A new platform, termed spatial crowdsourcing (SC), is emerging that enables a requester to commission workers to physically travel to some specified locations to perform a set of spatial tasks (i.e., tasks related to a geographical location and time). For spatial crowdsourcing to scale to millions of workers and tasks, it should be able to efficiently assign tasks to workers, which in turn consists of both matching tasks to workers and computing a schedule for each worker. The two current approaches for task assignment in spatial crowdsourcing, batched assignment and centralized online assignment, cannot scale as either task matching or task scheduling will become a bottleneck. Instead, we propose a distributed online assignment approach utilizing an auction-based framework where workers bid on every arriving task and the server determines the highest bidder, resulting in splitting the assignment responsibility between workers (for scheduling) and the server (for matching) and thus eliminating all bottlenecks. As a benchmark to evaluate the results of our online framework, we introduce a clairvoyant offline algorithm that gives the globally optimal assignment. Through several experiments, we evaluate the accuracy of the real-time algorithm by comparing it with the clairvoyant algorithm for small number of tasks. For larger real-world and synthetic datasets, we compare the accuracy and efficiency of our real-time algorithm with centralized online algorithms proposed for similar problems. We show how other algorithms cannot generate as good of an assignment because they fail to manage the dynamism and/or take advantage of the spatiotemporal characteristics of SC.

1. INTRODUCTION

Smartphones are ubiquitous: we are witnessing an astonishing growth in mobile phone subscriptions. The International Telecommunication Union estimates there are nearly 7 billion mobile subscriptions worldwide [1]. Meanwhile, the mobile phones’ sensors (e.g., cameras) are advancing and the network bandwidth is constantly increasing. Consequently, every person with a mobile phone can now act as a multi-modal sensor, collecting and sharing various types of high-fidelity spatiotemporal data instantaneously (e.g., picture, video, audio, location, time, speed, direction, and acceleration).

Exploiting this large crowd of potential workers and their mobility, a new mechanism for efficient and scalable data collection has emerged: Spatial Crowdsourcing (SC) [2]. Spatial crowdsourcing requires workers (e.g., willing individuals) to perform a set of tasks by physically traveling to certain locations at particular times. Spatial crowdsourcing is exploited in numerous industries, e.g., Uber, TaskRabbit, Waze, Gigwalk, etc., and has applications in numerous domains such as citizen-journalism, tourism, intelligence, disaster response and urban planning. With spatial crowdsourcing, a requester submits a set of spatiotemporal tasks to a spatial crowdsourcing server (SC-Server). Subsequently, the SC-Server has to select a worker to perform each task.

With Spatial Crowdsourcing, it is not enough to only match a task with a worker. An SC-Server must consider the schedule of every worker when matching a task to workers and only consider those workers who are able to fit the task in their schedule. For example in [3, 4] the focus is on matching tasks with workers and the workers’ schedule is ignored. On the other hand, Deng et. al. [5] consider scheduling tasks for a single worker while assuming the tasks have already been matched. They show there is no guarantee that the worker is going to be able to schedule all its matched tasks. In this paper, we define the task assignment in SC consisting of two phases, a matching phase and a scheduling phase, which need to happen in tandem. Neither of these phases should be ignored, otherwise, the resulting solution is rendered infeasible for real-world applications.

Two recent studies consider both matching and scheduling in spatial crowdsourcing [6, 7]. However, neither studies can provide real-time assignments. This is because in order to match a task to the best worker, the server has to perform the scheduling phase for multiple workers. Scheduling multiple workers is time consuming and cannot be done in real-time. Consequently, both studies utilize a batched assignment scheme, where the assignment is delayed for a period of time (i.e., batching time interval) during which all the arrived tasks are batched to be matched and scheduled during the next time interval. That is, while the SC-Server is busy processing a set of tasks, the next batch of tasks arrive at the server. Once the server completes the previous batch, it starts processing those tasks that have been queued. Once the tasks are batched and processed together, suddenly the matching phase becomes complex because many tasks need to be matched to many workers. This in turn, adds to the running time and increases the batching time interval.

A long batching time interval (e.g., 10 minutes) has two main disadvantages. First, the duration of the batching time interval should be subtracted from the tasks’ deadline, leaving each task with less available time to be scheduled. Second, batch scheme
can no longer generate real-time assignments which is the requirement of many SC applications. For example, an Uber user wants to know if there is a driver that can service her as soon as she submits the request. Instead in online assignment, a task is assigned to a worker as soon as it arrives at the SC-Server. This requires the server to perform matching and scheduling in real-time. With online assignment, at each point of time the SC-Server is processing only one task and hence, the matching phase becomes a one-to-many matching where there are multiple workers and only one task. Consequently, the complex many-to-many matching phase of batched assignment is reduced to only selecting the best worker that can fit the task in its schedule. Even though matching is fast with online assignment, the server has to still perform scheduling for multiple workers. Therefore, the scheduling phase becomes the bottleneck in online assignment. As shown in [8], scheduling for a single worker can be done fast, however a centralized SC-Server is not capable of scheduling multiple workers in real-time. Towards this end, we propose Auction-SC; an auction-based framework for real-time task matching and scheduling. In this framework, we decentralize the scheduling problem by utilizing the workers.

With Auction-SC, the server broadcasts a task to the workers upon the task’s arrival. Each worker \(^1\) submits a bid for that task based on its current schedule and location. To compute its bid, each worker has to consider only its own schedule so the bid computation phase can be done in real-time. Once every worker submits its bid to the SC-Server, matching the task to a worker reduces to selecting the best bid.

We introduce a branch-and-bound scheduling algorithm where for each new task, the worker performs an exhaustive search to find out whether it can fit the incoming task into its schedule. We show that at each point of time the number of remaining tasks for each worker (the number of tasks that the worker has scheduled and not completed yet) is in a range that even the branch-and-bound algorithm can be completed in real-time. However, in our experiments, we show that even replacing the branch-and-bound algorithm with a polynomial time approximate algorithm, will not affect the quality of the assignment significantly.

In addition to the branch-and-bound algorithm, we propose a more complex bidding technique that takes into consideration the spatial distribution of the tasks seen so far. The key idea is that having more workers in areas with more tasks can increase the quality of the assignment. This consideration increases the complexity of the bidding phase such that it may impact the scalability of the framework. However, we show that the SC-Server can still manage a throughput of 200 tasks per second with this complex bidding technique.

With a real-world SC system, the SC-Server does not know about the time and location of future tasks and workers, until they arrive. Therefore, it is impossible for the SC-Server to generate a globally optimal assignment. As a benchmark to better evaluate the quality of the real-time assignments made by Auction-SC, for the first time, we propose an algorithm for finding the globally optimal assignment in an SC environment. We assume there exists a clairvoyant which knows exactly at what time and which location new tasks and workers will arrive and for how long they will be available. We prove that the clairvoyant algorithm is computationally expensive and cannot be performed on large workloads with thousands of tasks and workers. However, in our experiments, for small workloads, we compare the results of Auction-SC with the globally optimal assignment.

\(^1\)Hereafter, we use the term "worker" interchangeably to refer to both the human worker and the software running on her mobile device unless clear distinction is needed.

We conduct many experiments on both real world and synthetically generated workloads to evaluate different aspects of Auction-SC. For our real world data we use a dataset of hundreds of thousands of geo-tagged Flickr images. We show that with a centralized approach the SC-Server cannot process more than 5 tasks per second. However, with the auction based framework, the throughput of the system can go as high as 200 tasks per second.

The remainder of this paper is organized as follows. In Section 2 we formally define the task assignment problem in SC and prove it is NP-Complete. We review the related work in Section 3. Next, we propose our offline clairvoyant algorithm in Section 4 and analyze its time complexity. We introduce Auction-SC in Section 5 and propose several bidding rules within the auction-based framework. We show the results of our experiments on both real world and synthetic data in Section 6 and conclude the paper with guidelines for future work (Section 7).

2. PRELIMINARIES

In this section, we define the terminologies used in the paper and give a formal definition of the problem under consideration. In addition, we analyze the complexity of the problem and its hardness.

2.1 Problem Definition

We start by defining some terminologies in order to formally define the task assignment problem in spatial crowdsourcing.

**Definition 1 (Spatial Task).** A spatial task \( t \) is a task to be performed at location \( t.l \) with geographical coordinates, i.e., latitude and longitude. The task becomes available at \( t.r \) (release time) and expires at \( t.d \) (deadline). Also, \( t.v \) is the obtained reward after completing \( t \).

It should be pointed out that in a spatial crowdsourcing environment, a spatial task \( t \) can be executed only if a worker is at location \( t.l \). For example, if the query is to report the traffic situation at a specific location, someone has to actually be present at the location to be able to report the traffic. From here on, whenever we use task we are actually referring to a spatial task. Now, we formally define a worker.

**Definition 2 (Worker).** A worker \( w \) is any entity, e.g., a person, willing to perform spatial tasks. We show the current location of the worker by \( w.l \). Each worker has a list of tasks assigned to it, \( w.T \), and a maximum number of tasks it is willing to perform, \( w.max \). Also \( w.s \) and \( w.e \) show the availability of the worker such that the worker is available during the time interval \((w.s, w.e)\).

Throughout the paper we assume every worker moves one unit of length per unit of time. Therefore, we can assume that \( distance \ ( (a,b) \) is also the time required to move from point \( a \) to point \( b \).

**Definition 3 (Schedule).** We call an ordered list of tasks a schedule. We show \( s \) as \((t_1, \ldots, t_n)\) where \( n \) is the number of tasks in \( s \). If we show the \( i^{th} \) task in \( s \) with \( s_i \), we say worker \( w \) is able to perform schedule \( s \) if and only if:

\[
\forall i, 1 \leq i \leq n \sum_{j=1}^{i} distance (s_{j-1}, s_j) \leq s_i.d
\]

where \( s_0.l \) is the current location of the \( w \).

At each point in time, each worker keeps track of a schedule and completes the tasks based on their order in its schedule.
### Table 1: List of notations

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t.l</td>
<td>location of task t</td>
</tr>
<tr>
<td>t.r</td>
<td>release time of task t</td>
</tr>
<tr>
<td>t.d</td>
<td>deadline of task t</td>
</tr>
<tr>
<td>t.v</td>
<td>value of task t</td>
</tr>
<tr>
<td>w.l</td>
<td>location of worker w</td>
</tr>
<tr>
<td>w.e</td>
<td>the time worker w becomes available</td>
</tr>
<tr>
<td>w.c</td>
<td>the time worker w becomes unavailable</td>
</tr>
<tr>
<td>w.t</td>
<td>list of tasks assigned to worker w</td>
</tr>
<tr>
<td>w.max</td>
<td>maximum number of tasks worker w can perform</td>
</tr>
<tr>
<td>w.PTS</td>
<td>list of all potential task subsets worker w can perform</td>
</tr>
</tbody>
</table>

### Definitions

**Definition 4 (Matching).** Assuming we have a set of workers $W$ and a set of tasks $T$, we call $M \subseteq W \times T$ a matching if for each $t \in T$ there is at most one $w \in W$ such that $(w, t) \in M$. We call $(w, t) \in M$ a match and say $t$ has been matched to $w$. For each matching $M$, we define the value (benefit) of $M$ as:

$$Value(M) = \sum_{(w, t) \in M} t.v$$

A matching $M$ is valid if and only if, for every worker $w$, there exists a schedule $s$, such that $(w, t_i) \in M \implies t_i \in s$.

Now we can formally define the Task Assignment in Spatial Crowdsourcing (TASC) as follows:

**Definition 5 (Task Assignment in SC).** Given a set of workers $W$, a set of spatial tasks $T$ and a cost function $d : (W \cup T) \times T \to \mathbb{R}$ where $d((a, b))$ is the distance between $a$ and $b$, the goal of the TASC($W, T, d$) problem is to find a valid matching $M$ with maximum value.

It is important to note that with task assignment in SC, the goal is to find a valid matching. This means that in addition to finding a matching between tasks and workers, the SC-Server has to also find a schedule for each worker to perform the tasks. Throughout this paper we use the terms matching phase and scheduling phase to refer to the two different aspects of task assignment in SC. We focus on maximizing the total number of assignment and hence we assume every task has a value equal to 1. Table 1 lists the notations we frequently use in this paper.

### 2.2 Complexity Analysis

The equivalent decision problem for TASC is to decide if there exists a matching $M$ with value $K$ and is denoted as TASC($W, T, d, K$). In this section we use a slightly modified version of the well known Hamiltonian Path Problem. We call it the Minimum Length Hamiltonian Problem (Min-Ham-Path) and define it as follows:

**Definition 6 (Min-Ham-Path).** Given a directed graph $G(V, E)$ where each edge $e \in E$ is assigned a length $l : E \to \mathbb{R}$, a source node $s$ and a length $L \in \mathbb{R}$, the Min-Ham-Path problem $(G, l, s, L)$ is to decide whether there exists a path in $G$ that starts from $s$, visits every other node exactly once and has a length of at most $L$.

**Theorem 1.** The Min-Ham-Path problem is NP-Hard.

**Proof.** In order to prove the NP-Hardness of Min-Ham-Path we show Ham-Path $\leq_P$ Min-Ham-Path. The Hamiltonian Path problem asks the following question: Given a directed graph $G(V, E)$ does there exist a path that goes through every node exactly once?

Given an instance of the Ham-Path problem $(G)$ we modify graph $G(V, E)$ and generate a new graph $G'(V', E')$ where $V' = V \cup \{o\}$ and $E' = E \cup \{(o, v) : v \in V\}$. Also, for every $e \in E'$ we assume $l(e) = 1$.

Now we show that Ham-Path$(G)$ is true if and only if Min-Ham-Path$(G', l, o, n)$ is true where $n$ is the number of vertices in $G$. If Min-Ham-Path returns a path of length $n$, we can remove the first edge from the path which results in a Hamiltonian Path for the Ham-Path$(G)$ problem. On the other hand every Hamiltonian Path on graph $H$ has length $n - 1$. By adding vertex $o$ and connecting it to the starting vertex, we end up with a Hamiltonian Path of length $n$ on $G'$.

**Theorem 2.** The TASC problem is NP-Complete.

**Proof.** We start the proof by showing that the decision problem of TASC is verifiable in polynomial time. Given a matching $M$, we can check that no task is assigned to more than one worker in polynomial time. Furthermore, based on the definition of a valid match, checking the validity of a matching $M$ can be done in polynomial time as well. Finally, we can find the value of $M$ by adding the value of every task in $M$.

We show TASC is NP-Hard by proving the reduction Min-Ham-Path $\leq_P$ TASC. Given an instance of the Min-Ham-Path problem $(G(V, E), l, o, K)$ we reduce it to an instance of the TASC($W, T, l', n - 1$) problem such that $W = \{o\}, T = V \setminus \{o\}$. For every task $t$ we set $t.v = 1$, $t.r = 0$ and $t.d = K$. Also for every $e \in E$, $l'(e) = l(e)$. In addition for every $e' \in (W \times T) \cup (T \times T)$ where $e' \notin E$ we set $l'(e') = \infty$.

Finally we show the result of Min-Ham-Path$(G, l, o, K)$ is true if TASC($W, T, l', n - 1$) is true where $n$ is the number of vertices in $G$. Considering that $|T| = n - 1$ and $t.v = 1$ for every $t \in T$, if there exists a matching with size $n - 1$ it means every task has been assigned to the single worker. Also, since the deadline of every task is $K$ the worker visits every task no later than $K$. Therefore, the path that the worker traverses starts at $o$ and goes through every other vertex $v \in V \setminus \{o\}$, where the length of the path is no more than $K$.

### 3. RELATED WORK

Spatial crowdsourcing research focuses on the task assignment [2, 4, 9, 5, 8, 7, 6], trust [3, 9] and privacy issues [10]. Kazemi and Shahabi [2] formulate task assignment in spatial crowdsourcing as a matching problem with the primary objective of maximizing the number of matched tasks, and Alfarra et al. [4] scaled up the matching algorithm in a distributed setting. Reliable task assignment addressing trust issues in spatial crowdsourcing have been studied in [3, 9]. In [9] Cheng et al. proposed a partitioning heuristic which recursively divides tasks and workers via KMeans clustering to improve the assignment efficiency. All these studies utilize the batch scheme. The methods used in [2, 9, 4] cannot be applied to a single task for real-time assignment. Furthermore, neither of these studies consider the schedule of a worker when matching tasks and workers. In their settings a worker is able to complete a task if it is matched to it which is not the same in our case. On the other hand, Deng et al. [5] studied the scheduling problem from the worker’s perspective and developed both exact and heuristic algorithms to help the worker find the best schedule for all the tasks assigned to it. However, their work assumes that
each worker has been pre-assigned with some tasks, and thus only deals with scheduling tasks for a single worker. Recent studies have focused on simultaneous task assignment and scheduling [8, 6, 9]. In [8], Li et. al. perform a real-time task assignment for a single worker. Knowing the worker’s destination, they recommend a route to the worker to be able to complete as many tasks as possible on its way to the destination point. Deng et. al. [6] and Chen et. al. [7] perform assignment and scheduling for multiple workers. However, similar to [2, 9, 4] they tackle the assignment problem by batching tasks’ and workers’ arrivals and departures, and then perform matching and scheduling periodically (e.g., every 10 minutes) for each batch.

Our work is also related to some combinatorial optimization problems such as Vehicle Routing Problem (VRP) [11]. The general setting of VRP is to serve a number of customers with a fleet of vehicles and the objective is to minimize the total travel cost of those vehicles. Compared with VRP, with spatial crowdsourcing our objective is to maximize the number of completed tasks, whereas VRP aims to minimize the total travel time. In addition, the spatial workers in our problem setting are not located at one or several fixed depots, and each worker can show up at any unique location. In our setting the spatial tasks are also not guaranteed to be completed by the workers. Finally, in a spatial crowdsourcing platform, we need to provide efficient solutions for potentially millions of tasks and workers. This is different from the solutions in VRP which take hundreds of seconds even for one hundred delivery points.

Auction frameworks have been used in various domains such as in dynamic multi-agent environments [12, 13] and online advertising [14]. Bidding strategies from other domains cannot be used in spatial crowdsourcing as they do not consider the spatial and temporal characteristics of SC. In this paper, we develop bidding strategies that are unique to a spatial crowdsourcing environment. One aspect of auction frameworks that has been widely studied is how to prevent bidders from submitting malicious bids to give themselves unfair advantage [15]. This problem does not apply to our Auction-SC framework because here, the workers do not directly interact with the SC-Server. Instead, the software on a worker’s phone receives tasks, computes bids and submits them to the server. If a task is assigned to a worker, the worker will be notified with an updated schedule through the software. The worker has no control over the software and hence, cannot cheat the system.

4. EXACT CLAIRVOYANT ALGORITHM

In order to have a benchmark to evaluate our real-time algorithms, in this section, we define a clairvoyant off-line algorithm. We assume there exists a clairvoyant which has a global knowledge of tasks and workers. In this case, the clairvoyant can assign tasks to workers such that the value of the assignment is maximized. We propose a two step algorithm which finds the optimal solution to the TASC problem. In the first step we find all potential subsets of tasks that a single worker is able to complete. Having the output of step 1, in the next step we find an assignment with maximum value. In the following, we describe different steps of the algorithm and later discuss the complexity of it. We also show how to adapt the algorithm to address other spatial crowdsourcing frameworks. This adaption does not deteriorate the performance of the algorithm, and in most cases the performance is even improved.

4.1 Discovering Potential Task Subsets

In the first step of the algorithm we focus on finding all task subsets that a worker \( w \) is able to complete. We can define a Potential Task Subset as:

$$\text{Definition 7 } \text{(Potential Task Subset (PTS))} \quad \text{We call } \text{PTS} \subset T \text{ a potential task subset for worker } w \text{ if there exists a valid schedule } s \text{ for worker } w \text{ such that } t \in \text{PTS} \implies t \text{ contains } t. \quad \text{We define the value of PTS as:}$$

$$\text{Value}(\text{PTS}) = \sum_{t \in \text{PTS}} t.v$$

For each worker \( w \), we define \( w.\text{PTS} \) as the set of all such potential task subsets.

In order to find all PTSs for a single worker, a straightforward method is to go through all subsets of \( T \) and check whether or not it is a PTS for the worker. It is trivial to see that such approach requires \( O(n!) \) permutations. Therefore, we utilize a branch and bound algorithm and use the following propositions for pruning the search space.

**Proposition 1.** For every \( t \in T \) and \( w \in W \) if \( (t.r, t.d) \cap (w.s, w.e) = \emptyset \) then for every \( \text{PTS} \in w.\text{PTS} \) we have \( t \not\in \text{PTS} \).

**Proposition 2.** For any \( w \in W \) for every \( \text{PTS} \in w.\text{PTS} \) we have \( |\text{PTS}| \leq w.\text{max} \).

**Proposition 3.** For every \( w \in W \) if \( \text{PTS} \in w.\text{PTS} \) for every \( \text{PTS} \subset \text{PTS} \) we have \( \text{PTS} \in w.\text{PTS} \).

We model the entire solution space as a tree (Fig. 2) and use a depth-first approach to search the solution space. By Proposition 2 we know that we only have to extend the tree up to depth \( w.\text{max} \) levels for worker \( w \). Each node of the tree corresponds to a subset of \( T \). Therefore, for any node \( n \) of the tree, if the corresponding subset of \( n \) is not in \( w.\text{PTS} \), according to Proposition 3, then we can prune the entire sub-tree rooted at \( n \).

**Algorithm 1** Find\( \text{PTSs}(T, W) \)

**Input:** \( T \) is the set of all tasks and \( W \) is the set of all workers

**Output:** \( \text{PTSs} \) is the list containing all potential task subsets for different workers.

1. for \( w \) in \( W \) do
2. \( \text{PTSs}[w] = \text{SearchBranch}(\emptyset, T, w) \)
3. end for
4. return \( \text{PTSs} \)

Algorithm 2 finds all the PTSs under a specific branch of the search space. The current subset, \( \text{pts'} \), is expanded with a new task, \( t \), if \( t \) temporally overlaps with \( t \) (lines 4-8). If the new subset \( \text{pts''} \) is a PTS itself we continue the search for more PTSs in the sub-tree rooted at \( \text{pts''} \) (lines 9-15). The subset \( \text{pts''} \) is a PTS only if a valid schedule for worker \( w \) exists that contains every \( t \in \text{pts''} \). Finally in Algorithm 1 for each worker we search for PTSs starting the search at the root of the tree.
4.2 Finding the Maximal Assignment

Having found all PTSs for each worker in step 1, the next step of the algorithm focuses on finding the assignment with the maximum value. For this purpose, we construct a graph such that each PTS from step 1 is a node in the graph. We call this graph the PTS-Graph. Subsequently, we show that the maximum weighted clique in the PTS-Graph corresponds to the assignment with the maximum value.

The PTS-Graph is a multi-layered graph such that for each worker we add a layer to the graph. With each layer \( l \), for each PTS inside \( w_l \), PTS we add a node to layer \( l \). The weight of this node is equal to the value of the corresponding PTS. We use an arbitrary ordering to name the nodes within each layer of the PTS-Graph. Assuming layer \( l \) contains \( m \) nodes, we name them \( n_{lm} \) through \( n_{lm} \). Also, we define the set of edges of the PTS-Graph as:

\[
E = \{(n_{ak}, n_{bj}) | a \neq b \land (PTS_{ai} \cap PTS_{bj} = \emptyset)\}
\]

where \( PTS_{lm} \) refers to the corresponding PTS of node \( n_{lm} \). In other words, there is an edge between two nodes in different layers if the corresponding PTSs of the two nodes do not contain the same task.

Given the PTS-Graph \( G(V, E) \), we associate an assignment of tasks to workers with a set of vertices \( N \subset V \) as follows: if \( n_{lm} \in N \), then every task \( t \in PTS_{lm} \) is assigned to \( w_l \).

We claim that the optimal assignment corresponds to the nodes of the Maximum Weight Clique in PTS-Graph.

**Definition 8 (Maximum Weight Clique).** For undirected graph \( G=(V,E) \) where each \( v \in V \) is assigned a value \( w(v) \), the Maximum Weight Clique is the clique for which the sum over the weight of all vertices in the clique, is larger than any other clique in the graph.

**Theorem 3.** If \( C^* \) is a maximum weighted clique in the PTS-Graph, then the assignment of tasks to workers corresponding to \( C^*, A^* \) is a maximum value assignment for the TASC problem.

**Proof.** Based on how we generated the PTS graph, no two vertices in \( C^* \) can contain the same task in their corresponding PTS. Therefore, any task \( t \) is assigned to at most one worker. Next we need to show no other assignment \( A \) can have a larger value than \( A^* \). If such an assignment \( A \) existed, then the clique \( C \) corresponding to \( A \), has a larger weight than \( C^* \) which conflicts the fact that \( C^* \) is the maximum weighted clique. \( \Box \)

There exists a vast body of literature on solving the maximum weight clique problem [16]. Here we briefly explain one of these algorithms and refer the interested readers to [17].

In Algorithm 3 we start with an arbitrary node ordering. Östegård [17] suggests a node ordering where \( v_1 \) is the node with the largest total sum of the weights of its adjacent nodes and so on so forth. Algorithm 3 computes a value \( C(k), 1 \leq k \leq n \) which determines the largest weight of a clique, only considering nodes \{\( v_1, v_{k+1}, \ldots, v_n \)\}. If \( w(i) \) denotes the weight of \( v_i \), then we have \( C(n) = w(i) \) and other values for \( C(k) \) are computed in a backtrack search. For \( 1 \leq k \leq n - 1 \), \( C(k) > C(k+1) \) only when a corresponding clique exists and contains \( v_k \). If such clique does not exist then \( C(k) = C(k+1) \). In order to find such \( C(k) \) clique, a branch and bound approach is used to search all possible cliques.

Details of the branch and bound algorithm can be found in [17].

**Algorithm 3 MaximumWeightClique(V, E)**

**Input:** \( V \) is an ordering of the nodes and \( E \) is the set of edges in the graph

**Output:** \( C \) a subset of \( V \) as the maximum weighted clique

1: \( C = \{v_n\} \)
2: \( \max = w(n) \)
3: for \( k : n - 1 \) down to 1 do
4: \( (C_k, w) \) = FindMaxClique\((V, E, v_k)\)
5: if \( w > \max \) then
6: \( C = C_k \)
7: \( \max = w \)
8: end if
9: end for
10: return \( C \)

4.3 Complexity Analysis

We analyze the time complexity of our proposed exact algorithm as follows. For simplicity we assume \( |T| = n, |W| = m \) and \( |w, \max| = p \). For Algorithm 2 we can assume that we are running the \( ExistsValidSchedule() \) method for each node of the search space tree(Fig. 2) at most once. At level \( i \) of the search space tree we have at most \( C(n, i) \) nodes. Also for each node in level \( i \) the size of the task subsets is \( i \). Hence, a naive implementation for the \( ExistsValidSchedule() \) method runs in \( O(i^{i+1}) \). Therefore the amortized time complexity of Algorithm 2 is:

\[
\sum_{i=1}^{n} \binom{n}{i} O(i^{i+1}) = O(n^{n-p}p^{p+1})
\]

Since we run the SearchBranch method for each worker once, we can conclude that the time complexity of Algorithm 1 is \( O(nm^{n-p}p^{p+1}) \).
In Algorithm 3 we implement the FindMaxClique() method, utilizing a branch and bound algorithm described in [17]. The time complexity of this algorithm is $O(n^n)$ for a graph of size $n$. Therefore, for a graph of size $n$, the overall time complexity of Algorithm 3 is $O(n^{n+1})$.

We end the section with a discussion on how generalizing the spatial crowdsourcing framework can affect the algorithm introduced in this section. In [3] it is assumed that each task requires a certain level of confidence and only workers with a trust level higher than that are able to perform the task. Moreover, in some spatial crowdsourcing frameworks [2, 3, 5], workers define a spatial range and only perform tasks within this region. Constraints like these only remove a subset of tasks from the list of tasks a certain worker is able to perform. Consequently, the search space of Algorithm 1 gets reduced and we end up with fewer numbers of PTSs per worker. This in turn, reduces the size of the PTS-Graph presented at the end of this section.

5. AUCTION-SC FRAMEWORK

In practice, an SC system works similar to a complex event processing (CEP) engine [18] where new tasks arrive at the SC-Server as an input stream. With a real-world SC scenario, the SC-Server will find out about a task and its properties only when it is released. The complexity of the many-to-many matching in addition to the need for immediacy (e.g., Uber) render the batch scheme impractical for real-world scenarios. Furthermore, in an online centralized SC-Server scheduling multiple workers becomes the bottleneck and hence, real-time assignment is not guaranteed. In this section we introduce Auction-SC which has neither shortcomings and generates real-time online assignments by splitting the matching and scheduling responsibilities between the server and the workers, respectively. First we explain the auction framework and how tasks are dispatched to workers. Next, we discuss how workers compute their bids and at the end we provide a cost analysis of this framework.

5.1 Task Dispatchment in Auction-SC

Auction methods have been effectively used for assignment problems in dynamic multi-agent environments [12, 13]. The main advantage of auction methods are their simplicity and the fact that they allow for a decentralized implementation. Auction-SC considers workers as bidders and tasks as goods. Furthermore, the SC-Server plays the role of a central auctioneer in Auction-SC. At a very high level, in Auction-SC, once a new tasks is submitted to the SC-Server (auctioneer), the server presents the task to the workers (bidders). Depending on the bidding rule, which is common among all workers, each worker computes its own bid and submits the bid to the server. The bidding process is performed as a sealed-bid auction where workers simultaneously submit bids and no other workers knows how much the other workers have bid. The SC-Server selects the worker with either the lowest or highest bid (depending on the bidding rule) as the winner and matches the task with the worker.

Broadcasting every incoming task to all available workers incurs a large communication cost on the system. Auction-SC lowers the communication cost by only sending any incoming task to eligible workers that are defined as:

**Definition 9 (Eligible Worker).** An available worker $w$ is said to be eligible for performing a newly released task $t$, if and only if:

$$\text{distance}(w, t) \leq w.d - t.r \land \text{distance}(w, t) \leq t.d - t.r$$

In other words, an available worker $w$ is eligible for performing task $t$, if it has enough time to reach the location of $t$ before either $t$ expires or $w$ leaves the system. The SC-Server maintains a spatial index on the location of the workers. With Auction-SC, we use a grid index since (1) the workers have to send updated locations to the server only if they change cells and (2) the server does not need to know the exact location of the worker to be able to filter out non-eligible workers. A detailed analysis of the communication cost is presented at the end of this section.

Algorithm 4 OnlineTASC($W, t$)

Input: $W$ is the set of currently available workers and $t$ is a task the has just released
Output: Either $w \in W$ as the worker task $t$ should be assigned to or null if no worker is selected
1: $w_{selected} = null$
2: $Bids = \emptyset$
3: for $w \in W$ do
4:  $bid = w$.ComputeBid($t$)
5:  $Bids \leftarrow \langle w, bid \rangle$
6: end for
7: $w_{selected} = \text{SelectBestBid}(Bids)$
8: return $w_{selected}$

Algorithm 4 outlines the process of assigning an incoming task $t$. $W$ in line 3 is the set of eligible workers for task $t$. Notice that all the iterations of the for loop in Algorithm 4 (lines 3-6) run in parallel. The ComputeBid($t$) method (line 4) that each worker executes depends on the implemented bidding rule. Similarly, the SelectBestBid($t$) method (line 7) returns the worker with either the highest or lowest bid. In case of a tie, the SelectBestBid($t$) method, randomly selects one worker among the ones with optimal bid.

5.2 Worker’s Bid Computation

With Auction-SC, every worker computes its bid using a pre-defined bidding rule. A worker’s bid represents how good it is to be matched with the task. When computing a bid for a task, the workers have no knowledge about other tasks that might arrive in the future. Consequently, they have to make a greedy decision based on their current status. First we introduce four simple bidding rules based on heuristics that are used for other problems that seem to be similar to task assignment in SC, hence we call them non-SC rules. These problems do not consider either the spatial or temporal dynamism of SC. We use non-SC rules in the experiments to show the importance of both spatial and temporal aspects of SC. The four non-SC rules used in our experiments are:

- **Random (Rnd)**: As a baseline approach, we consider a rule where every eligible worker submits a bid of value 1. The SC-Server selects the winner randomly from the set of eligible workers.

- **Ranking (Rnk)**: Based on the ideas in Online Bipartite Matching [19], the workers are ranked from 1 to $n$ (the workers’ order do not change over time). Each eligible worker submits a bid of value 1 and the SC-Server selects the eligible worker with the lowest rank as the winner. This heuristic does not consider either spatial or temporal aspects of SC.

- **Nearest Neighbor (NN)**: Similar to the the Spatial Matching problem [20], for this bidding rule we give priority to the closest worker to the location of the task. This means a worker closer to the location of the incoming task is better suited to perform the task compared to a worker farther away. To compute a bid, each worker needs to compute the distance between the location of the task and its own location and submit the computed distance as its bid. Once every worker has submitted its bid, the SC-Server chooses
the worker with the minimum bid as the winner. This bidding rule only considers spatial constraints of SC.

Most Free Time (MFT): In this bidding rule, based on the ideas in Online Scheduling [21], we give priority to workers that have more time before they leave the system. At each point of time, a worker computes its free time as the duration between the time it finishes its current schedule and the time it leaves the system. Each worker submits a bid equal to its free time and the SC-Server selects the worker with the highest bid as the winner. This bidding rule only considers the temporal dynamism of SC.

In addition to the four non-SC rules, we propose two bidding rules that consider both spatial and temporal constraints of SC and hence, we call them SC rules. The intuition behind the first bidding rule is that the less time a worker spends completing an incoming task, the higher the chances are that it is available in the future to perform more tasks. Our second SC rule, tends to move workers to areas that there is a higher chance for a task to show up in the future. We describe how workers compute their bids and how the server selects the winner in the following.

Best Insertion (BI): Intuitively, if a worker spends less time to complete a task it will likely have more time for performing other tasks. In BI, the server gives priority to workers that can better insert the incoming task into their schedules. Auction-SC considers the extra time each worker will need to complete the new task in addition to its current schedule.

Each worker starts with finding a potential optimal schedule where the new task is added to its current schedule. For this process, the worker uses a branch and bound algorithm to check all possible orders of its uncompleted tasks in addition to the new task. Running an exhaustive search for large numbers of tasks can be time consuming. However, in our experiments on both real world and synthetic data, we realized that the number of uncompleted tasks at each point of time for each worker, remains in a range where even the exhaustive search can be done in real-time for a single worker. The reason is that, as time passes and new tasks arrive, the worker also completes other tasks and removes them from its schedule. In cases where computing the bid takes too long, we can replace this computation with an approximate algorithm (e.g., the Nearest Insertion algorithm [22]). In our experiments, we show that running an approximate algorithm for the scheduling phase, reduces the accuracy of the assignment by less than 5%.

In order to compute a bid, each worker computes the finish time of the potential optimal schedule in case it is matched with the new task \( f_2 \). The worker also knows the finish time of its current schedule \( f_1 \). For each worker, \( f_2 - f_1 \) is the extra time it requires to complete the new task in addition to tasks in its current schedule. Therefore the bid each worker submits to the SC-Server is equal to \( f_2 - f_1 \). After receiving the bids from every worker, the SC-Server assigns the task to the worker with the lowest bid.

Best Distribution (BD): BI does not consider the spatial distribution of tasks. It might be beneficial to assign a task located at a task-dense area to a worker with high remaining availability, even if the worker has to move from its current location significantly. The general idea behind this rule is to try to move workers to locations that have a higher chance of having a task in the future. Ideally, we want the spatial distribution of the available workers \( S_W \) to be as close as possible to the overall spatial distribution of the tasks \( S_T \).

One can argue that knowing \( S_T \) contradicts the assumption that the SC-Server has no spatiotemporal knowledge about future tasks. Even if the SC-Server knows \( S_T \), it does not mean it also knows the exact locations at which the tasks are going to be released. Nevertheless, with Auction-SC we do not make the assumption that \( S_T \) is known a priori. Instead, we assume the SC-Server starts with an empty distribution and keeps updating it as new tasks arrive.

For \( S_W \) and \( S_T \) we use a grid index similar to the one Auction-SC maintains for choosing eligible workers when a task arrives. For each event occurring inside a cell, we add to the weight of that cell. To compute \( S_T \), for every task we add a value of 1 to the cell containing the task. As for \( S_W \), we need to consider the availability of the worker. For example, if the maximum number of tasks worker \( w \) can perform is \( n \), and it already has been assigned \( m \) tasks, we say the availability of worker \( w \) is \( n - m \). In this case, when computing \( S_W \), we add \( n - m \) units to the cell covering \( w \). For both \( S_T \) and \( S_W \), by normalizing the weights of the cells, we can compute the probability of an event occurring in each cell.

Having \( S_W \) and \( S_T \), we need a metric to determine the closeness of these two distributions. Several methods have been used to compute the similarity between two distributions. Among the more commonly used methods, we can name the Kullback-Leibler divergence [23] and Jensen-Shannon divergence [24]. The problem with these methods is that they do not take the distance between the cells into consideration. For example, Fig. 3 shows two spatial distribution for two tasks (red dots) and two workers (blue squares). Clearly, the distribution of workers in Fig. 3(b) is more similar to \( S_T \) compared to Fig. 3(a). However, with the Jensen-Shannon divergence metric, \( JSD(S_T, S_W) \) is equal to \( JSD(S_T, S_W^2) \). Considering the Kullback-Leibler divergence or any other metric that does not take into account the spatial relationship between the cells, ends up with the same results.

In this paper, we use the Earth Mover’s Distance metric since it has the ability to incorporate the spatial aspect of the distributions when computing their similarity.

**Definition 10 (Earth Mover’s Distance).** The Earth Mover’s Distance (EMD) is a measure of distance between two probability distributions over a region \( D \). If the distributions are interpreted as two different ways of piling up a certain amount of dirt over region \( D \), the EMD is the minimum cost of turning one pile into the other; where the cost is assumed to be the amount of dirt moved, times the distance by which it is moved [25].

If we want to compute the EMD between two distributions \( A \) and \( B \), for each grid cell \( i \), we call cell \( i \) a supplier iff \( P_A(i) > P_B(i) \) and a consumer iff \( P_A(i) < P_B(i) \). If \( P_A(i) = P_B(i) \), cell \( i \) is neither a supplier nor a consumer. For each supplier \( i \), we say \( a_i = P_A(i) - P_B(i) \) is the total supply of \( i \). Also, the total demand for consumer \( j \) is shown as \( d_j = P_B(j) - P_A(j) \). Now we can model the problem as a bipartite network flow problem where on one side we have the supplier nodes and on the other side we have consumer nodes. The weight of each edge \( c_{ij} \) between supplier \( i \) and consumer \( j \), is the cost of moving one unit of mass from \( i \) to \( j \). Consequently, finding EMD reduces to finding the Minimum Cost Flow for this bipartite graph. The Minimum Cost Flow problem can be formalized as the following linear programming problem:
\[
\text{minimize } \sum_{i \in I} \sum_{j \in J} c_{ij} f_{ij} \\
\text{subject to: } f_{ij} \geq 0 \quad i \in I, j \in J \quad (1) \\
\sum_{i \in I} f_{ij} = b_j \quad j \in J \quad (2) \\
\sum_{j \in J} f_{ij} \leq a_i \quad i \in I \quad (3)
\]

We can solve this LP problem efficiently using the simplex method [26].

We assume the SC-Server maintains \( S_W \) and \( S_T \) and shares it with the eligible workers. First, each worker has to compute a potential schedule by inserting the incoming task to its current schedule. If the worker is able to find a new schedule, it can locally modify \( S_W \) considering how its location and availability changes in the potential schedule, i.e., \( S_W' \). Subsequently, the worker computes the EMD between \( S_W' \) and \( S_T \) and submits the value as its own bid. The SC-Server selects the worker with the lowest bid as the winner.

### 5.3 Cost Analysis

We end the discussion on Auction-SC with a detailed analysis of the communication cost in this framework. Communication cost can be looked at from two different perspectives: response time and throughput. With Auction-SC, once the task arrives at the SC-Server, it is broadcast to a number of workers. Therefore, the SC-Server can send all messages (packets) in parallel at the same time. In return, all workers are submitting their bids in parallel as well. Considering current network transmission speeds, even in cellular networks, the response time of transmitting a single packet of data in Auction-SC is negligible.

The other aspect of communication cost analysis is the throughput of the network. Specially when at scale, the increase in the number of sent messages can saturate the network bandwidth and cause trouble. In a centralized architecture, tasks are not broadcast to workers and there is no bid submission. Hence it seems that communication cost is not a factor in a centralized approach. However, due to the dynamism of SC, coordination between the workers and the centralized server is inevitable and they need to communicate with each other. Here we show that with regard to the number of messages transmitted between the workers and the SC-Server, there is not much difference between a centralized approach and Auction-SC.

As explained earlier, with Auction-SC we implement a grid index that keeps track of the location of available workers. The SC-Server uses this grid index to identify eligible workers and only broadcast the new task to them. The worker has to notify the server about its location only if its movement causes it to switch cells. We assume, on average a workers switches cells \( \bar{\alpha} \) times and show the total number of transmitted messages as:

\[
|M_{\text{Centralized}}| = \bar{\alpha} n + 2|T|n_e
\]

Similar to Auction-SC, \( \bar{\alpha} n \) is the total number of messages sent to the server to change the workers’ cells. We can see the number of messages transmitted in a centralized server is similar to those with Auction-SC. Thus, when compared to a centralized architecture, Auction-SC does not increase the throughput of the network.

### 6. EXPERIMENTS

Unlike regular crowdsourcing, there is no publicly available platform for running real-world spatial crowdsourcing campaigns. With platforms such as Amazon Mechanical Turk, workers do not perform spatial crowdsourcing tasks. For this reason, we simulated various spatial crowdsourcing campaigns on an Inter(R) Core(TM)2 Duo 3.16GHz PC with 4GB memory running Microsoft Windows 7.

#### 6.1 Dataset

Due to its commercial value, real-life SC systems such as Uber and TaskRabbit do not make their datasets available to public. However, we were able to use a large collection of geo-tagged images from Flickr [28] as an input to Auction-SC. The dataset consists of \( ^\sim 15 \) million images with attributes such as location, time taken, time uploaded, the user who took the picture, etc.

We correspond each image with a spatial task considering that the task was to take a photo at a specific location. We set the deadline of the task to the time the image was taken and the duration of the task to the interval between the time the image was taken and the time it was uploaded. Each user in the dataset corresponds to a worker. The original location of the worker is randomly selected. The worker is available during the interval between the first time he took a photo and the last time he uploaded one. We ran 5 separate instances, each of which with images in a different metropolitan area. Table 2 shows the total number of tasks (and workers) for each city.

<table>
<thead>
<tr>
<th>City</th>
<th># of Tasks</th>
<th># of Workers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Los Angeles</td>
<td>219,332</td>
<td>19,081</td>
</tr>
<tr>
<td>New York</td>
<td>390,229</td>
<td>17,603</td>
</tr>
<tr>
<td>London</td>
<td>366,034</td>
<td>18,480</td>
</tr>
<tr>
<td>Paris</td>
<td>237,344</td>
<td>14,275</td>
</tr>
<tr>
<td>Beijing</td>
<td>23,335</td>
<td>1,598</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>City</th>
<th># of Tasks</th>
<th># of Workers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Los Angeles</td>
<td>219,332</td>
<td>19,081</td>
</tr>
<tr>
<td>New York</td>
<td>390,229</td>
<td>17,603</td>
</tr>
<tr>
<td>London</td>
<td>366,034</td>
<td>18,480</td>
</tr>
<tr>
<td>Paris</td>
<td>237,344</td>
<td>14,275</td>
</tr>
<tr>
<td>Beijing</td>
<td>23,335</td>
<td>1,598</td>
</tr>
</tbody>
</table>

Table 2: Number of tasks/worker for each city in Flickr dataset
We also generated a synthetic dataset with realistic streaming workload based on the methodology proposed in [29]. To generate a workload suitable for SC systems we modeled three different sets of parameters:

**Temporal Parameters:** In [30], it is shown that with crowdsourcing environments, workers and tasks arrive following Poisson processes [31]. The default Poisson arrival rates for tasks and workers are \( \mu_t = 20/\text{min} \) and \( \mu_w = 3/\text{min} \), respectively. Subsequently, the duration of the tasks and workers were randomly sampled from closed range of \([1, 4]\) hours and \([1, 8]\) hours, respectively.

**Spatial Parameters:** Fig. 4 shows the spatial distribution of tasks from our real-world dataset in Los Angeles. As depicted, the tasks are not uniformly distributed in space. The spatial distribution is rather skewed, meaning that the density of the tasks at certain areas is higher. To model the same behavior with our synthetic workloads, we created 6 two dimensional Gaussian clusters with randomly selected means and standard deviations. Eighty percent of the tasks are sampled within the clusters and the rest are uniformly distributed.

**Static Parameters:** In addition to the spatiotemporal parameters, we consider two other parameters. The default workload size of each experiment is 10K tasks. The task arrival rate and the number of tasks determine the duration of the simulation. Based on the duration of the simulation and the worker arrival rate, the total number of workers may vary.

The maximum number of tasks a worker can perform, i.e., \( w_{\text{max}} \), is a uniformly random number from the closed interval \([8, 12]\).

6.2 Online vs. Offline

As the first set of experiments, we compare how the results of the real-time algorithms perform with the optimal solution computed using the offline clairvoyant algorithm explained in Section 4. Because of the high complexity of the offline algorithm, we are not able to run tests with large workloads and thus we use workloads with 100 tasks. The experimental results of Fig. 5 show that the best real-time algorithms perform close to 60% of the optimal solution. Our results are consistent with studies that compute a competitive ratio [32] of \( 1 - e^{-1} \) for the online matching problem with random inputs [33].

6.3 Assignment Quality

In this section we evaluate the quality of the assignments using different real-time non-clairvoyant algorithms. First, we compare them using the Flickr and the synthetic datasets. Subsequently, using the synthetic datasets, we show how the spatial and temporal settings of the problem can affect the performance of the real-time assignment quality.

Fig. 6 compares the assignment quality of different real-time algorithms. As we can see, both SC approaches outperforms the best non-SC algorithm by almost 25%. The main reason as explained in Section 5 is that, the SC rules perform scheduling when matching tasks with workers. Furthermore, the BD approach outperforms BI by at most 10%. This is not surprising as BD tends to “move” workers to areas where future tasks are more likely to appear, thus achieving higher assignment in long term. Among the non-SC approaches NN outperforms other rules by almost 2 times more completed tasks. The reason MFT does not outperform baseline approaches (Rnd and Rnk) can be explained by what we call a radical move. A radical move is when the SC-Server assigns a task to a worker which requires it to move a relatively long distance to reach the location of that task. Since we do not consider any spatial proximity to the task with MFT, there is a high chance to end up with assignments resulting in radical moves. With NN and BI the general idea is to prevent radical moves as much as possible. With BD, although radical moves occur, but only if the worker moves to areas where there will be more tasks to complete.

In order to study the effect of temporal parameters of SC, we ran several experiments using different pairs of task arrival rates (\( t_{\text{rate}} \)) and worker arrival rates (\( w_{\text{rate}} \)). In Fig. 7 we show the effect of increasing \( t_{\text{rate}} \) and \( w_{\text{rate}} \) on the quality of the assignment. The level of grayness corresponds to the percentage of completed tasks where black and white representing 100% and 0% respectively. As we can see with small number of workers, as we increase the task arrival rate, the percentage of completed tasks decreases where at the top left corner of each plot we get close to 0%. On the other hand in Figs. 7(d) to 7(f) for NN, BI and BD, with small number of incoming tasks, as we increase \( w_{\text{rate}} \), eventually all tasks will be completed. Fig. 7 clearly shows that NN, BI and BD outperform Rnd, Rnk and MFT independent of the \( t_{\text{rate}} \) and \( w_{\text{rate}} \).

To better evaluate the leading real-time approaches, NN, BI and BD, in Fig. 8 we performed a pair-wise comparison by taking their task completion rates. For example, Fig. 8(a) shows the difference between BI and NN. We observe that these three approaches perform similarly at the two extreme cases discussed in Fig. 7, i.e., high task-low worker and low task-high worker. BI and BD outperform NN up to 30% when the problem is more complex, i.e., outside the extreme cases. An interesting observation in Figs. 8(a) and 8(b) is that BI and BD outperform NN by a much larger margin at scale (higher \( t_{\text{rate}} \) and \( w_{\text{rate}} \)). The reason is that with higher higher \( t_{\text{rate}} \) and \( w_{\text{rate}} \) more workers are moving around and more
tasks come and leave so in general the spatiotemporal dynamism of
the system increases. BI and BD cope with the dynamism by guar-
anteeing a task gets assigned to worker that can complete it. On
the contrary, NN ignores the schedule of a worker during matching
and this becomes more important as there is more dynamism in the
system.

We mentioned earlier that with the SC-rules, the workers per-
form an exhaustive search to find out if they can fit a new task into
their schedule. As workers accept new tasks, they also complete
some other tasks so as we observed in our experiments, performing
an exhaustive search did not cause any scalability issues. Neverthe-
less, one might want to replace the exhaustive search with a poly-
nomial time approximate algorithm. With ApproxBI, we use the
insertion algorithm from [22] that runs in $O(n^2)$. Fig. 9 shows the
change affects the quality of the assignment by less than 5%. The
difference caused by ApproxBI is that the workers that are eligible
for a task using BI, may not be able to fit the task in their sched-
ules using ApproxBI due to the approximation. As a result, using
ApproxBI the server may not be able to assign some tasks even if
they can be completed using BI. Fortunately, as shown in Fig. 9,
that does not happen very often regardless of $r_{rate}$ and $w_{rate}$.

The next set of experiments compare the effect of the spatial dis-
tribution of tasks. We compared the quality of the final assignment
for three different distribution. Even though real-world data usu-
ally follow a skewed spatial distribution (Section 6.1), the results of
these experiments show that regardless of the distribution, SC rules
outperform non-SC rules. With the first distribution, the location of
the tasks follow a spatial Poisson process [34]. The other two dis-
tributions are a Uniform 2D distribution and a Skewed distribution.
The results in Fig. 10 show that SC rules generate assignments at
least 20% better than non-SC rules. Also, we can see with Poisson
and Uniform distributions, there is not much difference between BI
and EMD. The reason is that both distributions generate tasks at
completely random locations. Consequently, tasks are released at
every area with the same probability and hence EMD and BI be-
come similar.

The final set of experiments regarding assignment quality, com-
pare bathed assignment with online assignment results. For bathed
assignment we used the LALS algorithm from [6].

Earlier we explained for some use cases (e.g., Uber), batched as-
signment does not even satisfy application requirements. Neverthe-
less, the results in Fig. 11 show even if non real-time assignments
are tolerable, the quality of batched assignment is not as good as
the online assignment. One reason is that the LALS algorithm per-
forms the matching phase and then attempts to schedule tasks for
their matched workers. All tasks that could not be scheduled for
their matched workers, will go back to the matching phase and the
process continues until all tasks are scheduled or there is no more
worker to match with a task. When performing the matching, the

2The reported runtimes in [6] outperform those of [7]
schedule of the worker is not considered and hence a task might end up getting matched to and scheduled for a worker that was not the best worker. This in turn, can lower the chances of that worker to get assigned to a new task in the future. The second reason is that while a task is waiting at the server to get processed with the next batch, depending on the length of the batching time interval, it will lose some portion of its availability time which in turn, can lower the chances of the task fitting a worker’s schedule.

6.4 Scalability

The last set of experiments focus on measuring the scalability of two system architectures, i.e., centralized and Auction-SC (decentralized). We compare the scalability of the real-time Auction-SC algorithms with the equivalent implementation of the same algorithms on a single centralized SC-Server.

We can measure the scalability of SC systems by their throughput: the number of tasks processed per second, or equivalently, the processing time per task, shown in Fig. 12. Because of the decentralized architecture of Auction-SC, in Fig. 12 we see that the average processing time of a single task does not change as the arrival rate of workers increases. On the contrary, with the centralized architecture, the average processing time of a single task increases linearly as we increase the number of workers and is several orders of magnitude higher than Auction-SC approaches. Although BD consumes more time than other real-time algorithms, but only takes up to 3 milliseconds.

![Figure 12: Average processing time for a single task](image)

For a CEP engine, it is also common to measure the queuing delay of events [35] once they arrive at the system as a metric for the scalability of the systems. In Fig. 13 we compare the average queuing delay of tasks in the two architectures after running them for 1 hour. We can see that the centralized system suffers from queuing delays with less than 10 tasks/second. On the other hand, with Auction-SC, even for BD, we do not observe queuing delays for up to 500 tasks/second. Fig. 13(b) also shows that BD incurs higher delay than BI but results in higher completion rates (Figs. 6 and 8). Users of Auction-SC can choose between BD and BI to balance their needs for assignment quality and efficiency.

![Figure 13: Average queuing delay](image)

To summarize the results of our experiments, we showed that when SC bidding rules are used, the quality of the assignment is much higher as compared to when a non-SC rule is used. The consideration of scheduling at the time a task is being matched is the main reason for the better overall assignments. When running on a single centralized SC-Server, neither one of bidding rules can scale. Furthermore, the time required to process a single task increases linearly as more workers are added to the system in a centralized architecture. Scalability of a single centralized server suffers more when SC bidding rules are used due to their higher computation complexity. However, with Auction-SC we solve the scalability problem by splitting the scheduling and matching responsibilities between workers and the SC-Server. Consequently, Auction-AC can afford to execute complex SC bidding rules, resulting in a very high quality assignment. Table 3 shows the summary of our experimental results.

7. CONCLUSION AND FUTURE WORK

In this paper, we studied the problem of real-time task assignment in spatial crowdsourcing. We showed that neither of the two current approaches for task assignment in spatial crowdsourcing, batched assignment and centralized online assignment, can scale as either task matching or task scheduling will become the bottleneck. Therefore, we introduced an auction-based framework in which we split the matching and scheduling responsibilities between the SC-Server and workers, respectively. We showed that by exploiting the spatiotemporal aspects of SC, with our proposed algorithms, the workers will be able to complete up to 30% more tasks as compared to non-SC approaches. The decentralized architecture of our framework allows for performing such high quality assignments at scale.

In this paper, we assumed each task can be performed instantaneously, e.g., taking a picture. Once that assumption is relaxed, we will face new challenges with scheduling the tasks. We also
assumed each task requires the worker to travel to a single location. However, in other applications the worker may need to visit multiple locations for a single task, e.g., in the Uber application the worker has to pick up a passenger at one location and drop him off at a second location. We plan to extend our Auction-SC framework to incorporate these two features.

8. REFERENCES


<table>
<thead>
<tr>
<th></th>
<th>non-SC bidding rules</th>
<th>SC bidding rules</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rnd</td>
<td>Rnk</td>
</tr>
<tr>
<td>Centralized</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scalability</td>
<td>Bad</td>
<td>Bad</td>
</tr>
<tr>
<td>Assignment Quality</td>
<td>Very Bad</td>
<td>Very Bad</td>
</tr>
<tr>
<td>Auction-SC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scalability</td>
<td>Very Good</td>
<td>Very Good</td>
</tr>
<tr>
<td>Assignment Quality</td>
<td>Very Bad</td>
<td>Very Bad</td>
</tr>
</tbody>
</table>

Table 3: Summary of Experimental Results