Jurnal Personalia Pelajar 29(1): 9-23
https://doi.org/10.17576/personalia.2901.2026.02
ISSN 0128-2735

A Crowdsourcing Approach to Ad-Hoc on-Site Information Collection
(Pendekatan Penyumberan Khalayak untuk Pengumpulan Maklumat di Lokasi Secara Ad-Hoc)

YATARO FUJINAGA, HIROYOSHI ITO, NOBUTAKA SUZUKI & ATSUYUKI MORISHIMA

ABSTRACT

Spatial Crowdsourcing (SC) is used to complete real-world tasks that must be performed at specific locations, such as disaster assessment, traffic monitoring, and infrastructure inspection. Many SC studies assume the Server Assigned Tasks (SAT) model, where the AI running on the server assigns tasks directly to workers. However, this model does not reflect important characteristics of real SC systems. Workers may reject task requests, and the number of available workers is often limited. Because of these factors, the order in which workers are asked and which workers are asked can strongly affect the number of tasks that are eventually completed. Yet, this effect has not been fully examined in prior research. To address this issue, this study proposes a framework called request flows. A request flow is a sequential process in which the server sends a task request to one worker at a time and selects the next worker based on whether the previous worker accepted or rejected the request. The first contribution of this paper is that we show the impact of choosing good request flows. Using synthetic datasets, we found that the design of the request flow can change the expected number of completed tasks by up to 12%. Thus, choosing a good request flow is an important task of the assignment AI on the server. However, the problem is that choosing good request flows is computationally expensive. The second contribution is that we compared different methods to approach the problem and found that a greedy algorithm can find a request flow that achievesabout 90% of the performance of the optimal solution while running in practical computation time. These results indicate that modeling and optimizing request flows can substantially improve task completion in realistic SC settings.

Keywords: Spatial Crowdsourcing, Task Assignment

ABSTRAK

Penyumberan Khalayak Spatial digunakan untuk menyelesaikan tugasan dunia sebenar yang mesti dilaksanakan di lokasi tertentu, seperti penilaian bencana, pemantauan trafik dan pemeriksaan infrastruktur. Banyak kajian SC mengandaikan model Tugasan Ditugaskan Pelayan (Server Assigned Tasks - SAT), di mana kecerdasan buatan (AI) yang berjalan pada pelayan memberikan tugasan secara terus kepada pekerja. Walau bagaimanapun, model ini tidak mencerminkan ciri-ciri penting sistem SC yang sebenar. Pekerja mungkin menolak permintaan tugasan, dan bilangan pekerja yang tersedia selalunya terhad. Disebabkan faktor-faktor ini, urutan pekerja yang diminta dan pekerja mana yang dipilih boleh memberi kesan kuat terhadap jumlah tugasan yang akhirnya dapat diselesaikan. Namun, kesan ini masih belum diperiksa sepenuhnya dalam kajian terdahulu. Bagi menangani isu ini, kajian ini mencadangkan satu rangka kerja yang dipanggil aliran permintaan (request flows). Aliran permintaan ialah proses berurutan di mana pelayan menghantar permintaan tugasan kepada seorang pekerja pada satu-satu masa, dan memilih pekerja seterusnya berdasarkan sama ada pekerja sebelumnya menerima atau menolak permintaan tersebut. Sumbangan pertama kertas kajian ini adalah menunjukkan impak daripada pemilihan aliran permintaan yang baik. Menggunakan set data sintetik, kami mendapati bahawa reka bentuk aliran permintaan boleh mengubah jangkaan bilangan tugasan yang diselesaikan sehingga 12%. Oleh itu, memilih aliran permintaan yang baik merupakan tugas penting bagi AI penugasan pada pelayan.Walau bagaimanapun, masalahnya ialah pemilihan aliran permintaan yang baik memerlukan kos pengiraan (computational cost) yang tinggi. Sumbangan kedua kajian ini adalah kami membandingkan kaedah yang berbeza untuk menangani masalah tersebut dan mendapati bahawa algoritma tamak (greedy algorithm) mampu mencari aliran permintaan yang mencapai sekitar 90% daripada prestasi penyelesaian optimum, sambil berjalan dalam masa pengiraan yang praktikal. Keputusan ini menunjukkan bahawa pemodelan dan pengoptimuman aliran permintaan dapat meningkatkan penyelesaian tugasan secara sgnifikan dalam persekitaran SC yang realistik.

Kata Kunci: Penyumberan Khalayak Spatial, Penugasan Tugasan

INTRODUCTION

In recent years, crowdsourcing, which involves outsourcing tasks to a large number of unspecified individuals via the Internet, has been widely adopted in many applications and became one of the technologies that yield high social impacts (Howe, 2009). Crowdsourcing is an approach to solving various problems by leveraging the knowledge, skills, and time of a large number of people who are not affiliated with any specific organization or company. Representative examples include platforms such as Amazon Mechanical Turk (Amazon Mechanical Turk, 2025) for data annotation tasks like image labeling and text classification, designs for soliciting logo and website designs. On these platforms, workers can participate in tasks from anywhere with an Internet connection, such as their homes or cafes, with work primarily conducted entirely online.

Among various forms of crowdsourcing, Spatial Crowdsourcing (SC) has recently attracted considerable attention. SC is an approach that assigns location-based tasks to a large number of workers and requires them to complete the tasks on-site. Unlike traditional crowdsourcing where tasks are performed online, SC requires workers to physically travel to specific locations and perform tasks or collect information that can only be obtained on-site. Examples include photographing specific buildings or landscapes, surveying product shelf arrangements and prices in stores, and inspecting road damage conditions. These tasks can only be completed when workers physically visit the locations, observe with their own eyes, and record and report information using mobile devices such as smartphones.

SC has the potential to provide powerful solutions to problems that have traditionally been difficult to address. A well-known example of the successful use of spatial crowdsourcing is the Red Balloon Challenge (John C Tang, 2011). This challenge, organized by the U.S. Defense Advanced Research Projects Agency (DARPA) in 2009, tasked participants with locating ten red weather balloons placed at unknown locations across the United States. Using crowdsourcing, the team that won the challenge identified all ten balloons in approximately nine hours.

Traditionally, tasks requiring in-person fieldwork have been carried out by a limited number of designated personnel. Because the number of available workers is restricted, such approaches struggle to cover large geographic areas. In contrast, SC enables the mobilization of a vast number of volunteers, making it possible to execute large-scale field tasks more efficiently. Examples of collecting and utilizing information from large numbers of individuals include disaster damage assessment [Mohammad H Vahidnia, 2020], weather monitoring [Luan Tran, 2018], traffic condition surveillance (Waze, 2025), and map updating (OpenStreetMap, 2025). By leveraging workers' mobile devices, it becomes possible to collect location data and images captured on site and transmit them to a central server, enabling efficient acquisition of location-dependent information.

A wide range of platforms already rely on tasks that require workers to physically travel (Yongxin Tong Z. Z., 2020). Examples include ride-sharing services such as Uber (Uber, 2025) and Didi (Didi, 2025); micro-task platforms such as Gigwalk (Gigwalk, 2025), which assigns in-store product checks; TaskRabbit (TaskRabbit, 2025), which handles household micro-tasks such as furniture assembly and moving; and food delivery services such as GrubHub (GrubHub, 2025), Instacart, and UberEats (UberEats, 2025). These applications demonstrate that tasks requiring physical movement play an important role in society and that SC already has significant real-world impact.

One of the central objectives in SC is to maximize the number of completed tasks from a given task set. In "division-based" SC, multiple workers collectively cover the task set, and the key issue becomes determining which worker should perform which task in order to ensure efficient execution. However, maximizing the number of completed tasks in SC is not straightforward for three major reasons. First, the number of workers available in SC is significantly smaller than in online crowdsourcing systems, where large numbers of workers can participate simultaneously. Second, workers in SC exhibit distinctive behavioral tendencies: they prefer to select tasks themselves and tend to work on tasks near their residence or along their movement paths. As a result, server-driven task assignment, widely used in traditional crowdsourcing is often impractical. Third, task completion is inherently uncertain. Even if a worker is assigned a set of tasks, there is no guarantee that they will complete them. Therefore, assignment decisions must take into account each worker's probability of completing their tasks.

To address these behavioral characteristics of SC workers, this study introduces a framework called a request flow (Figure 1). A request flow determines the order in which workers are requested and specifies, depending on whether each worker accepts or rejects the request, which worker should be contacted next. This framework allows the system to dynamically adapt to workers' acceptance outcomes and is expected to improve task completion even when the number of

available workers is limited.

In this study, we clarify the extent to which the design of the request flow affects the number of completed tasks. We also propose algorithms that construct effective request flows within practical computational time. Based on these objectives, we formulate the following two research questions:

i. RQ1: To what extent does the choice of request flow influence the final number of completed tasks?
ii. RQ2: How can an effective request flow be constructed within realistic computational time to maximize the number of task completions?

To address these RQs, this study presents the following two findings:

i. Finding 1: The design of the request flow can result in variations of up to approximately 12% in the number of completed tasks.
ii. Finding 2: By employing a greedy algorithm, we can obtain a request flow that achieves approximately 90% of the performance compared to the optimal solution.

The novelty of this study lies in formulating the maximization of completed tasks in Spatial Crowdsourcing as a sequential decision-making problem that incorporates worker behavioral characteristics, an aspect that has been largely overlooked in prior work.

The remainder of this thesis is organized as follows. Chapter 2 reviews related work in Spatial Crowdsourcing and positions the contribution of this study. Chapter 3 defines the problem setting, incorporating worker behavior and the sequential structure of request flows, and formulates the evaluation metrics. Chapter 4 presents the proposed methods, including both an exact algorithm that enumerates optimal request flows and greedy heuristics that compute effective flows within practical computational time. Chapter 5 describes the experimental settings and results, demonstrating both the impact of request flows on task completion and the performance of the proposed algorithms. Finally, Chapter 6 concludes the thesis and discusses remaining challenges in optimizing request flows in Spatial Crowdsourcing.

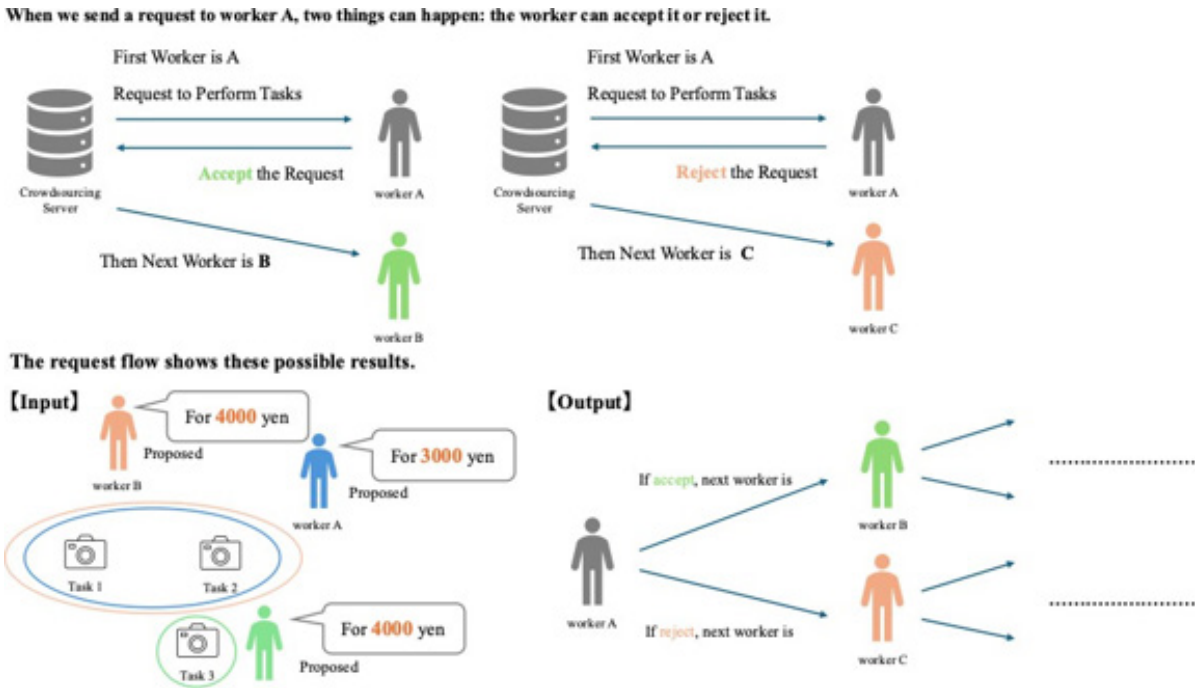

FIGURE 1. What is the Request Flow

RELATED WORK

Task Assignment

Task assignment research in Spatial Crowdsourcing (SC) aims to design assignment strategies that optimize specific performance metrics. Such metrics include the number of assigned tasks, the number of completed tasks (Shahabi, 2012) (Hien To, 2015) (Leyla Kazemi, 2013) (Curry U. u., 2016) (Yan Zhao K. Z., 2019), worker rewards (Yu Li, 2015) (Peng Cheng X. L., 2016), and the quality of assignment results (Peng Cheng X. L.,

2017) (Chunyan Miao, 2016).

In addition to simple optimization, several studies incorporate budget constraints into the assignment problem (Peng Cheng X. L., 2016) (Peng Cheng X. L., 2017) (Chunyan Miao, 2016) (Xiaofeng Gao, 2020). For example, when workers are compensated monetarily, some studies aim to maximize the number of completed tasks under a limited budget (Xu, 2020), while others seek to maximize result quality (Long Tran-Thanh, 2014) (Yunhui Li, 2021).

Task assignment approaches can be broadly categorized into Server Assigned Task (SAT) and Worker Selected Task (WST) models. In SAT models, the AI running on the server assigns tasks directly to workers, making it easier to optimize specific metrics. In contrast, in WST models, workers autonomously choose tasks; thus, the server cannot directly control worker behaviors, making optimization more difficult. Consequently, most existing studies adopt the SAT model, and research addressing WST remains limited.

A notable example of WST research is by Deng et al. (Dingxiong Deng, 2013), who optimize the execution order of tasks chosen by workers so as to minimize their travel distance. Considering travel distance is particularly specific to SC, and this study demonstrates that even in WST settings, optimizing the order of task execution is important.

Worker Preferences for Tasks Involving Physical Movement

Previous research has pointed out that worker behavior in SC differs from that in online crowdsourcing. Alt et al. (Florian Alt, 2010) report that, for tasks requiring physical movement, workers tend to select tasks located near their residence or daily travel routes. Based on such behavioral tendencies, several studies utilize workers' historical mobility patterns for task assignment (Peng Cheng X. L., 2016) (Cen Chen, 2014). In the context of taxi dispatching, user destination prediction based on mobility history has also been used to improve dispatch efficiency (Lingyu Zhang, 2017). Other studies incorporate worker preferences to improve assignment quality (Yan Zhao J. X., 2019) (Liwei Deng, 2024). Because tasks in SC are tied to fixed spatial locations, the distance between workers and tasks significantly affects assignment outcomes (Shahabi, 2012) (Leyla Kazemi, 2013). Distant tasks are less likely to be accepted, whereas nearby tasks are more likely to be completed. This arises because spatial proximity induces worker preferences toward specific tasks.

For this reason, the WST model (where workers choose tasks voluntarily) may better reflect the characteristics of SC than the SAT model. The method for determining worker payments is another crucial factor in task assignment. Research on incentive mechanisms includes the posted-price model (Yongxin Tong L. W., 2018) and auction-based models (Lam, 2022) (Ying Hu, 2018), in which workers propose the cost required to perform tasks. Lam et al. propose a framework in which worker payments are determined based on costs proposed by workers using their task results.

Uncertainty in Task Completion and Acceptance Models

Many existing studies assume that assigned tasks are always completed (Shahabi, 2012) (Hien To, 2015) (Leyla Kazemi, 2013). However, because SC tasks require physical movement, workers may be unable to complete the assigned tasks in practice. Indeed, prior work notes the inherent uncertainty associated with task completion in SC (Yongxin Tong Z. Z., 2020). Motivated by such observations, several studies have proposed probabilistic models of worker rejection behavior and incorporated them into assignment strategies (Xinglin Zhang, 2016) (Curry U. U., 2014) (Chen, 2017).

Batch Assignment vs. Sequential Assignment

Many prior studies adopt a batch assignment model, where the server obtains all task and worker information at once and determines an optimal assignment in a single step (Shahabi, 2012) (Hien To, 2015) (Leyla Kazemi, 2013). While this approach is computationally efficient, it cannot incorporate workers' subsequent behaviors (such as rejection) which may lead to inefficient assignments in practice.

Alt et al. (Florian Alt, 2010) point out that SC typically involves fewer available workers than online crowdsourcing. Because the number of workers is limited in SC, acceptance or rejection by a single worker can significantly affect the number of completed tasks. Therefore, sequential assignment, which selects the next worker based on previous workers' acceptance outcomes, is considered more effective for utilizing limited workers than batch assignment. Sequential assignment shares with online assignment problems (Yongxin Tong J. S., 2016) (Yongxin Tong J. S., 2016) (Tianshu Song, 2017) the need to make decisions in stages without full prior information. In both settings, decisions rely on the information available at the current time. However, sequential assignment in SC differs fundamentally from general online assignment.

In online assignment problems, decisions are made based on the arrival order of tasks or workers. In SC sequential assignment, decisions are made based on the acceptance outcomes (accept/reject) of workers to whom the server has already issued requests. Thus, whereas online assignment depends on sequentially arriving *inputs*, the sequential assignment in this study depends on sequentially observed *worker responses*.

Applications

In recent years, SC, where workers perform tasks in the physical world using location information, has been increasingly deployed in real-world services. Representative examples include ride-sharing services such as Uber (Uber, 2025) and Didi (Didi, 2025), traffic and hazard reporting via Waze (Waze, 2025), same-day delivery via Instacart (Instacart, 2025) and Postmates (Postmates, 2025), in-store audit tasks via Gigwalk (Gigwalk, 2025), and household support tasks (e.g., furniture assembly, cleaning) via TaskRabbit (TaskRabbit, 2025). These applications share the common characteristic that tasks arise at specific locations and require workers to physically travel. Consequently, the design of task assignment methods must account for real-world constraints such as travel cost and uncertainty in task acceptance, creating strong demand for SC-specific research.

Summary

The "request flow" studied in this work is positioned as a sequential task assignment method that explicitly incorporates SC-specific constraints, acceptance uncertainty, limited worker availability, and worker-driven task selection. This study provides a systematic model for maximizing expected task completion while dynamically adapting to worker responses, offering a perspective distinct from prior research assuming batch assignment.

PROBLEM DEFINITION

This chapter defines the optimization problem of request flows under budget constraints, which is the focus of this study. The problem consists of three components: input, output, and objective functions. The input specifies the information provided to the problem, the output represents the solution, and the objective functions evaluate the quality of a solution. We begin by defining the input components, followed by the definition of a request flow. Then, we introduce the objective functions, and finally define the optimal request flow based on the input and objectives.

Input

The input consists of a set of workers W, a set of tasks T, a set of proposals P, and a budget B. The worker set is $W = \{w_1, w_2, \ldots, w_m\}$, representing the workers who have submitted proposals to the server. The task set is $T = \{t_1, t_2, \ldots, t_n\}$, representing the set of spatial tasks prepared by the server. The proposal set is $P = \{p_1, p_2, \ldots, p_m\}$, where each proposal corresponds to a worker.

For each worker $w_i \in W$, the proposal p_i consists of: a task set $S_i \subseteq T$ that the worker is willing to perform, a cost c_i paid to the worker upon acceptance, an acceptance probability q_i, representing the probability that the worker accepts a request and completes the tasks. The budget B denotes the maximum amount of money available for worker payments.

A request refers to asking a worker w_i to complete the task set S_i specified in their proposal.

Upon receiving a request, worker w_i accepts it with probability q_i and rejects it with probability $1 - q_i$. If the worker accepts, the server pays c_i, and the worker completes all tasks in S_i. If the worker rejects, no payment is made, and no tasks are completed. A request flow describes the order in which workers are requested and indicates how subsequent workers are selected based on the acceptance or rejection of previous workers. For example, the server may first request worker 1; if worker 1 accepts, the next request is sent to worker 2; otherwise, the next request is sent to worker 3. A request flow is represented as a binary decision tree, where each node corresponds to the state at the moment a request is issued: (U, b, R) where $U \subseteq T$: set of completed tasks so far, $b \in [0, B]$: remaining budget, $R \subseteq W$: set of workers already requested.

The request flow is defined as: $f = (V, E, r)$ where V is the set of nodes, E is the set of edges, and $r \in V$ is the root node (initial state). Each node $v \in V$ represents a state (U, b, R, w_i). Each edge $e = (v \rightarrow v')$ represents a request to worker w_i and the acceptance result $a_i \in \{0, 1\}$, where: $a_i = 1$ means acceptance, $a_i = 0$ means rejection. The acceptance probability is represented as: $P(a_i = 1) = q_i$

The state transition is defined as follows: If $a_i = 1$ (accepted): $v' = (U \cup S_i, b - c_i, R \cup \{w_i\})$, If $a_i = 0$ (rejected): $v' = (U, b, R \cup \{w_i\})$

A request flow terminates at a leaf node vl when it satisfies one of two stopping conditions. First, under the Budget Rule, the system halts if no remaining worker can be hired within the available budget; formally, this occurs when $b < \min\_\{w_i \notin R\}$

c_i . Second, under the Marginal Coverage Rule, the flow also terminates when no remaining worker can contribute additional task coverage, namely, when every unrequested worker w_i satisfies $S_i \subseteq U$ indicating that their coverage set is already fully contained in the set of covered tasks.

Objective Functions

We define two objective functions: expected coverage and expected cost. Expected coverage represents the expected number of completed tasks when following a request flow. Coverage is defined as the number of completed tasks normalized by |T|.

Higher coverage is desirable. Expected cost is the expected amount of money paid to workers when following the request flow. Lower expected cost is desirable.

Let γ be a path from the root r to any leaf v_l in request flow f, and let Γ(f) denote the set of all possible paths. Then the expected coverage is defined as: E[Cov(f)] = Σ_{γ ∈ Γ(f)} P(γ) · |U(v_l)|. Similarly, the expected cost is: E[Cost(f)] = Σ_{γ ∈ Γ(f)} P(γ) · (B − b(v_l)). The probability of path γ is: P(γ) = (Π_{ i ∈ Accept(γ)} q_i) · (Π_{j ∈ Reject(γ)} (1 − q_j)), Accept(γ) is the set of workers who accept on path γ, Reject(γ) is the set of workers who reject on path γ.

Output

The output of the problem is the request flow that maximizes expected coverage. We denote the optimal request flow by f_{opt}, defined as: f_{opt} = argmax_{f ∈ F} E[Cov(f)]. If multiple request flows achieve the same expected coverage, we select the one with minimum expected cost.

PROPOSED METHODS

This chapter presents algorithms for constructing request flows based on the input defined in the previous chapter. Before describing the algorithms in detail, we provide an overview of the methods used in this study and their respective roles. The algorithms evaluated in this study can be categorized into three types according to their characteristics and objectives: exact methods, greedy methods, and randomized methods.

Exact methods guarantee the computation of an optimal solution with respect to the defined objective function. In this study, we employ an exhaustive search method that enumerates all possible request flows. Although exact methods always find the optimal solution, their search space grows extremely large, making them impractical for large-scale instances. We therefore use the optimal solution obtained by the exact method as a benchmark to evaluate the performance of the greedy and randomized approaches.

Greedy methods make the locally optimal choice at each step. In this study, a greedy method selects the worker with the highest score computed for the current state of the system. Because of their relatively low computational cost compared to exact methods, greedy methods can be applied to larger problem instances.

Randomized methods select solutions probabilistically. Here, we use a method that selects workers at random when constructing request flows. This method provides a baseline for evaluating algorithmic performance and represents a lower bound on expected solution quality. The following sections describe the algorithms, their pseudocode, and their computational complexity.

Exhaustive Search

This section describes the exhaustive search method, which enumerates all possible request flows and identifies the optimal one. Exhaustive search evaluates every possible request flow over the worker set W and selects the flow that maximizes expected coverage.

Greedy

While exhaustive search can be used to obtain the optimal request flow, its computational cost grows exponentially with the number of workers, making it impractical for problems of realistic size. To address this issue, we propose greedy algorithms (Figure 2) that can compute high-quality solutions within practical running time. The greedy methods presented in this study operate iteratively as follows. Given a state (U, b, R) at the moment of issuing a request, the algorithm considers the set of candidate workers $W_1 = W - R$ and selects the worker w* that maximizes a predefined score function:

w* = argmax (Score(w | U, b, R)), subject to cost(w) ≤ b, w ∈ W_1

This study introduces four greedy algorithms (Marginal Coverage Gain (MCG) Greedy, Cost-Normalized Marginal Coverage Gain (CN-MCG) Greedy, Expected Marginal Coverage Gain (EMCG) Greedy, Cost-Normalized Expected Marginal Coverage Gain (CN-EMCG) Greedy). All four methods share the same flow-construction procedure; they differ only in

their score functions. In other words, the only difference among the algorithms lies in how they compute the score used to determine which worker should be requested next. Let S_W denote the task set proposed by worker w, p_W the acceptance probability, and c_W the associated cost. Given a state (U, b, R), each greedy algorithm evaluates a candidate worker by computing a score that reflects (i) the marginal coverage, that is, the number of additional tasks the worker can newly cover, (ii) the worker's acceptance probability, and (iii) the cost required to hire the worker. MCG-Greedy considers only the marginal coverage and assigns to each worker the score $| S_W \setminus U |$. CN-MCG-Greedy incorporates the cost factor by normalizing this marginal coverage with respect to the hiring cost, assigning the score $| S_W \setminus U | / c_W$. EMCG-Greedy integrates the acceptance probability into the evaluation and scores workers by $p_W \cdot | S_W \setminus U |$, thereby prioritizing workers who are more likely to accept the request while contributing larger coverage. Finally, CN-EMCG-Greedy combines the effects of marginal coverage, acceptance probability, and cost, computing the score $p_W \cdot | S_W \setminus U | / c_W$, which favors workers who offer high marginal coverage with high acceptance probability relative to their cost.

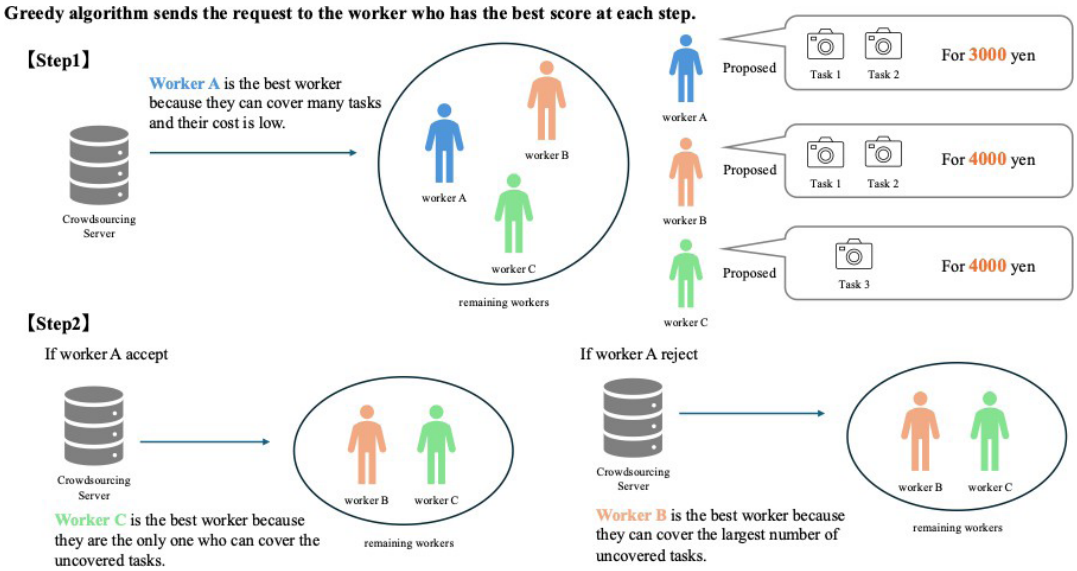

FIGURE 2. Greedy Algorithm

Randomized Method

The randomized method selects workers without using any score function. Instead, it uniformly selects one worker at random from the set of workers R who have not yet been requested. Unlike the greedy methods proposed in this study, the randomized method does not evaluate workers based on their cost c_W, acceptance probability p_W, or proposed task set S_W. Therefore, it serves as a baseline representing the lower bound of algorithmic performance. In the randomized method, the algorithm first enumerates the workers who satisfy both the budget rule and the marginal coverage rule at the time a request is to be issued. Then, it randomly selects one worker uniformly from this set and sends a request to the selected worker. The system state is updated according to the worker's acceptance or rejection outcome. The algorithm repeats this process (enumerating feasible workers and selecting one uniformly at random) until no worker satisfies the rules.

EXPERIMENT

Experimental Overview

The experiments in this study consist of two major components: (1) performance variation of request flows, and (2) algorithm performance comparison. The first experiment analyzes how the design of a request flow influences the expected coverage. Specifically, we compare the request flow with the highest expected coverage and the flow with the lowest expected coverage, and measure the difference between them. The second experiment evaluates which algorithms are able to construct request flows with high expected coverage. We use synthetic data to evaluate both the variation in request flow performance and the performance of the algorithms. Synthetic data is used because it allows the independent manipulation of factors that affect expected coverage. With real-world data, factors such as the task set, worker set, worker proposals (task sets, costs, and acceptance probabilities), and budget cannot

be adjusted independently, making it difficult to isolate the effect of each factor. By generating synthetic data under controlled parameter settings, we ensure that each factor can be independently studied. In this study, we analyze how each factor influences performance variation and algorithm performance by varying only the parameter of interest while fixing all other parameters. The factors examined include budget, task redundancy, cost distribution, acceptance probability distribution, number of tasks, and number of workers. We describe each factor and the parameters controlling it below.

Budget: The budget determines the maximum amount of money available for payments to workers. A smaller budget restricts the number of workers that can be requested. We control the budget using the *budget ratio* α. Let α be the budget ratio and let $C_{all} = \Sigma_i c_i$ denote the total cost required to request all workers. The budget B is defined as: $B = \alpha \times C_{all}$. The domain of α is [0, 1], where values closer to 1 correspond to larger available budgets.

Task Redundancy: Task redundancy quantifies the extent to which workers propose overlapping task sets. Higher redundancy implies that multiple workers propose the same tasks. Instead of defining redundancy directly, we control it through the number of workers per task N, which is the primary factor generating redundancy. If each task is independently proposed by N randomly selected workers, the relationship between task redundancy D and N is: $D = |T| \times (N / |W|)^2$. Given fixed numbers of workers |W| and tasks |T|, adjusting N allows direct control of D. The domain of N is [1,|W|], with values closer to |W| producing higher redundancy.

Cost: Cost represent the payment required for a worker to complete their proposed tasks. We assume that worker costs follow either a normal distribution or a log-normal distribution, and we adjust distribution parameters to control their variability.

Normal Distribution: The cost range is [min, max], and we define the midpoint: $m = (\min + \max) / 2$ as the mean of the normal distribution. To examine the effect of cost variability, we vary the standard deviation σ. The baseline standard deviation σ_0 is defined as: $\sigma_0 = (\max - \min) / 6$ reflecting that 99.7% of values in a normal distribution lie within ±3σ of the mean. We then define: $\sigma = s \times \sigma_0$ where s is a scaling factor.

Log-normal Distribution: For the log-normal distribution Lognormal(μ, σ), the mean is: $\exp(\mu + \sigma^2 / 2) = m$ Thus, μ is set to: $\mu = \log(m) - \sigma^2 / 2$ We vary σ to modify the skewness of the distribution while keeping the mean fixed. Larger σ values produce heavier right tails, representing environments with a few high-cost workers. We evaluate how the skewness (controlled by σ) influences performance variation and algorithmic behavior.

Acceptance Probability: Acceptance probability represents the likelihood that a worker accepts a request and completes their tasks .We assume that acceptance probabilities follow either a normal distribution or a beta distribution.

Normal Distribution: Since acceptance probabilities range from [0, 1], we use: min = 0, max = 1, mean = m = 0.5.The baseline standard deviation is: $\sigma_0 = (\max - \min) / 6 = 1 / 6$ and $\sigma = s \times \sigma_0$ As with costs, we vary σ to examine how acceptance probability variability affects performance.

Beta Distribution: For Beta(α, β), the mean is: $\text{mean} = \alpha / (\alpha + \beta)$ We fix α and vary β to control skewness. Smaller β produces right-skewed distributions (a few workers with high acceptance probabilities), while larger β produces left-skewed distributions (most workers have low acceptance probabilities). Thus, β controls both direction and concentration of skewness. We evaluate how variations in β affect performance variation and algorithm performance.

Number of tasks |T|: the size of the task set.

Number of workers |W|: the size of the worker set.

Experiment 1: Analysis of Performance Variation in Request Flows

Experiment 1 aims to quantify how each experimental variable (budget ratio, task redundancy, cost distribution, acceptance probability distribution, number of tasks, number of workers) affects the expected coverage of request flows. For each problem instance, we measure the performance variation, defined as the difference between the best and worst expected coverage across all flows.

Performance variation depends strongly on instance characteristics such as small budgets or high task redundancy. For a problem instance I_s and the set of all request flows F, performance variation Δ(I_s, F) is: $\Delta(I\_s, F) = \max\_\{F_i \in F\} E[\text{Cov}(F_i)] - \min\_\{F_i \in F\} E[\text{Cov}(F_i)]$. A large Δ(I_s, F) indicates that the choice of request flow has significant impact, and that flow optimization is important. For each parameter setting, we generate 100 problem instances and compute the performance variation for each. The average across the 100 instances is used as the performance variation for that parameter setting. This approach reduces the influence of instance-specific randomness. Parameter values for Experiment 1 are listed in Table X, with defaults in bold.

Experiment 2: Algorithm Performance Comparison

Experiment 2 evaluates five algorithms (Random method, Marginal Coverage Gain Greedy, Expected Marginal Coverage Gain Greedy, Cost-Normalized Marginal Coverage Gain Greedy, Cost-Normalized Expected Marginal Coverage Gain Greedy)

Three evaluation criteria are considered: We evaluate the algorithms using three criteria. First, relative performance is defined as the ratio of the expected coverage achieved by a given algorithm to that of the optimal solution. Second, robustness is assessed by computing the standard deviation of this relative performance across 100 independent instances. Finally, running time refers to the average computation time required to construct a request flow, where the running time of Exhaustive Search is normalized to 1, and the values for the other algorithms are reported relative to this baseline. As in Experiment 1, we vary only one parameter at a time while keeping all others fixed at their default values. For each parameter setting, we generate 100 instances, run all five algorithms on every instance, and compute both the relative performance and the running time. The reported results are obtained by averaging these metrics over the 100 instances. Experiments were conducted on a machine with: CPU: 3.8 GHz 8-core Intel Core i7, Memory: 64 GB DDR4 (2667 MHz).

Result

Analysis of Performance Variation in Request Flows

We analyze how the performance gap between request flows changes under different experimental parameters (Figure 3). First, with respect to the budget ratio, the gap becomes largest when the budget ratio is 0.5, while it is almost zero at 0.1 and 0.9. This is because differences in ordering matter only when the budget is moderate. When the budget is very small, only a few workers can be requested in the first place; when the budget is large, almost all workers can be requested. In both cases, the influence of the request flow is limited. Next, regarding the number of workers per task, the performance gap is largest when each task has only one worker, and it becomes smaller as more workers per task are available. As the number of workers per task increases, the sets of tasks proposed by workers become more similar, reducing the impact of differences in worker preferences across flows.

For the cost distribution, the performance gap widens when the standard deviation of a normal distribution is large, or when a log-normal distribution becomes highly skewed. With a normal distribution, greater variance increases the chance of spending the budget early when assigning expensive workers, which amplifies the gap between flows. With a log-normal distribution, a stronger skew introduces extremely high-cost workers. Flows that prioritize such workers differ more from those that do not, leading to larger coverage differences.

A similar trend appears for the acceptance-probability distribution. Under a normal distribution, a larger standard deviation increases the spread of acceptance probabilities among workers, which strengthens the effect of flow ordering. Under a beta distribution, when β is small, a few high-probability workers appear among many low-probability ones. Because the request order strongly determines whether these rare high-probability workers are requested, the flows show larger performance differences.

Finally, for the number of tasks, the performance gap is largest when there are 10 tasks and becomes smaller as the number of tasks increases. Since the number of workers and workers per task is fixed, adding more tasks increases task overlap. Workers then tend to propose similar task sets, and differences caused by the request flow become less pronounced.

Overall, the results show that performance differences between request flows become more visible when workers' proposals (costs, acceptance probabilities, and proposed task sets) are diverse. Conversely, when these proposals become more homogeneous, the impact of the request flow diminishes.

Algorithm Performance Comparison

We present the results on the performance differences among the algorithms. Overall, the experiments show that MCG-Greedy and CN-EMCG-Greedy consistently achieved the highest expected coverage across most parameter settings, outperforming the other algorithms in a stable manner. This indicates that greedy selection based on marginal gains provides robust performance under a wide range of conditions. Several settings, however, exhibited distinctive behaviors. When the cost distribution is highly skewed toward expensive workers, as in the log-normal distribution (Table 4), CN-EMCG-Greedy performed better than MCG-Greedy. The reason is that the cost-normalized method selects workers with high cost-effectiveness, whereas MCG-Greedy does not consider cost and may choose very expensive workers early, rapidly depleting the budget and reducing the number of workers that can subsequently be requested. As a result, its expected coverage becomes lower. Moreover, EMCG-Greedy, which also ignores cost, showed a

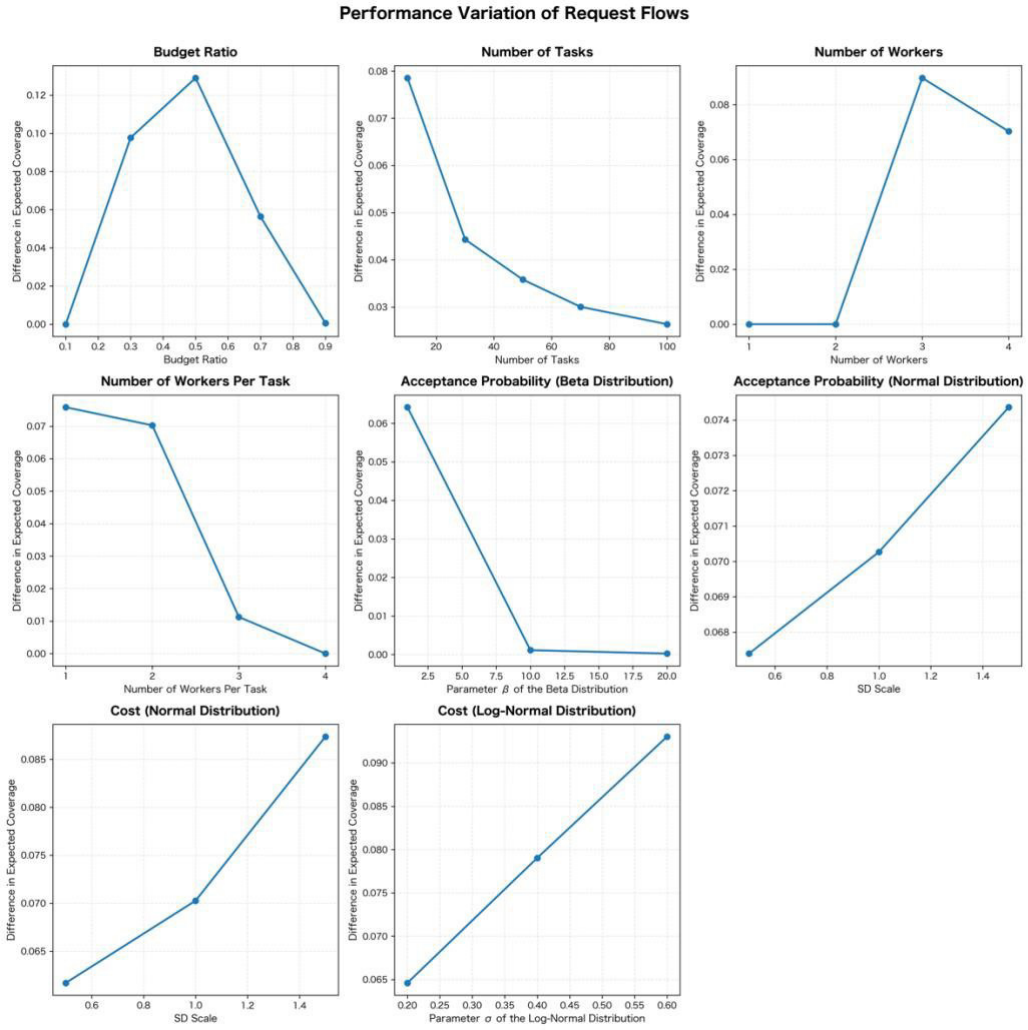

FIGURE 3. Performance Variation of Request of Flows

substantial performance drop as the skewness toward high costs increased. These observations suggest that incorporating cost into the selection criteria is essential when worker costs are highly skewed.

A second observation concerns the acceptance-probability distribution. When this distribution is heavily skewed toward low values, as in the beta distribution with a small β (Table 6), the relative performance changed: MCG-Greedy achieved the highest performance, followed by EMCG-Greedy and then CN-EMCG-Greedy. In all other settings, EMCG-Greedy generally performed worse than the other algorithms, but under this strong skew, MCG-Greedy performed comparably to the cost-aware method. This suggests that, when most workers have very low acceptance probabilities, prioritizing workers with higher acceptance probability becomes an effective strategy.

A third finding is that as the number of workers per task increases, the task sets proposed by workers become more homogeneous (Table 2). Consequently, the marginal coverage gain becomes nearly identical across workers, and performance differences between algorithms shrink rapidly. This shows that the influence of algorithm choice weakens when heterogeneity among workers is small.

Regarding computation time (Figure 4), CN-EMCG-Greedy was the fastest overall, followed by MCG-Greedy. When the variance in worker costs was large, the algorithms more frequently terminated early due to the selection of highly expensive workers, which reduced the search space and shortened computation time.

In summary, MCG-Greedy and CN-EMCG-Greedy demonstrated high and stable performance across a wide range of settings. At the same time, in environments with strong skew in worker costs or acceptance probabilities, incorporating these factors into the selection strategy provides notable performance benefits. These findings offer practical guidance for choosing appropriate request strategies in real-world spatial crowdsourcing systems.

TABLE 1. Performance and Robustness by Budget Ratio

Budget Ratio	0.2	0.4	0.6	0.8
Randomized Method	0.7160 ± 0.4493	0.8616 ± 0.1122	0.9516 ± 0.0425	0.9890 ± 0.0149
MCG-Greedy	0.7200 ± 0.4513	0.9986 ± 0.0084	0.9979 ± 0.0061	0.9996 ± 0.0023
CN-MCG-Greedy	0.7179 ± 0.4503	0.9083 ± 0.1002	0.9642 ± 0.0349	0.9933 ± 0.0123
EMCG-Greedy	0.7197 ± 0.4511	0.9547 ± 0.0569	0.9888 ± 0.0203	0.9976 ± 0.0069
CN-EMCG-Greedy	0.7200 ± 0.4513	0.9976 ± 0.0104	0.9975 ± 0.0066	0.9996 ± 0.0023

TABLE 2. Performance and Robustness by Number of Workers Per Task

Number of Workers Per Task	1	2	3	4
Randomized Method	0.9121 ± 0.0779	0.9516 ± 0.0425	0.9923 ± 0.0195	1.0000 ± 0.0000
MCG-Greedy	0.9996 ± 0.0041	0.9979 ± 0.0061	0.9997 ± 0.0022	1.0000 ± 0.0000
CN-MCG-Greedy	0.9320 ± 0.0874	0.9642 ± 0.0349	0.9953 ± 0.0112	1.0000 ± 0.0000
EMCG-Greedy	0.9939 ± 0.0127	0.9888 ± 0.0203	0.9969 ± 0.0146	1.0000 ± 0.0000
CN-EMCG-Greedy	0.9996 ± 0.0041	0.9975 ± 0.0066	0.9997 ± 0.0022	1.0000 ± 0.0000

TABLE 3. Performance and Robustness by SD Scale (Cost: Normal Distribution)

SD Scale	0.5	1	2
Randomized Method	0.9610 ± 0.0318	0.9516 ± 0.0425	0.9345 ± 0.0650
MCG-Greedy	0.9990 ± 0.0033	0.9979 ± 0.0061	0.9964 ± 0.0107
CN-MCG-Greedy	0.9688 ± 0.0304	0.9642 ± 0.0349	0.9598 ± 0.0479
EMCG-Greedy	0.9916 ± 0.0114	0.9888 ± 0.0203	0.9707 ± 0.0543
CN-EMCG-Greedy	0.9958 ± 0.0066	0.9975 ± 0.0066	0.9973 ± 0.0063

TABLE 4. Performance and Robustness by σ (Cost: Log-Normal Distribution)

Parameter σ of the Log-Normal Distribution	0.2	0.4	0.6
Randomized Method	0.9576 ± 0.0341	0.9377 ± 0.0586	0.9287 ± 0.0761
MCG-Greedy	0.9983 ± 0.0057	0.9965 ± 0.0111	0.9942 ± 0.0199
CN-MCG-Greedy	0.9679 ± 0.0314	0.9628 ± 0.0417	0.9623 ± 0.0484
EMCG-Greedy	0.9898 ± 0.0171	0.9721 ± 0.0531	0.9578 ± 0.0686
CN-EMCG-Greedy	0.9965 ± 0.0076	0.9976 ± 0.0064	0.9935 ± 0.0175

TABLE 5. Performance and Robustness by SD Scale (Acceptance Probability: Normal Distribution)

SD Scale	0.5	1.0	2.0
Randomized Method	0.9536 ± 0.0359	0.9516 ± 0.0425	0.9460 ± 0.0648
MCG-Greedy	0.9984 ± 0.0042	0.9979 ± 0.0061	0.9974 ± 0.0101
CN-MCG-Greedy	0.9657 ± 0.0306	0.9642 ± 0.0349	0.9614 ± 0.0485
EMCG-Greedy	0.9948 ± 0.0134	0.9888 ± 0.0203	0.9813 ± 0.0355
CN-EMCG-Greedy	0.9977 ± 0.0050	0.9975 ± 0.0066	0.9913 ± 0.0236

TABLE 6. Performance and Robustness by β (Acceptance Probability: Beta Distribution)

Parameter β of the Beta Distribution	1.0	10.0	20.0
Randomized Method	0.9614 ± 0.0436	0.9978 ± 0.0043	0.9991 ± 0.0020
MCG-Greedy	0.9983 ± 0.0069	1.0000 ± 0.0002	1.0000 ± 0.0000
CN-MCG-Greedy	0.9721 ± 0.0387	0.9982 ± 0.0057	0.9991 ± 0.0033
EMCG-Greedy	0.9820 ± 0.0253	0.9982 ± 0.0058	0.9993 ± 0.0026
CN-EMCG-Greedy	0.9810 ± 0.0267	0.9981 ± 0.0061	0.9993 ± 0.0026

TABLE 7. Performance and Robustness by Number of Tasks

Number of Tasks	10	30	50	70	100
Randomized Method	0.9472 ± 0.0469	0.9691 ± 0.0260	0.9737 ± 0.0203	0.9774 ± 0.0194	0.9808 ± 0.0182
MCG-Greedy	0.9970 ± 0.0076	0.9987 ± 0.0030	0.9982 ± 0.0072	0.9987 ± 0.0040	0.9986 ± 0.0040
CN-MCG-Greedy	0.9607 ± 0.0384	0.9768 ± 0.0215	0.9787 ± 0.0178	0.9813 ± 0.0157	0.9839 ± 0.0142
EMCG-Greedy	0.9921 ± 0.0133	0.9844 ± 0.0263	0.9861 ± 0.0177	0.9847 ± 0.0222	0.9862 ± 0.0237
CN-EMCG-Greedy	0.9965 ± 0.0088	0.9980 ± 0.0050	0.9974 ± 0.0077	0.9981 ± 0.0048	0.9983 ± 0.0040

TABLE 8. Performance and Robustness by Number of Workers

Number of Workers	1	2	3	4
Randomized Method	0.0000 ± 0.0000	1.0000 ± 0.0000	0.9306 ± 0.0808	0.9516 ± 0.0425
MCG-Greedy	0.0000 ± 0.0000	1.0000 ± 0.0000	0.9958 ± 0.0137	0.9979 ± 0.0061
CN-MCG-Greedy	0.0000 ± 0.0000	1.0000 ± 0.0000	0.9491 ± 0.0640	0.9642 ± 0.0349
EMCG-Greedy	0.0000 ± 0.0000	1.0000 ± 0.0000	0.9683 ± 0.0436	0.9888 ± 0.0203
CN-EMCG-Greedy	0.0000 ± 0.0000	1.0000 ± 0.0000	0.9919 ± 0.0195	0.9975 ± 0.0066

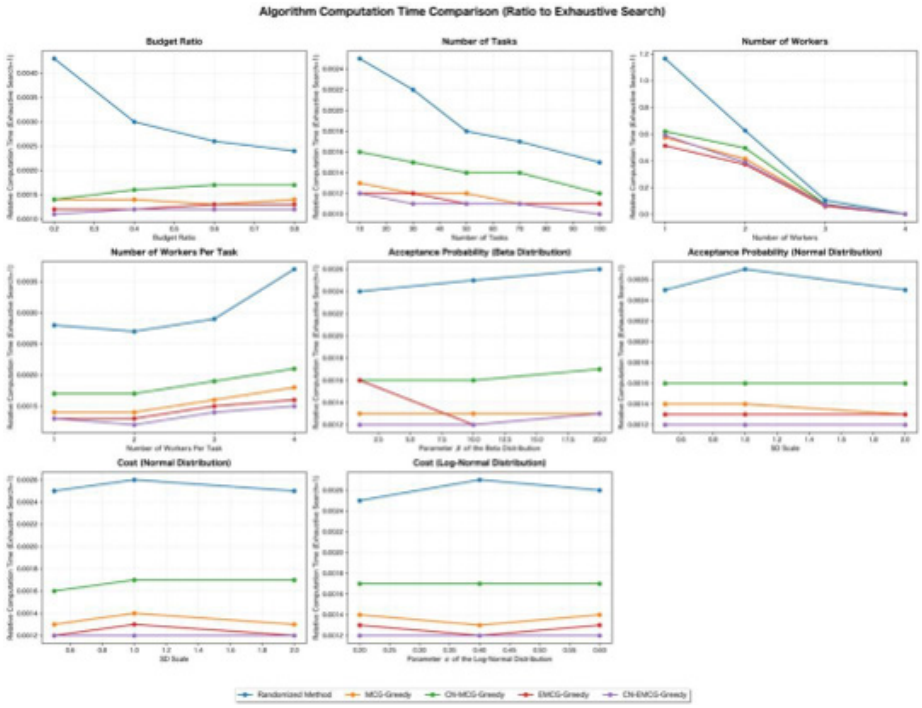

FIGURE 4. Algorithm Computation Time Comparison

CONCLUSION

This study proposed a new framework, called a request flow, for maximizing the number of completed tasks in Spatial Crowdsourcing (SC), and demonstrated its effectiveness. A request flow is a sequential assignment framework in which the next worker to request is chosen based on the acceptance result of the previous request. This structure makes it possible to model worker behaviors that cannot be handled by traditional batch assignment methods, such as situations in which workers propose their own tasks and costs, or situations in which workers may reject a request.

The experimental results revealed two main findings. First, the design of the request flow can change the number of completed tasks by up to approximately 10%. Second, an optimal or near-optimal request flow can be obtained efficiently: greedy algorithms such as MCG-Greedy and CN-EMCG-Greedy achieve around 90% of the optimal performance within practical computation time.

Future work includes extending the problem to the multi-objective setting of simultaneously optimizing both task completion and cost, conducting a more detailed analysis of interactions among experimental variables, and validating the approach using real-world datasets. These directions will further enhance the practicality and generality of the proposed request flow framework.

ACKNOWLEDGEMENTS

The author thanks to the advisors for their guidance and all lab members for their support. This work was partially supported by JSPS KAKENHI (JP22H00508) and JST CREST (Grant Number JPMJCR22M2), to which the author expresses sincere gratitude.

REFERENCES

Amazon Mechanical Turk. (2025, 11 27). Amazon Mechanical Turk. https://www.mturk.com Cen Chen, S.-F. C. (2014). Traccs: a framework for trajectory-aware coordinated urban crowd-sourcing. In Proceedings of the AAAI Conference on Human Computation and Crowdsourcing, Vol. 2, pp. 30–40.

Chen, L. Z. (2017). Maximizing acceptance in rejection-aware spatial crowd- sourcing. IEEE Transactions on Knowledge and Data Engineering, Vol. 29, No. 9, pp. 1943–1956.

Chunyan Miao, H. Y. (2016). Balancing quality and budget considerations in mobile crowdsourcing. Decision Support Systems, Vol. 90, pp. 56–64.

Curry, U. U. (2014). A multi-armed bandit approach to online spatial task assignment. In 2014 IEEE 11th Intl Conf on Ubiquitous Intelligence and Com-puting and 2014 IEEE 11th Intl Conf on Autonomic and Trusted Computing and 2014 IEEE 14th Intl Conf on Scalable Computing and Communications and Its Associated Workshops, pp. 212–219. IEEE.

Curry, U. u. (2016). Efficient task assignment for spatial crowdsourcing: A combinatorial fractional optimization approach with semi-bandit learning. Expert Systems with Applications, Vol. 58, pp. 36–56.

Didi. (2025, 11 27). Didi. https://didimobility.co.jp

Dingxiong Deng, C. S. (2013). Maximizing the number of worker's self-selected tasks in spatial crowdsourcing. In Proceedings of the 21st acm sigspatial international conference on advances in geographic information systems, pp. 324–333.

Florian Alt, A. S. (2010). Location-based crowdsourcing: extending crowdsourcing to the real world. In Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries, pp. 13–22.

Gigwalk. (2025, 11 27). Gigwalk. https://www.gigwalk.com

GrubHub. (2025, 11 27). GrubHub. https://www.grubhub.com

Hien To, C. S. (2015). A server-assigned spatial crowdsourcing framework. ACM Transactions on Spatial Algorithms and Systems (TSAS), Vol. 1,No. 1, pp. 1–28.

Howe, J. (2009). Crowdsourcing: Why the power of the crowd is driving the future of business.

Instacart. (2025, 11 27). Instacart. https://www.instacart.com

John C Tang, M. C.-W. (2011). Reflecting on the darpa red balloon challenge. Communications of the ACM, Vol. 54, No. 4, pp. 78–85.

Lam, T. S. (2022). Two-stage auction mechanism for long-term participation in crowdsourcing. IEEE Transactions on Computational Social Sys- tems, Vol. 10, No. 3, pp. 855–868.

Leyla Kazemi, C. S. (2013). Geotrucrowd: trustworthy query answer- ing with spatial crowdsourcing. In Proceedings of the 21st acm sigspatial international conference on advances in geographic information systems, pp. 314–323.

Lingyu Zhang, T. H. (2017). A taxi order dispatch model based on combinato- rial optimization. In Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, pp. 2151–2159.

Liwei Deng, Y. Z. (2024). Task recommendation in spatial crowdsourcing: A trade-off between diversity and coverage. In 2024 IEEE 40th International Conference on Data Engineering (ICDE), pp. 276–288.

Long Tran-Thanh, T. D. (2014). Budgetfix: budget limited crowdsourcing for interdependent task allocation with quality guarantees.

Luan Tran, H. T. (2018). A real-time framework for task assignment in hyperlocal spatial crowdsourcing. ACM Transactions on Intelligent Systems and Technology (TIST), Vol. 9, No. 3, pp. 1–26.

Mohammad H Vahidnia, F. H. (2020). Crowdsource mapping of target buildings in hazard: The utilization of smartphone technologies and geographic services. Applied Geomatics, Vol. 12, No. 1, pp. 3–14.

OpenStreetMap. (2025, 11 27). OpenStreetMap. https://www.openstreetmap.org

Peng Cheng, X. L. (2016). Task assignment on multi-skill oriented spatial crowdsourcing. IEEE Transactions on Knowledge and Data Engineering, Vol. 28, No. 8, pp. 2201–2215.

Peng Cheng, X. L. (2017). Prediction-based task assignment in spatial crowdsourcing. In 2017 IEEE 33rd International Conference on Data Engineering (ICDE), pp. 997–1008. IEEE.

Postmates. (2025, 11 27). Postmates. https://postmates.com

Shahabi, L. K. (2012). Geocrowd: enabling query answering with spatial crowdsourcing. In Proceedings of the 20th international conference on advances in geographic information systems, pp. 189–198.

Sindlinger, T. S. (2010). Crowdsourcing: Why the power of the crowd is driving the future of business.

TaskRabbit. (2025, 11 27). TaskRabbit. エラー! ハイパーリンクの参照に誤りがあります。

Tianshu Song, Y. T. (2017). Trichromatic online matching in real-time spatial crowdsourcing. In 2017 IEEE 33rd International Conference on Data Engineering (ICDE), pp. 1009–1020. IEEE.

Uber. (2025, 11 27). Uber. https://www.uber.com

UberEats. (2025, 11 27). UberEats. https://www.ubereats.com

Waze. (2025, 11 27). Waze. https://www.waze.com/

Xiaofeng Gao, S. C. (2020). Mab-based reinforced worker selection framework for budgeted spatial crowdsensing. Transactions on Knowledge and Data Engineering, Vol. 34, No. 3, pp. 1303–1316.

Xinglin Zhang, Z. Y. (2016). On reliable task assignment for spatial crowdsourcing. IEEE Transactions on Emerging Topics in Computing, Vol. 7, No. 1, pp. 174–186.

Xu, J.-X. L. (2020). Budget-aware online task assignment in spatial crowdsourcing. World Wide Web, Vol. 23, No. 1, pp. 289–311.

Yan Zhao, J. X. (2019). Preference-aware task assignment in spatial crowdsourcing. In Proceedings of the AAAI conference on artificial intelligence, Vol. 33, pp. 2629–2636.

Yan Zhao, K. Z. (2019). Destination-aware task assignment in spatial crowdsourcing: A worker decomposition approach. IEEE Transactions on Knowledge and Data Engineering, Vol. 32, No. 12, pp. 2336–2350.

Ying Hu, Y. W. (2018). An incentive mechanism in mobile crowdsourcing based on multi-attribute reverse auctions. Sensors, Vol. 18, No. 10, p. 3453.

Yongxin Tong, J. S. (2016). Online minimum matching in real-time spatial data: experiments and analysis. Proceedings of the VLDB Endowment, Vol. 9, No. 12, pp. 1053–1064.

Yongxin Tong, J. S. (2016). Online mobile micro-task allocation in spatial crowdsourcing. In 2016 IEEE 32Nd international con- ference on data engineering (ICDE), pp. 49–60. IEEE.

Yongxin Tong, L. W. (2018). Dynamic pricing in spatial crowdsourcing: A matching-based approach. In Proceedings of the 2018 international conference on management of data, pp. 773–788.

Yongxin Tong, Z. Z. (2020). Spatial crowdsourcing: a survey. The VLDB Journal, Vol. 29, No. 1, pp. 217–250.

Yu Li, M. L. (2015). Oriented online route recommendation for spatial crowdsourcing task workers. In International Symposium on Spatial and Temporal Databases, pp. 137–156. Springer.

Yunhui Li, L. C. (2021). Tasc-madm: Task assignment in spatial crowdsourcing based on multiattribute decision-making. Security and Communication Networks, Vol. 2021, No. 1, p. 5448397.

Yataro Fujinaga*
University of Tsukuba,
Tsukuba, Japan.

*Corresponding author: yataro.fujinaga.2023b@gmail.com