario we are aware of is that of M Khaledi et al [41], in which an incentive mechanism formed as optimal auction is devised to offload computation jobs to nearby mobile nodes. Although the mobility of agents is seriously considered, this work focuses on the availability of agent resources to determine auction intervals dynamically.

6 Conclusion

In this paper, we propose the problem of the *task graph scheduling in mobile crowdsourcing* (TGS-MC), which assigns location-dependent subtasks contained in one complex M-C task to dynamic mobile workers, such that the whole task completion time, i.e., makespan, and execution idle time are simultaneously minimized, and worker reliability constraints can be met. As the TGS-MC problem is NP-Complete, we propose two heuristic algorithms, including BFS-based priority scheduling algorithm *BFSPriD*, and evolutionary multitasking-based *EMTTSch* algorithm. Experimental results have shown the effectiveness of the proposed approaches compared against baseline algorithms.

Acknowledgments

This work is supported by the National Key Research and Development Program of China (No. 2018YFB2100800), the National Natural Science Foundation of China (No. U2001207, 61972319), and the Fundamental Research Funds for the Central Universities (No.31020180QD139).

References

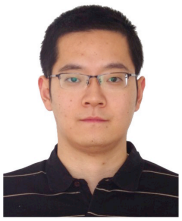
[1] Y Wang, X Jia, Q Jin, et al. Mobile crowdsourcing: framework, challenges, and solutions. Concurrency and Computation: Practice and experience, 2017, 29(3): 1-17.
[2] Y Tong, L Chen, C Shahabi. Spatial crowdsourcing: challenges, techniques, and applications. Proceedings of the VLDB Endowment, 2017, 10(12): 1988-1991.
[3] L Kazemi, C Shahabi. Geocrowd: enabling query answering with spatial crowdsourcing. In Proceedings of the 20th international conference on advances in geographic information systems. ACM, 2012: 189-198.
[4] SK née Müller, C Tekin, et al. Context-Aware Hierarchical Online Learning for Performance Maximization in Mobile Crowdsourcing. IEEE/ACM Transactions on Networking, 2018, 26(3): 1334-1347.
[5] S Ghosh, CK Huyck, M Greene, et al. Crowdsourcing for rapid damage assessment: The global earth observation catastrophe assessment network (GEO-CAN). Earthquake Spectra, 2011, 27(S1): S179-S198.
[6] H Gao, G Barbier, R Goolsby. Harnessing the crowdsourcing power of social media for disaster relief. IEEE Intelligent Systems, 2011, 26(3): 10-14.
[7] C Chen, D Zhang, X Ma, B Guo, L Wang, et al. Crowddeliver: planning city-wide package delivery paths leveraging the crowd of taxis. IEEE Transactions on Intelligent Transportation Systems, 2017, 18(6): 1478-1496.
[8] JF Rougs, B Montreuil. Crowdsourcing delivery: New interconnected business models to reinvent delivery. In Proceedings of 1st international physical internet conference. 2014: 1-19.
[9] H Jiang, S Matsubara. Efficient Task Decomposition in Crowdsourcing, PRIMA 2014, LNAI 8861, pp: 65C73, 2014.
[10] J Cheng, J Teevan, ST Iqbal, et al. Break it down: A comparison of macro-and microtasks. In Proceeding of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI), ACM, 2015: 4061-4064.
[11] Y Tong, L Chen, Z Zhou, HV Jagadish, et al. SLADE: a smart large-scale task decomposer in crowdsourcing. IEEE Transactions on Knowledge and Data Engineering, 2018, 30(8): 1588-1601.
[12] A Kittur, B Smus, S Khamkar, RE Kraut. Crowdforge: Crowdsourcing complex work. In Proceedings of the 24th annual ACM symposium on User interface software and technology. ACM, 2011: 43-52.
[13] L Tran-Thanh, TD Huynh, A Rosenfeld, et al. Crowdsourcing Complex Workflows under Budget Constraints. In Proceedings of Conference on Artificial Intelligence (AAAI). 2015: 1298-1304.
[14] C Miao, H Yu, Z Shen, et al. Balancing quality and budget considerations in mobile crowdsourcing. Decision Support Systems, 2016, 90: 56-64.
[15] L Wang, Z Yu, et al. Multi-objective Optimization based Allocation of Heterogeneous Spatial Crowdsourcing Tasks, IEEE Transactions on Mobile Computing, 2018, 17(7): 1637-1650.
[16] M Walter. Multi-project management with a multi-skilled workforce: a quantitative approach aiming at small project teams. Springer, 2014.
[17] A Janiak, MY Kovalyov. Job sequencing with exponential functions of processing times. Informatica, 2006, 17(1): 13-24.
[18] AM Greenberg, WG Kennedy, ND Bos. Social computing, behavioral-cultural modeling and prediction. In Proceedings of the 6th International Conference Proceedings. April. 2013.
[19] L Wang, Z Yu, D Zhang, B Guo, et al. Heterogeneous Multi-Task Assignment in Mobile Crowdsensing Using Spatiotemporal Correlation, IEEE Transactions on Mobile Computing, 2019, 18(1): 84-97.
[20] SRB Gummidi, X Xie, TB Pedersen. A Survey of Spatial Crowdsourcing. ACM Transactions on Database Systems, 2019, 44(2): 1-46.
[21] S Ijaz, EU Munir. MOPT: list-based heuristic for scheduling workflows in cloud environment. The Journal of Supercomputing, 2018: 1-29.
[22] Huijun Wang, Oliver Sinnen. List-Scheduling versus Cluster-Scheduling. IEEE Transactions on Parallel and Distributed Systems, 2018, 29(8): 1736-1749.

[23] K He, X Meng, Z Pan, L Yuan, et al. A novel task-duplication based clustering algorithm for heterogeneous computing environments. IEEE Transactions on Parallel and Distributed Systems, 2018, 30(1): 2-14.
[24] Y Xu, K Li, L He, L Zhang, K Li. A hybrid chemical reaction optimization scheme for task scheduling on heterogeneous computing systems. IEEE Transactions on parallel and distributed systems, 2014, 26(12): 3208-3222.
[25] U ul Hassan, E Curry. Efficient task assignment for spatial crowdsourcing: A combinatorial fractional optimization approach with semi-bandit learning. Expert Systems with Applications, 2016, 58: 36-56.
[26] L Zhou, L Feng, J Zhong, et al. Evolutionary multitasking in combinatorial search spaces: A case study in capacitated vehicle routing problem. In Proceedings of 2016 IEEE Symposium Series on Computational Intelligence. IEEE, 2016: 1-8.
[27] L Feng, L Zhou, J Zhong, et al. Evolutionary multitasking via explicit autoencoding. IEEE transactions on cybernetics, 2018, 49(9): 3457-3470.
[28] G Wu, X Shen, H Li, et al. Ensemble of differential evolution variants. Information Sciences, 2018, 423: 172-186.
[29] J Peng, J Wang, H Zhang, et al. An outranking approach for multi-criteria decision-making problems with simplified neutrosophic sets. Applied Soft Computing, 2014, 25:336-346.
[30] X Zhou, G Zhang, J Sun, et al. Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT. Future Generation Computer Systems, 2019, 93: 278-289.
[31] J Kim, S Sterman, AAB Cohen, et al. Mechanical novel: Crowdsourcing complex work through reflection and revision. In Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing. ACM, 2017: 233-245.
[32] A Chatterjee, M Borokhovich, et al. Efficient and flexible crowdsourcing of specialized tasks with precedence constraints. IEEE/ACM Transactions on Networking, 2018, 26(2): 879-892.
[33] P Cheng, X Lian, Z Chen, R Fu, et al. Reliable diversity-based spatial crowdsourcing by moving workers. Proceedings of the VLDB Endowment, 2015, 8(10): 1022-1033.
[34] Z Liu, Z Li, K Wu. UniTask: A Unified Task Assignment Design for Mobile Crowdsourcing based Urban Sensing. IEEE Internet of Things Journal, 2019.
[35] M Li, Y Zheng, X Jin, et al. Task Assignment for Simple Tasks with Small Budget in Mobile Crowdsourcing. In Proceedings of 14th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN). IEEE, 2018: 68-73.
[36] H Arabnejad, JG Barbosa. List scheduling algorithm for heterogeneous systems by an optimistic cost table. IEEE Transactions on Parallel and Distributed Systems, 2013, 25(3): 682-694.
[37] H Kanemitsu, M Hanada, et al. Clustering-based task scheduling in a large number of heterogeneous processors. IEEE Transactions on Parallel and Distributed Systems, 2016, 27(11): 3144-3157.
[38] X Qiu, L Zhai, H Wang, et al. Time-Minimized Offloading for Mobile Edge Computing Systems. IEEE Access, 2019: 135439-135447
[39] R Singh. An optimized task duplication based scheduling in parallel system. Int. J. Intell. Syst. Appl.(IJISA), 2016, 8(8): 26-37.
[40] R Khazankin, B Satzger, et al. Optimized execution of business processes on crowdsourcing platforms. In Proceedings of the collaborative computing, 2012: 443-451.
[41] M Khaledi, M Khaledi, et al. Profitable Task Allocation in Mobile Cloud Computing. arXiv: Networking and Internet Architecture, 2016.

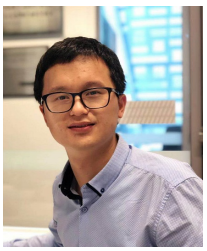
Liang Wang received his Ph.D. degree in computer science from Shenyang Institute of Automation (SIA), Chinese Academy of Sciences, Shenyang, China, in 2014. He is currently an Associate Professor with Northwestern Polytechnical University, Xi'an, China. His research interests include ubiquitous computing, mobile crowd sensing, and data mining.

Zhiwen Yu received the Ph.D. degree in computer science from Northwestern Polytechnical University, Xi'an, China, in 2006. He is currently a Professor and the Vice-Dean of the School of Computer Science, Northwestern Polytechnical University, Xi'an, China. He was an Alexander Von Humboldt Fellow with Mannheim University, Germany, and a Research Fellow with Kyoto University, Kyoto, Japan. His research interests include ubiquitous computing and HCI.

Qi Han received the Ph.D. degree in computer science from the University of California, Irvine, CA, USA, in August 2005. She is an Associate Professor of computer science with the Colorado School of Mines, Golden, CO, USA. Her research interests include mobile crowd sensing and ubiquitous computing.



Dingqi Yang received his Ph.D. in Computer Science from Pierre and Marie Curie University (Paris VI) and Institut Mines-TELECOM/TELECOM SudParis. He is currently a senior researcher with University of Macau, China. His research interests lie in ubiquitous computing, social media data analytics, and smart city applications.



Shirui Pan received his Ph.D. degree in computer science from University of Technology Sydney (UTS), Australia. He is a Research Fellow with the Centre for Artificial Intelligence at UTS. His current research interests include data mining and artificial intelligence.

Yuan Yao received his Ph.D. degree in computer science from the Northwestern Polytechnical University in 2015. He is currently a postdoctoral researcher with Department of Computing, The Hong Kong Polytechnic University, Hong Kong. His research interests are in the area of realtime and embedded system, stochastic modeling and analysis of communication systems.

Daqing Zhang is a Professor at Institut MinesT'el'ecom, TELECOM SudParis, France and Peking University, China. He obtained his Ph.D from University of Rome "La Sapienza" and the University of L'Aquila, Italy in 1996. His research interests include urban computing, contextaware computing, and ambient assistive living. He is the associate editor in 3 international journals including IEEE Pervasive Computing, Proceedings of ACM IMWUT and ACM TIST. He also served in the technical committee for top ubiquitous computing conferences such as UbiComp, Pervasive, PerCom, etc..