Multi-Objective Task Assignment in Crowdsourcing

Xiao Chen
Department of Computer Science, Texas State University, San Marcos, TX 78666
Email: xc10@txstate.edu

***Abstract*—Crowdsourcing coordinates a large group of online workers to perform small tasks published by requesters on crowdsourcing platforms. While task assignment problems within these platforms have been extensively studied, multi-objective task assignment remains an area with limited exploration. Specifically, existing research on multi-objective optimization problems has primarily focused on two objectives in the context of spatial crowdsourcing. In this paper, we address this gap by investigating a constrained three-objective task assignment in a general crowdsourcing environment. Our study establishes three key objectives from the perspective of requestors: maximize the amount of work finished; minimize the total cost; and minimize the variance among requestors' finished work under the limited capabilities of workers. To solve this multi-objective problem, we first adopt the Particle Swarm Optimization (PSO) algorithm to handle the multi-objective task assignment. Next, inspired by the solution to the knapsack problem and the fair queueing algorithm in network scheduling, we design a heuristic algorithm (HA). Finally, we incorporate the results from HA into the initial swarm of PSO to create a hybrid algorithm, HYB. HYB leverages the strengths of both approaches: PSO's exploration ability and HA's directness. We conduct simulations to compare the performance of the three algorithms. Our results demonstrate that HYB generates the best outcomes, HA performs moderately, while PSO lags behind. We conclude that leveraging heuristic insights alongside optimization techniques enhances both PSO and HA, especially given the large combinatorial nature of the multi-objective task assignment problem.**

***Index Terms*—crowdsourcing, heuristic, multi-objective, particle swarm optimization, task assignment**

I. Introduction

Crowdsourcing [2] coordinates large online groups to perform small tasks that solve problems beyond individual capabilities. It is used in voting, information sharing, gaming, creative systems [21], and mobile crowd sensing (MCS) [9]. Key components include requesters who publish tasks, workers who complete them, and platforms like Amazon Mechanical Turk [1] that manage interactions.

One of the critical challenges in crowdsourcing is the task assignment problem. Task assignment involves the crowdsourcing platform assigning the most suitable crowd workers to tasks to achieve specific objectives. Ensuring a reasonable and effective task assignment is essential for enhancing the service quality and efficiency of crowdsourcing systems. While task assignment problems are well-studied in crowdsourcing platforms [6], [7], [10], [12], [14], [17], our focus in this paper lies on multi-objective task assignment problems from the perspective of requesters - a topic that has received less attention. Although some papers have explored multi-objective optimization problems [11], [18], [19], [20], most of them are limited to two objectives. Notably, [4] addressed three objectives, but their work was specific to spatial crowdsourcing. In contrast, our research investigates a three-objective task assignment problem in a general crowdsourcing context. Additionally, we tackle a constrained optimization problem in which we consider worker ability limits.

The key differences and contributions of our work compared to others are as follows:

- Problem Formulation: We formulate a constrained multi-objective task assignment problem from the requesters' perspective. Our three objectives are to maximize the amount of finished work, minimize the total cost, and balance requestors' finished work under the limited capabilities of workers.
- Algorithm Approach: Initially, we adopt the Particle Swarm Optimization (PSO) algorithm to solve the problem. Subsequently, we propose a heuristic algorithm (HA). Finally, we combine both to create a hybrid algorithm (HYB).
- Simulations: We conduct simulations to compare these methods and draw meaningful conclusions.

The remainder of this paper is structured as follows: Section II reviews the related work. Section III defines a constrained multi-objective optimization problem. Section IV presents our first solution based on the Particle Swarm Optimization algorithm. Section V introduces a heuristic algorithm as our second solution. Section VI proposes a hybrid approach. Section VII presents simulation results comparing these methods. And the conclusion is provided in Section VIII.

II. Related Work

In this section, we review the related work and highlight the distinctions between our approach and that of others.

In their research, the authors of [11] proposed a multi-objective optimization problem related to spatial crowdsourcing (SC). Specifically, they focused on optimizing two objectives: travel costs and task reliability. To tackle this problem, they transformed it into a minimum-cost maximum-weight bipartite matching problem. Their approach involved introducing the distance reliability ratio (DRR), which relies on combinatorial fractional programming. Additionally, they extended their method by adapting two algorithms: a combinatorial multi-armed bandit model with semi-bandit learning to estimate worker reliability. Tran et al. [18] utilized a genetic algorithm to solve a multi-objective optimization (MOO) problem in hyper-local spatial crowdsourcing. Their objectives were to

maximize task coverage while minimizing the highest workload across workers, all within budget constraints. Wang et al. [19] investigated a spatial crowdsourcing multi-objective task allocation problem. Their goal was to search for a set of representative Pareto-optimal allocation solutions that maximize assigned task coverage while simultaneously minimizing incentive costs. They proposed effective heuristic methods, including multi-round linear weight optimization and enhanced multi-objective particle swarm optimization algorithms, to achieve adequate Pareto-optimal allocation. The authors of [20] studied a Multi-Objective Optimization Task Assignment problem and proposed a Weighted and Multi-Objective Particle Swarm Combination (WAMOPSC) algorithm to maximize both the platform's and crowd workers' utility. All of the papers mentioned above addressed two-objective optimization problems within the field of spatial crowdsourcing.

The only paper we found that addressed the three-objective optimization problem is [4]. In their study, the authors concentrated on task scheduling with three objectives: maximizing the number of completed tasks, minimizing total travel costs, and ensuring workload balance among workers. To tackle this problem, they devised an algorithm using particle swarm optimization and further improved it by incorporating a ranking strategy based on task entropy and execution duration.

Similar to the work in [4], this paper investigates a three-objective task assignment optimization problem. Unlike [4], our problem is defined in a general crowdsourcing environment. In addition, our goals are presented from the perspective of the requestors - an aspect that has received relatively little attention. Furthermore, our problem introduces constraints: each worker has a defined ability or capacity, and task assignments should not exceed an individual worker's ability.

III. Problem Formulation

In this section, we formulate a multi-objective task assignment optimization problem.

A. Problem Definition

In our model, we assume that there is a set of tasks represented as $\mathcal{T} = \{t_i\}$, which originate from a group of requestors $\mathcal{R} = \{r_q\}$. Each task i is described by a tuple of four elements: an id, a size s_i, a requestor associated with it, and an order number u representing its position among the tasks from that requestor. On the other hand, there exists a set of workers denoted as $\mathcal{W} = \{w_j\}$ who are actively seeking tasks. Each worker j is characterized by an id, a unit charge ξ_j, and an ability a_j to complete tasks. We assume that a task cannot be divided and can only be assigned to a single worker.

We introduce a binary-valued indicator matrix X, such that $X_{ji} = 1$ if task i is completed by worker j, and $X_{ji} = 0$ otherwise. From the perspective of the requestors, the problem is formally defined as follows:

$$\begin{aligned} \text{maximize} \quad & \sum_j \sum_i X_{ji} s_i \\ \text{minimize} \quad & \sum_j \sum_i X_{ji} s_i \xi_j \\ \text{minimize} \quad & Var([\sum_{i \in r} X_{ji} s_i, \text{ all } r \in \mathcal{R}]) \\ \text{subject to} \quad & \sum_j X_{ji} s_i \leq a_j, \forall j \end{aligned} \tag{1}$$

The first objective of the problem is to maximize the total amount of work completed by summing the sizes of all the completed tasks. The second one is to minimize the total cost of the finished tasks by considering the unit charge of each worker. And the third objective is to balance the finished work among the requestors by minimizing the variance of the finished work among the requestors. The constraint states that the total amount of work assigned to a worker should not exceed their ability.

B. Problem Hardness

Theorem 1: Our defined problem is NP-hard.

Proof. If we focus solely on the first objective of maximizing the total amount of finished work, our problem is closely related to the Multiple Knapsacks Problem (MKP) [5]. In MKP, we deal with disjoint subsets of items that must be distributed into different knapsacks. Each subset can be placed in a separate knapsack, and the goal is to find these disjoint subsets of items while maximizing the total value, all while adhering to the weight limits of each knapsack.

In our specific context, we assign disjoint subsets of tasks to workers to maximize the total amount of finished work. Each subset of tasks can be assigned to a different worker, as long as the total size of each subset does not exceed the corresponding worker's ability. Notably, MKP has been proven to be NP-hard [5]. Therefore, our problem, even with just one objective, falls into the NP-hard category. When we introduce two additional objectives, our problem becomes even more complex than MKP, reinforcing its NP-hardness. □

IV. Solution 1: Particle Swarm Optimization Algorithm (PSO)

To solve the defined multi-objective optimization problem, a useful tool is the Particle Swarm Optimization (PSO) algorithm [13]. PSO is a population-based optimization technique that finds the optimal solution by iteratively adjusting a swarm of particles. In the first solution, we will adapt PSO to our problem space to identify the optimal solution.

Here are the main steps and features of PSO:

1) Initialization:
   - Randomly generate a swarm S composed of a set of particles, with each particle a valid assignment of tasks to workers.
   - A valid assignment ensures that tasks assigned to workers can be completed within their abilities.

- If a task cannot find an appropriate worker, the corresponding worker is set to zero.

2) Features:
   - Velocity and Position Update:
     - Particles start with an initial velocity of zero, and their initial positions correspond to the initial assignment of tasks to workers.
     - Particles update their velocity and position to explore and exploit the search space.
     - These updates continue iteratively until convergence.
   - Fitness:
     - A measure of how good a particle is with respect to the optimization problem.
   - Local Best ($pbest_k$):
     - Represents the best solution (evaluated by fitness) achieved so far in particle k.
     - Each particle maintains its own local best solution.
   - Global Best ($gbest$):
     - Represents the best solution (evaluated by fitness) achieved by any particle in the entire swarm.

In short, PSO leverages the collective behavior of particles to efficiently search for optimal solutions. By balancing exploration and exploitation, it aims to find the best assignment of tasks to workers. The details of the adapted PSO algorithm for solving our problem are presented in Fig. 1.

Algorithm 1: Particle Swarm Optimization Algorithm (PSO)

Inputs: a list of workers and tasks, G (maximum generation allowed), particle number

Output: an assignment of tasks to workers meeting three objectives

1: $g = 0$; /* initialize generation counter */
2: Initialize random swarm $M(0)$ with each particle a valid assignment of each task to a worker;
3: Initialize the velocity of each particle v_k^0 to zero;
4: The initial position of each particle w_k^0 is the first assignment;
5: Evaluate each particle in $M(0)$ using formula (4);
6: Find $p_{(k,lb)}^0$ of each particle k and p_{gb}^0;
7: **while** $g < G$ **do**
8: **for** $k = 1$; $k \leq Particle\_num$; $k++$ **do**
9: Update velocity v_k^{g+1} using formula (2);
10: Update position w_k^{g+1} using formula (3);
11: **end for**
12: Evaluate each particle in $M(g)$ using formula (4);
13: Update $p_{(k,lb)}^g$ of each particle k and p_{gb}^g;
14: g = g + 1;
15: **end while**

Fig. 1. Particle Swarm Optimization Algorithm (PSO)

In Fig. 1, the PSO algorithm takes as inputs a list of tasks $\mathcal{T}$, a list of workers $\mathcal{W}$, the maximum generation G, and the particle number $Particle\_num$, and generates a valid task assignment to meet our three objectives. In Steps (1) to (6), we start by initializing the generation counter g to zero, creating a random swarm of particles, denoted as $M(0)$, in generation 0. Each particle represents a valid potential assignment of tasks to workers. We initialize the velocity of each particle to zero, and the initial position is the first assignment. We evaluate each particle in $M(0)$ using the fitness formula in (4). Then, we obtain the local best $p_{(k,lb)}^0$ of each particle k and the global best p_{gb}^0 for generation 0. In the main loop for g from Step (7) to Step (15), for each particle k in the swarm, we update its velocity v_k^{g+1} in generation $g+1$ using formula (2) and its position w_k^{g+1}, which is the updated assignment, using formula (3). After each particle in generation g has been processed, we update the local best $p_{(k,lb)}^g$ for each particle k based on its own experience, and revise the global best p_{gb}^g among all particles. The details of Steps (9), (10), and (4)/(12) are described below.

A. Velocity Update

In Step (9), the velocity of particle k in generation $g+1$ is updated as follows:

$$v_k^{g+1} = \phi v_k^g + c_1 r_1 (p_{(k,lb)}^g - w_k^g) + c_2 r_2 (p_{gb}^g - w_k^g) \quad (2)$$

In the formula, v_k^g is the velocity of particle k in generation g, w_k^g denotes the position of particle k, $p_{(k,lb)}^g$ represents the local best of the k-th particle in generation g, and p_{gb}^g corresponds to the global best. Besides these, in PSO, the parameters ϕ, c_1, and c_2 play crucial roles in determining the behavior of the algorithm. Parameter ϕ is the inertia weight, which controls the balance between exploration and exploitation during the optimization process. It influences how much a particle's current velocity contributes to its next position update. Parameter c_1 is the cognitive component, which represents a particle's confidence in its own best position (local best). It influences how much a particle's historical best position affects its movement. The social component c_2 reflects the influence of the global best position (swarm best) on a particle's movement, encouraging particles to move toward the global best position found by other particles.

B. Position Update

In Step (10), the position of particle k in generation $g+1$ is adjusted as follows:

$$w_k^{g+1} = w_k^g + v_k^{g+1} \quad (3)$$

Particle k's position w_k^{g+1} in generation $g+1$ depends on its position in generation g and its velocity in generation $g+1$. Along with the velocity update, PSO dynamically adjusts these parameters to strike a balance between exploration and exploitation, ultimately converging toward optimal solutions. In our context, if a position update causes a task assignment to exceed the maximum worker number $Worker\_num$ or fall below 1, we bring it within the valid range by assigning the task to worker $Worker\_num$ or worker 1, respectively.

C. Particle Evaluation (fitness)

In Steps (5) and (12), the fitness of a particle is evaluated by summing the three components that correspond to the three objectives in our optimization problem. Before combining them, we preprocess these components to ensure they are on the same scale and contribute in the same direction during the optimization process.

To transform the components to the same scale, we employ min-max normalization [16] to map the values into the range $[0, 1]$. The formula for min-max normalization is as follows:

$$Normalized\ Value = \frac{Original\ Value - Min\ Value}{Max\ Value - Min\ Value}$$

Our optimization problem's first objective is to maximize the amount of work finished. We denote the normalized amount of work finished as $Nwork\_finished$.

Our second objective is to minimize the overall cost. To achieve this, we first normalize the total cost and denote it as $Ntotal\_cost$. Then, we maximize $1 - Ntotal\_cost$ to ensure it aligns in the same direction during the optimization process.

Our third objective aims to balance the amount of finished work among the requestors. A common metric for measuring balance is the variance of the amount of work finished among the requestors. We minimize the variance to achieve a good balance. After calculating the variance, we normalize it and denote it as $Nvariance$. Additionally, we adjust the goal to maximize $1 - Nvariance$ to maintain consistency with the other two objectives.

After preprocessing, the fitness of a particle is determined as follows to evaluate how well it aligns with our objectives.

$$\begin{aligned} fitness = Nwork\_finished + (1 - Ntotal\_cost) \\ + (1 - Nvariance) \end{aligned} \tag{4}$$

V. Solution 2: Heuristic Algorithm (HA)

Our second solution to the defined problem is a heuristic algorithm (HA) illustrated in Fig. 3. The idea was inspired by the heuristic solution to the knapsack problem and fair queueing in networks. In Step (1), we start by ordering the workers based on the ratio of their ability to unit cost in non-increasing order. This helps maximize the amount of finished work while minimizing the overall cost. In Step (2), to achieve a balance in the amount of finished work among the requestors, we adopt the concept of the fair queuing algorithm from the quality of service in computer networks [15]. In our context, the workers represent resources. We create a queue for each requestor and place their tasks in these queues, as shown in Fig. 2. Fair queuing achieves fairness by preventing requestors with large tasks from consuming more resources than other requestors when a limited resource is shared. We calculate the finish time of each task using the formulas provided in (5). For simplicity, and without affecting our main focus, we assume that the arrival time $Arrive(u)_r$ of the first task of requestor r is 0. Once all finish times of the tasks are calculated, we order the tasks by their finish times in a non-decreasing order.

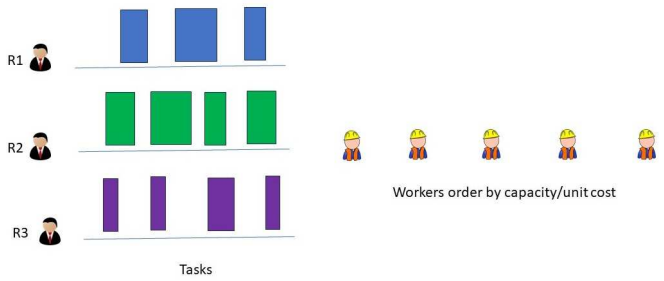


Fig. 2. Task assignment inspired by fair queuing

$$\begin{aligned} Arrive(u)_r &= \text{arrival time of } u\text{-th task of requestor } r \\ Size(u)_r &= \text{size of } u\text{-th task of requestor } r \\ Finish(u)_r &= max(Arrive(u)_r, Finish(u-1)_r) + Size(u)_r \end{aligned} \tag{5}$$

From Step (3) to (12), we assign each task to a worker. If task i is assigned to worker j, worker j must have sufficient ability to complete the task. Once assigned, worker j reduces his ability by the size of the task. If a worker is found, we break the internal loop, reorder the workers based on ability per unit cost, and proceed to the next task. If no worker can be found for a task, we assign the task to worker 0.

Algorithm 2: Heuristic Algorithm (HA)

Inputs: a list of workers and tasks
Output: an assignment of tasks to workers meeting three objectives

1: Order the workers in non-increasing order by ability/unit cost;
2: Order the tasks in non-decreasing order by finish times calculated using formulas in (5);
3: **for** $i = 1 : Task\_num$ **do**
4: **for** $j = 1 : Worker\_num$ **do**
5: **if** the size of task $i \leq$ worker j's ability **then**
6: Assign task i to worker j;
7: Deduct worker j's ability by the size of task i;
8: break;
9: **end if**
10: **end for**
11: Order the workers according to ability/unit cost;
12: **end for**

Fig. 3. Heuristic Algorithm (HA)

VI. Solution 3: Hybrid Algorithm combining PSO and HA (HYB))

Given a list of workers and tasks, there are numerous possible assignments. The PSO algorithm in Solution 1 starts from the initial particle positions for its exploration and exploitation dynamics. Proper initialization is crucial to achieving a balanced search that efficiently converges toward optimal solutions. The HA method in Solution 2 can directly produce a solution, but it lacks the exploration capability to find the optimal solution. To address the limitations of both

approaches, we propose a hybrid solution, HYB, that combines PSO and HA to optimally solve our defined problem.

In this hybrid approach, we use the assignment obtained from HA as the initial particle and combine it with other randomly generated particles to explore the search space. By starting with a good solution and continuing the exploration process, we aim to find an even better solution.

VII. Simulations

In this section, we conducted simulations using MATLAB to compare the methods we proposed above.

A. Algorithms Compared

We compared the following algorithms:

1) Particle Swarm Optimization Algorithm (PSO)
2) Heuristic Algorithm (HA)
3) Hybrid Algorithm combining PSO and HA (HYB)

B. Metrics

We employed the following metrics, each corresponding to one of the three objectives in our optimization problem:

1) Amount of Work Finished
2) Total Cost of Finished Work
3) Variance of the Amount of Finished Work among Requestors

In multi-objective optimization, a feasible solution that optimizes all objective functions simultaneously typically does not exist [3]. Here, we compare the assignments found by each of the three algorithms when the program terminates.

C. Settings

In our simulations, we randomly generated a list of workers and tasks. Each worker was associated with the following attributes: worker id, unit cost, and ability. The unit cost for each worker was randomly selected from the range $[1, 10]$, while the ability was chosen randomly from the range $[10, 30]$. On the other hand, each task possessed the following properties: task id, task size, its requestor, and its sequence number at the requestor. The task size was randomly drawn from the interval $[1, 10]$. The maximum task number of each requestor was set to 5.

Regarding the parameters for PSO and HYB, we set the particle number to 8. The inertial weight ϕ typically falls between 0.4 and 0.9. We used 0.72984, following [8]. The acceleration coefficients c_1 and c_2 were both set to 1.4.

D. Performance Evaluation

In our simulations, we explored two different numbers of requestors: 100 and 200. The number of workers was set to the product of the number of requestors and maximum task number divided by two. To compare the performance of the algorithms, we randomly generated 200 cases of workers and tasks. For each case, in PSO and HYB, we set the maximum number of generations G to start at 100 and increment by 50 until reaching 300. After running the algorithms, we calculated the average values for the three metrics.

The simulation results are presented in Figs. 4, 5, and 6. In Figs. 4(a) and 4(b), with 100 and 200 requestors, respectively, HYB achieved the highest value of finished work, followed by HA, and PSO performed the least effectively. Figs. 5(a) and 5(b) display the total cost of completing these tasks with 100 and 200 requestors. Here, HYB incurred the lowest cost, HA fell in the middle, and PSO had the highest cost. Lastly, the variance in the amount of finished work among the requestors is shown in Figs. 6(a) and 6(b). Once again, HYB exhibited the least variance, HA was intermediate, and PSO had the highest variance.

In summary, HYB outperforms the other two. It achieves the highest amount of finished work, the lowest cost, and the minimum variance. HA falls in the middle, while PSO performs the worst. Due to the explosive combinations of assignments, a brute-force strategy like PSO struggles to enumerate all possible solutions. The heuristic algorithm HA is a viable choice; its results are close to those of HYB, and it efficiently finds the solution without relying on generations. The superior performance of HYB demonstrates that adding good particles to the initial swarm improves both PSO and HA. Although HYB relies on generations for the optimal solution, its improvement in the three metrics justifies this approach.

VIII. Conclusion

In this paper, we have addressed a constrained three-objective task assignment optimization problem from the perspective of requestors in a general crowdsourcing environment. Our objectives have included maximizing the amount of finished work, minimizing the total cost, and reducing the variance of the amount of finished work among the requestors under the limited capabilities of workers. To tackle this problem, we have employed two distinct approaches: particle swarm optimization (PSO) and a heuristic algorithm (HA). Furthermore, we have proposed a hybrid approach, HYB, which combines PSO and HA. Through simulations, we have compared the performance of these three algorithms. The results have demonstrated that HYB achieves the best outcome in satisfying all three objectives, while HA falls in the middle and PSO performs the worst. Our conclusion highlights the importance of incorporating good particles from a heuristic algorithm into the swarm, as it enhances the performance of both PSO and HA. In the future, we will work on more multi-objective optimization problems in crowdsourcing.

References

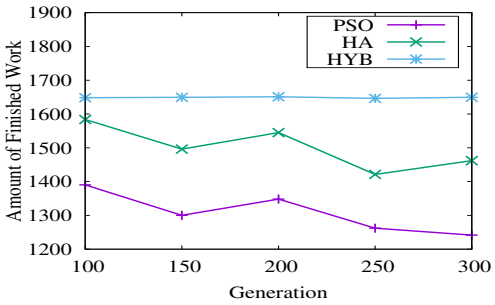
[1] Amazon mechanical turk. http://mturk.com.

[2] Crowdsourcing. https://en.wikipedia.org/wiki/Crowdsourcing.

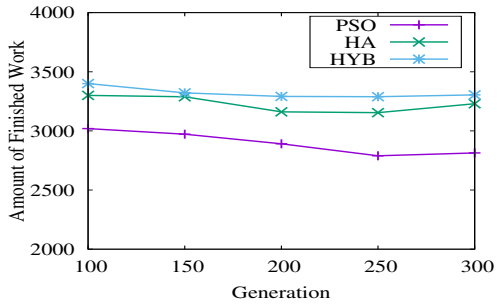
[3] Multi-objective optimization. https://en.wikipedia.org/wiki/Multi-objective_optimization.

[4] A. A. Alabbadi and M. F. Abulkhair. Multi-objective task scheduling optimization in spatial crowdsourcing. *Algorithms*, 14(3), 2021.

[5] V. Cacchiani, M. Iori, A. Locatelli, and S. Martello. Knapsack problems — an overview of recent advances. part ii: Multiple, multidimensional, and quadratic knapsack problems. *Computers & Operations Research*, 143:105693, 2022.

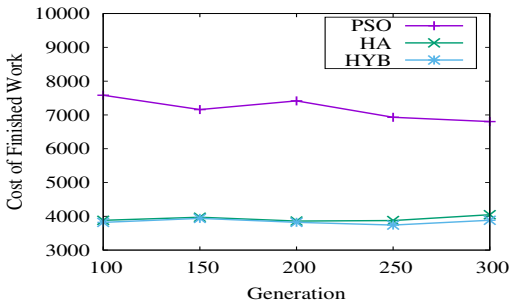


(a) Amount of finished work with 100 requestors

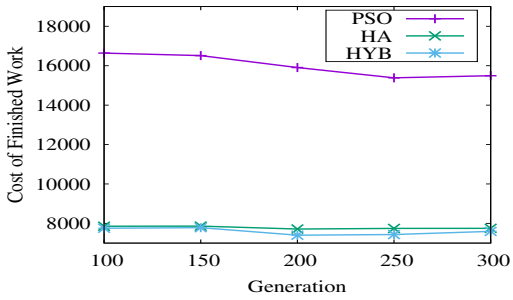


(b) Amount of finished work with 200 requestors

Fig. 4. Amount of finished work

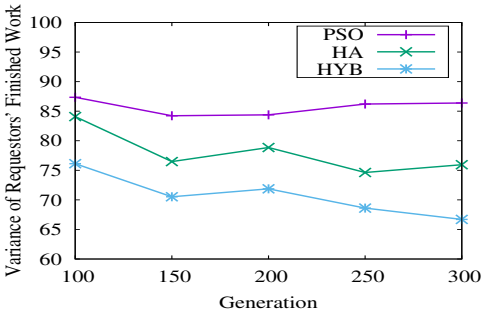


(a) Cost of finished work with 100 requestors

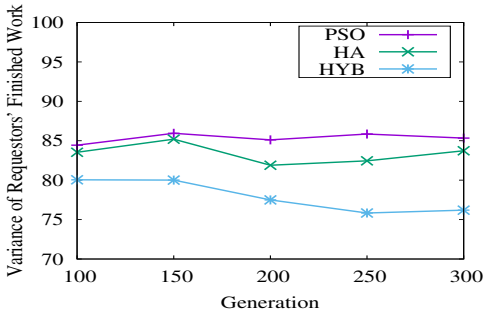


(b) Cost of finished work with 200 requestors

Fig. 5. Cost of finished work



(a) Variance of finished tasks with 100 requestors



(b) Variance of finished tasks with 200 requestors

Fig. 6. Variance of finished tasks

[6] P. Cheng, X. Lian, L. Chen, J. Han, and J. Zhao. Task assignment on multi-skill oriented spatial crowdsourcing. *IEEE Transactions on Knowledge and Data Engineering*, 28(8):2201–2215, 2016.

[7] D. Deng, C. Shahabi, and U. Demiryurek. Maximizing the number of worker's self-selected tasks in spatial crowdsourcing. In *21st SIGSPATIAL GIS*, pages 314—-323, 2013.

[8] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, 1995.

[9] R. Ganti, F. Ye, and H. Lei. Mobile crowdsensing: current state and future challenges . *IEEE Communication Magazine*, 49(11), 2011.

[10] J. P. Hanna, M. Albert, D. Chen, and P. Stone. Minimum cost matching for autonomous carsharing. In *IFAC-PapersOnLine*, volume 49, pages 254–259. 2016.

[11] U. Hassan and E. Curry. Efficient task assignment for spatial crowdsourcing: A combinatorial fractional optimization approach with semi-bandit learning. *Expert Systems with Applications*, 58:36–56, 2016.

[12] L. Kazemi and C. Shahabi. Geocrowd: enabling query answering with spatial crowdsourcing. In *21st SIGSPATIAL GIS*, pages 189–198, 2012.

[13] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995.

[14] Y. Li, M. L. Yiu, and W. Xu. Oriented online route recommendation for spatial crowdsourcing task workers. *Advances in Spatial and Temporal Databases*, pages 137–156, 2015.

[15] J. Nagle. On packet switches with infinite storage. *IEEE Transactions on Communications*, 35(4):435–438, 1987.

[16] M. Shantal, Z. Othmana, and A. Bakar. A novel approach for data feature weighting using correlation coefficients and min–max normalization. *Symmetry*, 15(12), 2023.

[17] H. To, C. Shahabi, and L. Kazemi. A server-assigned spatial crowdsourcing framework. *ACM Transactions on Spatial Algorithms and Systems*, 1(1), 2015.

[18] L. V. Tran, H. To, L. Fan, and C. Shahabi. A real-time framework for task assignment in hyperlocal spatial crowdsourcing. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 9:1 – 26, 2017.

[19] L. Wang, Z. W. Yu, Q. Han, B. Guo, and H. Y. Xiong. Multi-objective optimization based allocation of heterogeneous spatial crowdsourcing tasks. *IEEE Transactions on Mobile Computing*, 17(7):1637–1650, 2018.

[20] S. N. Wu, Y. J. Wang, and X. R. Tong. Multi-objective task assignment for maximizing social welfare in spatio-temporal crowdsourcing. *China Communications*, 18(11):11–25, 2021.

[21] M. Yuen, I. King, and K. Leung. A survey of crowdsourcing systems. In *IEEE PASSAT and IEEE SocialCom*, 2011.