Context-Aware Hierarchical Online Learning for Performance Maximization in Mobile Crowdsourcing

Sabrina Klos née Müller, *Student Member, IEEE*, Cem Tekin, *Member, IEEE*, Mihaela van der Schaar, *Fellow, IEEE*, and Anja Klein, *Member, IEEE*

***Abstract*— In mobile crowdsourcing (MCS), mobile users accomplish outsourced human intelligence tasks. MCS requires an appropriate task assignment strategy, since different workers may have different performance in terms of acceptance rate and quality. Task assignment is challenging, since a worker's performance 1) may fluctuate, depending on both the worker's current personal context and the task context and 2) is not known a priori, but has to be learned over time. Moreover, learning context-specific worker performance requires access to context information, which may not be available at a central entity due to communication overhead or privacy concerns. In addition, evaluating worker performance might require costly quality assessments. In this paper, we propose a context-aware hierarchical online learning algorithm addressing the problem of performance maximization in MCS. In our algorithm, a local controller (LC) in the mobile device of a worker regularly observes the worker's context, her/his decisions to accept or decline tasks and the quality in completing tasks. Based on these observations, the LC regularly estimates the worker's context-specific performance. The mobile crowdsourcing platform (MCSP) then selects workers based on performance estimates received from the LCs. This hierarchical approach enables the LCs to learn context-specific worker performance and it enables the MCSP to select suitable workers. In addition, our algorithm preserves worker context locally, and it keeps the number of required quality assessments low. We prove that our algorithm converges to the optimal task assignment strategy. Moreover, the algorithm outperforms simpler task assignment strategies in experiments based on synthetic and real data.**

***Index Terms*— Crowdsourcing, task assignment, online learning, contextual multi-armed bandits.**

Manuscript received May 8, 2017; revised November 6, 2017 and March 10, 2018; accepted March 29, 2018; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor J. Huang. Date of publication May 2, 2018; date of current version June 14, 2018. The work of S. Klos née Müller and A. Klein was supported by the German Research Foundation (DFG) under Project B3 within the Collaborative Research Center 1053–MAKI. The work of C. Tekin was supported by the Scientific and Technological Research Council of Turkey under 3501 Program under Grant 116E229. The work of M. van der Schaar was supported in part by ONR Mathematical Data Sciences Grant and in part by the NSF under Grant 1524417 and Grant 1462245. *(Corresponding author: Sabrina Klos née Müller.)*

S. Klos née Müller and A. Klein are with the Communications Engineering Laboratory, Technische Universität Darmstadt, 64289 Darmstadt, Germany (e-mail: s.klos@nt.tu-darmstadt.de; a.klein@nt.tu-darmstadt.de).

C. Tekin is with the Electrical and Electronics Engineering Department, Bilkent University, 06800 Ankara, Turkey (e-mail: cemtekin@ee.bilkent.edu.tr).

M. van der Schaar is with the Department of Electrical Engineering, University of California at Los Angeles, Los Angeles, CA 90095 USA, and also with the Department of Engineering Science, University of Oxford, Oxford OX1 2JD, U.K. (e-mail: mihaela@ee.ucla.edu).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/TNET.2018.2828415

I. Introduction

CROWDSOURCING (CS) is a popular way to outsource human intelligence tasks, prominent examples being conventional web-based systems like Amazon Mechanical Turk[1] and Crowdflower.[2] More recently, *mobile crowdsourcing* (MCS) has evolved as a powerful tool to leverage the workforce of mobile users to accomplish tasks in a distributed manner [1]. This may be due to the fact that the number of mobile devices is growing rapidly and at the same time, people spend a considerable amount of their daily time using these devices. For example, between 2015 and 2016, the global number of mobile devices grew from 7.6 to 8 billion [2]. Moreover, the daily time US adults spend using mobile devices is estimated to be more than 3 hours in 2017, which is an increase by more than 45% compared to 2013 [3].

In MCS, *task owners* outsource their tasks via an intermediary *mobile crowdsourcing platform* (MCSP) to a set of *workers*, i.e., mobile users, who may complete these tasks. An MCS task may require the worker to interact with her/his mobile device in the physical world (e.g., photography tasks) or to complete some virtual task via the mobile device (e.g., image annotation, sentiment analysis). Some MCS tasks, subsumed under the term *spatial* CS [4], are spatially constrained (e.g., photography task at point of interest), or require high spatial resolution (e.g., air pollution map of a city). In spatial CS, tasks typically require workers to travel to certain locations. However, recently emerging MCS applications are also concerned with *location-independent* tasks. For example, MapSwipe[3] lets mobile users annotate satellite imagery to find inhabitated regions around the world. The GalaxyZoo app[4] lets mobile users classify galaxies. The latter project is an example of the more general trend of citizen science [5]. On the commercial side, Spare5[5] or Crowdee[6] outsource micro-tasks (e.g., image annotation, sentiment analysis, and opinion polls) to mobile users in return for small payments. While location-independent tasks could as well be completed by users of static devices as in web-based CS, emerging MCS applications for location-independent tasks exploit that online mobile users complete such tasks on the go.

MCS – be it spatial or location-independent – requires an appropriate *task assignment* strategy, since not all workers

[1] https://www.mturk.com
[2] https://www.crowdflower.com/
[3] https://mapswipe.org/
[4] https://www.galaxyzoo.org/
[5] https://app.spare5.com/fives
[6] https://www.crowdee.de/

may be equally suitable for a given task. First, different workers may have different task preferences and hence different acceptance rates. Secondly, different workers may have different skills, and hence provide different quality when completing a task. Two assignment modes considered in the CS literature are the *server assigned tasks* (SAT) mode and the *worker selected tasks* (WST) mode [6]. In SAT mode, the MCSP tries to match workers and tasks in an optimal way, e.g., to maximize the number of task assignments, possibly under a given task budget. For this purpose, the MCSP typically gathers task and worker information to decide on task assignment. This sophisticated strategy may require a large communication overhead and a privacy concern for workers since the MCSP has to be regularly informed about the current worker contexts (e.g., their current positions). Moreover, previous work on the SAT mode often either assumed that workers always accept a task once assigned to it or that workers' acceptance rates and quality are known in advance. However, in reality, acceptance rates and quality are usually not known beforehand and therefore have to be learned by the MCSP. In addition, a worker's acceptance rate and the quality of completed tasks might depend not only on the specific task, but also on the worker's current context, e.g., the worker's location or the time of day [7]. This context may change quickly, especially in MCS with location-independent tasks, since workers can complete such tasks anytime and anywhere.

In contrast, in WST mode, workers autonomously select tasks from a list. This rather simple mode is often used in practice (e.g., on Amazon Mechanical Turk) since it has the advantage that workers automatically select tasks they are interested in. However, the WST mode can lead to suboptimal task assignments since first, finding suitable tasks is not as easy as it seems (e.g., time-consuming searches within a long list of tasks are needed and workers might simply select from the first displayed tasks [8]) and secondly, workers might leave unpopular tasks unassigned. Therefore, in WST mode, the MCSP might additionally provide personalized *task recommendation* (TR) to workers such that workers find appropriate tasks [7]. However, personalized TR typically requires workers to share their current context with the MCSP, which again may mean a communication overhead and a privacy concern for workers.

We argue that a task assignment strategy is needed which combines the advantages of the above modes: The MCSP should centrally coordinate task assignment to ensure that appropriate workers are selected, as in SAT mode. At the same time, the workers' personal contexts should be kept locally, as in WST mode, in order to keep the communication overhead small and to protect the workers' privacy. Moreover, task assignment should take into account that workers may decline tasks, and hence, the assignment should fit to the workers' preferences, as in WST mode with personalized TR. In addition, task assignment should be based both on acceptance rates and on the quality with which a task is completed. Since quality assessments (e.g., a manual quality rating from a task owner, or an automatic quality assessment using either local software in a mobile device or the resources of a cloud) may be costly, the number of quality assessments should be kept low. Finally, workers' acceptance rates and quality have to be learned over time.

Our contribution therefore is as follows: We propose a context-aware hierarchical online learning algorithm for performance maximization in MCS for location-independent tasks. Our algorithm for the first time *jointly* takes the following aspects into account: (i) Our algorithm learns worker performance online without requiring a training phase. Since our algorithm learns in an online fashion, it adapts and improves the worker selection over time and can hence achieve good results already during run time. By establishing regret bounds, we provide performance guarantees for the learned task assignment strategy and prove that our algorithm converges to the optimal task assignment strategy. (ii) We allow different task types to occur. We use the concept of *task context* to describe the features of a task, such as its required skills or equipment. (iii) We model that the worker performance depends (in a possibly non-linear fashion) on both the task context and the *worker context*, such as the worker's current location, activity, or device status. Our proposed algorithm learns this context-specific worker performance. (iv) Our algorithm is split into two parts, one part executed by the MCSP, the other part by *local controllers* (LCs) located in each of the workers' mobile devices. An LC learns its worker's performance in terms of acceptance rate and quality online over time, by observing the worker's personal contexts, her/his decisions to accept or decline tasks and the quality in completing these tasks. The LC learns from its worker's context only locally, and personal context is not shared with the MCSP. Each LC regularly sends performance estimates to the MCSP. Based on these estimates, the MCSP takes care of the worker selection. This hierarchical (in the sense of the coordination between the MCSP and the LCs) approach enables the MCSP to select suitable workers for each task under its budget based on what the LCs have previously learned. Moreover, workers receive personalized task requests based on their interests and skills, while keeping the number of (possibly costly) quality assessments low.

The remainder of this paper is organized as follows. Sec. II gives an overview on related work. Sec. III describes the system model. In Sec. IV, we propose a context-aware hierarchical online learning algorithm for performance maximization in MCS. In Sec. V, we theoretically analyze our algorithm in terms of its regret, as well as its requirements with respect to local storage, communication and worker quality assessment. Sec. VI contains a numerical evaluation based on synthetic and real data. Sec. VII concludes the paper.

II. Related Work

Research has put some effort in theoretically defining and classifying CS systems, such as web-based [9], mobile [1] and spatial [4] CS. Below, we give an overview on related work on task assignment in general, mobile and spatial CS systems (see Table I), as relevant for our scenario. Note that strategic behavior of workers and task owners in CS systems, e.g., concerning pricing and effort spent in task completion [10] is out of the scope of this paper. Also note that we assume that it is possible to assess the quality of a completed task. A different line of work on CS deals with quality estimation in case of missing ground truth, recently also using online learning [11].

Due to the dynamic nature of CS, with tasks and/or workers typically arriving dynamically over time, task assignment is often modeled as an online decision making problem [12]. For general CS systems, [13] proposed a competitive online task assignment algorithm for maximizing the utility of a task owner on a given set of task types, with finite number of

TABLE I
COMPARISON WITH RELATED WORK ON TASK ASSIGNMENT IN CROWDSOURCING

	[13]	[14]	[15]	[16]	[17]	[18]	[6], [19]	[20]	[21]	[22]	This work
Crowdsourcing Type	General	General	General	General	Mobile	Mobile	Spatial	Spatial	Spatial	Spatial	Loc.-Ind.
Task Assignment Mode	SAT	SAT	SAT	WST/TR	WST/TR	SAT	SAT	SAT	SAT	SAT	proposed
Different Task Types	Yes	No	Yes	Yes	Yes	No	Yes	Yes	No	Yes	Yes
Worker Context-Aware	No	No	No	No	Yes	No	Yes	Yes	Yes	No	Yes
Context-Spec. Performance	No	No	No	No	Yes	No	No	Yes	Yes	No	Yes
Worker Context Protected	N/A	N/A	N/A	N/A	Yes	N/A	No	No	Yes	N/A	Yes
Type of Learning	Online	Online	Batch	Online	Offline	Online	N/A	Online	N/A	Offline	Online
Regret Bounds	Yes	Yes	No	No	No	Yes	N/A	Yes	N/A	No	Yes

tasks per type, by learning the skills of sequentially appearing workers. While [13] considers sequentially arriving workers and their algorithm decides which task to assign to a worker, we consider sequentially arriving tasks and our algorithm decides which workers to assign to a task. Therefore, our algorithm can be applied to an infinite number of task types by describing a task using its context. In addition, our algorithm takes worker context into account, which may affect worker performance in MCS. In [14], a bounded multi-armed bandit model for expert CS is presented and a task assignment algorithm with sublinear regret is derived which maximizes the utility of a budget-constrained task owner under uncertainty about the skills of a finite set of workers with (known) different prices and limited working time. While in [14], the average skill of a worker is learned, our algorithm takes context into account, and thereby learns context-specific performance. In [15], a real-time algorithm for finding the top-k workers for sequentially arriving tasks is presented. First, tasks are categorized offline into different types and the similarity between a worker's profile and each task type is computed. Then, in real time, the top-k workers are selected for a task based on a matching score, which takes into account the similarity and historic worker performance. The authors propose to periodically update the performance estimates offline in batches, but no guarantees on the learning process are given. In contrast, we additionally take into account worker context, learn context-specific performance and derive guarantees on the learning speed. In [16], methods for learning a worker preference model are proposed for personalized TR in WST mode. These methods use the history of worker preferences for different tasks, but they do not take into account worker context.

For MCS systems, [17] proposes algorithms for optimal TR in WST mode that take into account the trade-off between the privacy of worker context, the utility to recommend the best tasks and the efficiency in terms of communication and computation overhead. TR is performed by a server based on a generalized context shared by the worker. The statistics used for TR are gathered offline via a proxy that ensures differential privacy guarantees. While [17] allows to flexibly adjust the shared generalized context and makes TRs based on offline statistics and generalized worker context, our approach keeps worker context locally and learns each worker's individual statistics online. In [18], an online learning algorithm for mobile crowdsensing is presented to maximize the revenue of a budget-constrained task owner by learning the sensing values of workers with known prices. While [18] considers a total budget and each crowdsensing task requires a minimum number of workers, we consider a separate budget per task, which translates to a maximum number of required workers, and we additionally take task and worker context into account.

A taxonomy for spatial CS was first introduced in [6]. The authors present a location-entropy based algorithm for SAT mode to maximize the number of task assignments under uncertainty about task and worker arrival processes. The server decides on task assignment based on centrally gathered knowledge about the workers' current locations. Shahabi and Kazemi [19] extend this framework to maximize the quality of assignments under varying worker skills for different task types. However, in contrast to our work, [6] and [19] assume that worker context is centrally gathered, that workers always accept assigned tasks within certain known bounds and that worker skills are known a priori. In [20], an online task assignment algorithm is proposed for spatial CS with SAT mode for maximizing the expected number of accepted tasks. The problem is modeled as a contextual multi-armed bandit problem, and workers are selected for sequentially arriving tasks. The authors adapt the LinUCB algorithm by assuming that the acceptance rate is a linear function of the worker's distance to the task and the task type. However, such a linearity assumption is restrictive and it especially may not hold in MCS with location-independent tasks. In contrast, our algorithm works for more general relationships between context and performance. In [21], an algorithm for privacy-preserving spatial CS in SAT mode is proposed. Using differential privacy and geocasting, the algorithm preserves worker locations (i.e., their contexts) while optimizing the expected number of accepted tasks. However, the authors assume that the workers' acceptance rates are identical and known, whereas our algorithm learns context-specific acceptance rates. In [22], exact and approximation algorithms for acceptance maximization in spatial CS with SAT mode are proposed. The algorithms are performed offline for given sets of available workers and tasks based on a probability of interest for each pair of worker and task. The probabilities of interest are computed beforehand using maximum likelihood estimation. On the contrary, our algorithm learns acceptance rates online and we provide an upper bound on the regret of this learning.

We model the problem formally as a *contextual multi-armed bandit* (contextual MAB) problem [23]–[33]. MABs are a type of *reinforcement learning* (RL). In general, RL, which has been used to solve various problems in networking [34], [35], deals with agents learning to take actions based on rewards. Specifically, in contextual MAB, an agent sequentially chooses among a set of actions with unknown expected rewards. In each round, the agent first observes some context information, which he may use to determine the action to select. After selecting an action, the agent receives a reward, which

may depend on the context. The agent tries to learn which action has the highest reward in which context, to maximize his expected reward over time.

In the related literature on contextual MAB, different algorithms make different assumptions on how context is generated and on how rewards are formed. For general contextual MAB with no further assumptions on how rewards are formed, [23] proposes the epoch-greedy algorithm. Also [24] for general contextual MAB with resource constraints and policy sets makes no further assumptions on how rewards are formed, except that they assume that the marginal distribution over the contexts is known. However, the algorithms in [23] and [24] work only for a finite set of actions and they assume that at each time step the tuples (context, rewards) are sampled from a fixed but unknown distribution (i.e., contexts are generated in an i.i.d. fashion). Other algorithms have stronger assumptions on how rewards are formed. For example, the LinUCB algorithm [25], [26], assumes that the expected reward is linear in the context. Such a linearity assumption is also used in the Thompson-sampling based algorithm in [27], and in the clustering algorithm in [28], where a clustering is performed on top of a contextual MAB setting. There are also works which assume a known similarity metric over the contexts. These algorithms group the contexts into sets of similar contexts by partitioning the context space. Then, they estimate the reward of an action under a given context based on previous rewards for that action in the set of similar contexts. For example, the contextual zooming algorithm [29] proposes a non-uniform adaptive partition of the context space. Moreover, [30], [31] use uniform and non-uniform adaptive partitions of the context space. In [32] and [33], these algorithms are applied to a wireless communication scenario. While the algorithms in [25]–[33] are more restrictive with respect to how rewards are formed, they are more general than [23], [24] in the sense that they do not require the contexts to be generated i.i.d. over time. Moreover, the algorithms in [29]–[33] also work for an infinite set of actions.

Algorithms for contextual MAB also differ with respect to their approach to balance the exploration vs. exploitation trade-off. While the epoch-greedy algorithm [23] and the algorithms in [30]–[33] explicitly distinguish between exploration and exploitation steps, the LinUCB [25], [26] algorithm, the clustering algorithm in [28] and the contextual zooming algorithm [29] follow an index-based approach, in which in any round, the action with the highest index is selected. Other algorithms, like the one for contextual MAB with resource constraints in [24], draw samples from a distribution to find a policy which is then used to select the action. Finally, algorithms like the Thompson-sampling based algorithm in [27] draw samples from a distribution to build a belief, and select the action which maximizes the expected reward based on this belief.

Our proposed algorithm extends [30]–[33] as follows: (i) While in [30]–[33], a learner observes some contexts and selects a subset of actions based on these contexts, our algorithm is decoupled to several learning entities, each observing the context of one particular action and learning the rewards of this action, and a coordinating entity, which selects a subset of actions based on the learning entities' estimates. In the MCS scenario, an action corresponds to a worker, the learning entities correspond to the LCs which learn the performance of their workers, and the coordinating entity corresponds to the MCSP, which selects workers based on the performance estimates from the LCs. (ii) While in [30]–[33], the same number of actions is selected per round, we allow different numbers of actions to be selected per round. In the MCS scenario, this corresponds to allowing different numbers of required workers for different tasks. Hence, in contrast to [30]–[33], the learning speed of our algorithm is affected by the arrival process of the numbers of actions to be selected. (iii) While in [30]–[33], each action has the same context space, we allow each action to have an individual context space of an individual dimension. In the MCS scenario, this corresponds to allowing workers to give access to individual sets of context dimensions. Therefore, in contrast to [30]–[33], the granularity of learning may be different for different actions. (iv) Finally, while in [30]–[33], all actions are available in any round, we allow actions to be unavailable in arbitrary rounds. In the MCS scenario, this corresponds to allowing that workers may be unavailable. Hence, in contrast to [30]–[33], the best subset of actions in a certain round depends on the specific set of available actions in this round.

III. System Model

A. Mobile Crowdsourcing Platform

We consider an MCSP, to which a fixed set $\mathcal{W}$ of $W := |\mathcal{W}|$ workers belongs. A worker is a user equipped with a mobile device, in which the MCS application is installed. Workers can be in two modes: A worker is called *available*, if the MCS application on the device is running. In this case, the MCSP may request the worker to complete a task, which the worker may then accept or decline. A worker is called *unavailable*, if the MCS application on the device is turned off.

Task owners can place location-independent tasks of different types into the MCSP and select a task budget. A task t is defined by a tuple (b_t, c_t), where $b_t > 0$ denotes the budget that the task owner is willing to pay for this task and $c_t \in \mathcal{C}$ denotes the task context. The task context is taken from a bounded C-dimensional task context space $\mathcal{C} := [0,1]^C$ and captures feature information about the task.[7] Possible features could be the skills or equipment required to complete a task (e.g., the required levels of creativity or analytical skills may be translated to continuous values between 0 and 1; whether a camera or a specific application is needed may be encoded as 0 or 1). The task owner has to pay the MCSP for each worker that completes the task after being requested by the MCSP. Specifically, we assume that the MCSP charges the task owner a fixed price $e_t \in [e_{\min}, e_{\max}]$ per worker that completes task t, where $e_{\min} > 0$ and $e_{\max} \geq e_{\min}$ correspond to lower and upper price limits, respectively. The price e_t depends on the task context c_t and is determined by the MCSP's fixed context-specific price list. We assume that for each task t, the budget b_t satisfies $b_t \in [e_t, We_t]$, so that the budget is sufficient to pay at least one and at most W workers for completing the task. Based on the budget b_t and the price e_t, the MCSP computes the maximum number $m_t := \lfloor \frac{b_t}{e_t} \rfloor \in \{1, \ldots, W\}$ of workers that should complete the task.

Following [13], [14], and [18], we assume that each task has the following properties: (i) As determined by budget and price, the task owner would like to receive replies from possibly several workers. (ii) It is possible to assess the quality

[7]We assume that tasks are described by C context dimensions. In each of the C context dimensions, a task is classified via a value between $[0, 1]$. Then, $c_t \in [0,1]^C$ is a vector describing task t's overall context.

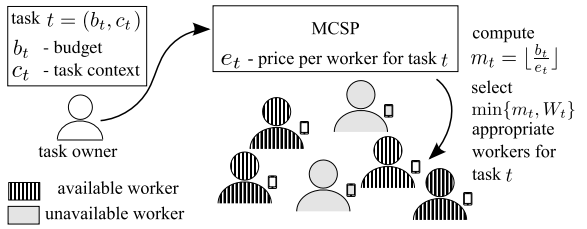


Fig. 1. System model. A task arrives at the MCSP. The MCSP has to select an appropriate subset of available workers for the task.

of a single worker's reply. (iii) The qualities of different workers' replies are independent. (iv) The qualities of the workers' replies are additive, i.e., if workers 1 and 2 complete the task with qualities A and B, the task owner receives a total quality of $A + B$. Such tasks belong to the class of *crowd solving* tasks [7], examples being translation and retrieval tasks [13].

We assume that tasks arrive at the MCSP sequentially and we denote the sequentially arriving tasks by $t = 1, \ldots, T$. For each arriving task t, if sufficient workers are available, the MCSP will request m_t workers to complete the task.[8] However, due to the dynamics in worker availability over time, the MCSP can only select workers from the set $\mathcal{W}_t \subseteq \mathcal{W}$ of currently available workers for task t, as defined by $\mathcal{W}_t := \{i : \text{ worker } i \text{ is available at arrival time of } t\}$, where the number of available workers[9] is denoted by $W_t := |\mathcal{W}_t| \in \{1, \ldots, W\}$. Hence, since the MCSP can select at most all available workers, it aims at selecting $\min\{m_t, W_t\}$ workers for task t.[10] The goal of the MCSP is to select a subset of $\min\{m_t, W_t\}$ workers which maximizes the worker performance for that task (see Fig. 1 for an illustration).

B. Context-Specific Worker Performance

The performance of a worker depends on (i) the worker's willingness to accept the task and (ii) the worker's quality in completing the task, where we assume that the quality can take values in a range $[q_{\min}, q_{\max}] \subseteq \mathbb{R}_{0,+}$. Both the willingness to accept the task and the quality may depend on the worker's current context and on the task context. Let $x_{t,i}$ denote the personal context of worker $i \in \mathcal{W}_t$ at the arrival time of task t, coming from a bounded X_i-dimensional personal context space $\mathcal{X}_i := [0,1]^{X_i}$. Here, we allow each worker i to have an individual personal context space $\mathcal{X}_i$, since each worker may allow access to an individual set of context dimensions (e.g., the worker allows access to a certain set of sensors of the mobile device that are used to derive her/his context). Possible personal context dimensions could be the worker's current location (in terms of geographic coordinates), the type of location (e.g., at home, in a coffee shop), the worker's current activity (e.g., commuting, working) or the current device status (e.g., battery state, type of wireless connection). We further call the concatenation $(x_{t,i}, c_t) \in \mathcal{X}_i \times \mathcal{C}$ the *joint (personal and task) context of worker* i. For worker i, this joint context is hence a vector of dimension $D_i := X_i + C$. We call $\mathcal{X}_i \times \mathcal{C} = [0,1]^{X_i} \times [0,1]^C \equiv [0,1]^{D_i}$ the *joint (personal and task) context space of worker* i. The reason for considering the joint context is that the performance of worker i may depend on both the current context $x_{t,i}$ and the task context c_t – in other words, the performance depends jointly on $(x_{t,i}, c_t)$.

Let $p_i(x_{t,i}, c_t)$ denote the performance of worker i with current personal context $x_{t,i}$ for task context c_t. The performance can be decomposed into (i) worker i's decision $d_i(x_{t,i}, c_t)$ to accept ($d_i(x_{t,i}, c_t) = 1$) or reject ($d_i(x_{t,i}, c_t) = 0$) the task and, in case the worker accepts the task, also on (ii) worker i's quality $q_i(x_{t,i}, c_t)$ when completing the task. Hence, we can write

$$p_i(x_{t,i}, c_t) := \begin{cases} q_i(x_{t,i}, c_t), & \text{if } d_i(x_{t,i}, c_t) = 1, \\ 0, & \text{if } d_i(x_{t,i}, c_t) = 0. \end{cases}$$

The performance is a random variable whose distribution depends on the distributions of the random variables $d_i(x_{t,i}, c_t)$ and $q_i(x_{t,i}, c_t)$. Since the decision $d_i(x_{t,i}, c_t)$ is binary, it is drawn from the Bernoulli distribution with unknown parameter $r_i(x_{t,i}, c_t) \in [0,1]$. Hence, $r_i(x_{t,i}, c_t)$ represents worker i's acceptance rate given the joint context $(x_{t,i}, c_t)$. The quality $q_i(x_{t,i}, c_t)$ is a random variable conditioned on $d_i(x_{t,i}, c_t) = 1$ (i.e., task acceptance) with unknown distribution and we denote its expected value by $\nu_i(x_{t,i}, c_t) := \mathrm{E}[q_i(x_{t,i}, c_t)]$. Hence, $\nu_i(x_{t,i}, c_t)$ represents the average quality of worker i with personal context $x_{t,i}$ when completing a task of context c_t. Therefore, the performance $p_i(x_{t,i}, c_t)$ of worker i given the joint context $(x_{t,i}, c_t)$ has unknown distribution, takes values in $[0, q_{\max}]$ and its expected value satisfies

$$\mathrm{E}[p_i(x_{t,i}, c_t)] = \theta_i(x_{t,i}, c_t),$$

where $\theta_i(x_{t,i}, c_t) := r_i(x_{t,i}, c_t)\nu_i(x_{t,i}, c_t)$.

C. Problem Formulation

Consider an arbitrary sequence of task and worker arrivals.[11] Let $y_{t,i}$ denote a binary variable which is 1 if worker i is requested to complete task t and 0 otherwise. Then, the problem of selecting, for each task, a subset of workers which maximizes the sum of expected performances given the task budget is given by

$$\begin{aligned} \max_{\{y_{t,i}\}_{i \in \mathcal{W}_t, t=1,\ldots,T}} \quad & \sum_{t=1}^{T} \sum_{i \in \mathcal{W}_t} \theta_i(x_{t,i}, c_t) y_{t,i} \\ \text{s.t.} \quad & \sum_{i \in \mathcal{W}_t} y_{t,i} \leq m_t \quad \forall t = 1, \ldots, T \\ & y_{t,i} \in \{0,1\} \quad \forall i \in \mathcal{W}_t, \ \forall t = 1, \ldots, T. \end{aligned} \tag{1}$$

First, we analyze problem (1) assuming full knowledge about worker performance. Therefore, assume that there was an entity that (i) was an omniscient *oracle*, which knows the expected performance of each worker in each context for each task context *a priori* and (ii) for each arriving task, this entity is *centrally* informed about the current contexts of all available workers. For such an entity, problem (1) is an integer linear programming problem, which can be decoupled to an independent sub-problem per arriving task. For a task t, if fewer workers are available than required, i.e., $W_t \leq m_t$,

[8] Note that each task is only processed once by the MCSP, even if not all m_t requested workers complete the task. In this case, the MCSP charges the task owner only for the actual number of workers that completed the task since only these workers are compensated. The task owner may submit the task to the MCSP again if she/he wishes more workers to complete the task.

[9] We assume that for each arriving task, at least one worker is available.

[10] If fewer than m_t workers are available, the MCSP will request all available workers to complete the task.

[11] In the following, by "*an arbitrary sequence of task and worker arrivals*", we mean, given arbitrary sequences of task budgets $\{b_t\}_{t=1,\ldots,T}$, task contexts $\{c_t\}_{t=1,\ldots,T}$, task prices $\{e_t\}_{t=1,\ldots,T}$, worker availability $\{\mathcal{W}_t\}_{t=1,\ldots,T}$ and worker contexts $\{x_{t,i}\}_{i \in \mathcal{W}_t, t=1,\ldots,T}$.

the optimal solution is to request all available workers to complete the task. However, if $W_t > m_t$, the corresponding sub-problem is a special case of a knapsack problem with a knapsack of size m_t and with items of identical size and non-negative profit. Therefore, the optimal solution can be easily computed in at most $O(W \log(W))$ by ranking the available workers according to their context-specific expected performance and selecting the m_t highest ranked workers. By $\mathcal{S}_t^* := \{s_{t,1}^*, \ldots, s_{t,\min\{m_t,W_t\}}^*\}$, we denote the optimal subset of workers to select for task t. Formally, these workers satisfy

$$s_{t,j}^* \in \underset{i \in \mathcal{W}_t \setminus \bigcup_{k=1}^{j-1}\{s_{t,k}^*\}}{\operatorname{argmax}} \theta_i(x_{t,i}, c_t) \text{ for } j = 1, \ldots, \min\{m_t, W_t\},$$

where $\bigcup_{k=1}^{0}\{s_{t,k}^*\} := \emptyset$. Note that $\mathcal{S}_t^*$ depends on the task budget b_t, context c_t, price e_t, the set $\mathcal{W}_t$ of available workers and their personal contexts $\{x_{t,i}\}_{i \in \mathcal{W}_t}$, but we write $\mathcal{S}_t^*$ instead of $\mathcal{S}_t^*(b_t, c_t, e_t, \mathcal{W}_t, \{x_{t,i}\}_{i \in \mathcal{W}_t})$ for brevity. Let $\mathcal{S}^* := \{\mathcal{S}_t^*\}_{t=1,\ldots,T}$ be the collection of optimal subsets of workers for the collection $\{1, \ldots, T\}$ of tasks. We call this collection the solution achieved by a *centralized oracle*, since it requires an entity with a priori knowledge about expected performances and with access to worker contexts to make optimal decisions.

However, we assume that the MCSP does not have a priori knowledge about expected performances, but it still has to select workers for arriving tasks. Let $\mathcal{S}_t := \{s_{t,1}, \ldots, s_{t,\min\{m_t,W_t\}}\}$ denote the set of workers that the MCSP selects and requests to complete task t. If for an arriving task, fewer workers are available than required, i.e., $W_t \leq m_t$, by simply requesting all available workers (i.e., $\mathcal{S}_t = \mathcal{W}_t$) to complete the task, the MCSP automatically selects the optimal subset of workers. Otherwise, for $W_t > m_t$, the MCSP cannot simply solve problem (1) like an omniscient oracle, since it does not know the expected performances $\theta_i(x_{t,i}, c_t)$. Moreover, we assume that a worker's current personal context is only locally available in the mobile device. We call the software of the MCS application, which is installed in the mobile device, a *local controller* (LC) and we denote by LC i the LC of worker i. Depending on the requirements of the specific MCS application, such as, concerning communication overhead and worker privacy, the LCs may be owned by either the MCSP, the workers, or a trusted third party [17], [21]. In any case, each LC has access to its corresponding worker's personal context, but it does not share this information with the MCSP.

Hence, the MCSP and the LCs should cooperate in order to learn expected performances over time and in order to select an appropriate subset of workers for each task. For this purpose, over time, the system of MCSP and LCs has to find a trade-off between exploration and exploitation, by, on the one hand, selecting workers about whose performance only little information is available and, on the other hand, selecting workers which are likely to have high performance. For each arriving task, the selection of workers depends on the history of previously selected workers and their observed performances. However, observing worker performance requires quality assessments (e.g., in form of a manual quality rating from a task owner, or an automatic quality assessment using either local software in the battery-constrained mobile device or the resources of a cloud), which may be costly. Our model and algorithm are agnostic to the specific type of quality assessment, as long as the LCs do have access to the quality assessments. In any case, we aim at limiting the number of performance observations in order to keep the cost for quality assessment low.

Next, we present a *context-aware hierarchical online learning algorithm*, which maps the history of previously selected workers and observed performances to the next selection of workers. The performance of this algorithm can be evaluated by comparing its loss with respect to the centralized oracle. This loss is called the *regret of learning*. For an arbitrary sequence of task and worker arrivals, the regret is formally defined as

$$R(T) = \mathrm{E}\left[\sum_{t=1}^{T} \sum_{j=1}^{\min\{m_t,W_t\}} \Big(p_{s_{t,j}^*}(x_{t,s_{t,j}^*}, c_t) - p_{s_{t,j}}(x_{t,s_{t,j}}, c_t)\Big)\right], \quad (2)$$

which is equivalent to

$$R(T) = \sum_{t=1}^{T} \sum_{j=1}^{\min\{m_t,W_t\}} \Big(\theta_{s_{t,j}^*}(x_{t,s_{t,j}^*}, c_t) - \mathrm{E}[\theta_{s_{t,j}}(x_{t,s_{t,j}}, c_t)]\Big). \quad (3)$$

Here, the expectation is taken with respect to the selections made by the learning algorithm and the randomness of the workers' performances.

IV. A Context-Aware Hierarchical Online Learning Algorithm for Performance Maximization in Mobile Crowdsourcing

The goal of the MCSP is to select, for each arriving task, a set of workers that maximizes the sum of expected performances for that task given the task budget. Since the expected performances are not known a priori by neither MCSP nor the LCs, they have to be learned over time. Moreover, since only the LCs have access to the personal worker contexts, a coordination is needed between the MCSP and the LCs. Below, we propose a hierarchical contextual online learning algorithm, which is based on algorithms [30]–[33] for the *contextual multi-armed bandit problem*. The algorithm is based on the assumption that a worker's expected performance is similar in similar joint personal and task contexts. Therefore, by observing the task context, a worker's personal context and her/his performance when requested to complete a task, the worker's context-specific expected performances can be learned and exploited for future worker selection.

We call the proposed algorithm *Hierarchical Context-aware Learning* (HCL). Fig. 2 shows an overview of HCL's operation. In HCL, the MCSP broadcasts the context of each arriving task to the LCs. Upon receiving information about a task, an LC first observes its worker's personal context. If the worker's performance has been observed sufficiently often before given the current joint personal and task context, the LC relies on previous observations to estimate its worker's performance and sends an estimate to the MCSP. If its worker's performance has not been observed sufficiently often before, the LC informs the MCSP that its worker has to be explored. Based on the messages received from the LCs, the MCSP selects a subset of workers. The LC of a selected worker requests its worker to complete the task and observes if the worker accepts or declines the task. If a worker was selected

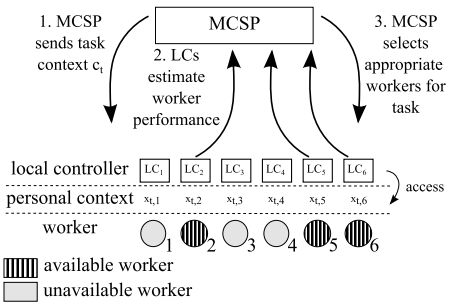


Fig. 2. Overview of operation of HCL algorithm for task t.

for exploration purposes and accepts the task, the LC additionally observes the quality of the completed task, i.e., depending on the type of quality assessment, the LC gets a quality rating from the task owner or it generates an automatic quality assessment using either local software or the resources of a cloud. The reason for only making a quality assessment when a worker was selected for exploration purposes is that quality assessment may be costly.[12] Hence, in this way, HCL keeps the number of costly quality assessments low.

In HCL, a worker's personal contexts, decisions and qualities are only locally stored at the LC. Thereby, (i) personal context is kept locally, (ii) the required storage space for worker information at the MCSP is kept low, (iii) if necessary, task completion and result transmission may be directly handled between the LC and the task owner, (iv) workers receive requests for tasks that are interesting for them and which they are good at, but without the need to share their context information, (v) even though an LC has to keep track of its worker's personal context, decision and quality, the computation and storage overhead for each LC is small.

In more detail, LC i operates as follows, as given in Alg. 1. First, for synchronization purposes, LC i receives the finite number T of tasks to be considered, the task context space $\mathcal{C}$ and its dimension C from the MCSP. Moreover, LC i checks to which of worker i's context dimensions it has access. This defines the personal context space $\mathcal{X}_i$ and its dimension X_i. Then, LC i sets the joint context space to $\mathcal{X}_i \times \mathcal{C}$ with size $D_i = X_i + C$. In addition, LC i has to set a parameter $h_{T,i} \in \mathbb{N}$ and a control function $K_i : \{1, \ldots, T\} \to \mathbb{R}_+$, which are both described below. Next, LC i initializes a uniform partition $\mathcal{Q}_{T,i}$ of worker i's joint context space $[0,1]^{D_i}$, which consists of $(h_{T,i})^{D_i}$ D_i-dimensional hypercubes of equal size $\frac{1}{h_{T,i}} \times \ldots \times \frac{1}{h_{T,i}}$. Hence, the parameter $h_{T,i} \in \mathbb{N}$ determines the granularity of the partition of the context space. Moreover, LC i initializes a counter $N_{i,q}(t)$ for each hypercube $q \in \mathcal{Q}_{T,i}$. The counter $N_{i,q}(t)$ represents the number of times before (i.e., up to, but not including) task t, in which worker i was selected to complete a task for exploration purposes when her/his joint context belonged to hypercube q. Additionally, for each hypercube $q \in \mathcal{Q}_{T,i}$, LC i initializes the estimate $\hat{\theta}_{i,q}(t)$, which represents the estimated performance of worker i for contexts in hypercube q before task t.

Then, LC i executes the following steps for each of the tasks $t = 1, \ldots, T$. For an arriving task t, LC i only takes actions if its worker i is currently available (i.e., $i \in \mathcal{W}_t$). If this is the case, LC i first receives the task context c_t sent by the MCSP.[13] Moreover, LC i observes worker i's current personal context $x_{t,i}$ and determines the hypercube from $\mathcal{Q}_{T,i}$ to which the joint context $(x_{t,i}, c_t)$ belongs.[14] We denote this hypercube by $q_{t,i} \in \mathcal{Q}_{T,i}$. It satisfies $(x_{t,i}, c_t) \in q_{t,i}$. Then, LC i checks if worker i has not been selected sufficiently often before when worker i's joint personal and task context belonged to hypercube $q_{t,i}$. For this purpose, LC i compares the counter $N_{i,q_{t,i}}(t)$ with $K_i(t)$, where $K_i : \{1, \ldots, T\} \to \mathbb{R}_+$ is a deterministic, monotonically increasing control function, set in the beginning of the algorithm. On the one hand, if worker i has been selected sufficiently often before ($N_{i,q_{t,i}}(t) > K_i(t)$), LC i relies on the estimated performance $\hat{\theta}_{i,q_{t,i}}(t)$, and sends it to the MCSP. On the other hand, if worker i has not been selected sufficiently often before ($N_{i,q_{t,i}}(t) \leq K_i(t)$), LC i

Algorithm 1 HCL@LC: Local Controller i of Worker i

Receive input from MCSP: T, $\mathcal{C}$, C
Receive input from worker i: $\mathcal{X}_i$, X_i
Set joint context space $\mathcal{X}_i \times \mathcal{C}$, set $D_i = X_i + C$
Set parameter $h_{T,i} \in \mathbb{N}$ and control function $K_i : \{1, \ldots, T\} \to \mathbb{R}_+$
Initialize context partition: Create partition $\mathcal{Q}_{T,i}$ of $[0,1]^{D_i}$ into $(h_{T,i})^{D_i}$ hypercubes of identical size
Initialize counters: For all $q \in \mathcal{Q}_{T,i}$, set $N_{i,q} = 0$
Initialize estimated performance: For all $q \in \mathcal{Q}_{T,i}$, set $\hat{\theta}_{i,q} = 0$
for each $t = 1, \ldots, T$ **do**
  if $i \in \mathcal{W}_t$ **then**
    Receive task context c_t
    Observe worker i's personal context $x_{t,i}$
    Find the set $q_{t,i} \in \mathcal{Q}_{T,i}$ such that $(x_{t,i}, c_t) \in q_{t,i}$
    if $N_{i,q_{t,i}} > K_i(t)$ **then**
      Send $\text{message}_i := \hat{\theta}_{i,q_{t,i}}$ to MCSP
    else
      Send $\text{message}_i :=$ "explore" to MCSP
    end if
    Wait for MCSP's worker selection
    if MCSP selects worker i **then**
      Give task context c_t to worker i
      Request worker i to complete task t
      Observe worker i's decision d
      if $\text{message}_i ==$ "explore" **then**
        if $d == 1$ **then**
          Observe worker i's quality q, set $p := q$
        else
          Set $p := 0$
        end if
        $\hat{\theta}_{i,q_{t,i}} = \frac{\hat{\theta}_{i,q_{t,i}} N_{i,q_{t,i}} + p}{N_{i,q_{t,i}} + 1}$
        $N_{i,q_{t,i}} = N_{i,q_{t,i}} + 1$
      end if
    end if
  end if
end for

[12] If quality assessment is cheap, HCL can be adapted to always observe worker quality. This may increase the learning speed.

[13] A worker being unavailable may mean that she/he is offline. Therefore, we here consider the LC to only take actions if its worker is available.

[14] If there are multiple such hypercubes, then, one of them is randomly selected.

sends an "explore" message to the MCSP. The control function $K_i(t)$ is hence needed to distinguish when a worker should be selected for exploration (to achieve reliable estimates) or when the worker's performance estimates are already reliable and can be exploited. Therefore, the choice of control function is essential to ensure a good result of the learning algorithm, since it determines the trade-off between exploration and exploitation. Then, LC i waits for the MCSP to take care of the worker selection. If worker i is not selected, LC i does not take further actions. However, if the MCSP selects worker i, LC i gives the task context information c_t to worker i via the application's user interface and requests worker i to complete the task. Then, LC i observes whether worker i declines or accepts the task. If worker i was selected for exploration purposes, LC i makes an additional counter update. For this, if worker i accepted the task, LC i additionally observes worker i's quality in completing the task (e.g., by receiving a quality rating from the task owner or by generating an automatic quality assessment) and sets the observed performance to the observed quality. If worker i declined the task, LC i sets the observed performance to 0. Then, based on the observed performance, LC i computes the estimated performance $\hat{\theta}_{i,q_{t,i}}(t+1)$ for hypercube $q_{t,i}$ and the counter $N_{i,q_{t,i}}(t+1)$. Note that in Alg. 1, the argument t is omitted from counters $N_{i,q}(t)$ and estimates $\hat{\theta}_{i,q}(t)$ since it is not necessary to store their respective previous values.

By definition of HCL, the estimated performance $\hat{\theta}_{i,q}(t)$ corresponds to the product of (i) the relative frequency with which worker i accepted tasks when the joint context belonged to hypercube q and (ii) the average quality in completing these tasks. Formally, $\hat{\theta}_{i,q}(t)$ is computed as follows. Let $\mathcal{E}_{i,q}(t)$ be the set of observed performances of worker i before task t when worker i was selected for a task and the joint context was in hypercube q. If before task t, worker i's performance has never been observed before for a joint context in hypercube q, we have $\mathcal{E}_{i,q}(t) = \emptyset$ and $\hat{\theta}_{i,q}(t) := 0$. Otherwise, the estimated performance is given by $\hat{\theta}_{i,q}(t) := \frac{1}{|\mathcal{E}_{i,q}(t)|} \sum_{p \in \mathcal{E}_{i,q}(t)} p$. However, in HCL, the set $\mathcal{E}_{i,q}(t)$ does not appear, since the estimated performance $\hat{\theta}_{i,q}(t)$ can be computed based on $\hat{\theta}_{i,q}(t-1)$, $N_{i,q}(t-1)$ and on the performance for task $t-1$.

In HCL, the MCSP is responsible for the worker selection, which it executes according to Alg. 2. First, for synchronization purposes, the MCSP sends the finite number T of tasks to be considered, the task context space $\mathcal{C}$ and its dimension C to the LCs. Then, for each arriving task $t = (b_t, c_t)$, the MCSP computes the maximum number m_t of workers, based on the budget b_t and the price e_t per worker. In addition, the MCSP initializes two sets. The set $\mathcal{W}_t$ represents the set of available workers when task t arrives, while $\mathcal{W}_t^{\text{ue}}$ is the so-called *set of under-explored workers*, which contains all available workers that have not been selected sufficiently often before. After broadcasting the task context c_t, the MCSP waits for messages from the LCs. If the MCSP receives a message from an LC, it adds the corresponding worker to the set $\mathcal{W}_t$ of available workers. Moreover, in this case the MCSP additionally checks if the received message is an "explore" message. If this is the case, the MCSP adds the corresponding worker to the set $\mathcal{W}_t^{\text{ue}}$ of under-explored workers. Note that according to Alg. 1 and Alg. 2, the set of under-explored workers is hence given by

$$\mathcal{W}_t^{\text{ue}} = \{i \in \mathcal{W}_t : N_{i,q_{t,i}}(t) \leq K_i(t)\}. \tag{4}$$

Algorithm 2 HCL@MCSP: Worker Selection at MCSP

```
Send input to LCs: T, C, C
for each t = 1, ..., T do
    Receive task t = (b_t, c_t)
    Compute m_t = ⌊b_t / e_t⌋
    Set W_t = ∅
    Set W_t^ue = ∅
    Broadcast task context c_t
    for each i = 1, ..., W do
        if Receive message_i from LC i then
            W_t = W_t ∪ {i}
            if message_i == "explore" then
                W_t^ue = W_t^ue ∪ {i}
            end if
        end if
    end for
    Compute W_t = |W_t|
    if W_t ≤ m_t then                                   ▷ SELECT ALL
        Select all W_t workers from W_t
    else
        Compute n_ue,t = |W_t^ue|
        if n_ue,t == 0 then                             ▷ EXPLOITATION
            Rank workers in W_t according to estimates from (message_i)_{i∈W_t}
            Select the m_t highest ranked workers
        else                                            ▷ EXPLORATION
            if n_ue,t ≥ m_t then
                Select m_t workers randomly from W_t^ue
            else
                Select the n_ue,t workers from W_t^ue
                Rank workers in W_t \ W_t^ue according to estimates from (message_i)_{i∈W_t\W_t^ue}
                Select the (m_t − n_ue,t) highest ranked workers
            end if
        end if
    end if
    Inform LCs of selected workers
end for
```

Next, the MCSP calculates the number W_t of available workers. If $W_t \leq m_t$, i.e., at most the required number of workers are available, the MCSP enters a *select-all-workers phase* and selects all available workers to complete the task. Otherwise, the MCSP continues by calculating the number $n_{\text{ue},t} := |\mathcal{W}_t^{\text{ue}}|$ of under-explored workers. If there is no under-explored worker, the MCSP enters an *exploitation phase*. It ranks the available workers in $\mathcal{W}_t$ according to the estimated performances, which it received from their respective LCs. Then, the MCSP selects the m_t highest ranked workers. By this procedure, the MCSP is able to use context-specific estimated performances without actually observing the workers' personal contexts. If there are under-explored workers, the MCSP enters an *exploration phase*. These phases are needed, such that all LCs are able to update their estimated performances sufficiently often. Here, two different cases may occur, depending on the number $n_{\text{ue},t}$ of under-explored workers. Either the number $n_{\text{ue},t}$ of under-explored workers is at least m_t, in which case the MCSP selects m_t under-

explored workers at random. Or the number $n_{\mathrm{ue},t}$ of under-explored workers is smaller than m_t, in which case the MCSP selects all $n_{\mathrm{ue},t}$ under-explored workers. Since it should select $m_t - n_{\mathrm{ue},t}$ additional workers, it ranks the available sufficiently-explored workers according to the estimated performances, which it received from their respective LCs. Then, the MCSP additionally selects the $(m_t - n_{\mathrm{ue},t})$ highest ranked workers. In this way, additional exploitation is carried out in exploration phases, when the number of under-explored workers is small. After worker selection, the MCSP informs the LCs of selected workers that their workers should be requested to complete the task. Note that since the MCSP does not have to keep track of the workers' decisions, the LCs may handle the contact to the task owner directly (e.g., the task owner may send detailed task instructions directly to the LC; after task completion, the LC may send the result to the task owner).

V. Theoretical Analysis

A. Upper Bound on Regret

The performance of HCL is evaluated by analyzing its regret, see Eq. (2), with respect to the centralized oracle. In this section, we derive a sublinear bound on the regret, i.e., we show that $R(T) = O(T^\gamma)$ with some $\gamma < 1$ holds. Hence, our algorithm converges to the centralized oracle for $T \to \infty$, since $\lim_{T\to\infty} \frac{R(T)}{T} = 0$ holds. The regret bound is derived based on the assumption that under a similar joint personal and task context, a worker's expected performance is also similar. This assumption can be formalized as follows.[15]

Assumption 1 (Hölder Continuity Assumption): There exists $L > 0$, $0 < \alpha \leq 1$ such that for all workers $i \in \mathcal{W}$ and for all joint contexts $(x, c), (\tilde{x}, \tilde{c}) \in \mathcal{X}_i \times \mathcal{C} \equiv [0,1]^{D_i}$, it holds that

$$|\theta_i(x,c) - \theta_i(\tilde{x},\tilde{c})| \leq L||(x,c) - (\tilde{x},\tilde{c})||_i^\alpha,$$

where $||\cdot||_i$ denotes the Euclidean norm in $\mathbb{R}^{D_i}$.

The theorem given below shows that the regret of HCL is sublinear in the time horizon T.

Theorem 1 (Bound for $R(T)$): Given that Assumption 1 holds, when LC i, $i \in \mathcal{W}$, runs Alg. 1 with parameters $K_i(t) = t^{\frac{2\alpha}{3\alpha+D_i}} \log(t)$, $t = 1, \ldots, T$, and $h_{T,i} = \lceil T^{\frac{1}{3\alpha+D_i}} \rceil$, and the MCSP runs Alg. 2, the regret $R(T)$ is bounded by

$$\begin{aligned} R(T) \leq\ & q_{\max} W \sum_{i\in\mathcal{W}} 2^{D_i} \left(\log(T) T^{\frac{2\alpha+D_i}{3\alpha+D_i}} + T^{\frac{D_i}{3\alpha+D_i}} \right) \\ & + \sum_{i\in\mathcal{W}} \frac{2 q_{\max}}{(2\alpha + D_i)/(3\alpha + D_i)} T^{\frac{2\alpha+D_i}{3\alpha+D_i}} \\ & + q_{\max} W^2 \frac{\pi^2}{3} + 2 \sum_{i\in\mathcal{W}} L D_i^{\frac{\alpha}{2}} T^{\frac{2\alpha+D_i}{3\alpha+D_i}}. \end{aligned}$$

Hence, the leading order of the regret is $O\left(q_{\max} W^2\ T^{\frac{2\alpha+D_{\max}}{3\alpha+D_{\max}}} \log(T)\right)$, where $D_{\max} := \max_{i\in\mathcal{W}} D_i$.

The proof of Theorem 1 is given in the supplementary material in Appendix A. Theorem 1 shows that HCL converges to the centralized oracle in the sense that when the number T of tasks goes to infinity, the averaged regret $\frac{R(T)}{T}$ diminishes. Moreover, since Theorem 1 is applicable for any finite number T of tasks, it characterizes HCL's speed of learning.

[15]Note that our algorithm can also be applied to data, which does not satisfy this assumption. In this case, the regret bound may, however, not hold.

B. Local Storage Requirements

The required local storage size in the mobile device of a worker is determined by the storage size needed when the LC executes Alg. 1. In Alg. 1, LC i stores the counters $N_{i,q}$ and estimates $\hat{\theta}_{i,q}$ for each $q \in \mathcal{Q}_{T,i}$. Using the parameters from Theorem 1, the number of hypercubes in the partition $\mathcal{Q}_{T,i}$ is $(h_{T,i})^{D_i} = \lceil T^{\frac{1}{3\alpha+D_i}} \rceil^{D_i} \leq (1 + T^{\frac{1}{3\alpha+D_i}})^{D_i}$. Hence, the number of variables to store in the mobile device of worker i is upper bounded by $2 \cdot (1 + T^{\frac{1}{3\alpha+D_i}})^{D_i}$. Hence, the required storage depends on the number $D_i = X_i + C$ of context dimensions. If the worker allows access to a high number X_i of personal context dimensions and/or the number C of task context dimensions is large, the algorithm learns the worker's context-specific performance with finer granularity and therefore the assigned tasks are more personalized, but also the required storage size will increase.

C. Communication Requirements

The communication requirements of HCL can be deduced from its main operation steps. For each task t, the MCSP broadcasts the task context to the LCs, which is one vector of dimension C (i.e., C scalars), assuming that the broadcast reaches all workers in a single transmission. Then, the LCs of available workers send their workers' estimated performances to the MCSP. This corresponds to W_t scalars to be transmitted (one scalar sent by each LC of an available worker). Finally, the MCSP informs selected workers about its decision, which corresponds to m_t scalars sent by the MCSP. Hence, for task t, in sum, $C + W_t + m_t$ scalars are transmitted. Among these, $C + m_t$ scalars are transmitted by the MCSP and one scalar is transmitted by each mobile device of an available worker.

We now compare the communication requirements of HCL and of its centralized version, called here CCL. In CCL, for each task, the personal contexts of available workers are gathered in the MCSP, which then selects workers based on the task and personal contexts and informs selected workers about its decision. The communication requirements of CCL are as follows: For each task t, the LC of each available worker i sends the current worker context to the MCSP, which is a vector of dimension D_i (i.e., D_i scalars). Hence, in sum, $\sum_{i\in\mathcal{W}_t} D_i$ scalars are transmitted. After worker selection, the MCSP requests selected workers to complete the task, which corresponds to m_t scalars sent by the MCSP. Moreover, the MCSP broadcasts the task context to the selected workers, which is one vector of dimension C (i.e., C scalars), assuming that the broadcast reaches all addressed workers in a single transmission. Hence, in total, $\sum_{i\in\mathcal{W}_t} D_i + m_t + C$ scalars are transmitted for task t. Among these, $C + m_t$ scalars are transmitted by the MCSP and D_i scalars are transmitted by each mobile device of an available worker.

We now compare HCL with CCL. The mobile device of any worker $i \in \mathcal{W}$ with $D_i > 1$ has to transmit less using HCL than using CCL. Moreover, under the assumption that any broadcast reaches all addressed workers using one single transmission, if $D_i \geq 1$ for all $i \in \mathcal{W}$ (i.e., each worker gives access to at least one personal context), the sum communication requirements (for all mobile devices and the MCSP in sum) of HCL are at most as high as that of CCL.

D. Worker Quality Assessment Requirements

Observing a worker's quality might be costly. HCL explicitly takes this into account by only requesting a quality

assessment if a worker is selected for exploration purposes. Here, we give an upper bound on the number $A_i(T)$ of quality assessments per worker up to task T.

Corollary 1 (Bound for Number of Quality Assessments up to Task T): Given that Assumption 1 holds, when LC i, $i \in \mathcal{W}$, runs Alg. 1 with the parameters given in Theorem 1, and the MCSP runs Alg. 2, the number $A_i(T)$ of quality assessments of each worker i up to task T is upper bounded by

$$A_i(T) \leq (1 + T^{\frac{1}{3\alpha+D_i}})^{D_i} \left(1 + \log(T) T^{\frac{2\alpha}{3\alpha+D_i}}\right).$$

The proof of Corollary 1 is given in the supplementary material in Appendix B. From Corollary 1, we see that the number of quality assessments per worker is sublinear in T. Hence, it holds $\lim_{T\to\infty} \frac{A_i(T)}{T} = 0$, so that for $T \to \infty$, the average rate of quality assessments approaches zero.

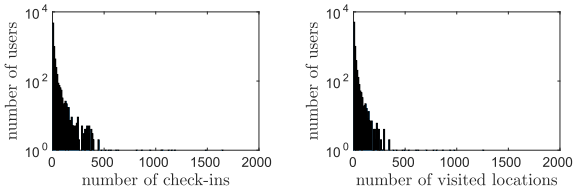
VI. Numerical Results

We evaluate HCL by comparing its performance with various algorithms based on synthetic and real data.

A. Reference Algorithms

The following algorithms are used for comparison.

- The *(Centralized) Oracle* has perfect a priori knowledge about context-specific expected performances and knows the current contexts of available workers.
- *LinUCB* assumes that the expected performance of a worker is linear in its context [25], [26]. Based on a linear reward function over contexts and previously observed context-specific worker performances, for each task, LinUCB chooses the m_t available workers with highest estimated upper confidence bounds on their expected performance. LinUCB has an input parameter λ_{LinUCB}, controlling the influence of the confidence bound. LinUCB is used in [20] for task assignment in spatial CS.
- *AUER* [36] is an extension of the well-known UCB algorithm [37] to the sleeping arm case. It learns from previous observations of worker performances, but without taking into account context information. Based on the history of previous observations of worker performances, AUER selects the m_t available workers with highest estimated upper confidence bounds on their expected performance. AUER has an input parameter λ_{AUER}, which controls the influence of the confidence bound.
- *ϵ-Greedy* selects a random subset of available workers with a probability of $\epsilon \in (0, 1)$. With a probability of $(1 - \epsilon)$, ϵ-Greedy selects the m_t available workers with highest estimated performance. The estimated performance of a worker is computed based on the history of previous performances [37], but without taking into account context.
- *Myopic* only learns from the last interaction with each worker. For task 1, it selects a random subset of m_1 workers. For each of the following tasks, it checks which of the available workers have previously accepted a task. If more than m_t of the available workers have accepted a task when requested the last time, Myopic selects out of these workers the m_t workers with the highest performance in their last completed task. Otherwise, Myopic selects all of these workers and an additional subset of random workers so that in total m_t workers are selected.
- *Random* selects a random subset of m_t available workers for each task t.



(a) (b)

Fig. 3. Statistics of used Gowalla-NY data set. (a) Check-ins. (b) Visited locations.

Note that, if an algorithm originally would have selected only one worker per task, we adapted it to select m_t workers per task. Also, above, we described the behavior of the algorithms for the case $m_t < W_t$. In the case of $m_t \geq W_t$, we adapted each algorithm such that it selects all available workers. Moreover, while we used standard centralized implementations of the reference algorithms, they could also be decoupled to a hierarchical setting like HCL.

B. Evaluation Metrics

Each algorithm is run over a sequence of tasks $t = 1, \ldots, T$ and its result is evaluated using the following metrics. We compute the *cumulative worker performance at T* achieved by an algorithm, which is the cumulative sum of performances by all selected workers up to (and including) task T. Formally, if the set of selected workers of an algorithm A for task t is $\{s^A_{t,j}\}_{j=1,\ldots,\min\{m_t,W_t\}}$ and $p_{s^A_{t,j}}(t)$ is the observed performance of worker $s^A_{t,j}$, the cumulative worker performance at T achieved by algorithm A is

$$\Gamma_T(A) := \sum_{t=1}^{T} \sum_{j=1}^{\min\{m_t,W_t\}} p_{s^A_{t,j}}(t).$$

As a function of the arriving tasks, we compute the *average worker performance up to t* achieved by an algorithm, which is the average performance of all selected workers up to task t. Formally, it is defined by

$$\frac{1}{\sum_{\tilde{t}=1}^{t} \min\{m_{\tilde{t}}, W_{\tilde{t}}\}} \sum_{\tilde{t}=1}^{t} \sum_{j=1}^{\min\{m_{\tilde{t}},W_{\tilde{t}}\}} p_{s^A_{\tilde{t},j}}(\tilde{t}).$$

C. Simulation Setup

We evaluate the algorithms using synthetic and real data. The difference between the two approaches lies in the arrival process of workers and their contexts. To produce synthetic data, we generate workers and their contexts based on some predefined distributions as described below. In case of real data, similar to, e.g., [6], [20], [22], we use a data set from Gowalla [38]. Gowalla is a location-based social network where users share their location by checking in at "spots", i.e., certain places in their vicinity. We use the check-ins to simulate the arrival process of workers and their contexts. The Gowalla data set consists of 6,442,892 check-ins of 107,092 distinct users over the period of February 2009 to October 2010. Each entry of the data set consists of the form (User ID, Check-in Time, Latitude, Longitude, Location ID). Similar to [22], we first extract the check-ins in New York City, which leaves a subset of 138,954 check-ins of 7,115 distinct users at 21,509 distinct locations. This resulting Gowalla-NY data set is used below. Fig. 3(a) and Fig. 3(b) show the

distributions of the number of check-ins and the number of distinct locations visited by the users in the Gowalla-NY data set, respectively.

For both synthetic and real data, we simulate an MCSP, to which a set of $W = 100$ workers belongs. For synthetic data, 100 workers are created in the beginning. For real data, we randomly select 100 users from the Gowalla-NY data set, which represent the 100 workers of the MCSP. Then we use this reduced Gowalla-NY data set containing the check-ins of 100 users. On top of this, the simulation is modeled as follows:

1) Task Properties: The task context is assumed to be uniformly distributed in $\mathcal{C} = [0, 1]$ (i.e., $C = 1$). Task owners have to pay a fixed price of $e = 0.75$ or $e = 1$ per requested worker that completes the task when the task context lies in $[0, 0.5]$ or $(0.5, 1]$, respectively. The quality of a completed task lies in the range $q_{\min} = 0$ and $q_{\max} = 5$. The task budget is sampled from a normal distribution with expected value 20 and standard deviation of 5, truncated between 1 and 100.

2) Worker Availability: For synthetic data, we let each worker be available with a probability of $\rho = 0.7$ (default value) for each arriving task. For the real data, we use a Binomial distribution with parameters $W = 100$ and $\rho = 0.7$ (default value) to sample the number of available workers W_t for an arriving task.[16] Having sampled W_t, we randomly draw samples from the reduced Gowalla-NY data set (consisting of the check-ins of 100 users) until these samples contain W_t distinct users. These W_t sampled users correspond to the available workers (i.e., users with higher number of check-ins in the reduced Gowalla-NY data set translate to workers that are more often available for the MCSP).

3) Worker Context: The personal context space of an available worker i is set to $\mathcal{X}_i = [0, 1]^2$ (i.e., $X_i = 2$). The first personal context dimension refers to the worker's battery state, which is sampled from a uniform distribution in $[0, 1]$. The second personal context dimension refers to the worker's location, which is sampled differently in case of synthetic and real data. For synthetic data, the worker's location is sampled from 5 different (personal) locations, using a weighted discrete distribution with probabilities $\{\frac{1}{2}, \frac{1}{3}, \frac{1}{12}, \frac{1}{24}, \frac{1}{24}\}$ to represent the fact that workers may use the MCS application different amounts of time in different places (e.g., at home more often than at work). For real data, we set the worker's location to be the check-in location of the respective user from the sample.[17]

4) Expected Worker Performance: We use two different models to generate expected worker performance.

a) Discrete performance model: The joint personal and task context space $\mathcal{X}_i \times \mathcal{C}$ (of dimension $D_i = 3$) is split into a uniform grid. For synthetic data, the space is split into 5 identical parts along each of the 3 dimensions, i.e., $5 \cdot 5 \cdot 5 = 125$ subsets are created. For real data, along the dimensions of task context and battery state, the context space is split into 5 identical parts each, but along the dimension of location context, the context space is split into l_i identical parts, where l_i corresponds to the number of distinct locations visited by the corresponding user from the reduced Gowalla-NY data set. Hence, $5 \cdot 5 \cdot l_i$ subsets are created. Then, for both synthetic and real data, in each of the subsets, the expected performance of a worker is a priori sampled uniformly at random from $[0, 5]$. Note that for the real data, since the expected performance differs per visited location, workers with higher number of visited locations have a higher number of different context-specific performances.

b) Hybrid performance model: We assume a continuous dependency of the expected performance on two of the context dimensions. Let $x_i^{(1)}$ and $x_i^{(2)}$ be worker i's battery state and location, respectively, and let c be the task context. We assume that the expected performance θ_i of worker i is given by

$$\theta_i\left(c, x_i^{(1)}, x_i^{(2)}\right) = q_{\max} \cdot w_i\left(x_i^{(2)}\right) \cdot \bar{f}_{\mu_i,\sigma_i^2}(c) \cdot \sqrt{x_i^{(1)}},$$

where $w_i\left(x_i^{(2)}\right)$ is a (discrete) location-specific weighting factor that is a priori sampled uniformly between $[0.5, 1]$ for each of worker i's (finitely many) locations. Moreover, $\bar{f}_{\mu_i,\sigma_i^2}$ is a Gaussian probability density function with mean μ_i and standard deviation σ_i, which is normalized such that its maximum value equals 1. For worker i, the mean μ_i is a priori sampled uniformly from $[0.1, 0.9]$ and the standard deviation is set to $\sigma_i = 0.1 \cdot \mu_i$. Hence, the expected performance is a continuous function of task context and battery state. The hybrid model has the following intuition: The expected performance of a worker is location-specific. Along the task context, the expected performance varies according to a worker-specific Gaussian distribution, i.e., each worker performs well at a specific type of tasks. Finally, the expected performance grows monotonically with the battery state, i.e., with more battery available, workers are more likely to perform well at tasks.

5) Instantaneous Worker Performance: For each occurring joint worker and task context, the instantaneous performance of a worker is sampled by adding noise uniformly sampled from $[-1, 1]$ to the expected performance in the given context (the noise interval is truncated to a smaller interval if the expected performance lies close to either 0 or $q_{\max}$).

D. Parameter Selection

HCL, LinUCB, AUER and ϵ-Greedy require input parameters. In order to find appropriate parameters, we generate 20 synthetic instances using the discrete performance model. Each instance consists of a sequence of $T = 10,000$ task and worker arrivals sampled according to Sec. VI-C. Then, we run each algorithm with different parameters on these instances. Note that for HCL, we set $\alpha = 1$, choose $h_{T,i} = \lceil T^{\frac{1}{3+D_i}} \rceil$, $i \in \mathcal{W}$, as in Theorem 1, and set the control function to $K_i(t) = f t^{\frac{2\alpha}{3\alpha+D_i}} \log(t)$, $t = 1, \ldots, T$, where the factor $f \in (0, 1]$ is included to reduce the number of exploration phases. Then, we search for an appropriate f. Table II shows the parameters at which each of the algorithms on average performed best, respectively. These parameters are used in all of the following simulations.

TABLE II
CHOICE OF PARAMETERS FOR DIFFERENT ALGORITHMS

Algorithm	Parameter	Selected Value
HCL	f	0.003
LinUCB	λ_{LinUCB}	1.5
AUER	λ_{AUER}	0.5
ϵ-Greedy	ϵ	0.01

E. Results Under the Discrete Performance Model

First, we generate 100 synthetic and 100 real instances, in both cases using $\rho = 0.7$ and the discrete performance

[16] In this way, the number of available workers in our experiments using the real and the synthetic data are distributed in the same way.

[17] If a user was sampled several times until we sampled W_t distinct users, we choose her/his first sampled check-in location.

TABLE III

COMPARISON OF CUMULATIVE WORKER PERFORMANCE AT T FOR $\rho = 0.7$ UNDER THE DISCRETE PERFORMANCE MODEL. FOR AN ALGORITHM A, THE TABLE SHOWS $\Gamma_T(A)/\Gamma_T(\text{HCL})$

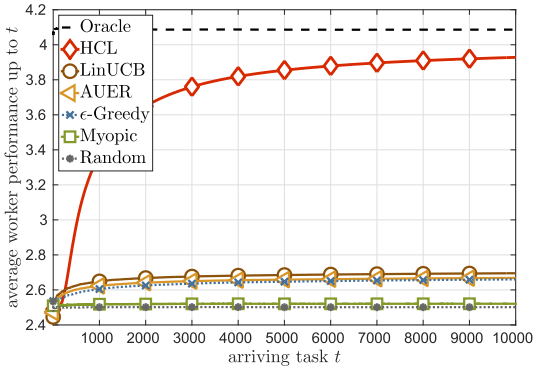
Algorithm	Synthetic Data	Real Data
Oracle	1.04	1.20
HCL	1.00	1.00
LinUCB	0.69	0.78
AUER	0.68	0.77
ϵ-Greedy	0.68	0.76
Myopic	0.64	0.74
Random	0.64	0.73

model. Each instance consists of a sequence of $T = 10,000$ task and worker arrivals sampled according to Sec. VI-C. Then, we run the algorithms on these instances and average the results.

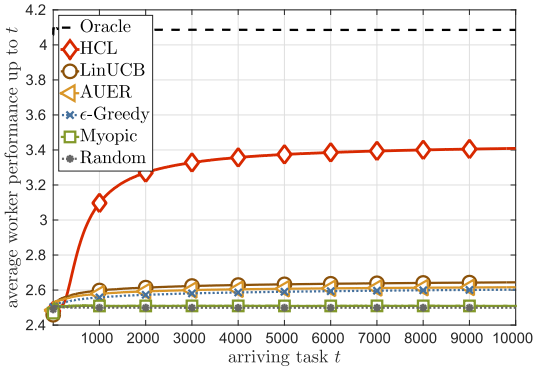
For both synthetic and real data, Table III compares the cumulative worker performance at T of an algorithm A with the one of HCL, by displaying $\Gamma_T(A)/\Gamma_T(\text{HCL})$. As expected, while Oracle outperforms all other algorithms due to its a priori knowledge, Random gives a lower bound on the achievable cumulative performance. HCL clearly outperforms LinUCB, AUER, ϵ-Greedy and Myopic, even though HCL observes worker performance only when requesting a worker for exploration purposes, while the other algorithms have access to worker performance whenever a worker is requested. This is due to the fact that HCL smartly exploits context. Moreover, HCL reaches a result close to the Oracle. In contrast, LinUCB, AUER, ϵ-Greedy and Myopic perform by far worse and lie close to the result of Random. This shows that algorithms which either do not take context into account (i.e., AUER, ϵ-Greedy and Myopic) or have a linearity assumption between context and performance (i.e., LinUCB), cannot cope with the non-linear dependency of expected worker performance on context. Comparing synthetic and real data, HCL has a better performance on the synthetic data, but it still reaches a good result on the real data, even though using real data, each worker has her/his own diversity in context arrival and hence in expected performance (since users in the Gowalla-NY data set have different numbers of visited check-in locations).

Fig. 4(a) and Fig. 4(b) show the average worker performance up to task t as a function of the sequentially arriving tasks $t = 1, \ldots, T$. We see that over the sequence of tasks, the average worker performance achieved by Oracle and Random stay nearly constant at around 4.1 and 2.5, respectively, for both synthetic and real data. LinUCB, AUER, ϵ-Greedy and Myopic increase the average worker performance slightly, starting between 2.4 and 2.5 at $t = 1$ and ending with average performance of between 2.5 and 2.7 at $t = T$. On the contrary, HCL is able to increase the average worker performance from 2.5 at $t = 1$ up to 3.9 (3.4) at $t = T$ for the synthetic (real) data. Hence, HCL learns context-specific worker performances and selects better workers over time.

Finally, we evaluate the impact of worker availability by varying the parameter ρ. For each value of ρ, we average the results over 100 synthetic instances and over 100 real instances for $T = 10,000$, respectively. Fig. 5(a) and 5(b) show the cumulative worker performance at T achieved by the algorithms for different ρ. For small $\rho = 0.1$, all algorithms yield approximately the same performance. This is as expected since given our modeling of task budget, for small ρ, the number of available workers is often smaller than the required number of workers. Since each of the algorithm enters a select-all-workers phase in this case, each algorithm performs optimally. For increasing worker availability ρ, the cumulative performance at T achieved by each of the algorithm increases. However, the gap between Oracle and HCL on the one hand, and the remaining algorithms on the other hand, is increasing for increasing ρ. For example, at $\rho \in \{0.3, 0.7, 1\}$, the cumulative performance achieved by HCL corresponds to $\{1.16, 1.46, 1.49\}$ ($\{1.07, 1.29, 1.34\}$) times the one achieved by the respective next best algorithm {AUER, LinUCB, LinUCB} ({ϵ-Greedy, LinUCB, LinUCB}) for the synthetic (real) data. Hence, the more workers are available, the more severe is the effect of not selecting the best workers and only HCL is able to cope with the more difficult worker selection.



(a)



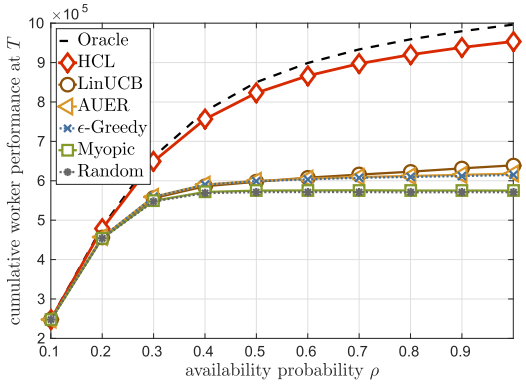
(b)

Fig. 4. Average worker performance up to task t for sequence $t = 1, \ldots, T$ for $\rho = 0.7$ under the discrete performance model. (a) Experiments with synthetic data. (b) Experiments with real data.

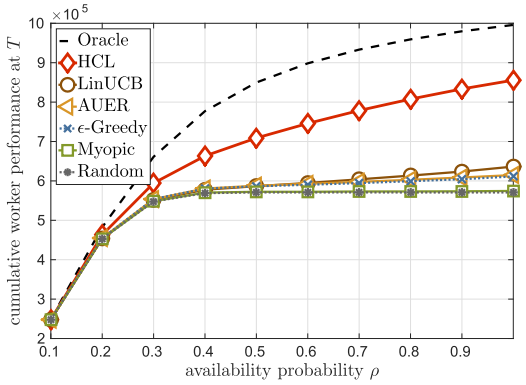
F. Results Under the Hybrid Performance Model

First, we run the algorithms on 100 real instances for $T = 10,000$ and $\rho = 0.7$ using the hybrid performance model. Fig. 6 shows the average worker performance up to task t as a function of the sequentially arriving tasks $t = 1, \ldots, T$.[18] The average worker performance achieved by Oracle and Random stay nearly constant at around 0.88 and 0.29 over the sequence of tasks. AUER, ϵ-Greedy and Myopic increase the average worker performance only slightly,

[18]Note that worker performance is differently distributed in the hybrid than in the discrete model, so that the absolute values presented in Sec. VI-F are not comparable to those in Sec. VI-E.



(a)



(b)

Fig. 5. Impact of worker availability on cumulative worker performance at T for $T = 10,000$ tasks under the discrete performance model. (a) Experiments with synthetic data. (b) Experiments with real data.

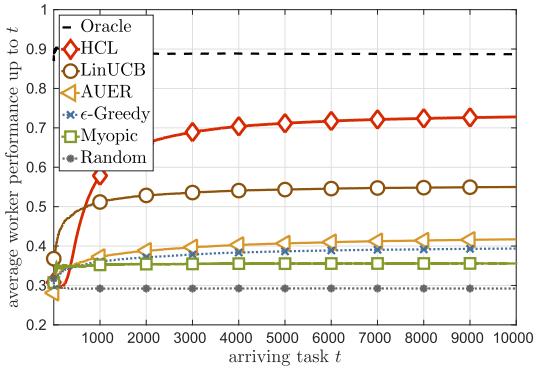


Fig. 6. Average worker performance up to task t for sequence $t = 1, \ldots, T$ for $\rho = 0.7$ under the hybrid performance model using real data.

from between 0.28 and 0.31 at $t = 1$ to between 0.36 and 0.42 at $t = T$. LinUCB has a larger increase from 0.37 at $t = 1$ to 0.55 at $t = T$. Compared to the discrete performance model, LinUCB performs better here due to the monotonic dependency of expected performance on battery state. Still, HCL has the largest increase from 0.31 at $t = 1$ up to 0.73 at $t = T$. Finally, we evaluate the impact of worker availability ρ. For each value of ρ, we average the results over 100 real instances for $T = 10,000$. Fig. 7 shows the cumulative worker performance at T achieved by the algorithms for different ρ. Again, for higher ρ, the algorithms achieve higher cumulative performances at T. While LinUCB performs better compared to the results under the discrete performance model, still, the gap in cumulative performance

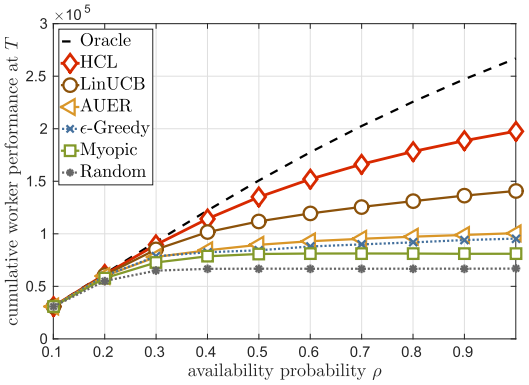


Fig. 7. Impact of worker availability on cumulative worker performance at T for $T = 10,000$ tasks under the hybrid performance model using real data.

between HCL and LinUCB is increasing for increasing ρ. For example, at $\rho \in \{0.3, 0.7, 1\}$, the cumulative performance achieved by HCL corresponds to $\{1.05, 1.32, 1.40\}$ times the one achieved by LinUCB.

VII. Conclusion

In this paper, we presented a context-aware hierarchical online learning algorithm, which learns context-specific worker performance online over time in order to maximize the performance in an MCS system for location-independent tasks. Our algorithm is split into two parts, one executed by LCs in the mobile devices of the workers, the other executed by the central MCSP. While the LCs learn their workers' performances, the MCSP assigns workers to tasks based on regular information exchange with the LCs. Our hierarchical approach ensures that the most suitable workers are requested by the MCSP. The learning in LCs ensures that personal worker context can be kept locally, but still workers are offered those tasks they are interested in the most. We showed that the requirements of our algorithm in terms of storage, communication and the number of quality assessments are small. Moreover, the algorithm converges to the optimal task assignment strategy.

References

[1] J. Ren, Y. Zhang, K. Zhang, and X. Shen, "Exploiting mobile crowdsourcing for pervasive cloud services: Challenges and solutions," *IEEE Commun. Mag.*, vol. 53, no. 3, pp. 98–105, Mar. 2015.

[2] *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021*. Accessed: Mar. 21, 2017. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.pdf

[3] *eMarketer Report*. Accessed: Mar. 21, 2017. [Online]. Available: https://www.emarketer.com/Report/US-Time-Spent-with-Mobile-Deep-Dive-Mobile-App-Web-Time/2001835

[4] Y. Zhao and Q. Han, "Spatial crowdsourcing: Current state and future directions," *IEEE Commun. Mag.*, vol. 54, no. 7, pp. 102–107, Jul. 2016.

[5] R. Bonney *et al.*, "Next steps for citizen science," *Science*, vol. 343, no. 6178, pp. 1436–1437, Mar. 2014.

[6] L. Kazemi and C. Shahabi, "GeoCrowd: Enabling query answering with spatial crowdsourcing," in *Proc. 20th ACM Int. Conf. Adv. Geogr. Inf. Syst. (SIGSPATIAL)*, 2012, pp. 189–198.

[7] D. Geiger and M. Schader, "Personalized task recommendation in crowdsourcing information systems—Current state of the art," *Decision Support Syst.*, vol. 65, pp. 3–16, Sep. 2014.

[8] L. B. Chilton, J. J. Horton, R. C. Miller, and S. Azenkot, "Task search in a human computation market," in *Proc. ACM SIGKDD Workshop Hum. Comput. (HCOMP)*, 2010, pp. 1–9.

[9] A. Doan, R. Ramakrishnan, and A. Y. Halevy, "Crowdsourcing systems on the World-Wide Web," *Commun. ACM*, vol. 54, no. 4, pp. 86–96, Apr. 2011.

[10] K. Ahuja and M. van der Schaar. (Sep. 2016). "Dynamic assessments, matching and allocation of tasks." [Online]. Available: https://arxiv.org/abs/1602.02439

[11] Y. Liu and M. Liu, "An online learning approach to improving the quality of crowd-sourcing," *IEEE/ACM Trans. Netw.*, vol. 25, no. 4, pp. 2166–2179, Aug. 2017.
[12] A. Slivkins and J. W. Vaughan, "Online decision making in crowdsourcing markets: Theoretical challenges," *SIGecom Exchanges*, vol. 12, no. 2, pp. 4–23, Dec. 2013.
[13] C.-J. Ho and J. W. Vaughan, "Online task assignment in crowdsourcing markets," in *Proc. 26th AAAI Conf. Artif. Intell.*, 2012, pp. 45–51.
[14] L. Tran-Thanh, S. Stein, A. Rogers, and N. R. Jennings, "Efficient crowdsourcing of unknown experts using bounded multi-armed bandits," *Artif. Intell.*, vol. 214, pp. 89–111, Sep. 2014.
[15] M. Safran and D. Che, "Real-time recommendation algorithms for crowdsourcing systems," *Appl. Comput. Inform.*, vol. 13, no. 1, pp. 47–56, Jan. 2017.
[16] V. Ambati, S. Vogel, and J. Carbonell, "Towards task recommendation in micro-task markets," in *Proc. 11th AAAI Conf. Hum. Comput.*, 2011, pp. 80–83.
[17] Y. Gong, L. Wei, Y. Guo, C. Zhang, and Y. Fang, "Optimal task recommendation for mobile crowdsourcing with privacy control," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 745–756, Oct. 2016.
[18] K. Han, C. Zhang, and J. Luo, "Taming the uncertainty: Budget limited robust crowdsensing through online learning," *IEEE/ACM Trans. Netw.*, vol. 24, no. 3, pp. 1462–1475, Jun. 2016.
[19] H. To, C. Shahabi, and L. Kazemi, "A server-assigned spatial crowdsourcing framework," *ACM Trans. Spatial Algorithms Syst.*, vol. 1, no. 1, pp. 2:1–2:28, Jul. 2015.
[20] U. Ul Hassan and E. Curry, "A multi-armed bandit approach to online spatial task assignment," in *Proc. 11th IEEE Int. Conf. Ubiquitous Intell. Comput. (UTC)*, Dec. 2014, pp. 212–219.
[21] H. To, G. Ghinita, L. Fan, and C. Shahabi, "Differentially private location protection for worker datasets in spatial crowdsourcing," *IEEE Trans. Mobile Comput.*, vol. 16, no. 4, pp. 934–949, Apr. 2017,
[22] L. Zheng and L. Chen, "Maximizing acceptance in rejection-aware spatial crowdsourcing," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 9, pp. 1943–1956, Sep. 2017.
[23] J. Langford and T. Zhang, "The epoch-greedy algorithm for multi-armed bandits with side information," in *Proc. 20th Int. Conf. Neural Inf. Process. Syst.*, 2007, pp. 817–824.
[24] A. Badanidiyuru, J. Langford, and A. Slivkins, "Resourceful contextual bandits," in *Proc. 27th Conf. Learn. Theory*, 2014, pp. 1–26.
[25] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *Proc. 19th ACM Int. Conf. World Wide Web (WWW)*, 2010, pp. 661–670.
[26] W. Chu, L. Li, L. Reyzin, and R. E. Schapire, "Contextual bandits with linear payoff functions," in *Proc. 14th Int. Conf. Artif. Intell. Stat. (AISTATS)*, 2011, pp. 208–214.
[27] S. Agrawal and N. Goyal, "Thompson sampling for contextual bandits with linear payoffs," in *Proc. 30th Int. Conf. Mach. Learn. (ICML)*, vol. 28. 2013, pp. 127–135.
[28] C. Gentile, S. Li, and G. Zappella, "Online clustering of bandits," in *Proc. 31st Int. Conf. Mach. Learn. (ICML)*, 2014, pp. 1–9.
[29] A. Slivkins, "Contextual bandits with similarity information," *J. Mach. Learn. Res.*, vol. 15, pp. 2533–2568, Jan. 2014.
[30] C. Tekin and M. van der Schaar, "Distributed online learning via cooperative contextual bandits," *IEEE Trans. Signal Process.*, vol. 63, no. 14, pp. 3700–3714, Jul. 2015.
[31] C. Tekin, S. Zhang, and M. van der Schaar, "Distributed online learning in social recommender systems," *IEEE J. Sel. Topics Signal Process.*, vol. 8, no. 4, pp. 638–652, Aug. 2014.
[32] S. Müller, O. Atan, M. van der Schaar, and A. Klein, "Context-aware proactive content caching with service differentiation in wireless networks," *IEEE Trans. Wireless Commun.*, vol. 16, no. 2, pp. 1024–1036, Feb. 2017.
[33] S. Müller, O. Atan, M. van der Schaar, and A. Klein, "Smart caching in wireless small cell networks via contextual multi-armed bandits," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–7.
[34] H.-P. Shiang and M. van der Schaar, "Delay-sensitive resource management in multi-hop cognitive radio networks," in *Proc. 3rd IEEE Symp. New Frontiers Dyn. Spectr. Access Netw.*, Oct. 2008, pp. 1–12.
[35] N. Mastronarde and M. van der Schaar, "Fast reinforcement learning for energy-efficient wireless communication," *IEEE Trans. Signal Process.*, vol. 59, no. 12, pp. 6262–6266, Dec. 2011.
[36] R. Kleinberg, A. Niculescu-Mizil, and Y. Sharma, "Regret bounds for sleeping experts and bandits," *Mach. Learn.*, vol. 80, nos. 2–3, pp. 245–272, 2010.
[37] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, no. 2, pp. 235–256, 2002.
[38] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: User movement in location-based social networks," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2011, pp. 1082–1090.
[39] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *J. Amer. Stat. Assoc.*, vol. 58, no. 301, pp. 13–30, 1963.
[40] E. Chlebus, "An approximate formula for a partial sum of the divergent p-series," *Appl. Math. Lett.*, vol. 22, no. 5, pp. 732–737, May 2009.

Sabrina Klos née Müller (S'15) received the B.Sc. and M.Sc. degrees in mathematics from Technische Universität Darmstadt, Germany, in 2012 and 2014, respectively, where she is currently pursuing the Ph.D. degree in electrical engineering as a member of the Communications Engineering Laboratory. Her research interests include machine learning and optimization methods and their applications to wireless networks.

Cem Tekin (M'13) received the B.Sc. degree in electrical and electronics engineering from Middle East Technical University, Ankara, Turkey, in 2008, and the M.S.E. degree in electrical engineering: systems, the M.S. degree in mathematics, and the Ph.D. degree in electrical engineering: systems from the University of Michigan, Ann Arbor, MI, USA, in 2010, 2011, and 2013, respectively. He is currently an Assistant Professor with the Electrical and Electronics Engineering Department, Bilkent University, Ankara. His research interests include machine learning, multiarmed bandit problems, data mining, and multi-agent systems. He received the University of Michigan Electrical Engineering Departmental Fellowship in 2008 and the Fred W. Ellersick Award for the best paper in MILCOM 2009.

Mihaela van der Schaar (M'99–SM'04–F'10) is currently a Man Professor with the University of Oxford and a Turing (Faculty) Fellow with The Alan Turing Institute. Her current research interests include machine learning, AI, and data science for medicine. She is also developing machine learning and data science methods to enable personalized education of students and professionals. Besides machine learning and data science, her research expertise spans signal processing, multimedia, communication networks, network science, game theory, and distributed systems. Her research work has been widely cited, and several of her papers received best paper awards, including the prestigious IEEE Circuits and Systems Society Darlington Award. Her research has also led to 33 U.S. patents and over 45 contributions to international standards. Her research has received many recognitions and awards, including the NSF CAREER award, the Okawa Foundation Award, three IBM Faculty Research Awards, the Philips Make a Difference Award, and three International Organization for Standardization Awards.

Anja Klein (M'96) received the Diploma and Dr.-Ing. (Ph.D.) degrees in electrical engineering from the University of Kaiserslautern, Germany, in 1991 and 1996, respectively. In 1996, she joined Siemens AG, Mobile Networks Division, Munich and Berlin, Germany. She was the Director of the Development Department and the Systems Engineering Department. She was active in the standardization of 3G mobile radio in ETSI and in 3GPP, for instance leading the TDD Group of RAN1 in 3GPP. In 2004, she joined the Technische Universität Darmstadt, Germany, as a Full Professor, heading the Communications Engineering Laboratory. She has authored over 300 refereed papers and has contributed to 12 books. She is an inventor and a co-inventor of over 45 patents in the field of mobile radio. Her main research interests are in mobile radio, including interference management, cross-layer design, relaying and multi-hop, computation offloading, smart caching, and energy harvesting. In 1999, she was named the Inventor of the Year by Siemens AG. She is a member of the Verband Deutscher Elektrotechniker-Informationstechnische Gesellschaft.