2021 IEEE 37th International Conference on Data Engineering (ICDE) | 978-1-7281-9184-3/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICDE51399.2021.00109

On Efficient and Scalable Time-Continuous Spatial Crowdsourcing

Ting Wang*†, Xike Xie*†, Xin Cao‡, Torben Bach Pedersen§, Yang Wang*† and Mingjun Xiao*†
*School of Computer Science and Technology, University of Science and Technology of China, China
†Suzhou Institute for Advanced Study, University of Science and Technology of China, China
‡ University of New South Wales, Australia
§ Aalborg University, Denmark
Email: {tingwt@mail, xkxie@, angyan@, xiaomj@}ustc.edu.cn, xin.cao@unsw.edu.au, tbp@cs.aau.dk

***Abstract*—The proliferation of advanced mobile terminals opened up a new crowdsourcing avenue, spatial crowdsourcing, to utilize the crowd potential to perform real-world tasks. In this work, we study a new type of spatial crowdsourcing, called time-continuous spatial crowdsourcing (TCSC *in short*). It supports broad applications for long-term continuous spatial data acquisition, ranging from environmental monitoring to traffic surveillance in citizen science and crowdsourcing projects. However, due to limited budgets and limited availability of workers in practice, the data collected is often incomplete, incurring data deficiency problem. To tackle that, in this work, we first propose an entropy-based quality metric, which captures the joint effects of incompletion in data acquisition and the imprecision in data interpolation. Based on that, we investigate quality-aware task assignment methods for both single- and multi-task scenarios. We show the NP-hardness of the single-task case, and design polynomial-time algorithms with guaranteed approximation ratios. We study novel indexing and pruning techniques for further enhancing the performance in practice. Then, we extend the solution to multi-task scenarios and devise a parallel framework for speeding up the process of optimization. We conduct extensive experiments on both real and synthetic datasets to show the effectiveness of our proposals.**

I. Introduction

Spatial crowdsourcing or crowdsensing refers to harnessing human knowledge or sensors of participants' smart phones to retrieve qualitative or quantitative details related to physical locations of crowdsourced tasks. For conventional spatial crowdsourcing, task assignment and fulfillment are often "atomic" in that either they are fully executed or not at all [1]–[6], [6]–[11]. In this work, we consider a special type of spatial crowdsourcing, called *time-continuous spatial crowdsourcing* (TCSC *in short*). TCSC is different, due to its temporal continuity, such that a spatial crowdsourced task takes long to finish, necessitating the time-sharing collaboration of multiple workers. It finds broad applications in capturing the presence and duration of environmental features, e.g., air/water pollution monitoring [12] and traffic surveillance [13]- [14], which are prevalent in citizen science projects [15].

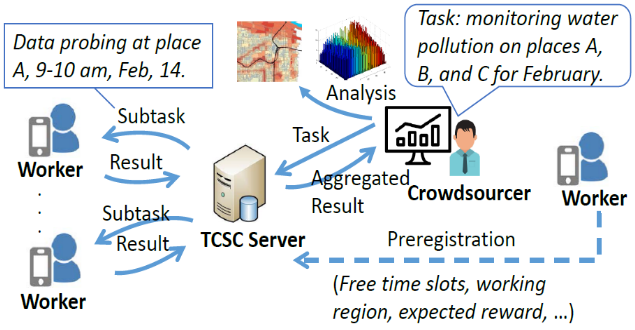

Data probing at place A, 9-10 am, Feb, 14.
Task: monitoring water pollution on places A, B, and C for February.
Analysis
Subtask
Result
Task
Aggregated Result
Worker
Crowdsourcer
Worker
Subtask
Result
TCSC Server
Preregistration
(Free time slots, working region, expected reward, ...)
Worker

Fig. 1. General TCSC Framework

For example, in Fig. 1, a crowdsourcer would like to analyze the microbial content in the water for a period. Upon receiving the task, the TCSC server looks up the records of preregistered workers' spatiotemporal information. Indicating the undertaking of workers giving to the TCSC server, the registered spatiotemporal information consists of workers' available time slots, working regions, and so on, e.g., $\{worker_1, \langle place\ A, 1-2pm\rangle, \langle place\ B, 7-8pm\rangle, \ldots\}$. The task is then decomposed into a set of subtasks. Each subtask corresponds to a specific time slot and location. Subtasks are assigned to appropriate workers according to the assignment policy. Workers finish the assignment, e.g., probing environmental values, and send their results to the server. The crowdsourced results are aggregated and delivered to the crowdsourcer.

Quality is essential for such applications. It is infeasible to accomplish a crowdsourced task for all time slots, due to limited budgets and availability of workers. So, the probed data in crowdsourced results is inherently incomplete. Interpolation (or extrapolation) alleviates the data incompletion problem by inferring unproved values with the probed ones. However, the interpolation error may further affect the data precision, incurring the so-called data deficiency problem. Ignorance of the facts of data incompletion and data imprecision would cause unreliable crowdsourced results. Thus, it is of paramount importance to consider the data quality problem in the TCSC setting. To this end, we propose a general entropy-based metric for summarizing the amount of incompleteness and impreciseness of the crowdsourced results, which enables quality-aware TCSC assignment and balances the plannable expense and observable essence.

There is a substantial difference between the TCSC problem and existing task assignment problems in spatial crowdsourc-

978-1-7281-9184-3/21/$31.00 ©2021 IEEE
DOI 10.1109/ICDE51399.2021.00109

ing. For example, existing works mostly focused on maximizing the total number of completed tasks [1], on maximizing the number of performed tasks for an individual worker [2], or on maximizing the reliability-and-diversity score of assignments [3]. These solutions cannot be directly applied, as none of them look into the temporal continuous nature and corresponding quality issues of the TCSC problem. To our best knowledge, we are the first to study the TCSC problem.

Nevertheless, the computational overhead of quality-aware TCSC assignment is high. Even a simplified version of the problem, i.e., single TCSC task assignment, is NP-hard, as shown in Section III. In this work, we study how the quality-aware TCSC assignment can be handled in an efficient and scalable way. For ease of presentation, we start with the simplified version, single-task assignment, with the target of maximizing the task quality under budget constraints. We prove that the problem is NP-hard and further show that it can be approximated by a polynomial-time solution with guaranteed ratios. We also devise novel pruning and indexing techniques for efficiency enhancement. Based on that, we introduce the multi-task case, where technical challenges arise in handling the correlations between a given set of single tasks. We devise a parallel framework by distributing multiple correlated tasks to independently running computation cores, so as to maximally utilize the independence between tasks.

The main contributions of this paper are as follows.

- We propose and formalize the novel TCSC problem.
- We prove that the problem is NP-hard and therefore study approximation algorithms for accelerating the task assignment with quality guarantees, for both single- and multi-task scenarios.
- We investigate novel indexing and pruning techniques for the efficiency of the single-task case.
- We devise an efficient parallelization framework for the multi-task case, by breaking ties of correlated task groups with devised synchronization mechanisms.
- We conduct extensive experiments on synthetic and real data to evaluate the efficiency and scalability.

The rest of this paper is organized as follows. Section II introduces preliminaries, including concepts, quality metrics, and properties. Section III starts with the single-task assignment. Section IV extends the solution to the multi-task scenario. Section V evaluates our proposals with extensive experiments. Section VI presents related works. Section VII concludes the paper. Table I summarizes the symbols and notations used.

II. Preliminaries

In this section, we introduce basic concepts, propose the quality metric, and investigate its properties. For ease of presentation, we start with the single-task case (Sections II and III), and extend it to the multi-task case (Section IV).

A. Basic Concepts

TCSC tasks and subtasks. A single TCSC task τ has its location $\tau.loc$ and duration $\tau.dur$. According to the batch size that tasks arrive in, the duration consists of at most m equal-sized time slots. Thus, τ can be represented by a set of subtasks $\{\tau^{(j)}\}_{j=1}^{m}$ such that each subtask $\tau^{(j)}$ takes $\tau.loc$ as its location, and the corresponding time slot as its duration. Formally, $\tau^{(j)}.loc = \tau.loc$ and $\tau^{(j)}.dur = \frac{\tau.dur}{m}$.

TABLE I
Summary of Notations

Symbol	Meaning
$\mathcal{T} = \{\tau_i\}$	a set of tasks
$\tau_i = \{\tau_i^{(j)}\}_{j=1}^{m}$	a set of m subtasks of task τ_i
$\tau_i^{(j)}$	a subtask of τ_i at j-th time slot
$c(\tau)$ and $c(\tau^{(j)})$	cost of task τ and subtask $\tau^{(j)}$
$W = \{w_i\}_{i=1}^{n}$	a set of n workers
$w_i^{(j)}$	worker w_i at j-th time slot
m	number of subtasks of a task
n	number of workers
$\|a, b\|_t$	distance in time between slots a and b
$q(.)$	quality metric function
$p(.)$	subtask finishing probability function
$\rho_{err}(.)$	interpolation error ratio function

Worker. Let $W = \{w_1, w_2, ..., w_n\}$ be a set of n workers. Each worker is registered with a set of consecutive states to the SC server, indicating whether s/he is online for providing crowdsourcing services. A worker w_i's temporal state can be represented by $w_i^{(j)}$, indicating the availability of worker w_i at time slot t_j.

Task Assignment. Task assignment is the mapping of workers to subtasks. In Fig. 2, there are 5 subtasks in τ. At each time slot, there exist a set of workers. In this example, workers $w_2^{(2)}$ and $w_4^{(4)}$ are assigned to subtasks $\tau^{(2)}$ and $\tau^{(4)}$, respectively. $\tau^{(3)}$ is not assigned to any worker as none are available at that time slot. $\tau^{(1)}$ is not mapped to any worker because of cost and budget limits, e.g., workers at time slot 1 are far away from $\tau^{(1)}$.

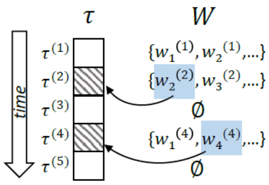


Fig. 2. An Example of TCSC Task Assignment ($m = 5$)

Cost. The cost of a subtask $\tau^{(j)}$ is denoted as $c(\tau^{(j)})$. Following the common setting of spatial crowdsourcing, we assume the travel cost of a subtask $c(\tau^{(j)})$ is the Euclidean distance from the location of a subtask $\tau^{(j)}$ and the assigned worker w[1]. For simplicity, we assume the unit cost of all workers is the same. Our work is general w.r.t. the type of cost. The cost of a task is the summation of all its subtasks' costs, i.e., $c(\tau) = \sum_{j=1}^{m} c(\tau^{(j)})$.

[1] If considering traveling distances as costs [4] [16], the nearest worker is usually selected in order to minimize the cost of taking a subtask. It is also possible to take the second nearest neighbor or the latter in multi-task scenarios, as covered by Section IV.

B. Quality Metric

Due to the limited budget or the availability of workers, it is necessary to measure the *quality* of a TCSC task, given the fact that a task cannot be fully assigned. There can be two possible states for a subtask, *executed* and *unexecuted*, corresponding to whether the subtask value is *probed* by assigned workers or *interpolated* by other probed subtasks. For example, in Fig. 2, $\tau^{(2)}$ and $\tau^{(4)}$ are executed subtasks, while the other 3 are unexecuted and need to be interpolated. Initially, all subtasks are "null". If some subtasks are assigned and probed, their states are changed from "unexecuted" to "executed". Meanwhile, remaining unexecuted subtasks can be inferred from the executed subtasks by interpolation (or extrapolation). So, the quality metric of a TCSC task should, 1) distinguish the quality between a probed item and an interpolated item; 2) be universal for covering the integrative action of different stated subtasks for the overall quality measurement.

Quality Metric. For condition 1), we define the concept of *subtask finishing probability*, based on the calculation of potential interpolation errors. For condition 2), we utilize an entropy function that returns a real-valued score for conveniently indicating the amount of inaccuracies in accordance with specific assignment strategies, as shown below.

Definition 1 **(Task Quality.)** *Let τ be a task consisting of m subtasks, $\tau = \{\tau^{(j)}\}_{1 \le j \le m}$. Each subtask $\tau^{(j)}$ is associated with a finishing probability $p^{(j)}$. The quality of τ, denoted by $q(\tau)$, is:*

$$q(\tau) = -\sum_{j=1}^{m} p^{(j)} \log_2 \left(p^{(j)}\right) \tag{1}$$

Next, we introduce the finishing probability $p^{(j)}$ for subtask $\tau^{(j)}$, that serves as building blocks for the quality metric.

Subtask Finishing Probability. The finishing probability of a TCSC task equals 1, if all its subtasks are done, in correspondence to an ideal case that all subtasks are executed. In practice, for a task consisting of m equally sized time slots, the finishing probability $p^{(j)}$ for each subtask $\tau^{(j)}$ is at most $\frac{1}{m}$. Without losing generality, we can thus use an error ratio ρ_{err} to measure the amount of information loss caused by interpolation errors.

$$p^{(j)} = \frac{1}{m}(1 - \rho_{err}(\tau^{(j)})) \tag{2}$$

Accordingly, the probability $p^{(j)}$ equals 0 for the "null" case, representing the zero knowledge about the subtask, and equals $\frac{1}{m}$ for the executed case, representing the total information gain of finishing the subtask. Next, we show the calculation of error ratio ρ_{err} for an interpolated subtask.

Interpolation Error Ratio. A common way of inferring a missing value from a discrete set of known values is known as inverse distance interpolation [17] [18] [19], which averages the values of its k nearest neighbors, i.e., k nearest subtasks on the timeline of a TCSC task.

Intuitively, the error ratio is proportional to the distances between the interpolated value and its neighboring values [17] [18]. Between two subtasks $\tau^{(i)}$ and $\tau^{(j)}$, the temporal distance is denoted as $|\tau^{(i)}, \tau^{(j)}|_i$, referring to the absolute difference of $\tau^{(i)}$ and $\tau^{(j)}$'s timestamps. For example, in Fig. 2, we have $|\tau^{(1)}, \tau^{(2)}| = 1$ and $|\tau^{(2)}, \tau^{(4)}| = 2$. Then, the interpolation error of an unexecuted subtask can be evaluated by the distances from a set of executed subtasks. Assume function $S_{kNN}(.)$ returns the set of k executed subtasks with the smallest distances. An unexecuted subtask $\tau^{(j)}$ can thus be interpolated by $S_{kNN}(\tau^{(j)})$, with the interpolation error measured by the error ratio function ρ_{err} as follows.

$$\rho_{err}(\tau^{(j)}) = \frac{\sum_{e \in S_{kNN}(\tau^{(j)})} |\tau^{(j)}, e|_i}{k \cdot m} \tag{3}$$

The value range of the error ratio ρ_{err} is from 0 to 1. The error ratio equals 100%, if none of the subtasks of a task is executed. Accordingly, a lower error ratio value is achieved, if the target subtask has more proximate executed subtasks (i.e., smaller interpolation distances). For example, in Fig. 2, $\tau^{(1)}$'s 2-NN results are $\tau^{(2)}$ and $\tau^{(4)}$, whose distances from $\tau^{(1)}$ are 1 and 3, respectively. So, $\rho_{err}(\tau^{(1)})$ can be calculated as $\frac{1+3}{2*100} = 0.02$. Since $\tau^{(2)}$ is an executed subtask, its error ratio equals zero [2]. Similarly, we can derive the error ratios and the subtask finishing probabilities for all the subtasks. Based on that, we can calculate the task quality, by Equation 1.

In summary, the concept of entropy is adopted to quantify the quality of crowdsourced results. The value of $q(\tau)$ ranges from 0, i.e., the lowest information degrees (none of the subtasks are executed), to $log_2 m$, i.e., the highest information degree (all subtasks are executed).

Extension for Worker Reliability. Our quality metric, i.e., Equation 1, is general in addressing the reliability issues [3], where workers are not assumed to be entirely trustable. Instead, each worker w_i is assumed to have a reliability score, represented by $\lambda_i \in [0, 1]$. Incorporating the reliability, the subtask finishing probability can be defined by both worker confidence and potential interpolation error. In particular, given subtask e, the finishing probability of an assigned task is $\frac{\lambda_e}{m}$, where λ_e is the reliability of the worker assigned to subtask e.

Due to the incorporation of worker reliability, for an interpolated subtask $\tau^{(j)}$, the maximum finishing probability is no longer $\frac{1}{m}$, but the product of $\frac{1}{m}$ and the average of $\{\lambda_e\}_{e \in S_{kNN}(\tau^{(j)})}$. The extended form of subtask probability is as Equation 4.

$$p^{(j)} = \frac{1}{m} \cdot \left[\frac{\sum_{e \in S_{kNN}(\tau^{(j)})} \lambda_e}{k} - \rho_{err}(\tau^{(j)})\right] \tag{4}$$

Accordingly, the error ratio is determined by the summation of interpolation distances weighted by the worker reliability λ_e

[2] It is possible that a subtask $\tau^{(j)}$ has less than k nearest neighbors, e.g., at the starting stage of subtask assignment. If $|S_{kNN}| < k$, we let $|\tau^{(j)}, e|_i$ be m, indicating the largest possible interpolation distance.

and then divided by $k \cdot m$, similar to Equation 5. The extended form of error ratio is as Equation 5.

$$\rho_{err}(\tau^{(j)}) = \frac{\sum_{e \in S_{kNN}(\tau^{(j)})} \lambda_e \cdot |\tau^{(j)}, e|_i}{k \cdot m} \tag{5}$$

Notice that probability $p^{(j)}$ equals 0, if $S_{kNN}(\tau^{(j)})$ is an empty set, meaning that null of subtasks are currently done. The probability $p^{(j)}$ is $\frac{\lambda_{\tau^{(j)}}}{m}$, if $\tau^{(j)}$ is an executed subtask. If $\tau^{(j)}$ is interpolated, and the reliability of each worker for executing $S_{kNN}(\tau^{(j)})$ equals 1, Equation 5 degenerates into Equation 3.

Summary. We have shown the effectiveness of the metric in capturing the quality of various scenarios. In the sequel, we show two important properties of the metric, submodularity and non-decreasingness, enabling efficient task assignment algorithms with approximation guarantees.

C. Properties

We derive the properties of the task quality metric. We first recall the definition of submodular functions.

Definition 2 *(***Submodular Function [20].***) Let S be a finite set, and 2^S be the power set of S. A submodular function is a set function $f : 2^S \to \mathbb{R}$, if for any $X, Y \subset S$ satisfying $f(X \cap Y) + f(X \cup Y) \leq f(X) + f(Y)$.*

Lemma 1 *(***Composite Submodular Functions [21].***) Assume a set V and a function $h : 2^V \to \mathbb{R}$ that returns a real value for any subset $S \subseteq V$. If h is a non-decreasing submodular function and ϕ is a non-decreasing concave function $\phi : \mathbb{R} \to \mathbb{R}$, then function $\phi(h(S))$ is non-decreasing and submodular.*

Lemma 2 *Function $q(\tau)$ is non-decreasing and submodular.*

Proof *The quality metric function $q(.)$ is a composite function of the entropy function and the finishing probability function, i.e., $p(.)$. It can be proved that function $p(.)$ is submodular and non-decreasing, as proved by Lemmas 6 and 7 in [22]. More, the entropy function is known as non-decreasing and concave. According to Lemma 1, the quality metric function $q(.)$ is non-decreasing and submodular w.r.t. τ.*

The submodularity property captures the effect of marginal benefits decreasing in the measuring the TCSC quality. As the number of workers on finishing a task is increasing, the marginal value of adding a new worker is decreasing. Also, the submodular property of the quality metric enables efficient optimization, e.g., submodular function maximization, where constant factor approximation algorithms are often available [23]. In the sequel, we discuss how the property is utilized and generalized for optimization problems in the TCSC task assignment applications.

III. Single Task Assignment

We proceed to study the single TCSC task assignment problem. In particular, we consider maximizing the quality of a single task with budget limits. We prove that the problem is NP-hard. For efficient task assignment, we devise heuristic algorithms that approximate the optimization targets in polynomial time with quality guarantees. Nevertheless, we show that the polynomial solution incurs overheads unaffordable for the real-time task assignment scenarios. Therefore, we propose novel indexing and pruning techniques to further enhance the task assignment efficiency.

We formalize and analyze the problem in Section III-A. We devise the approximation algorithm in Section III-B. We study efficient indexing and pruning techniques in Section III-C.

A. Problem Definition and Analysis

Problem 1 Single-task Quality Maximization with Fixed Budgets (sQM *in short***).** *Given a TCSC task τ and a fixed budget b, the sQM problem is to find an assignment for τ, such that the quality $q(\tau)$ is maximized, and the cost $c(\tau)$ does not violate the given budget b.*

$$\begin{aligned} &Maximize \;\; q(\tau) \\ &subject \;\; to \;\; \sum_{j=1}^{m} c(\tau^{(j)}) \leq b \end{aligned}$$

To solve the sQM problem is equivalent to evaluate a task assignment matrix yielding the maximum quality. To give a sense of the size of the solution space, we assume a given set of n workers and a task consisting of m subtasks/timeslots. Thus, there could be an exponential number of possible worker-and-subtask assignment pairs $O(m^n)$. Then, we prove that the sQM problem is NP-hard, by Lemma 3.

Lemma 3 *The sQM problem is NP-hard.*

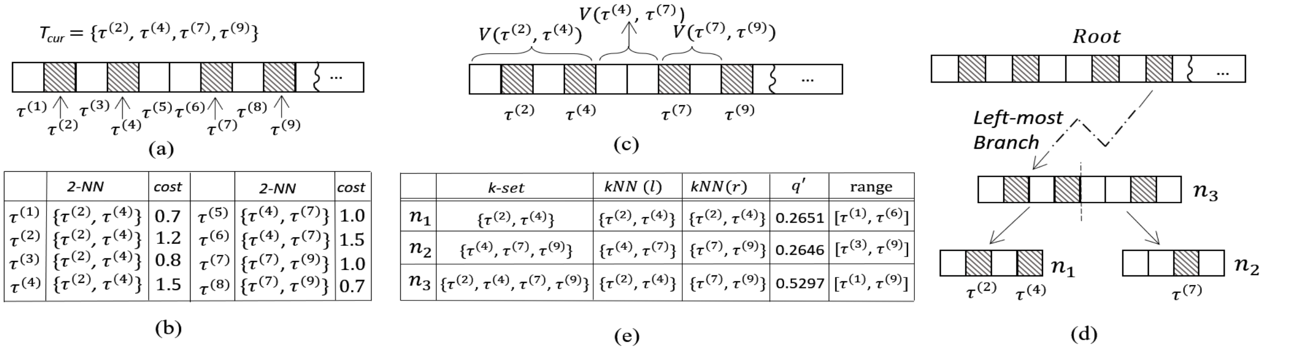
Proof *It is well known that maximizing a submodular function under a cardinality constraint (i.e., selecting at most k elements) is NP-hard [24]. If we consider the workers have unit cost, the budget constraint in our problem becomes the cardinality constraint. It has been proved in Lemma 2 that the quality function $q(\tau)$ (i.e., the objective function to be maximized) is submodular, and thus a special case of our sQM problem is also NP-hard.*

B. Approximation Algorithms

Hereby, we provide a suboptimal solution with guaranteed approximation ratios, based on the submodular property of the quality metric, as discussed in Section II-C. Since the problem is of budgeted maximization for a submodular function, we can have a heuristic algorithm, which repeatedly selects an element (e.g. a subtask) that maximizes the quality increment until the budget is exceeded. The process is detailed in Algorithm 1.

Hence, the heuristic value is defined as the quality increment divided by the corresponding cost. Let the currently assigned set of subtasks be T_{cur}. At each iteration, the algorithm greedily selects a subtask $\tau^{(*)}$ from the set $\tau - T_{cur}$, such that the heuristic value is maximized. Formally, the greedy rule is to find a subtask $\tau^{(*)}$ as follows.

$$\tau^{(*)} = \underset{\tau^{(j)}}{argmax} \frac{\Delta q(\tau)}{c(\tau^{(j)})} = \underset{\tau^{(j)}}{argmax} \frac{q(T_{cur} \cup \tau^{(j)}) - q(T_{cur})}{c(\tau^{(j)})}$$



	2-NN	cost		2-NN	cost
$\tau^{(1)}$	$\{\tau^{(2)}, \tau^{(4)}\}$	0.7	$\tau^{(5)}$	$\{\tau^{(4)}, \tau^{(7)}\}$	1.0
$\tau^{(2)}$	$\{\tau^{(2)}, \tau^{(4)}\}$	1.2	$\tau^{(6)}$	$\{\tau^{(4)}, \tau^{(7)}\}$	1.5
$\tau^{(3)}$	$\{\tau^{(2)}, \tau^{(4)}\}$	0.8	$\tau^{(7)}$	$\{\tau^{(7)}, \tau^{(9)}\}$	1.0
$\tau^{(4)}$	$\{\tau^{(2)}, \tau^{(4)}\}$	1.5	$\tau^{(8)}$	$\{\tau^{(7)}, \tau^{(9)}\}$	0.7

(b)

	k-set	kNN (l)	kNN(r)	q'	range
n_1	$\{\tau^{(2)}, \tau^{(4)}\}$	$\{\tau^{(2)}, \tau^{(4)}\}$	$\{\tau^{(2)}, \tau^{(4)}\}$	0.2651	$[\tau^{(1)}, \tau^{(6)}]$
n_2	$\{\tau^{(4)}, \tau^{(7)}, \tau^{(9)}\}$	$\{\tau^{(4)}, \tau^{(7)}\}$	$\{\tau^{(7)}, \tau^{(9)}\}$	0.2646	$[\tau^{(3)}, \tau^{(9)}]$
n_3	$\{\tau^{(2)}, \tau^{(4)}, \tau^{(7)}, \tau^{(9)}\}$	$\{\tau^{(2)}, \tau^{(4)}\}$	$\{\tau^{(7)}, \tau^{(9)}\}$	0.5297	$[\tau^{(1)}, \tau^{(9)}]$

(e)

Fig. 3. An Example. ($k = 2$, $t_s = 4$, $m = 100$): (a) Current state T_{cur}; (b) 2-Nearest Neighbors; (c) 1-dimensional Order-2 Voronoi Diagram; (d) Tree-structured Index; (e) Auxiliary Information.

```
Algorithm 1: Single task assignment algorithm
   Data: $b > 0$, a set of workers $W$, a task $\tau$
1  Initialize the states of subtasks $\{\tau^{(j)} \in \tau\}$ as NULL and
   initialize $T'_{cur}$ and $T_{cur}$ as two empty sets;
2  For each subtask $\tau^{(j)}$ get the corresponding cost $c(\tau^{(j)})$;
3  Execute the subtask $\tau^{(h)}$ yielding the highest quality but not
   exceeding the budget, $T'_{cur} \leftarrow \{\tau^{(h)}\}$;
4  while $C(\tau) \leq b$ do
5      for $\tau^{(j)} \in \tau - T_{cur}$ do
6          Compute $\frac{q(T_{cur} \cup \tau^{(j)}) - q(T_{cur})}{c(\tau^{(j)})}$;
7      $\tau^{(*)} \leftarrow argmax\left\{\frac{\Delta q(\tau)}{c(\tau^{(*)})} : \tau^{(*)} \in \tau\right\}$;
8      Update $\tau^{(*)}$'s state to Executed;
9      $T_{cur} \leftarrow T_{cur} \cup \tau^{(*)}$;
10 return $T'_{cur}$ or $T_{cur}$ with the highest quality as the final result;
```

We use an example to illustrate the process. Assume that there are 4 executed subtasks, i.e., $T_{cur} = \{\tau^{(2)}, \tau^{(4)}, \tau^{(7)}, \tau^{(9)}\}$, of a TCSC task, represented by shaded slots, as shown in Fig. 3 (a). Algorithm 1 enumerates all remaining slots/subtasks. At each iteration, a slot is selected for tentative execution in order to find the one with the highest heuristic value. If $\tau^{(1)}$ is chosen and tentatively executed, $\rho(\tau^{(1)})$ is reduced to 0 and the heuristic value increment is 0.0016. However, if $\tau^{(5)}$ is tentatively executed, $\tau^{(6)}$'s 2-NN result is changed from $\{\tau^{(4)}, \tau^{(7)}\}$ to $\{\tau^{(5)}, \tau^{(7)}\}$. Therefore, both $\rho(\tau^{(5)})$ and $\rho(\tau^{(6)})$ should be recalculated for getting the quality and the heuristic values. The process is repeated for all the unexecuted slots. Finally, $\tau^{(1)}$ is selected, since it derives the maximum heuristic value in this example. By setting the greedy strategy as such, Algorithm 1 guarantees a $(1-1/\sqrt{e})$ approximation to the optimal solution, as shown in [23].

Complexity Analysis. First, the number of iterations, i.e., the outer loop of Algorithm 1 (line 4), is at the level of $O(m)$. Second, the inner loop (line 5) is at the level of $O(m)$, since one needs to try all m subtasks and get their heuristic values in order to find the one maximizing the overall quality. Third, for each trial of a subtask, one needs to calculate the corresponding heuristic value, i.e., the overall quality increase of implementing the subtask, according to the quality metric function. The overall quality increase is the summation of individual quality increments of all other $m-1$ subtasks, so the complexity is $O(m)$ (line 6). Fourth, for quality increment of a individual subtask, k timeslots in the neighborhood should be visited. In our implementation, we maintain a sorted list for subtasks that are sorted in the ascending order of the corresponding time slots. During the query evaluation, $O(log(m))$ cost is used for finding the nearest assigned subtask, and then $O(k)$ cost is used for refining the exact k-NN. Therefore, the total time complexity is $O(m^3 log(m))$.

In summary, the approximation algorithm gives a polynomial alternative for tackling the NP-hard optimization problem. However, the computational overhead makes it impractical in real-time task assignment scenarios. In the sequel, we propose a series of techniques for better efficiency and scalability.

C. Efficient Heuristic Value Calculation

The idea of accelerating the algorithm is in two parts. First, for finding the maximum heuristic value, how to avoid unnecessary enumeration of all m subtasks. Second, the calculation of a heuristic value refers to the summation of partial qualities of all subtasks. For the heuristic value calculation, how to maximally reuse the computation so as to avoid unnecessary checking of all time slots.

Locality of k-NN Searching. We try to scale down the problem by considering the locality of k-NN searching. We observed from Fig. 3 (b) that if two slots are proximate, their k-NN results tend to be similar. For example, $\tau^{(1)}$ and $\tau^{(2)}$ share the same k-NN results, i.e., $\{\tau^{(2)}, \tau^{(4)}\}$. Theoretically, the solution space of k-NN searching over the m subtasks is a one-dimensional order-k Voronoi diagram. The domain space is a one-dimensional interval, i.e., from 1 to m. The diagram splits m slots into disjoint intervals, called Voronoi cells, such that the k-NN searching provides the same result if queries are within the same Voronoi cell. In Fig. 3 (c), for example, the Voronoi cell $V(\tau^{(2)}, \tau^{(4)})$ covers slots from $\tau^{(1)}$ to $\tau^{(4)}$, meaning that all slots from $\tau^{(1)}$ to $\tau^{(4)}$ take $\{\tau^{(2)}, \tau^{(4)}\}$ as their 2-NN results.

Such a structure facilitates the algorithm evaluation in two aspects. First, the time cost for k-NN searching can be reduced from $O(log(m))$ to constant time $O(1)$, as the diagram precomputes the solution space for the k-NN queries. Second, the diagram accelerates the calculation of heuristic values. For the tentative execution of a subtask, the heuristic value is calculated by enumerating all other slots which takes $O(m)$ (Algorithm 1, line 6).

Recall that the finishing probability of an unexecuted subtask depends on its k-NN interpolation. So, the value of finishing probability of an unexecuted subtask does not change, if the order-k Voronoi cell to which it belongs does change, with the tentative execution. This way, the problem of quality increment calculation is transformed to the reformulation of Voronoi cells w.r.t. the tentative subtask execution, which can be handled locally, since the diagram handles such updates locally [25]. We cover more details below.

The technical challenge arising is that there could be a large number of order-k Voronoi cells, making the gains of local computation not worthy of the overhead of the Voronoi diagram construction. In particular, the average number of order-k cells is $O(k(m-k))$ [25].

Approximated One-dimensional Voronoi Diagram. To handle that, we propose an approximate version of one-dimensional order-k Voronoi diagram. The idea is to use an aggregated binary tree for Voronoi cell indexing and Voronoi diagram approximation (Fig. 3 (d)). In the tree, each node represents a time segment $[l, r]$, where l and r are for the two slots on the segment's left and right ends, respectively. The root node is the interval of the entire m slots. For each node, we store the auxiliary information in the form of a quadruples, i.e., $\langle k\text{-}set, knn(l), knn(r), q' \rangle$, as shown in Fig. 3 (e).

The k-set of a node is the union of k-NN results for all its offsprings. $knn(l)$ and $knn(r)$ are k-NN results of the two end slots. The k-NN results are sorted in ascending order of the distance to l (or r), so that the distance from l (or r) to its k-th nearest neighbor can be fast retrieved, denoted by $k_{max}(l)$ (or $k_{max}(r)$). So, we can derive the influence range of a node as $[max(1, l - k_{max}(l)), min(m, r + k_{max}(r))]$, such that the quadruple of the node can be affected if a tentatively executed slot is within the influence range. q' is the partial quality value of the node. For subtask $\tau^{(j)}$, its partial quality equals $p^{(j)}log(p^{(j)})$. The quality of a node is the summation of partial qualities of all subtasks in its offspring. This way, the quality value can be fast retrieved by querying the upper level nodes of the tree structure. Upon updating, only necessary subtrees are retrieved and revised, following the style of updating an aggregated tree-structure.

Maximum Heuristic Value Calculation. With the tree structure, it takes much less computational overhead than enumerating all m slots. The process is implemented by traversing the tree in a best-first manner with an associated heap. The elements of the heap are tree nodes which are sorted in descending order of the upper bounds. The higher an element ranks in the heap, the more likely it corresponds to the maximum heuristic value increment.

Next, we study how to upper bound the effect of a tentative insertion. A tentative insertion affects the k-NN interpolation results so the quality value varies. The effect can be categorized into *inter-node* and *intra-node* cases[3].

For the intra-node case, if an unexecuted slot is tentatively executed, of the same node, another unexecuted slot's k-th NN distance is reduced to 1 at most. It corresponds to the extreme case that the slot is next to the tentative executed slot, whereas the distance between them is 1. So, the interpolation error ratio can be lower bounded as follows.

$$\rho_{err}(\tau^{(j)}) \geq \frac{1}{k \cdot m}\Big[\sum_{e \in S_{(k-1)NN}(\tau^{(j)})} |e, \tau^{(j)}|_i + 1\Big] \quad (6)$$

Therefore, the upper bound of a node's quality can be derived by the lower bound of ρ_{err} (Equation 6), since the value of $q(.)$ is inversely proportional to that of $\rho_{err}(.)$. So, the upper bound of a node's heuristic value is calculated by the maximum quality change divided by the minimum cost of all unexecuted subtasks in the node.

For the inter-node case, a tentatively executed slot of a node would also change the k-NN interpolation result of other nodes. For example, in Fig. 3 (d), we have two leaf nodes, n_1 and n_2. If $\tau^{(3)} \in n_1$ is tentatively executed, $\tau^{(5)} \in n_2$'s second nearest neighbor changes from $\tau^{(7)}$ to $\tau^{(3)}$. The corresponding quality value change should also be incorporated and updated.

At each iteration, the top element of the heap is popped up and the child nodes are inserted. If the top element is a time slot/subtask, the exact heuristic value increase can be obtained. Suppose the value is θ. Then, all other elements in the heap whose upper bounds are below θ can be pruned. The process repeats until the heap is empty. The slot that gives the maximum heuristic value is then obtained.

The advantages of using the tree structure are two-fold. First, using best-first searching for obtaining the slot with the potentially maximum heuristic value is expected to take $O(log(m))$ time. Second, the locality of k-NN searching is reflected by the decomposition of tree leaf nodes. Suppose that calculating the updated heuristic value of a leaf node takes constant time. Then, instead of enumerating $O(m)$ slots for calculating the updated heuristic value, one takes $O(log(m))$ time to retrieve and update the relevant leaf nodes, reducing the corresponding computation overhead from $O(m)$ to $O(log(m))$. The overall cost is $O(mlog^3(m))$.

Tree Construction. We consider the tree construction in an incremental manner. Suppose that a time slot e is to be tentatively executed, meaning that e is with the maximum current heuristic value. The process of tree construction is triggered accordingly, and is revoked recursively. At each iteration, we test if a subtree will be affected. If yes, we update the associated quadruples and forward the updates to the descendants. Otherwise, the entire subtree is skipped.

We summarize two cases to determine whether the current node will be affected and therefore updated by the tentative execution. Based on that, we can disqualify irrelevant nodes at a higher level to save the overhead of the tree construction.

[3] We consider the leaf node for ease of presentation. The calculation of upper bounds for non-leaf nodes can be done in a bottom-up manner, as the construction process of an aggregated tree-structure.

- *Case 1.* A node will be affected, if the tentatively executed slot is within the influence range of the node.
- *Case 2.* A node will not be affected, if its parent node is not affected, by e.

By doing so, we can find appropriate nodes for updating. The updates can thus be propagated to the leaf level so that the node splitting should be handled.

Splitting and Stopping Conditions. During splitting, a node is decomposed into two sub-nodes, with its time segment split into two equal sized sub-segments. The quadruple of the node can thus be partially inherited by its sub-nodes. For example, the left sub-node inherits the $knn(l)$ of its parent, because they share the same left end slot. In particular, we consider two stopping conditions.

- *Condition 1.* For a node with segment $[l, r]$, splitting stops, if $knn(l) = knn(r)$.
- *Condition 2.* If the length of a node's segment is smaller than a pre-specified threshold, t_s, splitting stops.

Condition 1 guarantees that current segment belongs to the same order-k Voronoi cell, so that there is no need for further splitting. Condition 2 limits the depth of the tree structure to $\lceil log_{t_s}(m) \rceil$. It serves as a knob for tuning the approximation accuracies so as to control the construction overhead of the tree structure. The correctness of Condition 1 is guaranteed by Lemma 8 in [22].

IV. MULTIPLE TASK ASSIGNMENT

The multi-task assignment problem is essential to the practical deployment of crowdsourcing platforms, where multiple tasks are submitted, scheduled, and executed, simultaneously. However, the computational overhead of multi-task assignment is high, even its simplified version, i.e., single-task assignment, is NP-hard. Even with the approximation solution, the algorithm has to iteratively retrieve a subtask from all given $|\mathcal{T}|$ tasks that maximizes the heuristic value, making the algorithm scale quadratically with $|\mathcal{T}|$, which is not scalable for handling a large number of tasks. A practical way of handling multi-task case is to fully exploit the hardware capabilities of the TCSC server, with parallel computing techniques.

In this section, we study the multi-task assignment scenario, by considering two variants regarding the settings of optimization targets. The first variant evaluates the overall quality by the summation of qualities of individual tasks, that belong to the given task set. The second variant improves the overall quality by reinforcing the "weakest" single task, i.e., maximizing the minimum single task quality. Both variants are on improving the overall quality of the given set of tasks with budget constraints[4]. In particular, we use q_{sum} and q_{min} to represent the optimization target functions, respectively.

A. Maximizing Summation Quality

The first optimization target is on maximizing the summation quality of all tasks as following.

Definition 3 Summation Quality. *Given a set of tasks $\mathcal{T} = \{\tau_1, \tau_2, ...\}$, we define the summation quality as:*

$$q_{sum}(\mathcal{T}) = \sum_{i=1}^{|\mathcal{T}|} q(\tau_i | \tau_i \in \mathcal{T}) \tag{7}$$

Problem 2 Multiple-task Summation Quality Maximization with Fixed Budgets (MSQM *in short***).** *Given a set of tasks $\mathcal{T} = \{\tau_1, \tau_2, ...\}$, the MSQM problem is to find an assignment for the tasks in $\mathcal{T}$, such that the summation quality is maximized, and the overall cost $\sum c(\tau_i)$ does not exceed the given budget b.*

$$\begin{aligned} &Maximize \;\; q_{sum}(\mathcal{T}) \\ &subject\ to \sum_{i=1}^{|\mathcal{T}|} c(\tau_i) \le b \end{aligned} \tag{8}$$

We can prove that the MSQM problem is NP-hard, by reducing it from the sQM problem, whose NP-hardness is proved by Lemma 3.

Lemma 4 *$q_{sum}(.)$ is submodular and non-decreasing.*

Proof *We have proved that $q(.)$ is non-decreasing and submodular, by Lemma 2. The summation function is known as both convex and concave. The $q_{sum}(.)$ function is a composite function of a summation function and $q(.)$ function. So, the lemma is proved, according to Lemma 1.*

Based on the properties of submodularity and non-decreasingness, the framework of single task assignment, i.e., Algorithm 1, can be applied for handling the multi-task assignment case. The heuristic value is set as the increase of the summation quality divided by the corresponding cost (of a tentatively selected subtask), following the same greedy strategy. Then, the algorithm is to iteratively retrieve a subtask from all given $|\mathcal{T}|$ tasks that maximizes the heuristic value, so that the solution space is $|\mathcal{T}|$ times the size of the single task case, making the algorithm scale quadratically with $|\mathcal{T}|$. The time complexity is $O(|\mathcal{T}|^2 m log^3(m))$. To improve the scalability, we aim to derive a parallelization framework for distributing the calculation workload onto multiple computation cores. Ideally, each task can be running independently on different cores, so that the time cost would be $\frac{|\mathcal{T}|}{\text{\# of cores}}$ times that of running a single task, assuming the value of $|\mathcal{T}|$ is larger than the number of cores. But, there occurs correlations between tasks, if two subtasks running on different cores "compete" for one worker at some time slot. It happens because it is possible that two subtasks choose the same worker with lowest costs for minimizing the budget decrement. We call this *worker conflicting*, as exemplified in Fig. 4 (a), where there are three tasks (τ_1 to τ_3) and two workers (w_1 and w_2). There exist conflicts between τ_2 and τ_3, since they both take w_1 as the worker with the lowest cost.

1) Group-level Parallelization: We can have a graph of independent groups, if taking each task as a node and drawing an edge between any two conflicted tasks. If a group of nodes do not have any connections with other groups, it is an independent group. The optimization process of such a group

[4] A dual version of our problem can be minimizing the task costs with quality constraints. It can be handled with the primal-dual method [26], which reduces the problem to the one studied in this work.

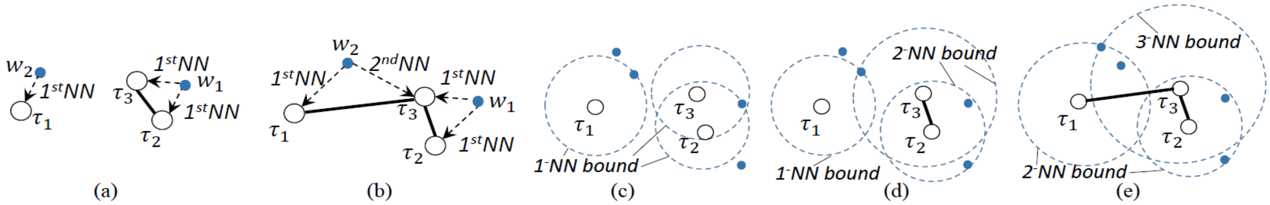


Fig. 4. Worker Conflicting and Group-level Parallelization: an example of considering travel distances between workers and subtasks as costs.

is independent of others and therefore independent groups can be run in parallel. For example, in Fig. 4 (a), tasks can be divided into two independent groups, $\{\tau_1\}$ and $\{\tau_2, \tau_3\}$, if any pair of tasks do not compete for workers with lowest costs.

Is it sufficient for considering the conflicts on the lowest costs for deriving the independent groups? The answer is NO. For example, in Fig. 4 (b), τ_2 and τ_3 have conflicts on w_1, so τ_3 opts for w_2, who ranks as the worker with the second lowest cost. Unfortunately, τ_3 further have conflicts with τ_1, which has w_2 as the worker with the lowest cost. In general, if two subtasks belonging to different tasks are to be executed, one of them has to choose a worker with the second lowest cost, or even the latter.

One may get the independence graph by gradually expanding the searching regions. For example, assume costs are calculated by the travelling distances from workers to tasks [16] [4]. Then, a subtask takes the nearest worker as the worker with the lowest cost. For a given task, we call the circle centered at the task's position with the distance between the task and its nearest neighbor as its 1-NN bound. The independence graph can be obtained by the following steps. Initially, we draw 1-NN bounds for each task, as shown in Fig. 4 (c). An edge between τ_2 and τ_3 is added, since the two tasks share the same worker that causes conflicts. Next, we draw 2-NN bounds for both τ_2 and τ_3, so that there exist enough workers for being assigned to the two conflicting tasks, as shown in Fig. 4 (d). However, it shows that another worker is within τ_1's 1-NN bound and τ_3's 2-NN bound, so the two tasks have conflicts and the edge between them is added. After that, we draw 2-NN bound for τ_1 and 3-NN bound for τ_3, as shown in Fig. 4 (e). In general, if a node of the independence graph is with degree d, the (d+1)-NN bound should be drawn. The process repeats until no conflicts are detected. A drawback of the gradual expanding method is on incurring large groups and heavyweight computation tasks, deteriorating the parallelization performance.

2) Task-level Parallelization: Hereby, we devise the task-level parallelization framework, as depicted in Fig. 5. We set a thread pool, with a master thread and a set of worker threads waiting for tasks to be concurrently executed. The master thread is for maintaining the thread pooling on heartbeat monitoring, conflicting controlling, scheduling, and logging. To support the functionalities of the master thread, there are several associated data structures, *Heartbeat Table*, *Conflicting Table*, and *Logging Table*.

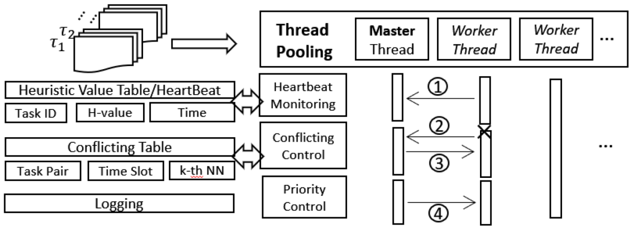


Fig. 5. Task-level Parallelization: ① Periodically Sending Heartbeats to Master Thread; ② Reporting to Master Thread that Conflicts Detected; ③ Looking Up Conflicting Table and Heartbeat Table, then Ask Worker Thread to Continue or Suspend; ④ Adjusting Priorities of Worker Threads.

Heartbeat Table stores periodically reported heuristic values from currently executed tasks. Logging Table traces the historical records of Heartbeat table. Conflicting Table stores a series of records for breaking the ties of conflicts. Assume three tasks, τ_1, τ_2, and τ_3, are conflicted at time slot t. The information, including the conflicting task sets and corresponding conflicting time slot, is stored at the Conflicting Table of the master thread. In this example, a tuple $\langle\{\tau_1, \tau_2, \tau_3\}, t, 1\rangle$ is recorded in the conflicting table. Here, 1 means the three tasks are to compete for the worker of 1-NN. Then, during the task processing, if τ_1 is to execute conflicting slot t, its associated thread sends a message to inform the master thread. Upon receiving the message, the master thread looks up the Conflicting table and the Heartbeat table to retrieve the current heuristic values of τ_1, τ_2, and τ_3. If τ_1's current heuristic value is higher than τ_2 and τ_3, it continues with the execution of slot t. Meanwhile, the master thread checks the availability of workers and updates the record on the conflicting table, by changing the field "*k-th NN*" from 1 to 2, so that τ_2 and τ_3 would compete for the worker with the 2^{nd} lowest cost next time, because the first one has been taken by τ_1. Otherwise τ_1 is suspended and the process continues.

Discussion. The task-level parallel approach is deterministic, meaning that the parallelized task assignment plan is consistent with the non-parallel plan. The master thread periodically stores and descendingly sorts the heuristic values that are collected in Heartbeat Table, so that the derived plan is the same as the serialized task execution (Algorithm 1). This way, the parallel algorithm follows the approximation framework with guaranteed ratio. On the other hand, in the parallel environment, it is hard to strictly control the stopping condition, i.e., the timeline when the given budget is exhausted. It is unavoidable that threads with lower heuristic values are executed earlier than those with higher values. But this can mostly be alleviated with our priority settings. We set priorities in accordance with the heuristic values of worker threads dynamically, so that the tasks with higher heuristic values are more likely to be processed. This is also consistent with the greedy strategy of Algorithm 1. The priorities of worker threads are initialized as infinity to avoid thread starvation.

B. Maximizing Minimum Quality

The second optimization target is on maximizing the minimum quality of all tasks, so that the overall quality is optimized. The problem is formalized.

Definition 4 *(Minimum Quality) Given a set of tasks* $\mathcal{T} = \{\tau_1, \tau_2, \dots\}$*, we define the minimum quality as:*

$$q_{min}(\mathcal{T}) = min\left\{q(\tau_i)\middle|\tau_i \in \mathcal{T}\right\} \tag{9}$$

Problem 3 Multi-task Minimum Quality Maximization with Fixed Budgets (MMQM). *Given a set of tasks* $\mathcal{T} = \{\tau_1, \tau_2, \dots\}$*, the MMQM problem is to find a task assignment for each task* $\tau_i \in \mathcal{T}$*, such that the minimum quality is maximized, and the overall cost* $\sum c(\tau_i)$ *does not exceed the given budget b.*

$$Maximize\ q_{min}(\mathcal{T})$$

$$Subject\ to\ \sum_{i=1}^{|\mathcal{T}|} c(\tau_i) \le b$$

We can prove the NP-hardness of the MMQM problem by reducing it to the sQM problem. The submodularity and non-decreasingness of $q_{min}(.)$ function can be proved by Lemma 5. Hence, the $(1-1/\sqrt{e})$ approximation ratio of q_{min} is achieved by iteratively executing the selected subtask from the task yielding the minimum quality. The subtask execution follows the framework of Algorithm 1. To fast retrieve the task with minimum quality, we maintain a heap for $\mathcal{T}$ tasks. Notice that there is no worker conflict issues for the MMQM problem, since the subtasks are executed in a sequence. So, the total time complexity is $O(mlog^3(m)log(|\mathcal{T}|))$.

Lemma 5 $q_{min}(\mathcal{T})$ *is submodular and non-decreasing.*

Proof *We have proved that* $q(.)$ *is non-decreasing and submodular, by Lemma 2. The minimization function is known as a concave function. The* $q_{min}(.)$ *function is a composite function of a minimization function and* $q(.)$ *function. So, the lemma is proved, according to Lemma 1.*

V. Experiments

We cover the experimental setup in Section V-A, and report the performance of our proposals in Sections V-B and V-C.

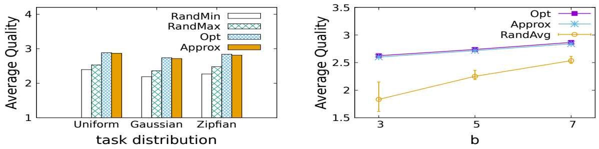
A. Experiment Setup

Dataset.[5] We use a real dataset[6] of 10,357 worker trajectories for representing workers' movements. For each worker trajectory, we randomly cut out a set of pieces, ranging from 1 to 5 time slots, as a worker's active slots. We use a public data generator[7] to generate a series of datasets to simulate the locations of TCSC tasks, following **uniform**, Gaussian, and Zipfian distributions. For parameters of Gaussian distribution, the mean is set as the domain center and the sigma is set as the $1/6$ of the domain sidelength, so that most of generated data are within the domain space. For Zipfian distribution, the exponent is set to 1, which is a common setting, and the only option of the generator. We also use a Beijing POI dataset for representing tasks' locations[8]. We set the cost for an assignment to be the distance that a worker moves to the assigned task, following the common setting of spatial crowdsourcing [16] [4]. We vary the number of TCSC tasks to test the scalability of our proposals by setting the number of tasks as 100, **300**, and 500, respectively. For each TCSC task, we set the task length (i.e., the number of subtasks) to 300, **500**, and 1000, respectively. The budget is set to \$50, **\$100**, \$200, corresponding to about 12.5%, 25%, and 50% of the average cost of a TCSC task in the default setting. By default, k is set to 3 for the k-NN interpolation, t_s is set to 4. and the number of cores is set to 10 for multi-task parallelization.

Implementation. All algorithms are implemented in Java and run a PC with Intel(R) Xeon(R) CPU $E5$-$2698v4$ @ 2.20GHz and 256GB main memory. By default, we use 12 cores for running experiments on multi-task assignment. Each reported value is the average of 20 runs.

B. Results on Quality

We test the effectiveness of our quality-aware task assignment method in Figure 6. We compare the quality of our method, Approx, with two competitors, OPT and Rand. OPT offers the optimal result by traversing the solution space. Rand accomplishes a task by randomly assigning a subtask to its nearest worker. The results with different data distributions is shown in Fig. 6 (a). In all testing, Approx achieves a high quality result which is: 1) close to the optimal result; 2) better than randomized heuristic algorithms. The randomized heuristic algorithm does not offer a deterministic solution, and therefore incurs quality fluctuations. The gap between Approx and Rand is bigger, if the budget is smaller, as shown in Fig. 6 (b), which is the essential scenario in TCSC problem.



(a) Quality vs. Distributions

(b) Quality vs. Budget

Fig. 6. Quality of Single-task Case

We further test the results for the multi-task case in Fig. 7 (a-d). In Fig. 7 (a) and (c), it can be observed that the quality of Approx is much better than its competitors, for both q_{sum} and q_{min} cases. We also examine how the quality change w.r.t. the

[5] Default parameters are bolded.

[6] https://www.microsoft.com/en-us/research/publication/t-drive-trajectory-data-sample/

[7] http://chorochronos.datastories.org/sites/default/files/algorithms /SpatialDataGenerator.zip

[8] https://ieee-dataport.org/documents/beijing-poi-datasets-geographical-coordinates-and-ratings

budget in Fig. 9 (b) and (d). In all cases, Approx gives much better quality than baselines. The gap between them can be smaller, if the budget is sufficiently large, which is consistent with the problem setting.

In summary, Approx offers a high quality task assignment solution, with a deterministic output and theoretical guarantees, which outperforms the baselines.

C. Results on Efficiency

We examine the efficiency and scalability of our proposal, by comparing two variants Approx and Approx*. The Approx solution is described by Algorithm 1, but without optimization techniques in Section III-C. Approx* improves Approx by: 1) using tree-structured order-k Voronoi diagrams to avoid redundant k-NN pre-computation; 2) using best-first searching and upper bound pruning for identifying the one with largest heuristic value. We test the efficiency in single-task assignment in Fig. 8 and multi-task assignment in Fig. 9.

Single task assignment. First, we test the efficiency of Approx and Approx*, by varying the number of subtasks (m) in Fig. 8 (a). Approx* improves over Approx by two orders of magnitude. As m increases, the improvement is more significant. It shows that the optimization techniques, i.e., the tree-structured order-k Voronoi diagram (Section III-C), bring in good scalability to the approximation framework. Second, we test the efficiency by varying the number of workers in Fig. 8 (b). The time cost keeps stable and increases only slightly w.r.t. $|W|$. The reasons are two-fold: 1) the increasing trend is moderate due to the good scalability of best-first NN searching algorithm; 2) the slight increase shows that, with larger $|W|$, the completion ratios of tasks increase, and are with higher costs. In all cases, Approx* outperforms Approx by at least two orders of magnitude, showing good efficiency and scalability in terms of m and $|W|$.

To understand how the efficiency is achieved, we make detailed analysis in Fig. 8 (c-e). The improvements made by Approx* are in two parts, as shown in Fig. 8 (c). First, Approx* utilizes the implementation of the approximation of order-k Voronoi diagram, and thus maximally reuses the computation of k-NN results. It can be observed that the cost of the interpolation (i.e., finding k nearest subtasks) can be reduced by 4 orders of magnitude. Second, the tree-based pruning techniques can further reduce the cost of heuristic value calculation by more than an order of magnitude. The little extra cost for Approx* on the tree-structure is well-spent, given the efficiency gained in the total execution time.

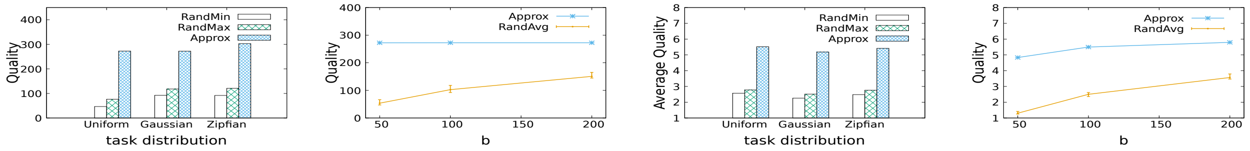
To examine the pruning effects supported by the tree structure, we report the pruning ratios, by varying m on different task distributions in Fig. 8 (d). The ratio is calculated by the dividing the number of slots executed with pruning (Section III-C) by the one without pruning. It can be observed that our methods prune away more than 70% subtask execution and therefore effectively accelerate the entire task processing. Similar trend is observed for the result on the real data. We report the time cost spent on the tree-structure construction by varying the value of fanout of the tree structure, t_s, in Fig. 8 (e). In all testing, the construction time is no more than 25 ms. Also, the time decreases w.r.t. the increase of t_s, since a larger t_s corresponds to a smaller number of tree nodes and therefore less construction time.

We continue to examine the effects of other factors on the efficiency. In Fig. 8 (f), we compare the two solutions by varying the distributions of tasks' locations. In all cases, the performance of Approx* dominates that of Approx by more than two orders of magnitude. More, the time cost of Approx* remains relatively stable with tasks' location distributions. We also test the effect of parameter k for data interpolation, in Fig. 8 (g). The time cost increases with k, since the cost of k-NN interpolation is higher for a bigger k. We study the effect of budgets in Fig. 8 (h). The time cost increases moderately w.r.t. b, since the number of executed subtasks also increases w.r.t. b. Zipfian distribution has the lowest construction time. A task tends to incur higher cost under skewed distributions, so that the number of executed subtasks is reduced and the corresponding time cost is less. In summary, Approx* dominates Approx in different parameter settings, and has better adaptivity to the skewness of data distributions.

Multiple task assignment. We provide the results on the summation quality case in Fig. 9 (a-f) and the results on the minimum quality case in Fig. 9 (g-h).

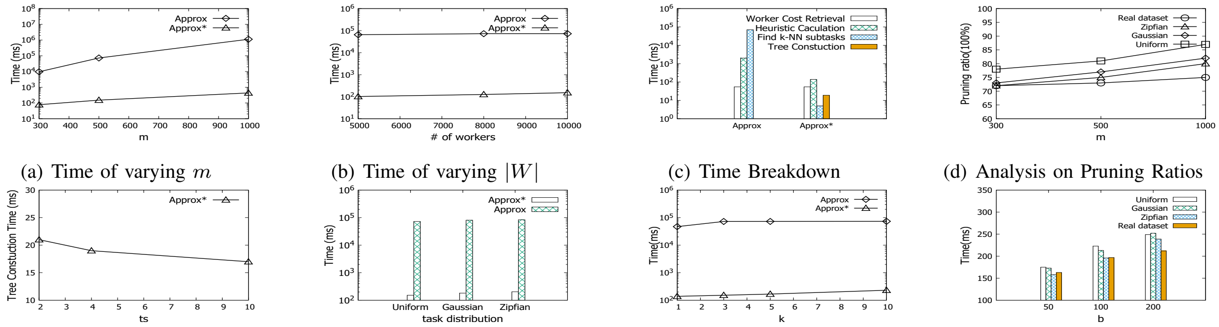
First, we compare the performance of the three variants, group-based parallelization, task-based parallelization, and the basic solution without parallelization in Fig. 9 (a). It shows that parallelization with smaller granularity achieves better scaleup. The task-based parallelization outperforms the other two. In particular, when the number of cores reaches 10, the task-based parallelization solution takes about an order of magnitude less running time than the basic solution, which is consistent with the analysis in Section IV-A. We then compare the two parallelization methods by varying task location distributions in Fig. 9 (b). We can see that the Gaussian and Zipf distributions incur higher costs. It is because that skewed datasets tend to incur larger numbers of worker conflicts. The cost increases moderately because of the good scalability achieved by the optimization techniques for indexing and scheduling. Also, the number of worker conflicts increases with the number of tasks, as reported in Fig. 9 (c).

We then report the scalability of our proposal w.r.t. the number of tasks in Fig. 9 (d). It can be observed that the task-based parallelization solution increases moderately w.r.t. the number of tasks, and grows only slowly. We examine the performance of the algorithm by varying the parameter m, in Fig. 9 (e). All methods increase moderately, where Zipfian and Gaussian distributions take longer than the uniform case. The result is consistent with our analysis, because skewed task distributions have a larger chance for incurring worker conflicts. We then make more analysis of the task-based parallelization method. We test the effect of thread priority setting in Fig. 9 (f). It reflects that threads with lower heuristic values are scheduled to execute earlier by the priority adjustment module, which breaks the ties of blocked threads and improves the performance of parallelization.



(a) Quality vs. Distributions (q_{sum}) (b) Quality vs. Budgets (q_{sum}) (c) Quality vs. Distributions (q_{min}) (d) Quality vs. Budgets (q_{min})

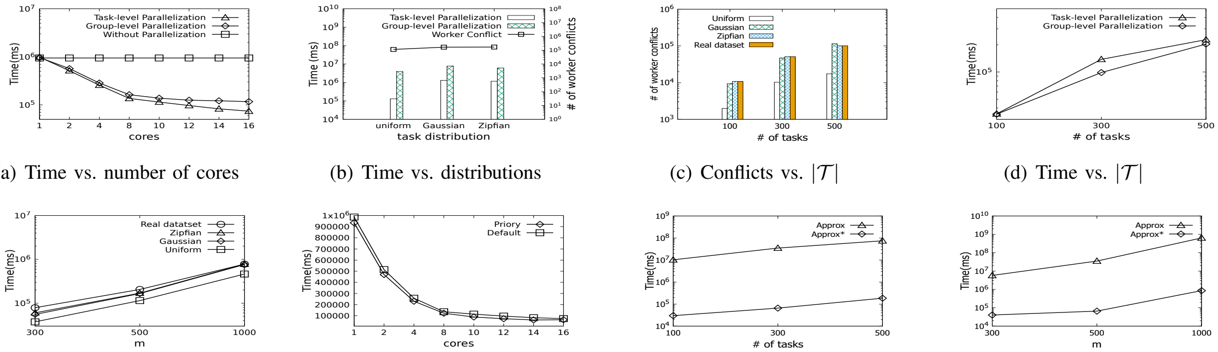
Fig. 7. Quality of Multi-task Case



(a) Time of varying m (b) Time of varying $|W|$ (c) Time Breakdown (d) Analysis on Pruning Ratios

(e) Analysis on Tree Construction (f) Effect of Distributions (g) Effect of k (h) Effect of Budgets

Fig. 8. Results of single-task assignment



(a) Time vs. number of cores (b) Time vs. distributions (c) Conflicts vs. $|\mathcal{T}|$ (d) Time vs. $|\mathcal{T}|$

(e) Time vs. m (f) Effect of Priority Setting (g) Time vs. $|\mathcal{T}|$ (MMQM) (h) Time vs. m (MMQM)

Fig. 9. Results of multi-task assignment

We show the results on the minimum quality metric in Fig. 9 (g) and (h). First, we examine the time costs by varying $|\mathcal{T}|$ in Fig. 9 (g). It can be observed that the time cost increases w.r.t. the number of tasks. Second, we test the result by varying m in Fig. 9 (h). It can be observed that the running time increases as m increases. In both experiments, Approx* steadily outperforms Approx, demonstrating better scalability in terms of the number of tasks and subtasks.

VI. Related Work

There are many studies in spatial crowdsourcing, requiring workers traveling to locations of spatial tasks and performs tasks, such as taking photos/videos, repairing a house, and waiting in line at shopping malls. These works focused on assigning available workers to tasks with distinct goals, such as maximizing the number of assigned tasks [1] [5] [27], minimizing the total travel costs of all workers [7] [8], maximizing the quality score [3] [5] [28], maximizing quality task assignment by considering both present and future workers/tasks [6], or minimizing maximum task assignment delay [9]. Existing works cannot be directly used to handle the quality issues of the TCSC problem. According to our comprehensive survey [16], TCSC is related to the categories of data collection, and of task matching with quality constraints. We thus review existing works in the two categories.

Regarding applications of data collection, there exist papers on floorplan generation [29], traffic anomalies detection [30], voluntary services [31], and geo-spatial linked open data postprocessing [32], etc. They mostly collect data in a specified spatiotemporal context, and do not address the issues in long-term data acquisition. Our work can support extending these works for continuous data acquisition, e.g., monitoring routing behaviours, by incorporating the quality-aware techniques.

Regarding quality constraints, most existing papers are on the quality of task responses, based on the workers' expertise, reputation, or reliability [33] [10] [3] [34]. They usually involve a pre-task qualification test [33], or the assignment based on the expertise [10], or abilities [3] [34] of the worker. These papers are similar to the TCSC problem in the sense that they require data aggregation from multiple workers, but the aggregation methods are totally different. To our best knowledge, the most relevant work is [3], which considers the diversity (or distribution) of spatial and temporal tasks. D-ifferently, they do not consider the mutual interaction between the interpolated and crowdsourced data. So, the optimization target and corresponding techniques are totally different.

To summarize, a TCSC task has a temporally continuous nature, and requires time-sharing collaboration of multiple workers, necessitating quality-aware data management.

VII. Conclusion and Future Works

In this paper, we study the problem of TCSC, which enables time-sharing collaboration among multiple workers towards long-term continuous spatial crowdsourcing applications. We propose an entropy-based quality metric for measuring the incompleteness of the crowdsourced results. Based on that, we study quality-aware task assignment algorithms with budget constraints for both single- and multi-task cases. For both variants, we prove its NP-hardness and submodularity of quality functions, so that a unified approximation framework can be applied. We devise novel indexing and parallel mechanisms for accelerating the processing. Extensive experiments on real and synthetic datasets show that our proposals achieve good efficiency and scalability. In the future, we will extend the approximation framework and optimization techniques for supporting spatiotemporal interpolation scenarios.

Acknowledgments

This work is partially supported by NSFC (No. 61772492, 62072428, 61672479, 62072427, 61872330, 61572457), Anhui NSF Distinguished Young Scholarship (No. 1908085J24), ARC DE190100663, and CAS Pioneer Hundred Talents Program. We thank anonymous reviewers for the insightful comments on crowdsourcing with spatiotemporal interpolation. Xike Xie is the corresponding author.

References

[1] L. Kazemi and C. Shahabi, "Geocrowd: enabling query answering with spatial crowdsourcing," in *SIGSPATIAL*, 2012, pp. 189–198.

[2] D. Deng, C. Shahabi, and U. Demiryurek, "Maximizing the number of worker's self-selected tasks in spatial crowdsourcing," in *SIGSPATIAL*, 2013, pp. 324–333.

[3] P. Cheng, X. Lian, Z. Chen, R. Fu, L. Chen, J. Han, and J. Zhao, "Reliable diversity-based spatial crowdsourcing by moving workers," *PVLDB*, pp. 1022–1033, 2015.

[4] Y. Tong, L. Wang, Z. Zhou, B. Ding, L. Chen, J. Ye, and K. Xu, "Flexible online task assignment in real-time spatial data," *PVLDB*, pp. 1334–1345, 2017.

[5] H. To, C. Shahabi, and L. Kazemi, "A server-assigned spatial crowdsourcing framework," *ACM Transactions on Spatial Algorithms and Systems*, vol. 1, no. 1, pp. 2:1–2:28, 2015.

[6] P. Cheng, X. Lian, L. Chen, and C. Shahabi, "Prediction-based task assignment in spatial crowdsourcing," in *ICDE*, 2017, pp. 997–1008.

[7] D. Deng, C. Shahabi, and L. Zhu, "Task matching and scheduling for multiple workers in spatial crowdsourcing," in *SIGSPATIAL*, 2015, pp. 21:1–21:10.

[8] X. Zhang, Z. Yang, Y. Liu, and S. Tang, "On reliable task assignment for spatial crowdsourcing," *IEEE Transactions on Emerging Topics in Computing*, vol. 7, no. 1, pp. 174–186, 2019.

[9] Z. Chen, P. Cheng, Y. Zeng, and L. Chen, "Minimizing maximum delay of task assignment in spatial crowdsourcing," in *ICDE*, 2019, pp. 1454–1465.

[10] 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.

[11] P. Cheng, L. Chen, and J. Ye, "Cooperation-aware task assignment in spatial crowdsourcing," in *ICDE*, 2019, pp. 1442–1453.

[12] J. Li, J. Wu, and Y. Zhu, "Selecting optimal mobile users for long-term environmental monitoring by crowdsourcing," in *IWQos*, 2019, pp. 8:1–8:10.

[13] C. Zhang, L. Zhu, C. Xu, X. Du, and M. Guizani, "A privacy-preserving traffic monitoring scheme via vehicular crowdsourcing," *Sensors*, vol. 19, no. 6, p. 1274, 2019.

[14] N. Sabir, A. A. Memon, and F. K. Shaikh, "Threshold based efficient road monitoring system using crowdsourcing approach," *Wireless Personal Communications*, vol. 106, no. 4, pp. 2407–2425, 2019.

[15] https://www.nationalgeographic.org/encyclopedia/citizen-science/, accessed:2020-01-10.

[16] S. R. B. Gummidi, X. Xie, and T. B. Pedersen, "A survey of spatial crowdsourcing," *ACM Transactions on Database Systems*, vol. 44, no. 2, pp. 8:1–8:46, 2019.

[17] J. Gao and P. Revesz, "Voting prediction using new spatiotemporal interpolation methods," in *DG.O*, 2006, pp. 293–300.

[18] J. Gao, "Adaptive interpolation algorithms for temporal-oriented dataset-s," in *TIME*, 2006, pp. 145–151.

[19] O. Babak and C. V. Deutsch, "Statistical approach to inverse distance interpolation," *Stochastic Environmental Research and Risk Assessment*, vol. 23, no. 5, pp. 543–553, 2009.

[20] T. M. Cover and J. A. Thomas, *Elements of information theory*. John Wiley & Sons, 2012.

[21] H. Lin and J. Bilmes, "A class of submodular functions for document summarization," in *ACL*, 2011, pp. 510–520.

[22] T. Wang, X. Xie, X. Cao, T. B. Pedersen, Y. Wang, and M. Xiao, "On efficient and scalable time-continuous spatial crowdsourcing full version," *Technical Report*, 2020. [Online]. Available: https://arxiv.org/pdf/2010.15404.pdf

[23] A. Krause and C. Guestrin, "A note on the budgeted maximization of submodular functions," 2005. [Online]. Available: http://reports-archive.adm.cs.cmu.edu/anon/cald/CMU-CALD-05-103.pdf

[24] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions—i," *Mathematical Programming*, vol. 14, no. 1, pp. 265–294, 1978.

[25] A. Okabe, B. Boots, K. Sugihara, and S. Chiu, *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley, 2000.

[26] C. Chekuri and A. Kumar, "Maximum coverage problem with group budget constraints and applications," in *APPROX-RANDOM*, 2004, pp. 72–83.

[27] P. Cheng, X. Jian, and L. Chen, "An experimental evaluation of task assignment in spatial crowdsourcing," *PVLDB*, pp. 1428–1440, 2018.

[28] F. Daniel, P. Kucherbaev, C. Cappiello, B. Benatallah, and M. Allahbakhsh, "Quality control in crowdsourcing: A survey of quality attributes, assessment techniques, and assurance actions," *ACM Computing Surveys*, vol. 51, no. 1, pp. 7:1–7:40, 2018.

[29] M. Alzantot and M. Youssef, "Crowdinside : Automatic construction of indoor floorplans," in *GIS*, 2012, pp. 99–108.

[30] B. Pan, Y. Zheng, D. Wilkie, and C. Shahabi, "Crowd sensing of traffic anomalies based on human mobility and social media," in *GIS*, 2013, pp. 344–353.

[31] X. Xie, P. Jin, M. L. Yiu, J. Du, M. Yuan, and C. S. Jensen, "Enabling scalable geographic service sharing with weighted imprecise voronoi cells," *TKDE*, vol. 28, no. 2, pp. 439–453, 2016.

[32] R. Karam and M. Melchiori, "Improving geo-spatial linked data with the wisdom of the crowds," in *EDBT Workshop*, 2013, pp. 68–74.

[33] X. Liu, M. Lu, B. C. Ooi, Y. Shen, S. Wu, and M. Zhang, "CDAS: A crowdsourcing data analytics system," *PVLDB*, pp. 1040–1051, 2012.

[34] L. Kazemi, C. Shahabi, and L. Chen, "Geotrucrowd: trustworthy query answering with spatial crowdsourcing," in *SIGSPATIAL*, 2013, pp. 304–313.