Task Publication Time Recommendation in Spatial Crowdsourcing

Xuanlei Chen
University of Electronic Science and Technology of China
China
202122080924@std.uestc.edu.cn

Yan Zhao*
Aalborg University
Denmark
yanz@cs.aau.dk

Kai Zheng
University of Electronic Science and Technology of China
China
zhengkai@uestc.edu.cn

ABSTRACT

The increasing proliferation of networked and geo-positioned mobile devices brings about increased opportunities for Spatial Crowdsourcing (SC), which aims to enable effective location-based task assignment. We propose and study a novel SC framework, namely Task Assignment with Task Publication Time Recommendation. The framework consists of two phases, task publication time recommendation and task assignment. More specifically, the task publication time recommendation phase hybrids different learning models to recommend the suitable publication time for each task to ensure the timely task assignment and completion while reducing the waiting time of the task requester at the SC platform. We use a cross-graph neural network to learn the representations of task requesters by integrating the obtained representations from two semantic spaces and utilize the self-attention mechanism to learn the representations of task-publishing sequences from multiple perspectives. Then a fully connected layer is used to predict suitable task publication time based on the obtained representations. In the task assignment phase, we propose a greedy and a minimum cost maximum flow algorithm to achieve the efficient and the optimal task assignment, respectively. An extensive empirical study demonstrates the effectiveness and efficiency of our framework.

CCS CONCEPTS

• **Networks** → *Location based services*; • **Information systems** → *Recommender systems.*

KEYWORDS

task publication time recommendation, task assignment, spatial crowdsourcing

ACM Reference Format:
Xuanlei Chen, Yan Zhao, and Kai Zheng. 2022. Task Publication Time Recommendation in Spatial Crowdsourcing. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3511808.3557466

*Corresponding author: Yan Zhao.

1 INTRODUCTION

With the development and widespread use of GPS-equipped smart devices and wireless mobile network (e.g., 5G network) in recent years, people can move as sensors and participate some location-based tasks such as monitoring traffic condition and reporting local hot spots. Spatial Crowdsourcing (SC) is a recently proposed concept and framework that has been widely used in many applications. In SC, the platforms collect spatial tasks and require workers to move to specific locations physically to complete the assigned tasks.

Extensive studies on SC [8, 20, 30, 36, 38, 41–43] have contributed many techniques for task assignment in different application scenarios, which are based generally on the assumption that the task publication time is specified by a task requester. However, in practice, unsuitable task publication time leads to a long waiting time for task requesters at an SC platform or failure of task assignment/completion. For example, if a task requester publishes a task at a location where workers are insufficient at the current time or in the near future, the task requester has to wait a long time at the SC platform. In such a case, if the SC platform recommends a suitable publication time for the task, the task requester does not need to focus on the implementation of the task until the recommended task publication time, which will facilitate the task requester. Additionally, when a task requester publishes a task in such a situation where all workers are unavailable at the current time or in the next few timeslots, it is likely to cause the failure of the timely task assignment and completion.

Recent studies have explored recommendation problems in SC, such as task recommendation [12, 14, 22, 23, 35]. Traditional task recommendation methods capture worker preference using static features, such as the inferred worker rates on tasks, worker skills and task categories [2, 11]. For example, Ambati et al. [2] build a worker preference model based on interests and skills, which are learned from implicit or explicit feedback provided by workers to recommend tasks. Yuen et al. [35] use the Probabilistic Matrix Factorization (PMF) to learn worker preference based on worker rates on tasks inferred from their interacting behaviours and task categories. However, these methods ignore sequential patterns of worker mobility traces. In addition, workers' task execution time is significant while modelling workers, which is largely overlooked by existing methods. Also, most of the previous studies mainly focus on the recommendation for workers, ignoring the importance of task publication time recommendation for task requesters in SC.

To tackle these issues, we develop a data-driven SC framework, called Task Assignment with Task Publication Time Recommendation (TAPR), that aims to enable effective task assignment with task publication time prediction. Specifically, the TAPR framework consists of a task publication time recommendation and a task assignment phase. In the first phase, given the historical task-publishing sequence data, we firstly learn the representations of task requesters in two semantic relations (i.e., requester-task and requester-timeslot interactions) with a graph neural network, respectively. We design a gate mechansim to integrate the two representations, which can control the proportion of information in the two semantic spaces in the final representations. Then we use the self-attention mechanism to learn the multi-perspective relations of the task-publishing sequences. Finally, we can predict the suitable publication time via a fully connected layer. In the task assignment phase, we propose a greedy algorithm that gives efficient task assignment based on the recommended task publication time. For achieving the optimal task assignment that maximizes the task completion rate while minimizing the average waiting time of task requesters, we further propose a Minimum Cost Maximum Flow (MCMF) algorithm based on the publication time recommendation, where the task assignment problem is transformed into a MCMF problem in a network-flow graph.

In summary, our work has four primary contributions.

1) To the best of our knowledge, this is the first work in SC that recommends task publication time for task requesters for ensuring the timely task assignment and completion while reducing the waiting time of task requesters at the SC platform.

2) We consider the multiple factors which may influence the task requester's preference for task publication timeslot and use the self-attention mechanism of Transform to capture the multi-perspective relations of task-publishing sequences.

3) We propose a greedy and an optimal algorithm for achieving efficient and effective task assignment.

4) We report on experiments using real data, offering evidence of the effectiveness and efficiency of the proposed framework.

The remainder of this paper is organized as follows. In Section 2, the proposed problem is given. Then we present the task publication time recommendation model and the task assignment algorithms in Section 3, followed by the experimental results in Section 4. Section 5 introduces the related work. Finally, we conclude the paper in Section 6.

2 PROBLEM DEFINITION

We proceed to present necessary preliminaries and then define the problem addressed. Table 1 lists the notation used throughout the paper.

Definition 1 (Spatial Task). *A spatial task, denoted by $s = (q, l, rt, p, e, r)$, has a task requester q, a location $s.l$, a registration time $s.rt$ which must be earlier than or equal to the publication time, a publication time $s.p$, an expiration time $s.e$, and a reward $s.r$.*

Definition 2 (Task-publishing History). *Given a task requester $s.q$ who has published n tasks in a time period, we define the task-publishing history as a time-ordered task sequence, $S_p^{s.q} = (s_1, s_2, ..., s_n)$, where $s_i.p \leq s_{i+1}.p$ $(1 \leq i < n)$.*

With spatial crowdsourcing, the query of a spatial task s can be answered only if a worker is physically located at that location $s.l$.

Table 1: Summary of Notation

Symbol	Definition
s	Spatial task
$s.q$	Task requester of task s
$s.l$	Location of spatial task s
$s.rt$	Registration time of spatial task s
$s.p$	Publication time of spatial task s
$s.e$	Expiration time of spatial task s
$s.r$	Reward of spatial task s
$S_p^{s.q}$	A historical task-publishing sequence of $s.q$
w	Worker
$w.l$	Current location of worker w
$w.d$	Reachable distance of worker w
$w.on$	Online time of worker w
$w.off$	Offline time of worker w
$AWS(s)$	Available worker set of task s
A	A spatial task assignment
$A.C$	Task completion rate in task assignment A
$A.T$	Total waiting time in task assignment A

Note that with the single task assignment mode [16], an SC server should assign each spatial task to only one worker.

Definition 3 (Worker). *A worker, denoted by $w = (l, d, on, off)$, has a location $w.l$, a reachable radius $w.d$, an online time $w.on$, and an offline time $w.off$. The reachable range of worker w is a circle with $w.l$ as the center and $w.d$ as the radius, within which w can accept assignments.*

A worker is online when being ready to accept tasks. After the offline time, the worker cannot accept tasks. In our work, a worker can accept and handle only one task at a time, which is reasonable in practice.

Definition 4 (Available Worker Set). *The available worker set for a task s, denoted as $AWS(s)$, is a set of workers that satisfy the following conditions: $\forall w \in AWS(s)$:*

1) worker w is in an online mode, i.e., $w.on \leq t_{now} \leq w.off$, and

2) task s is located in the reachable range of worker w, i.e., $d(w.l, s.l) \leq w.d$, and

3) worker w can arrive at the location of task s before it expires, i.e., $t_{now} + t(w.l, s.l) \leq s.e$,

where t_{now} is the current time, $d(w.l, s.l)$ is the travel distance (Euclidean distance) between location $w.l$ and location $s.l$, and $t(w.l, s.l)$ is the travel time between $w.l$ and $s.l$.

Definition 5 (Spatial Task Assignment). *Given a set of workers $W = \{w_1, w_2, ..., w_{|W|}\}$ and a set of tasks $S = \{s_1, s_2, ..., s_{|S|}\}$, we define A as a spatial task assignment, which consists of a set of tuples of form (w, s), where a spatial task s is assigned to worker w, satisfying all the workers' and tasks' spatio-temporal constraints.*

We use $A.C$ to denote the task completion rate in task assignment A, which is the ratio between the number of tasks completed before expiration time and the total number of tasks. Besides, we use $A.T = \sum_{s \in A.S} \mathcal{T}(s.q)$ to denote the total waiting time of all task requesters in task assignment A, where $A.S$ denotes the task set of A, and $\mathcal{T}(s.q)$ is the waiting time of task requester $s.q$ at the SC platform that is the task duration of s (i.e., elapsed time from publication to completion of s). The problem investigated can be stated as follows.

Publication Recommendation based Task Assignment (PR-TA). Given a set of online workers W and a set of tasks S to be

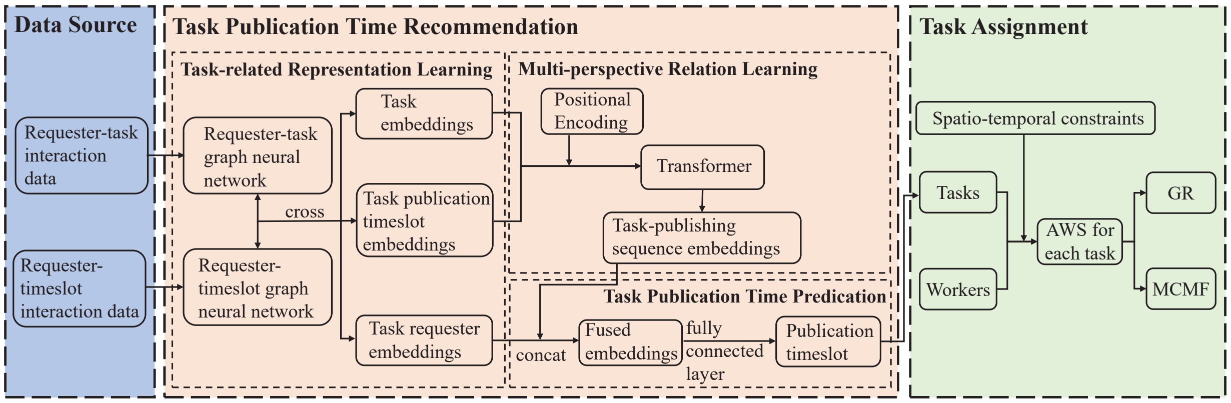

Data Source
Requester-task interaction data
Requester-timeslot interaction data
Task Publication Time Recommendation
Task-related Representation Learning
Requester-task graph neural network
cross
Requester-timeslot graph neural network
Task embeddings
Task publication timeslot embeddings
Task requester embeddings
Multi-perspective Relation Learning
Positional Encoding
Transformer
Task-publishing sequence embeddings
Task Publication Time Predication
concat
Fused embeddings
fully connected layer
Publication timeslot
Task Assignment
Spatio-temporal constraints
Tasks
Workers
AWS for each task
GR
MCMF

Figure 1: Framework Overview

published by task requesters at the current time instance on an SC platform, our problem is to recommend a suitable publication time for these tasks and find an optimal task assignment A_{opt} that achieves the following goals:

1) primary optimization goal: maximize the task completion rate., i.e., $\forall A_i \in A$ ($A_{opt}.C \geq A_i.C$), where A denotes all possible assignments.

2) secondary optimization goal: minimize the average waiting time of task requesters without compromising the primary optimization goal.

3 ALGORITHM

The main novelty of the Task Assignment with Task Publication time Recommendation (TAPR) framework is that the SC server learns the suitable task publication time for each task requester, based on which the task assignment is performed. In this section, we first give an overview of the framework, and then provide specifics on each component in the framework.

3.1 Framework Overview

The TAPR framework is comprised of two components: task publication time recommendation and task assignment, as illustrated in Figure 1.

The publication time recommendation component includes three main parts: Task-related Representation Learning, Multi-perspective Relation Learning, and Task Publication Time Prediction. Taking the requester-task interaction data and requester-timeslot interaction data which are obtained through the historical task-publishing sequential data as input, we first construct two graph neural networks, i.e., requester-task and requester-timeslot graph neural networks. Then we use a cross-graph neural network to learn the task-related representations (including task embeddings, task publication timeslot embeddings, and task requester embeddings). Next, we adopt the self-attention mechanism of Transformer to find the multi-perspective relations of the task-publishing sequences with positional encoding and get their embeddings. After that, we construct a fully connected layer to predict the task publication time by taking the concatenation of the representations of the task-publishing sequences and the corresponding task requesters as fused embeddings.

The second component needs to assign tasks to suitable workers. We first calculate the Available Worker Set (AWS) for each task, from which we can select the most suitable worker for the task during the subsequent task assignment. We propose a greedy task assignment (GR) algorithm that tries to assign the nearest worker to each task from the unassigned workers, until all the workers are exhausted or all the tasks are assigned. To achieve the optimal task assignment, we further design a Minimum Cost Maximum Flow (MCMF) algorithm that balances the distance between workers and tasks and the number of completed tasks.

3.2 Task Publication Time Recommendation

In this section, we introduce our proposed method for task publication time recommendation in detail. We predict the publication time of tasks for the task requesters via a fully connected layer based on the fused feature vectors of the task requesters obtained by the cross-graph neural network and the vectors of the task-publishing sequences encoded by the Transformer.

3.2.1 Task-related Representation Learning. Graph convolution neural network (GCN) can deal with the data of graph structure well and extract the features of data for model training. Based on GCN, we add a cross component to transfer graph information from different semantic spaces. By using the cross-graph neural network to aggregate the graph information from requester-task interaction graph and requester-timeslot interaction graph, we can obtain the task-related representation containing richer semantic information, which is beneficial for the recommendation.

Given the task-publishing sequences, we assume that there are K task requesters Q, L tasks S, and M publication timeslots T. Then a requester-task interaction matrix and a requester-timeslot interaction matrix can be defined as $X^{K \times L} = \{x_{qs} | q \in Q, s \in S\}$ and $Y^{K \times M} = \{y_{qt} | q \in Q, t \in T\}$, respectively. Next, we set $x_{qs} = 1$ and $y_{qt} = 1$ when observing that a task requester q publishes a task s in timeslot t in the task-publishing sequence. Otherwise, we set $x_{qs} = 0$ and $y_{qt} = 0$.

We can get two interaction graphs according to the interaction matrix obtained from the historical task-publishing sequence data, i.e., requester-task interaction graph $G_{qs}(V_{qs}, E_{qs})$ and requester-timeslot interaction graph $G_{qt}(V_{qt}, E_{qt})$. Each node of G_{qs} represents a task requester or a task, and each edge of G_{qs} shows that there is an interaction between a task requester node and a task node. Similar to G_{qs}, each node of G_{qt} represents a task requester or a publication timeslot, and each edge of G_{qt} shows that there is an interaction between a task requester node and a timeslot node.

The initial embeddings of a task requester, a task, and a task publication timeslot are represented as $\boldsymbol{Q}, \boldsymbol{S}, \boldsymbol{T}$, respectively, the embedding sizes of which keep the same.

We adopt Light Graph Convolutional Neural (GCN) network [15] rather than GCN to obtain task-related representation. Compared to general GCN, Light GCN can extract the complicated topology from requester-task and requester-timeslot interaction graphs by only using the neighborhood aggregation. Layer combination is used to replace the self-connections of nodes to achieve the same effect. Except for that, Light GCN also cancels the nonlinear activation function and the feature transformation matrix when performing layer propagation, since both of the operations are not effective and efficient enough for the training process of the model on recommendation [17, 29].

Firstly, we get the adjacency matrix of the requester-task graph based on the requester-task interaction matrix X, which is shown in Equation 1.

$$A_{qs} = \begin{bmatrix} 0 & X \\ X^T & 0 \end{bmatrix} \tag{1}$$

After getting the adjacency matrix, Light GCN can update the node embeddings of each layer according to the aggregations of their neighbors. During the updating process, we can extract features of the requester-task interactions graph. The propagation process of each layer is shown as follows:

$$\left[\frac{\boldsymbol{Q}_{(l_1)}}{\boldsymbol{S}_{(l_1)}}\right] = \hat{A}_{qs}\left[\frac{\boldsymbol{Q}_{(l_1-1)}}{\boldsymbol{S}_{(l_1-1)}}\right], \tag{2}$$

$$\boldsymbol{Q}_{qs} = \sum_{i=0}^{l_1} \alpha_i^{qs} \boldsymbol{Q}_{(i)}, \quad \boldsymbol{S}_{qs} = \sum_{i=0}^{l_1} \alpha_i^{qs} \boldsymbol{S}_{(i)}, \tag{3}$$

where $\hat{A}_{qs}$ is a laplacian matrix, $\hat{A}_{qs} = D_{qs}^{-\frac{1}{2}} A_{qs} D_{qs}^{-\frac{1}{2}}$, D_{qs} is the degree matrix of A_{qs}, and l_1 denotes the number of layers in the network. $\boldsymbol{Q}_{(i)}$ and $\boldsymbol{S}_{(i)}$ represent the i-th layer feature of the task requester and the corresponding task, respectively. As shown in Equation 3, combined with features of each layer, we can get the final task requester feature matrix $\boldsymbol{Q}_{qs}$ and the task feature matrix $\boldsymbol{S}_{qs}$ in the requester-task interaction graph. α_i^{qs} is a learnable parameter used to denote the weight of the feature matrix of the i-th layer when generating the final embeddings. Finding that learning the weight parameter α_i cannot promote the performance while making the model more complicated, we give each layer the same weight when calculating the final embeddings and the weight parameter $\alpha_i = \frac{1}{l_1+1}$.

As for the requester-timeslot graph, its adjacency matrix is similar to that of the requester-task graph and can be defined as follows.

$$A_{qt} = \begin{bmatrix} 0 & Y \\ Y^T & 0 \end{bmatrix} \tag{4}$$

The propagation process of each layer in the requester-timeslot graph is also similar to that of the requester-task graph. The equations are shown as follows:

$$\left[\frac{\boldsymbol{Q}_{(l_1)}}{\boldsymbol{T}_{(l_1)}}\right] = \hat{A}_{qt}\left[\frac{\boldsymbol{Q}_{(l_1-1)}}{\boldsymbol{T}_{(l_1-1)}}\right], \tag{5}$$

$$\boldsymbol{Q}_{qt} = \sum_{i=0}^{l_1} \alpha_i^{qt} \boldsymbol{Q}_{(i)}, \quad \boldsymbol{T}_{qt} = \sum_{i=0}^{l_1} \alpha_i^{qt} \boldsymbol{T}_{(i)}, \tag{6}$$

where $\hat{A}_{qt}$ is similar to $\hat{A}_{qs}$, $\hat{A}_{qt} = D_{qt}^{-\frac{1}{2}} A_{qt} D_{qt}^{-\frac{1}{2}}$, $\boldsymbol{Q}_{qt}$ denotes the final feature vector of task requester, and $\boldsymbol{T}_{qt}$ denotes the final feature vector of task publication timeslot in the requester-timeslot graph.

After the above operations, we can obtain the embeddings of the tasks, publication timeslots and task requesters in two semantic spaces. Considering that embeddings with more semantic information can be useful to achieve more personalized recommendation, we design a gate mechanism to integrate the information in the two semantic spaces and update the task requester embeddings by Equation 7.

$$\boldsymbol{Q}^* = f * \boldsymbol{Q}_{qt} + (1-f) * \boldsymbol{Q}_{qs}, \tag{7}$$

where $\boldsymbol{Q}_{qs}$ and $\boldsymbol{Q}_{qt}$ denote the feature vectors of the task requester at the l_1-th layer in G_{qs} and G_{qt}, respectively. Next, f denotes the gate mechanism and its formula is as follows:

$$f = \phi(W(\boldsymbol{Q}_{qs} \oplus \boldsymbol{Q}_{qt} + b), \tag{8}$$

where ϕ represents the sigmoid function mapping the input values to the range $[0, 1]$, $\oplus$ represents matrix concatenation, W and b are learnable matrices.

Finally, we can obtain the final embeddings of task requester set $\boldsymbol{Q}^*$, task set $\boldsymbol{S}$, and publication timeslot set $\boldsymbol{T}$, which can be used in the next component.

3.2.2 Multi-perspective Relation Learning. There are multiple factors which may influence the task requester's preference for task publication timeslot. Therefore, in this part, we exploit the self-attention mechanism of Transformer [28] to learn the multi-perspective relations in the task-publishing sequences.

By summing the embeddings of the corresponding task, publication timeslot, and task position in the task-publishing sequence, we can get a new entity embedding in the task-publishing sequence, which can be calculated as follows:

$$\boldsymbol{e}_i = \boldsymbol{s}_i + \boldsymbol{t}_i + \boldsymbol{pos}_i, \tag{9}$$

where $\boldsymbol{s}_i$ denotes the embedding of task s_i, $\boldsymbol{t}_i \in$ denotes the embedding of timeslot t_i, and $\boldsymbol{pos}_i$ is the position embedding of the task i in the task-publishing sequence. After the multi-perspective fusion, the previous embedding of task s_i can be denoted by the entity embedding $\boldsymbol{e}_i$.

After that, we can get the task-publishing sequence embedding based on the embeddings of tasks in the sequence. Then we utilize the self-attention mechanism of Transformer encoder to extract the contextual information in the task-publishing sequence. The basic propagation between layers can be defined as follows:

$$H_l = Transformer(H_{l-1}), \tag{10}$$

where $l \in [1, l_2]$, l_2 denotes the number of layers in Transformer encoder, and H_l denotes the output matrix containing contextual information of the i-th layer. Each encoder layer of Transformer consists of two sub-layers, which are multi-headed self-attention mechanism and fully connected feed-forward network. The residual connection and normalisation are added to each sub-layer. The propagation detail of each layer can be shown as follows:

$$Z_l = LayerNorm(MHA(H_{l-1}) + H_{l-1}), \tag{11}$$

$$H_l = LayerNorm(FFN(Z_l) + Z_l), \tag{12}$$

where MHA is a multi-headed self-attention mechanism, and FFN is a two-layer fully connected feed-forward network. In particular, for the multi-headed self-attention mechanism of the Transformer, the propagation process in the i-th layer can be shown in Equations 13-15:

$$\mathcal{Q}_i = H_{l-1}\mathcal{W}_i^{\mathcal{Q}}, \quad \mathcal{K}_i = H_{l-1}\mathcal{W}_i^{\mathcal{K}}, \quad \mathcal{V}_i = H_{l-1}\mathcal{W}_i^{\mathcal{V}}, \tag{13}$$

$$head_i = Softmax(\frac{\mathcal{Q}_i\mathcal{K}_i^T}{\sqrt{d_k}})\mathcal{V}_i, \tag{14}$$

$$\hat{Z}_l = (head_1 \oplus head_2, ..., \oplus head_n)\mathcal{W}_l^O, \tag{15}$$

where $\mathcal{Q}$, $\mathcal{K}$, and $\mathcal{V}$ denote the query, key, and value matrices, which can be obtained by multiplying $H_{l-1} \in \mathbb{R}^{b \times d_h}$ by weight matrices $\mathcal{W}_i^{\mathcal{Q}}$, $\mathcal{W}_i^{\mathcal{K}}$, and $\mathcal{W}_i^{\mathcal{V}} \in \mathbb{R}^{d_h \times d_k}$, respectively. Next, d_h denotes the hidden size, and d_k denotes the dimension of a head. $\hat{Z}_l$ denotes the output matrix of a multi-headed self-attention in the l-th layer, which can be obtained by concatenating the output matrix of all heads. n is the number of heads. We use $\mathcal{W}_l^O \in \mathbb{R}^{d \times d_h}$ to denote the weight matrix when performing concatenation, where d is the result of multiplying d_k and n.

Finally, we can get the representation $\overline{Z}$ of the entire task-publishing sequence by averaging the representations of all tasks in the sequence.

3.2.3 Task Publication Time Prediction. The goal of this part is to predict the probability distribution of publication timeslots of tasks that will be published by requesters. We split the day into 96 timeslots and choose to predict the publication timeslot of tasks because of the sparsity of task-publishing sequences.

We construct a fully connected layer to predict task publication time. The input of the layer can be obtained by concatenating the vector of the task-publishing sequence with the vector of the task requester, as shown in Equation 16.

$$\hat{Y}_t = \rho(F(\mathcal{Q}^* \oplus \overline{Z})), \tag{16}$$

where $\mathcal{Q}^*$ is the feature vector of the task requester integrating the information from two different semantic spaces, $\overline{Z}$ denotes the vector of the task-publishing sequence, $\oplus$ denotes concatenation, and F denotes a fully connected layer, the unit of which is 96 since a day is divided into 96 timeslots. Next, ρ is a softmax function, and $\hat{Y}_t$ is the prediction for the task publication timeslot.

We use the cross-entropy loss function as the loss function of the task publication timeslot prediction, which is defined in Equation 17.

$$\mathcal{L}_t = -\sum_{n}^{C_{m,t}} \sum_{l}^{D} Y_{t,nl} \ \ln \hat{Y}_{t,nl}, \tag{17}$$

where $C_{m,t}$ is the set of task-publishing sequence indices with labelled publication timeslots, and D equals 96 (a day can be divided into 96 timeslots) , which denotes the output dimension of FC. Next, Y_t denotes publication timeslots label indicator matrix.

The overall loss function of the prediction component can be defined as follows:

$$\mathcal{L}_{loss} = \mathcal{L}_t + \lambda \left\|\Theta\right\|^2, \tag{18}$$

where λ denotes the L_2 regularization coefficient. And all trainable parameters of the model are included in Θ. After the training, the publication timeslot of task s can be calculated by $max(\hat{Y}_{t,n})$ under the limitation of the task registration time and expiration time, where n denotes the index of the task-publishing sequence to which task s belongs, and $max(\cdot)$ denotes the maximum function.

3.3 Task Assignment

In this section, we will present the task assignment algorithms, which are based on the recommended publication timeslots of tasks. Specifically, when a task requester requests to publish a task, we give a recommended publication timeslot to the requester, based on which we assign a suitable task to the requester. In the sequel, we first detail how to generate available worker sets for tasks, which will be used throughout the task assignment process, and then propose two algorithms to assign tasks, including a Greedy (GR) and a Minimum Cost Maximum Flow (MCMF) algorithm.

3.3.1 Available Worker Set (AWS) Generation. Given time instance set, $\mathbb{T} = \{t, ..., t+n\}$, a worker can only complete a small subset of tasks because of the constraints of workers' reachable distance and valid time as well as tasks' expiration time. As a result, given a set of online workers and published tasks at time instance t, we should find the available worker set ($AWS(s)$) for each task s based on the spatio-temporal constraints. Worker w in $AWS(s)$ should satisfy the following conditions: $\forall w \in AWS(s)$

1) worker w is in an online mode, i.e., $w.on \leq t_{now} \leq w.off$, and

2) task s is located in the reachable range of worker w, i.e., $d(w.l, s.l) \leq w.d$, and

3) worker w can arrive at the location of task s before it expires, i.e., $t_{now} + t(w.l, s.l) \leq s.e$,

where t_{now} is the current time, $d(w.l, s.l)$ is the travel distance between location $w.l$ and location $s.l$, and $t(w.l, s.l)$ is the travel time between location $w.l$ and location $s.l$.

3.3.2 Greedy Task Assignment. After getting the $AWS(s)$ of each task s, a simple strategy is to assign the nearest worker who is not assigned yet to the task, until all the tasks are assigned or all the workers are exhausted, which is called Greedy Task Assignment (marked as GR) algorithm.

The execution process of GR is shown in Algorithm 1. The input of Algorithm 1 is a worker set W, a task set S and an empty task assignment A (line 2). During each iteration, the algorithm begins to randomly select a task $s \in S$ from the remaining ones and assigns the nearest worker who is not assigned yet in $AWS(s)$ to the selected task. And then the algorithm adds task s to the task assignment A (lines 2–6). Finally, we can obtain the task assignment, A (line 7).

3.3.3 MCMF-based Task Assignment. Taking the distance between workers and tasks as the priority, we transform the task assignment problem into a Minimum Cost Maximum Flow (MCMF) problem based on the obtained available worker sets.

```
Algorithm 1: Greedy Task Assignment
  Input: $W$, $S$
  Output: A feasible assignment result $A$ and the
          corresponding number of assigned tasks $|A|$
1 $A \leftarrow \varnothing$ ;
2 for each task $s \in S$ do
3     $Opt_w \leftarrow$ find the nearest worker in $AWS(s)$ ;
4     $A = A \cup s$ ;
5     $W = W - Opt_w$ ;
6     $S = S - s$ ;
7 return $A$ and $|A|$
```

MCMF is based on a flow network-based graph in time instance t_i and the graph can be denoted by $G_i = (V, E)$, in which V and E refers to the set of vertices and edges, respectively. Suppose that we have a online worker set, $W_i = \{w_1, ..., w_n\}$ and an available task set, $S_i = \{s_1, ..., s_n\}$ at time instance t_i. Then we can conclude that $|V|$ and $|E|$ is equal to $|W_i| + |S_i| + 2$ and $|W_i| + |S_i| + m$, respectively, where m denotes the number of available assignments satisfying the spatial and temporal constraints for all the workers. Next, we use $|\mathbb{A}_i^w|$ to denote the number of available assignments for worker w. Then m can be obtained by summing the number of available assignments for all the workers, i.e., $m = \sum_{w \in W_i} |\mathbb{A}_i^w|$.

For the construction of vertices, we firstly create a source vertice src (denoted by v_0) and a destination vertice dst (denoted by $v_{|W_i|+|S_i|+1}$). And then we map each worker w_j to a vertex v_j, and each task s_k to a vertex $v_{|W_i|+k}$, respectively.

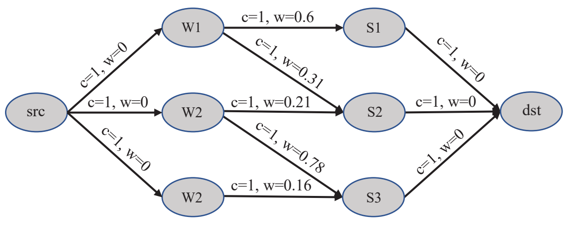


Figure 2: Flow Network-based Graph

There is an example of such a network flow graph with three workers and three tasks at one time instance in Figure 2. The steps of edge construction can be shown as follows:

1) We construct $|W_i|$ edges to connect src and the vertices mapped from W_i. The capacity of each edge is set 1, since each worker can only accept at most one task before one is completed at a time instance. And the cost of these edges is set to 0.

2) We create $|S_i|$ edges to connect the vertices mapped from S_i and dst. Similar to the edge connecting src and the vertices mapped from W_i, the capacity of each edge is set to 1, because each task can not be assigned to two or more workers at a time instance. The cost of these edges is also set to 0.

3) We create an edge to connect the vertex v_j (mapped from worker w_j) to the vertex $v_{|W_i|+k}$ (mapped from s_k) if s_k can be assigned to w_j (i.e., $\langle w_j, s_k \rangle \in \mathbb{A}_i^{w_j}$) according to the constraints of space and time. For the edge connecting the vertex v_j and $v_{|W_i|+k}$, its capacity is set to 1 and its cost is equal to the travel distance from worker w_j to task s_k.

After the construction of vertices and edges, we can convert the task assignment problem into a MCMF problem in the direct flow graph G_i from src to dst, aiming to maximize flow of the graph while minimizing the distance cost. Specifically, we maximize the flow of the graph by using the Ford-Fulkerson [10] algorithm and then minimize the distance cost with linear programming [16].

4 EXPERIMENTAL EVALUATION

Table 2: Statistics of Dataset

Dataset	Gowalla
Number of users	52979
Number of Pois	121851
Number of check-ins	3300986
Density	0.0511%
Average time between	51.28 hours
Collection period	2009/02-2010/10

4.1 Experimental Setup

Data Preparation. The experiments are carried out on a real dataset, Gowalla, which is an open source check-in dataset collected from location-based social networks. Some basic statistics of the dataset are demonstrated in Table 2. In Gowalla, the geo-tagged check-ins are used to simulate our problem. Each user is regarded as a task requester, each POI is regarded as a published task, and its check-in time is regarded as the task publication time. As Gowalla does not contain task registration time and task execution time, we randomly generate task registration time from range $[30min, 75min]$ based on the last check-in time of each check-in sequence and generate the task execution time from range $[0, 75min]$ based on task publication time. For each task requester, we randomly select a location of published tasks from his historical task-publishing sequence as the location of the task requested to be published. For each worker, we generate the locations of workers following a uniform distribution within a 2D space (with latitude from -80° to 80° and longitude from -150° to 150°) and also uniformly generate the arrival times of workers according to the distribution of task registration time. Moreover, we set the granularity of a time instance as 15 minutes (i.e., 10:00 am-10:15 am), during which the task requests and available workers will be packed and input to our framework. We run the algorithms from 12:00 am of the day and try to assign tasks to the suitable workers in different time instances (from the next time instance of the current time instance to the previous time instance of the task deadline) in the experiments.

Model Parameters. As for the model, Adam optimizer is used to speed up the training process. After analyzing, we determine the settings of the parameters. The learning rate is set to 0.001, the batch size is set to 200, the λ is set to 1e-7, the embedding size is set to 80, the dropout rate is set to 0.2, the number of layers of the cross-graph neural network (l_1) is set to 3 and the number of layers of Transformer encoder (l_2) is set to 3, and the number of self-attention heads (n) is set to 4. All the experiments are implemented on an Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz, and NVidia GeForce RTX 1080Ti GPU.

4.2 Experimental Results

4.2.1 Performance of Task Publication Time Recommendation. We first evaluate the performance of Task Publication Time Recommendation (TPTR) phase. We split all the task-publishing sequences into 80% for training and 20% for testing chronologically. For each sample, we truncate the task-publishing sequences of the same length (20 by default) [32].

Evaluation Methods. We perform an ablation analysis on our Task Publication time Recommendation (TPR) model and take them as baseline algorithms.

1) QS+QT: We adopt the light GCN to learn the embeddings of task requesters, tasks, and timeslots through Requester-Task graph (QS) and Requester-Timeslot graph (QT), and use the sum of the two obtained task requesters embeddings as the final task requester embeddings.

2) QS+QT+C: Based on QS+QT, we design a Cross graph (C) neural network to jointly learn the representations of task requesters in different semantic spaces and control how much information flows across two graphs.

3) QS+QT+MP: Based on QS+QT, we use Transformer to learn the Multi-Perspective (MP) relations of the task-publishing sequence.

4) QS+QT+C+MP (TPR): We use both the Cross graph neural network (C) and the Multi-Perspective (MP) components to predict task publication timeslots for task requesters.

Metrics. To evaluate the accuracy of the above models, we adopt the widely-used measure, *Recall@N* ($N \in 1, 2, 3$). For each task requester, *Recall@N* indicates how well the top-N recommended timeslots match the labeled publication timeslot and its four adjacent timeslots before and after.

Table 3: *Recall@N* of Different Methods for TPTR

Methods	Recall@1	Recall@2	Recall@3
QS+QT	0.0996	0.1820	0.2567
QS+QT+C	0.1015	0.1836	0.2578
QS+QT+MP	0.1374	0.2227	0.2825
TPR	**0.1425**	**0.2322**	**0.3037**

Results. Table 3 shows the evaluation results. QS+QT only combines the interactions of requester-task graph and requester-timslot graph and performs worst. After adding the cross graph neural work component or the multi-perspective component to the initial model, the performance of the model is further improved, i.e., QS+QT+C outperforms QS+QT by 0.41%–1.92% in terms of *Recall@N* ($N = 1, 2, 3$), and QS+QT+MP outperforms QS+QT by an astounding margin (up to 37.91%). It shows the multi-perspective component is more effective than the cross graph neural network component. By adopting both of the two components above, the TPR model gets the best performance, which demonstrates its superiority in task publication time recommendation.

4.2.2 Performance of Task Assignment. Next we evaluate the performance of task assignment.

Evaluation Methods. We study the following methods.

1) DE+GR: The greedy task assignment method, where the task registration time is directly regarded as the task publication time.

2) FQ+GR: The greedy task assignment method based on the task publication time recommended by a frequency-based method (FQ). FQ compares the difference between task publication time and task execution time in historical data and selects the one with the highest occurrence frequency, based on which FQ calculates the recommended task publication time.

3) TPR+GR: The greedy task assignment method based on the task publication time recommended by our model, TPR.

4) DE+MCMF: The MCMF task assignment method, where the task registration time is directly regarded as the task publication time.

5) FQ+MCMF: The MCMF task assignment method based on the task publication time recommended by FQ.

6) TPR+MCMF: The MCMF task assignment method based on the task publication time recommended by TPR.

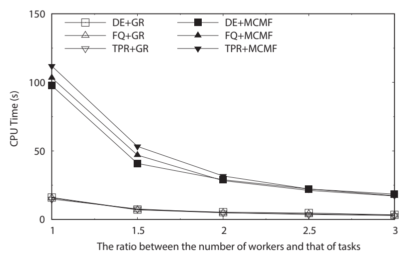
Metrics. Three main metrics are compared among the above algorithms, i.e., CPU time, completion rate of tasks, and average waiting time of task requesters, for finding the final task assignment. Table 4 shows our experimental settings, where the default values of all parameters are underlined.

Table 4: Experiment Parameters

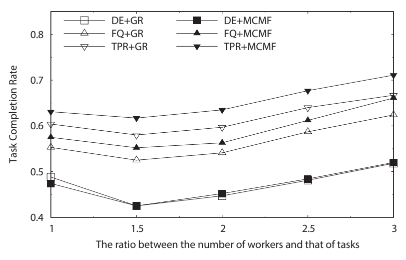
Parameter	Value
The ratio between the number of workers and that of tasks, $\|W\|/\|S\|$	$\underline{1}$, 1.5, 2, 2.5, 3
Valid time of tasks (h), $e - p$	0.5, 0.75, 1, 1.25, $\underline{1.5}$
Valid time of workers (h), $off - on$	0.25, 0.5, 0.75, 1, $\underline{1.25}$
Reachable distance of workers (km), r	2, 2.5, 3, 3.5, $\underline{4}$

Effect of $|W|/|S|$. We study the effect of $|W|/|S|$, the ratio between the number of workers and that of tasks, where the number of workers is fixed to 3000. Intuitively, a high $|W|/|S|$ value represents a worker-dense area. In Figure 3(a), the CPU time of all the methods decreases with the increase of $|W|/|S|$. Moreover, the CPU time of the MCMF-related methods (i.e., DE+MCMF, FQ+MCMF, and TPR+MCMF) declines faster than GR-related methods (i.e., DE+GR, FQ+GR, and TPR+GR), since the time complexity of MCMF methods is higher and more affected by the total number of tasks and workers than Greedy methods. As can be seen in Figure 3(b), as $|W|/|S|$ grows, the task completion rate decreases first (when $|W|/|S| < 1.5$) and then increases (when $|W|/|S| \geq 1.5$). All methods perform best when $|W|/|S| = 3$, since each task has more candidate available workers and more tasks can be completed in such a case. Also, TPR-related methods (i.e., TPR+GR and TPR+MCMF) achieve the highest task completion rate, followed by FQ-related methods (i.e., FQ+GR and FQ+MCMF) and DE-related methods (i.e., DE+GR and DE+MCMF), which demonstrates the advantage of our TPR model for task publication time recommendation. As is shown in Figure 3(c), the average waiting time of all methods except DE+GR almost keeps stable with the increasing $|W|/|S|$, showing their good scalability that can adapt to various $|W|/|S|$. The MCMF method performs worse than the Greedy method when combining with TPR while it performs better than the Greedy method when combining DE or FQ in terms of the average waiting time. This is because MCMF must make a tradeoff between maximizing the number of completed tasks and minimizing the distance from workers to tasks, which leads to that MCMF cannot always reduce the average waiting time compared to the Greedy method. Apparently, the average waiting time of TPR-related methods is still less than others, which demonstrates the superiority of TPR.

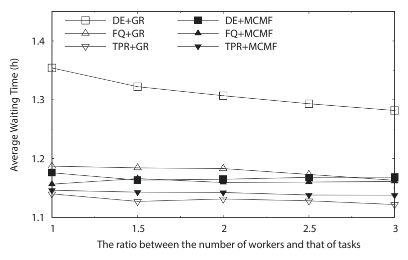
Effect of $e - p$. Next, we study the effect of $e - p$, the valid time of tasks. As illustrated in Figure 4(a), we can see that the CPU time of



(a) CPU Time

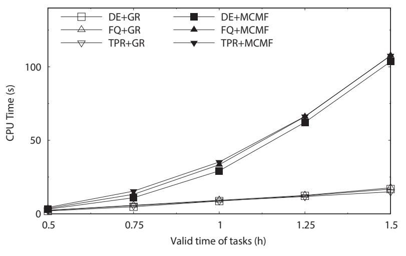


(b) Task Completion Rate

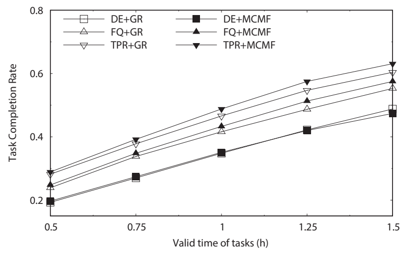


(c) Average Waiting Time

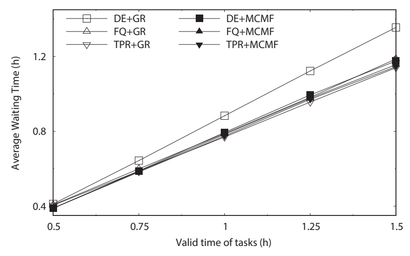
Figure 3: Performance of Task assignment: Effect of $|W|/|S|$



(a) CPU Time



(b) Task Completion Rate



(c) Average Waiting Time

Figure 4: Performance of Task assignment: Effect of $e - p$

all methods shows an upward trend, since longer valid time of tasks means more tasks can be completed. Although MCMF-related methods are more time-consuming than GR-related methods, the computing efficiency of MCMF-related methods is acceptable. There is no doubt that the task completion rate increases with the extension of the task valid time. As can be seen in Figure 4(b), TPR-related methods perform the best followed by FQ-related methods and DE-related methods. Moreover, all MCMF-related methods except DE+MCMF can always achieve a higher task completion rate than GR-related methods, which shows the effectiveness of MCMF. As for DE-related methods, they publish tasks directly without considering the ratio of tasks and workers at each time, which leads to longer waiting time for task requesters. So the MCMF method may sacrifice some task completion rate to achieve less waiting time during task assignment. In Figure 4(c), the average waiting time is on the rise when $e - p$ increases, and TPR+GR achieves less average waiting time compared to other methods.

Effect of $off - on$**.** Next, we study the effect of $off - on$, which represents the valid time of workers. In Figure 5(a), with the increase of $off - on$, the CPU time grows slowly for most methods. Among all the methods, TPR-GR has the least CPU time. As shown in Figure 5(b), the task completion rate of all the methods also increases smoothly since more tasks can be completed when the valid time of workers gradually extends. Moreover, MCMF-related methods still have the best performance compared to their counterparts. Besides, TPR-related methods achieve a higher task completion rate followed by FQ-related methods and DE-related methods. Another observation is that the average waiting time of all the methods keeps stable when $off - on$ increases, as depicted in Figure 5(c). And not surprisingly, TPR-related methods achieve the least average waiting time during task assignment.

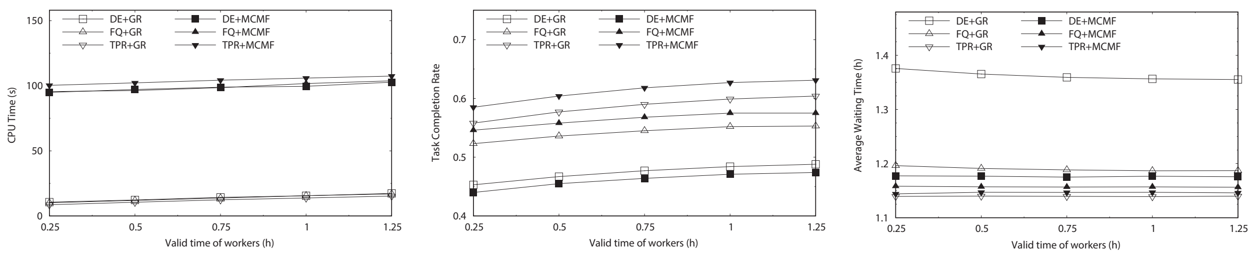
Effect of r**.** Finally, we study the effect of r. In Figure 6(a), the CPU time of MCMF-related methods shows an ascending trend while that of GR-related methods shows a completely opposite trend when r grows. This is because the greater reachable distance of workers accelerates the process of finding available workers for GR-related methods while increasing the number of edges in the network flow graph for MCMF-related methods, which leads to more iterations. With the increasing reachable distance of workers, the number of candidate available workers for tasks is enlarged, which promotes the task completion rate of all the methods in Figure 6(b). In addition, TPR+MCMF still achieves the highest task completion rate among all the methods and TPR-related methods are more outstanding than their counterparts. When $r = 2$, all the methods cannot assign tasks well but TPR-related methods still perform the best. As illustrated in Figure 6(c), the average waiting time of MCMF-related methods grows faster than that of GR-related methods. This is because MCMF aims to achieve a higher task completion rate globally and may not always choose the nearest workers of tasks at each time when the number of available workers of tasks grows.

5 RELATED WORK

5.1 Task Assignment in Spatial Crowdsourcing

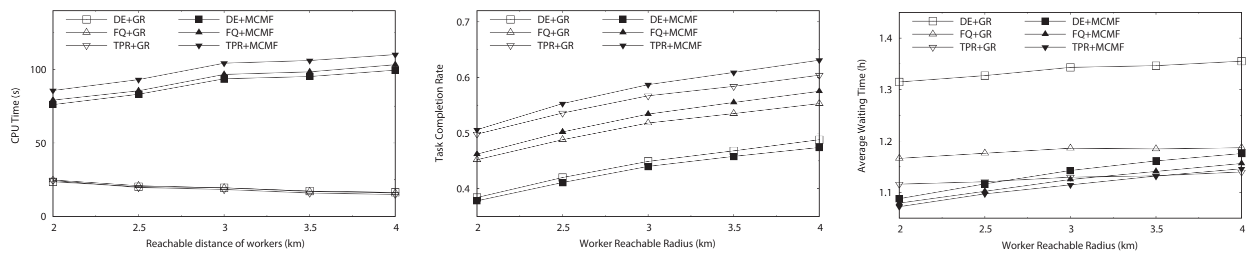
Spatial Crowdsourcing (SC) is an increasing popular category of crowdsourcing in the era of mobile Internet and sharing economy, where spatio-temporal tasks must be completed at a specific location on time [27]. SC has attracted extensive attention from both the academia and the industry [13, 18, 19, 31, 37–39], and there have been many successful SC platforms such as GrubHub[1] and Uber[2].

[1]https://get.grubhub.com/
[2]https://www.uber.com/



(a) CPU Time (b) Task Completion Rate (c) Average Waiting Time

Figure 5: Performance of Task assignment: Effect of $off - on$



(a) CPU Time (b) Task Completion Rate (c) Average Waiting Time

Figure 6: Performance of Task assignment: Effect of r

Task assignment is considered as one of the most fundamental challenges in SC [6].

An SC platform can assign tasks to suitable workers according to different optimization objectives during task assignment such as maximizing the total number of assigned tasks [26, 33], maximizing the total payoff of workers [4, 9, 25], and minimizing the total travel cost of the assigned workers [3, 7]. For example, Zhao et al. [40] propose a preference-aware spatial task assignment system and use a tensor-decomposition algorithm to learn worker preference, based on which they assign tasks with the Minimum Cost Maximum Flow (MCMF) method, aiming to maximize the total number of task assignments. Dickerson et al. [9] design an adaptive algorithm based on an offline-guide-online technique with the purpose of maximizing the total payoff of workers. It firstly solves a linear program benchmark and then uses the offline solution to simulate the online matching procedure.

5.2 Recommendation in Spatial Crowdsourcing

In SC systems, task recommendation can help workers to find their appropriate tasks and task requesters to receive high-quality task results [1, 5, 11, 21, 24, 34] recently. For task recommendation, Chen et al. [5] study the offline scenario where all tasks are known in advance, which is not practical in real-world applications. As an improvement of the study [5], Yuan et al. [35] consider dynamic scenarios of new workers and new tasks in an SC system and propose a Task Recommendation (called TaskRec) framework based on a unified probabilistic matrix factorization to recommend suitable tasks for workers. However, the above studies mainly offer task or route recommendation for workers, which fall short in terms of considering recommendation for task requesters. In this paper, we focus on the task publication time recommendation from the perspective of task requesters rather than that of workers.

6 CONCLUSION

Due to the rapid development of mobile technologies and dramatic proliferation of advanced mobile devices equipped with sensors, recent years witness the prosperity of the Spatial Crowdsourcing (SC) market, which consists of location-specific tasks and requires workers to physically be at specific locations to complete them. In this paper, we offer solutions to a problem called Publication Recommendation based Task Assignment in SC, aiming to enable effective task assignment with the task publication time recommendation. Specifically, we achieve task publication time recommendation by jointly learning the representations of task requesters in different semantic spaces with the cross-graph neural network and learning the multi-perspective relations of the task-publishing sequences with the self-attention. Based on that, we perform effective task assignment in SC. To the best of our knowledge, this is the first work in SC that recommends the publication time for a task requester when a request is proposed and performs task assignment based on the recommendation. Extensive experiments on real data demonstrate the effectiveness of our proposed solutions.

7 ACKNOWLEDGMENTS

This work is partially supported by NSFC (No. 61972069, 61836007 and 61832017), and Shenzhen Municipal Science and Technology RD Funding Basic Research Program (JCYJ20210324133607021).

REFERENCES

[1] Abdulrahman Alamer, Jianbing Ni, Xiaodong Lin, and Xuemin Shen. 2017. Location privacy-aware task recommendation for spatial crowdsourcing. In *WCSP*.

1–6.
[2] Vamsi Ambati, Stephan Vogel, and Jaime Carbonell. 2011. Towards task recommendation in micro-task markets. In *AAAI*.
[3] Nikhil Bansal, Niv Buchbinder, Anupam Gupta, and Joseph Seffi Naor. 2014. A randomized O (log2 k)-competitive algorithm for metric bipartite matching. *Algorithmica* 68, 2 (2014), 390–403.
[4] Brian Brubach, Karthik Abinav Sankararaman, Aravind Srinivasan, and Pan Xu. 2016. New algorithms, better bounds, and a novel model for online stochastic matching. *ESA* 57, 24 (2016), 1–16.
[5] Cen Chen, Shih-Fen Cheng, Hoong Chuin Lau, and Archan Misra. 2015. Towards city-scale mobile crowdsourcing: Task recommendations under trajectory uncertainties. In *IJCAI*. 1113–1119.
[6] Lei Chen and Cyrus Shahabi. 2016. Spatial crowdsourcing: Challenges and opportunities. *Bulletin of the Technical Committee on Data Engineering* 39, 4 (2016), 14.
[7] Zhao Chen, Peng Cheng, Yuxiang Zeng, and Lei Chen. 2019. Minimizing maximum delay of task assignment in spatial crowdsourcing. In *ICDE*. 1454–1465.
[8] Yue Cui, Liwei Deng, Yan Zhao, Bin Yao, Vincent W Zheng, and Kai Zheng. [n.d.]. Hidden poi ranking with spatial crowdsourcing. In *KDD*.
[9] John P Dickerson, Karthik A Sankararaman, Aravind Srinivasan, and Pan Xu. 2018. Assigning tasks to workers based on historical data: Online task assignment with two-sided arrivals. In *AAMAS*. 318–326.
[10] Lester Randolph Ford and Delbert R Fulkerson. 1956. Maximal flow through a network. *Canadian journal of Mathematics* 8 (1956), 399–404.
[11] Dawei Gao, Yongxin Tong, Jieying She, Tianshu Song, Lei Chen, and Ke Xu. 2017. Top-k team recommendation and its variants in spatial crowdsourcing. *DSE* 2, 2 (2017), 136–150.
[12] David Geiger and Martin Schader. 2014. Personalized task recommendation in crowdsourcing information systems—Current state of the art. *Decision Support Systems* 65 (2014), 3–16.
[13] Srinivasa Raghavendra Bhuvan Gummidi, Xike Xie, and Torben Bach Pedersen. 2019. A survey of spatial crowdsourcing. *TODS* 44, 2 (2019), 1–46.
[14] Danhuai Guo, Yingqiu Zhu, Wei Xu, Shuo Shang, and Zhiming Ding. 2016. How to find appropriate automobile exhibition halls: Towards a personalized recommendation service for auto show. *Neurocomputing* 213 (2016), 95–101.
[15] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *SIGIR*. 639–648.
[16] Leyla Kazemi and Cyrus Shahabi. 2012. Geocrowd: enabling query answering with spatial crowdsourcing. In *SIGSPATIAL*. 189–198.
[17] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
[18] Xiang Li, Yan Zhao, Jiannan Guo, and Kai Zheng. 2020. Group task assignment with social impact-based preference in spatial crowdsourcing. In *DASFAA*. 677–693.
[19] Xiang Li, Yan Zhao, Xiaofang Zhou, and Kai Zheng. 2020. Consensus-Based Group Task Assignment with Social Impact in Spatial Crowdsourcing. *Data Science and Engineering* 5, 4 (2020), 375–390.
[20] Yunchuan Li, Yan Zhao, and Kai Zheng. 2021. Preference-aware Group Task Assignment in Spatial Crowdsourcing: A Mutual Information-based Approach. In *ICDM*. 350–359.
[21] Dan Lu, Qilong Han, Hongbin Zhao, and Kejia Zhang. 2017. Optimal Task Recommendation for Spatial Crowdsourcing with Privacy Control. In *ICPCSEE*. 412–424.
[22] Jiangang Shu and Xiaohua Jia. 2016. Secure task recommendation in crowdsourcing. In *GLOBECOM*. 1–6.
[23] Jiangang Shu, Xiaohua Jia, Kan Yang, and Hua Wang. 2018. Privacy-preserving task recommendation services for crowdsourcing. *TCC* 14, 1 (2018), 235–247.
[24] Dezhi Sun, Ke Xu, Hao Cheng, Yuanyuan Zhang, Tianshu Song, Rui Liu, and Yi Xu. 2019. Online delivery route recommendation in spatial crowdsourcing. *WWWJ* 22, 5 (2019), 2083–2104.
[25] Hing-Fung Ting and Xiangzhong Xiang. 2015. Near optimal algorithms for online maximum edge-weighted b-matching and two-sided vertex-weighted b-matching. *Theoretical Computer Science* 607 (2015), 247–256.
[26] Yongxin Tong, Libin Wang, Zhou Zimu, Bolin Ding, Lei Chen, Jieping Ye, and Ke Xu. 2017. Flexible online task assignment in real-time spatial data. *PVLDB* 10, 11 (2017), 1334–1345.
[27] Yongxin Tong, Zimu Zhou, Yuxiang Zeng, Lei Chen, and Cyrus Shahabi. 2020. Spatial crowdsourcing: a survey. *PVLDB* 29, 1 (2020), 217–250.
[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *NIPS* 30 (2017), 5998–6008.
[29] Xin Wang, Jin Liu, Xiao Liu, Xiaohui Cui, and Hao Wu. 2020. A novel dual-graph convolutional network based web service classification framework. In *ICWS*. IEEE, 281–288.
[30] Ziwei Wang, Yan Zhao, Xuanhao Chen, and Kai Zheng. 2021. Task Assignment with Worker Churn Prediction in Spatial Crowdsourcing. In *CIKM*. 2070–2079.
[31] Jinfu Xia, Yan Zhao, Guanfeng Liu, Jiajie Xu, Min Zhang, and Kai Zheng. 2019. Profit-driven Task Assignment in Spatial Crowdsourcing.. In *IJCAI*. 1914–1920.
[32] Dingqi Yang, Benjamin Fankhauser, Paolo Rosso, and Philippe Cudre-Mauroux. 2020. Location prediction over sparse user mobility traces using RNNs: Flashback in hidden states!. In *IJCAI*. 2184–2190.
[33] Guanyu Ye, Yan Zhao, Xuanhao Chen, and Kai Zheng. 2021. Task Allocation with Geographic Partition in Spatial Crowdsourcing. In *CIKM*. 2404–2413.
[34] Xicheng Yin, Hongwei Wang, Wei Wang, and Kevin Zhu. 2020. Task recommendation in crowdsourcing systems: A bibliometric analysis. *Technology in Society* 63 (2020), 101337.
[35] Man-Ching Yuen, Irwin King, and Kwong-Sak Leung. 2015. Taskrec: A task recommendation framework in crowdsourcing systems. *Neural Processing Letters* 41, 2 (2015), 223–238.
[36] Junwei Zhang, Fan Yang, Zhuo Ma, Zhuzhu Wang, Ximeng Liu, and Jianfeng Ma. 2020. A decentralized location privacy-preserving spatial crowdsourcing for internet of vehicles. *TITS* 22, 4 (2020), 2299–2313.
[37] Yan Zhao, Xuanhao Chen, Liwei Deng, Tung Kieu, Chenjuan Guo, Bin Yang, Kai Zheng, and Christian S. Jensen. 2022. Outlier Detection for Streaming Task Assignment in Crowdsourcing. In *WWW*.
[38] Yan Zhao, Jiannan Guo, Xuanhao Chen, Jianye Hao, Xiaofang Zhou, and Kai Zheng. 2021. Coalition-based task assignment in spatial crowdsourcing. In *ICDE*. 241–252.
[39] Yan Zhao, Yang Li, Yu Wang, Han Su, and Kai Zheng. 2017. Destination-aware Task Assignment in Spatial Crowdsourcing. In *CIKM*. 297–306.
[40] Yan Zhao, Jinfu Xia, Guanfeng Liu, Han Su, Defu Lian, Shuo Shang, and Kai Zheng. 2019. Preference-aware task assignment in spatial crowdsourcing. In *AAAI*. 2629–2636.
[41] Yan Zhao, Kai Zheng, Yue Cui, Han Su, Feida Zhu, and Xiaofang Zhou. 2020. Predictive task assignment in spatial crowdsourcing: a data-driven approach. In *ICDE*. 13–24.
[42] Yan Zhao, Kai Zheng, Jiannan Guo, Bin Yang, Torben Bach Pedersen, and Christian S Jensen. 2021. Fairness-aware task assignment in spatial crowdsourcing: Game-theoretic approaches. In *ICDE*. 265–276.
[43] Yan Zhao, Kai Zheng, Yang Li, Han Su, Jiajun Liu, and Xiaofang Zhou. 2019. Destination-aware Task Assignment in Spatial Crowdsourcing: A Worker Decomposition Approach. *TKDE* 32, 12 (2019), 2336–2350.