A Comparison of New Swarm Task Allocation Algorithms in Unknown Environments with Varying Task Density

Grace Cai
MIT
Cambridge, US
gracecai@mit.edu

Noble Harasha
MIT
Cambridge, US
nharasha@mit.edu

Nancy Lynch
MIT
Cambridge, US
lynch@mit.edu

ABSTRACT

Task allocation is an important problem for robot swarms to solve, allowing agents to reduce task completion time by performing tasks in a distributed fashion. Existing task allocation algorithms often assume prior knowledge of task location and demand or fail to consider the effects of the geometric distribution of tasks on the completion time and communication cost of the algorithms. In this paper, we examine an environment where agents must explore and discover tasks with positive demand and successfully assign themselves to complete all such tasks. We first provide a new discrete general model for modeling swarms. Operating within this theoretical framework, we propose two new task allocation algorithms for initially unknown environments – one based on N-site selection and the other on virtual pheromones. We analyze each algorithm separately and also evaluate the effectiveness of the two algorithms in dense vs. sparse task distributions. Compared to the Levy walk, which has been theorized to be optimal for foraging, our virtual pheromone inspired algorithm is much faster in sparse to medium task densities but is communication and agent intensive. Our site selection inspired algorithm also outperforms Levy walk in sparse task densities and is a less resource-intensive option than our virtual pheromone algorithm for this case. Because the performance of both algorithms relative to random walk is dependent on task density, our results shed light on how task density is important in choosing a task allocation algorithm in initially unknown environments.

KEYWORDS

Task Allocation; Robot Swarms; Geometric Swarm Algorithms

ACM Reference Format:
Grace Cai, Noble Harasha, and Nancy Lynch. 2023. A Comparison of New Swarm Task Allocation Algorithms in Unknown Environments with Varying Task Density. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 9 pages.

1 INTRODUCTION

Robot swarms are simple, distributed units that are able to work together to achieve emergent collective behaviours [11]. We contribute a general, theoretical framework to model these swarms, which can be leveraged, as in this work, to implement various swarm algorithms. Swarm algorithms often draw inspiration from swarms in nature such as birds, ants, and bees [20, 27, 30]. Swarm algorithms provide a scalable and fault-tolerant solution to problems such as search-and-rescue [9] and environmental monitoring [10]. One of the most well-studied swarm problems is task allocation [13], which aims to assign agents to tasks in an optimal manner. Here, a *task* refers to an abstract notion: a task is simply a location of interest in the environment which requires some action(s) by agents. This could be a food item in foraging, survivors in search-and-rescue, mines for robots to defuse, and more.

Many classes of task allocation algorithms assume that task locations and demand for agents are known, and try to optimize an assignment of agents to tasks [3, 4, 35]. However, in many applications, such as finding and defusing mines [32], task information is not initially known. Algorithms which do consider task allocation in unknown environments [5, 16] run limited testing on the effects of task density, an important factor in algorithm performance.

In this paper, we consider the problem of assigning agents to tasks with positive demand in an initially unknown environment. We assume each agent can only be assigned to one task. Within this setup, we contribute two new algorithms and compare them to the Levy Walk (RW), which is used in nature for foraging [29]. We also show how task density makes our different algorithms better suited towards different task environments.

The first algorithm, our house hunting task allocation algorithm (HHTA), is inspired by swarm house-hunting models [28]. While the house hunting problem aims for agents to agree on one of many locations in the environment, the task allocation problem aims for agents to split themselves proportionally to task demand amongst all tasks in the environment. In our HHTA algorithm, agents use their starting location as a home base that they can return to after discovering tasks in the environment. The home base functions as a central point of communication and allows for agents to recruit each other to do tasks, serving the same function as the home nest in many swarm house hunting algorithms.

The second algorithm is our propagation-based algorithm (PROP), which uses a regular grid of cheap, simple agents to propagate task demand information outwards to neighboring propagator agents. We assign a separate type of agent with more advanced computing powers to read the information and use it to probabilistically decide which task to head towards. The propagation of task demand information via cheap agents is inspired by virtual pheromones [1, 2], a commonly used nature-inspired technique in swarm algorithms.

By comparing both algorithms to the Levy flight, we show that it is harder for PROP to do well with very dense tasks, as a large influx of propagated information can confuse agents. Our other algorithm, HHTA, does worse when tasks are mid to high density because inter-agent communication about tasks is not worth it compared to a random walk, which is highly likely to encounter tasks quickly.

Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), A. Ricci, W. Yeoh, N. Agmon, B. An (eds.), May 29 – June 2, 2023, London, United Kingdom.

However, it does better than RW when tasks are sparse as the cost of communication about task location is justified when tasks are harder to find. It is also less resource intensive compared to PROP. We also evaluate the effects of varying individual parameters within several task densities in order to better understand our new algorithms.

Our results demonstrate how different task allocation algorithms do well in environments with different task density and invite further examination on the performance of other task allocation algorithms in different types of task environments.

Section 2 provides the inspiration for our two proposed algorithms, explaining house hunting and virtual pheromones in further depth. Section 3 describes our general formal model and then dives into our two specific algorithms, presented under this model. Our simulation results and comparison between the two new algorithms in sparse and dense task environments can be found in Section 4. Section 5 discusses our results, and Section 6 concludes the paper and provides ideas for future work. The full version of this paper can be found at [6], containing pseudocode and other additional aids such as example simulations and a reference table of parameter notation. The full simulation code can be found at [15].

2 BACKGROUND

Task allocation is a well studied problem and has been classified into many subproblems. Per the taxonomy defined in [13] our task allocation problem is of the single-task agents, multi-robot tasks variety, which means that agents can only do one task at a time, but tasks may require multiple agents.

When task demands and locations are known, this problem becomes a coalition formation problem, where we wish to form agents into groups that are best suited to do each task. This problem can be thought of as a set partitioning problem [13], and adaptations to distributed swarms have been proposed [4, 35].

Other strategies for when tasks are fixed at known locations model tasks as a graph where agents can travel between edges [3, 14, 19]. These algorithms optimize for a flow rate between edges in the graph so that agents can satisfy all task demands quickly. Another strategy in this case, based on Optimal Mass Transport [33], is to treat the tasks with demands as sinks and the tasks with agents as sources in a min cost flow problem. However, both strategies require prior knowledge of task locations.

Our problem differs from coalition formation and the graph-based task allocation problems because we are assuming that agents have no initial knowledge of task location or demands. In this case, we want to discover tasks and communicate information about them as quick as possible so that agents can satisfy all task demands.

One solution to task allocation in an environment with unknown tasks is to have agents form local clusters and run Optimal Mass Transport locally [36]. Other task allocation algorithms, such as auction-based algorithms, perform a similar type of agent clustering to assign tasks [17]. Our two algorithms by contrast are fully distributed and computationally simple, without the need for grouping to locally run a complex centralized algorithm. This allows us to save the time needed to form agent clusters and allows agents to be cheaper to implement due to low computation cost.

2.1 Levy Flight

The Levy flight is a random walk that has been observed in foraging animals and adapted to swarm algorithms [12, 29] as well. The Levy Flight has shown to be an optimal forgaing algorithm, which is very relevant to the situation in which task locations and demands are unknown. As such, we will be using this random walk as a baseline to compare against for our two new algorithms.

2.2 House Hunting

Several ant species engage in a house-hunting behaviour when their home nest is destroyed [26, 27]. First, ants explore nearby for nest sites. If a site is found, the ant waits a period of time inversely proportional to the site quality before returning to the home nest to lead others to the new nest. This process of recruitment is known as forward tandem running (FTR). Once the encounter rate of other ants in the candidate nest reaches a critical threshold known as the quorum threshold, ants switch to carrying members of the colony to the new nest. This carrying behaviour is 3 times faster than FTR and accelerates the move to the new site [27].

Ant house hunting has inspired the corresponding swarm problem of N-site selection [34], where agents must choose the highest quality site from N initially unknown candidates. One common N-site selection model has agents transition between four main states: Uncommitted Interactive, Uncommitted Latent, Favoring Interactive, and Favoring Latent [28]. Some works also include a fifth Committed state [7, 8, 21]. In this type of model, Uncommitted Interactive agents explore the arena for new sites, while Uncommitted Latent agents stay in the home nest. Once an Uncommitted Interactive agent discovers a site, it can decide to favor the site. Favoring agents can be interactive, meaning they return to the home nest to recruit other favoring agents, or latent, meaning they stay in their favored site to build up quorum. Lastly, if agents detect a sufficient number of others in a new candidate site, they can transition into the committed state to finalize their decision.

Task allocation can be thought of as an extension to the house hunting problem, where instead of trying to send all agents to one location, we want to send agents to multiple locations according to the demand at each one. This idea has been used in Berman [3] and Halasz's [14] work to develop task allocation algorithms for a known graph of tasks where agents can traverse along the edges. We extend this idea further by using inspiration from site selection algorithms to develop our novel HHTA algorithm, in which, unlike [3, 14], task locations are initially unknown. In HHTA, agents use a home nest which functions as a location for recruiting other agents to tasks and communicating with other agents. The four main states of the HHTA algorithm share parallels to the Uncommitted Interactive, Uncommitted Latent, Favoring Active, and Committed states described above which are further explained in Section 3.3.

2.3 Virtual Pheromones and Potential Fields

Ants leave pheromones in their environment when foraging to guide other ants to any discovered food sources [1]. This strategy of leaving information in the environment has inspired swarms to implement virtual pheromones (pheromones represented by computational data instead of chemical signals). For example, [2] used

physically deployable beacons that robots could leave in the environment to store information in, [22] simulated pheromone trails by leveraging depots to store target-rich locations (pheromone waypoints) found by other robots, and [23] set up a virtual pheromone approach with a pre-deployed network of beacons that acted as a grid of locations to leave information in. One cheap way to implement virtual pheromones is using wireless sensor motes to store and propagate information [31].

Pheromones are frequently used in conjunction with potential fields or particle swarm optimization techniques. Potential field algorithms model objects in the environment as either positive charges or negative charges, with agents experiencing attraction or repulsion from the objects based on the electric force between them. Particle swarm optimization [25] follows a similar physics approach, except the attractive and repulsive forces were based on springs as opposed to charges. These techniques are employed in navigation tasks, where potential fields and pheromones can work together to guide robots around obstacles and towards a target in space [24]. Pheromones are also employed in foraging tasks to help robots efficiently find what they are foraging for [18].

We apply the ideas of virtual pheromones in our novel PROP algorithm, which uses simple mote-like agents to leave task information in the environment. Task-performing robots use this information when searching for tasks in the task allocation process. The use of virtual pheromones allows us to easily notify task-performing robots of nearby tasks. We also use potential fields as inspiration for how a robot's motion should be influenced when it learns of multiple potential tasks through pheromones in the environment. Robots are more attracted to tasks with higher demand and tasks that are closer to their current location, so tasks can be thought of like charges which robots can feel the force of.

3 MODEL

We first describe our new discrete general model for modeling swarms. Then we discuss the individual restrictions, parameters, and agent algorithms needed for task allocation.

3.1 General Model

We assume a finite set R of agents, with a state set SR of potential states. Agents move on a discrete rectangular grid of size $M \times N$, formally modelled as directed graph $G = (V, E)$ with $|V| = MN$. Edges are bidirectional, and we also include a self-loop at each vertex. Vertices are indexed as (x, y), where $0 \leq x \leq M-1$, $0 \leq y \leq N-1$. Each vertex also has a state set SV of potential states.

Local Configurations: A *local configuration* $C'(v)$ captures the contents of vertex v. It is a triple $(sv, myagents, srmap)$, where $sv \in SV$ is the vertex state of v, $myagents \subseteq R$ is the set of agents at v, and $srmap : myagents \rightarrow SR$ assigns an agent state to each agent at v.

Local Transitions: The transition of a vertex v may be influenced by the local configurations of nearby vertices. We define an **influence radius** I, which is the same for all vertices, to mean that vertex indexed at (x, y) is influenced by all valid vertices $\{(a, b) | a \in [x-I, x+I], b \in [y-I, y+I]\}$, where a and b are integers. We can use this influence radius to create a local mapping M_v from local coordinates to the neighboring local configurations. For a vertex v at location (x, y), we produce M_v such that $M_v(a, b) \rightarrow C'(w)$ where w is the vertex located at $(x+a, y+b)$ and $-I < a, b < I$. This influence radius is representative of a sensing and communication radius. Agents can use all information from vertices within the influence radius to make decisions.

We have a local transition function δ, which maps all the information associated with a vertex and its influence radius at one time to new information that can be associated with the vertex and the agents at that vertex for the following time.

Formally, for a vertex v, δ probabilistically maps M_v to a quadruple of the form $(sv_1, myagents, srmap_1, dirmap_1)$, where $sv_1 \in SV$ is the new state of the vertex, $srmap_1 : myagents \rightarrow SR$ is the new agent state mapping for agents at the vertex, and $dirmap_1 : myagents \rightarrow \{R, L, U, D, S\}$ gives directions of motion for agents currently at the vertex. Note that R, L, U, and D mean right, left, up, and down respectively, and S means to stay at the vertex. The local transition function δ is further broken down into two phases as follows.

Phase One: Each agent in vertex v uses the same transition function α, which probabilistically maps the agent's state $sr \in SR$, location (x, y), and the mapping M_v to a new suggested vertex state sv', agent state sr', and direction of motion $d \in \{R, L, U, D, S\}$. We can think of α as an agent state machine model.

Phase Two: Since agents may suggest conflicting new vertex states, a rule Q is used to select one final vertex state. The rule also determines for each agent whether they may transition to state sr' and direction of motion d or whether they must stay at the same location with original state sr.

Probabilistic Execution: The system operates by probabilistically transitioning all vertices v for an infinite number of rounds. During each round, for each vertex v, we obtain the mapping M_v which contains the local configurations of all vertices in its influence radius. We then apply δ to M_v to transition vertex v and all agents at vertex v. For each vertex v we now have $(sv_v, myagents_v, srmap_v, dirmap_v)$ returned from δ.

For each v, we take $dirmap_v$, which specifies the direction of motion for each agent and use it to map all agents to their new vertices. For each vertex v, its new local configuration is just the new vertex state sv_v, the new set of agents at the vertex, and the $srmap$ mapping from agents to their new agent states.

3.2 Task Allocation Problem Definition

Consider T tasks $[t_0 \ldots t_{T-1}]$ arranged at a subset of vertices in our general model, with at most one task at each vertex. Specifically, the task locations can be described as $l = [(x_0, y_0), \ldots, (x_{T-1}, y_{T-1})]$, where $l_i = (x_i, y_i)$ is the vertex location of task t_i and $i \neq j \rightarrow l_i \neq l_j$ (each task has a distinct location). We wish to distribute agents among the tasks to achieve a certain distribution $a = [a_0, ..., a_{T-1}]$ where a_i represents the number of agents doing task i and $\sum a_i = kR \leq R$ (meaning k% of all agents is enough to complete all tasks).

We assume that when an agent senses a task within its influence radius, it is able to detect the demand of that task. Since agents can also detect how many agents are at the task, they can use this information to compute the *residual demand*, defined as the difference between the task demand and the number of agents already at the task. We denote the residual demand at task i by t_i^{rd}. We assume that the desired task distribution does not change over

time, and that the task is complex enough that each agent can only do one task over the course of the algorithm.

In order to represent tasks in both of our algorithms, the vertex state set SV contains the following variables: `is_task`, whether the vertex is a task; `demand`, the task demand if the vertex is a task; `residual_demand`, the residual demand if the vertex is a task; `task_location`, the x, y coordinates of the vertex if it is a task.

We go into more detail on the agent states and transitions for our two algorithms in Sections 3.3 and 3.4. One other detail to note about task allocation is that in phase two of δ, we reconcile conflicting proposed vertex states. This shows up in task allocation when multiple agents attempt to claim the same task. When this happens, if there are s agents trying to claim the task but only $rd < s$ residual demand, then only rd agents are allowed to transition their state to having claimed the task (these rd agents are chosen arbitrarily). Otherwise, if $rd > s$, all agents will be allowed to claim the task.

3.3 House Hunting Task Allocation Algorithm

In our house-hunting inspired algorithm (HHTA), agents start out at a square home location with lower left corner (x_h^1, y_h^1) and upper right corner (x_h^2, y_h^2). Call the set of home vertices $\mathcal{H}$. We assume that $\forall i, l_i \notin \mathcal{H}$, meaning no tasks are located at the home location. In HHTA, the vertex state set SV needs the additional variable `is_home`, indicating whether the vertex is a home vertex or not.

Agents can be in one of four core states: Home (H), Exploring (E), Recruiting (R), or Committed (C). Home agents wait in home nest for news of tasks. Exploring agents explore the arena for tasks. Home agents have a $P_E = \frac{L * P_e}{1 - P_e}$ chance of converting to exploring agents, and exploring agents have a $P_H = L$ chance of converting to home agents, where L is defined as $1/(M + N)$ and P_e is the expected fraction of exploring agents. The transitions between H and E agents indicate that agents are expected to explore for $M + N$ time steps (enough to reach the corners of the grid) before returning home. It also ensures that the expected fraction of E agents out of the total number of E and H agents is P_e. The factor of L is inspired by house hunting algorithms, where L is defined as the inverse of the average site round trip so that exploring agents will have enough time to reach candidate sites before returning home.

An exploring agent has a P_{t_i} chance of finding task i. Once it finds task i, it has a $c = \max(P_c, 1/t_i^{rd})$ chance of becoming a Committed agent, and a $1 - c$ chance of becoming a Recruiting agent. Here P_c is the base probability of committing, and $1/t_i^{rd}$ makes it so that at low residual demands, agents have a higher chance of committing to the task right away. If a task has residual demand 1, for instance, any agent which discovers it will commit to the task right away, completing the task instead of trying to recruit others for it.

Committed agents have fully committed to a task and stay at that task. The Committed state is similar to the Committed state in house hunting, where agents have decided on a new nest site and have moved to it. Recruiting agents head back to the home nest to tell Home agents about the task they have found. Agents recruiting for site i have a $1/t_i^{rd}$ chance to stop recruiting and become committed to task i. Recruiting agents have a r_m chance of sending a message to each agent within their influence radius at each time step, where r_m is the message rate. Therefore, a Home agent has an $P_{r_i} = I_{1-r_m}(R_{t_i} - 1, 2)$ chance of receiving at least

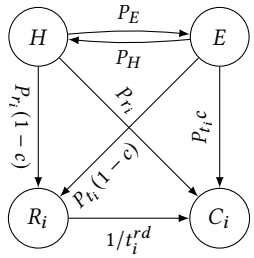

Figure 1: State model of the four core states. The subscript i denotes that an agent is recruiting for or committed to task i.

one recruiting message for task i. Here, R_{t_i} is the number of agents recruiting for task i that are within sensing radius, and I is the regularized incomplete beta function. If a Home agent receives a message from a recruiting agent, it has a P_c chance of committing to the task and heading towards it, and a $1 - P_c$ chance of recruiting for the task. Note that the residual demand information for C and R agents may become stale as more agents commit to tasks. A diagram of the transitions between these core states can be found in Figure 1.

In order to execute the core state transitions, the agent state set SR comprises of the following variables: `core_state`, which can be H, E, R or C; `id`, the agent id, taking on values from $0 \ldots |R| - 1$; `L`, defined as $L = 1/(M + N)$; `P_commit`, the probability P_c; `P_explore`, the probability P_e; `message_rate`, the message rate r_m; `angle`, the agent's current angle of travel; `starting_point`, a random walk parameter tracking where the agent started from; `travel_distance`, the length of the current leg of the random walk; `destination_task`, the agent's destination if they have just found a task or are headed towards their committed task; `home_destination`, the agent's destination if they are headed to a home vertex; `recruitment_task`, the task an agent is recruiting for; and `committed_task`, the task an agent has committed to.

The agent transition function α uses these state variables to implement the transitions between the four core states.

3.4 Task Propagation Algorithm

In our task propagation algorithm (PROP), we distinguish between two types of agents – MN propagators and F followers. Propagators are simple, mote-like agents. One of them is assigned to each vertex to allow vertices to propagate task information to each other. Followers are more advanced agents which are able to perform the tasks in the task allocation problem. Followers follow the signals left by propogators in order to find tasks.

Similarly to HHTA, all agents are initially deployed at a rectangular home location with lower left corner (x_h^1, y_h^1) and upper right corner (x_h^2, y_h^2). However, agents in PROP do not utilize this home location after starting the algorithm. First, all MN propagators travel to the vertex which they are assigned to, taking $\frac{M+N}{2}$ time for all agents to reach their assigned vertex.

Each propagator has an influence radius of 1 and also stores in their state a mapping $\mathcal{M}_T$ from task locations l_i to residual demands $t_i^{rd'}$, representing that they have heard that task i at location l_i has

residual demand $t_i^{rd'}$. After all propagators are in place, propagators that are at a task t_i spread the tuple $(t_i^{rd'}, l_i)$ to all other propagators in their influence radius. Every r_p time steps, a propagator takes all new task information (if it has new information it did not already propagate) it has received and spreads that information to all other propagators in its influence radius with the following conditions: information about task i can only be spread to agents whose assigned vertex v is located within the bounds $[x_i - I, x_i + I]$ for the x coordinate and $[y_i - I, y_i + I]$ for the y coordinate, and the Euclidian distance between t_i and v must be less than or equal to d_p. Here, r_p is the integer propagation timeout and d_p is the maximum propagation radius.

Because the residual demand of a task changes over time, the propagator at task i will have to send new information whenever the residual demand decreases. When a propagator which already has task information $\mathcal{M}_{\mathcal{T}}(l_i) \rightarrow t_i^{rd'}$ receives new information about a task $(t_i^{rd''}, l_i)$, it updates the task information for task i to be $\mathcal{M}_{\mathcal{T}}(l_i) \rightarrow min(t_i^{rd'}, t_i^{rd''})$ in order to have the most up-to-date information. Since the residual demand of a task is always decreasing as more and more agents join the task, we know the smaller residual demand is the more accurate one.

After all propagators have reached their assigned vertex, followers try to use the information of propagators in order to find tasks to head towards. At every time step, a follower first checks the vertices within its influence radius for a task with non-zero residual demand, and starts moving towards that task if it exists. If no task is found in its influence radius, a follower located at (x, y) looks at the propagator assigned to location (x, y) in order to get information about potential task locations it could head towards. It compiles all non-zero residual demands into the resulting mapping M_F, which maps from task locations l_i to residual demands $t_i^{rd'}$. If M_F is non-empty (there is at least one task location with non-zero residual demand) then the probability that a follower located at (x, y) heads towards task location $l_i \in M_F$ is:

$$\frac{\frac{M_F(l_i)}{L_2(l_i,(x,y))^2}}{\sum_{l_j \in D(M_F)} \frac{M_F(l_j)}{L_2(l_j,(x,y))^2}} \tag{1}$$

This means that the probability of a follower heading towards a task has an inverse square relationship with L_2 distance between the task location and the agent's location, and is also weighted by the residual demand of the task itself. This equation is determined so that agents are less likely to travel to tasks that are further away from them, but more likely to travel to a task if it has higher residual demand. If the mapping M_F is empty (the agent has no task information), it takes a random step in one direction $\{L, D, R, U\}$ (following a Levy flight random walk) in order to explore.

Once a follower agent reaches a task with non-zero residual demand, it stays there indefinitely, "completing the task" and decrementing the task's residual demand by one.

In order to execute the algorithm, the agent state set contains the following variables: `type`, the type of agent, which can be 'propagator' or 'follower' and `id`, the agent id, which takes on values from $0 \ldots |MN + F| - 1$. The following additional variables are in SR and are only used by propagator agents: `task_info`, the mapping $\mathcal{M}_{\mathcal{T}}$; `propagation_rate`, the propagation timeout r_p; and `propagation_ctr`, the number of rounds since an agent last propagated task information. Lastly, the variables in SR used only by follower agents are: `destination_task`, the agent's destination if they have just found a task or are headed towards their committed task; `committed_task`, the task an agent has committed to; `angle`, a random walk parameter denoting angle of travel; `starting_point`, a random walk parameter tracking where the agent started from; and `travel_distance`, the length of the current leg of the random walk.

The agent transition functions α for propagator and follower agents, respectively, use these state variables to implement the desired transitions at each time step.

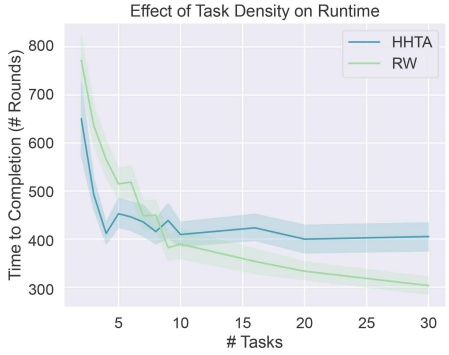

Figure 2: The effect of number of tasks on completion time for HHTA and RW

4 RESULTS

Our algorithms were tested in simulation [15] using Pygame on a grid of size $M = N = 50$, with a 3×3 home area in the center of the grid. All simulations were run using 100 task-performing agents and the total task demand summed to 80. In the trials for the HHTA algorithm, agents had an influence radius of 2. In the trials for the PROP algorithm, propagators had an influence radius of 1 and followers had an influence radius of 2.

For each set of trials, we evaluated task completion time, defined as the time necessary for the total residual demand to become 0. In subsections 4.1 and 4.2, we also measure the average number of messages sent per run per agent. For the HHTA algorithm, whenever a Home agent is notified of a task by a Recruiting agent, the Recruiting agent's message count is incremented. For the PROP algorithm, the message count is incremented when a propagator shares new task information with one of its neighbors. We do not track the message count for follower agents since it is a negligible portion of total messages.

4.1 Effects of Task Density on HHTA Performance

To examine the effects of task density on the HHTA algorithm's performance, we measured task completion time and average number of messages sent per agent for $T \in \{2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30\}$. For each value of T (the number of tasks), we ran 100 trials with $r_m = \frac{1}{6}$, $P_e = \frac{2}{3}$, $P_c = \frac{3}{10}$. Figure 2 shows the resulting average

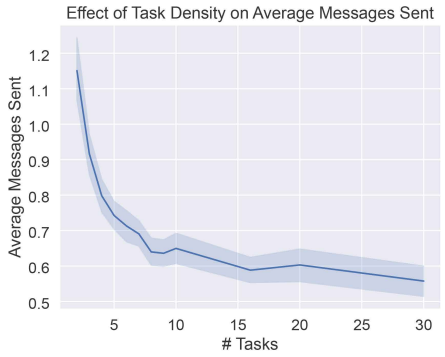

Figure 3: The effect of number of tasks on average messages sent per agent for HHTA

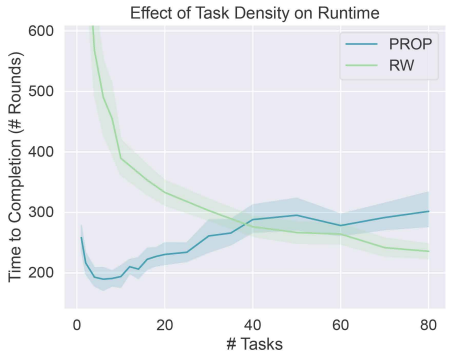

Figure 4: The effect of number of tasks on completion time for PROP and RW

task completion time for varying task densities. The HHTA algorithm outperforms the random walk by about 100 rounds in very sparse task setups when $T \leq 6$ and performs comparably when $7 \leq T \leq 10$, but for denser task setups, the cost of returning to the home nest to recruit others is too high compared to the random walk (Welch's T-test, p=0.05). We can approximate the area covered by detectable tasks as $\frac{T(2I+1)^2}{NM}$, where $(2I+1)^2$ is the size of the influence radius (in reality, the ratio would be a bit smaller as the detectable range for tasks can intersect). This means that for our choice of parameters, the HHTA algorithm outperforms the random walk when about 6% or less of the total task area has an immediately detectable task.

Figure 3 shows the average number of messages sent per agent for the HHTA algorithm. (Note that the random walk algorithm uses no communication). Note that on average, each agent sends less than 1.2 messages per round using HHTA. Note also that agents send less messages on average as density increases. Since the total task demand is fixed at 80, a larger number of tasks indicates less demand per task on average, making agents in the HHTA algorithm less likely to enter the task state (where messages are sent) and remain in it.

4.2 Effects of Task Density on PROP Performance

To examine the effects of task density on the PROP algorithm's performance, we once again measured task completion time and average number of messages sent per propagator agent for $T \in \{1, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 25, 30, 35, 40, 50, 60, 70, 80\}$. For each value of T (the number of tasks), we ran 20 trials with $r_p = 3$ and $d_p = 25$. Figure 4 shows the resulting average task completion time for varying task densities. The PROP algorithm outperforms the random walk significantly (Welch's T-test, p=0.05) in sparser task setups ($T \in [1, 25]$), with fewer tasks exaggerating this performance gap nonlinearly. In moderately dense task setups ($T \in [30, 60]$), the two algorithms' runtimes are comparable, and in our most dense task setups ($T \in [70, 80]$), the random walk begins to increasingly outperform the PROP algorithm to a significant extent (Welch's T-test, p=0.05). Intuitively, as the density of tasks in the environment increases, follower agents are more likely to find tasks in their influence radius (benefiting RW). Conversely, more tasks means more task information within each propagator agent, overloading and misguiding the follower agents during their decision process (harming PROP). After a certain point, too much propagated information results in that information declining in its specificity and thus usefulness.

Figure 5 shows the average number of messages sent per propagator agent per round . Note that on average, each propagator agent sends less than 1.3 messages to other agents per round; however, given the grid space's size, a large communication cost is still incurred as there are 2, 500 propagator agents. Regarding the effect of task density on these communication costs, agents generally send more messages as the number of tasks increases. When there are more tasks and thus more vertices close to tasks, it takes less time for most of the propagator agents to receive some information initially that they can begin propagating. Additionally, having more tasks means that *some* task's demand gets updated more often, resulting in there being new information (as messages) that needs to be propagated more often. This increasing trend becomes less dramatic at higher task densities, likely due to the fact that with enough tasks, the overlapping propagation radii all cover roughly the same area. As shown in Figure 4, recall that higher task densities result in higher completion times for the PROP algorithm. Therefore, higher task densities do not only result in agents sending more messages per round, but the total number of messages sent over an entire run increases even more dramatically along with the number of tasks.

4.3 Effects of P_c on HHTA Completion Time for Varying Task Density

We explored the effects of varying P_c on completion time for varying T. We ran 100 trials for each value of $P_c \in \{\frac{i}{10}, 0 \leq i \leq 9\}$ using $r_m = \frac{1}{6}$ and $P_e = \frac{2}{3}$. The results can be seen in Figure 6.

For $P_c \leq 0.7$, there was no significant difference between the completion times at different task densities (Welch's T-test, p=0.05). However, for $P_c \in \{0.8, 0.9\}$, the completion time for $T = 4$ is higher than the completion time for $T = 16$ (Welch's T-test, p=0.05). Our results show that only for large P_c do we see a significant difference in performance at different task densities. This makes sense, as a larger proportion of committing agents means agents mostly find tasks by discovering them independently, which is harder in the

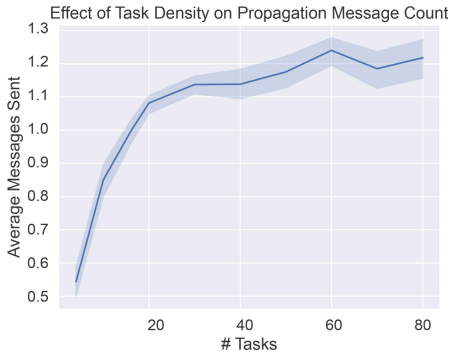

Figure 5: The effect of number of tasks on average messages sent per propagator agent per round (for PROP)

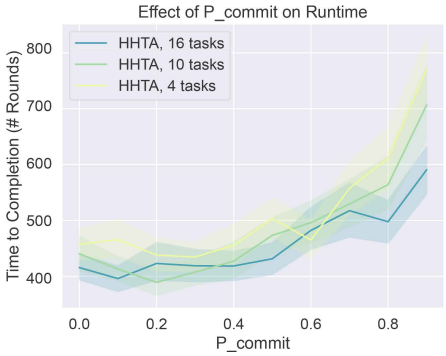

Figure 6: The effect of P_c on HHTA completion time

sparse case. We also note that from $P_c = 0.4$ to $P_c = 0.8$, task completion time follows an increasing trend, indicating that higher recruitment (lower P_c) allows agents to complete tasks faster.

4.4 Effects of P_e on HHTA Completion Time for Varying Task Density

We also explored varying P_e, running 100 trials each for $P_e \in \{\frac{i}{10}, 1 \leq i \leq 10\}$, with $T \in \{4, 10, 16\}$ and using $r_m = \frac{1}{6}$ and $P_c = \frac{3}{10}$. The results can be seen in Figure 7.

For $P_e \leq 0.8$, there was no significant difference between the HHTA completion time at different task densities (Welch's T-test, p=0.05) with the exception of $T = 10$ vs. $T = 4$ at $P_e = 0.2$, with $p = 0.03$. However, there was a significant difference in completion time between $T = 4$ and $T = 16$ when $P_e = 1.0$ and $P_e = 0.9$. Our results show that HHTA has a consistent completion time regardless of task density other than for large $P_e \in \{0.9, 1.0\}$, meaning a large majority of agents are exploring (making the algorithm more similar to random walk). When P_e is high, it is harder to complete the sparse problem because exploring is harder in a sparse environment.

Our results also show that a more even balance of P_e (the proportion of Exploring agents) vs. $1 - P_e$ (the proportion of Home agents) leads to a faster completion time. When P_e is too low, not enough agents are exploring, making it harder to find tasks. When P_e is too high, not enough agents are available in the home nest to be recruited when tasks are found.

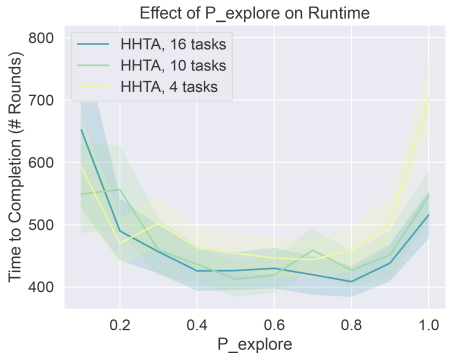

Figure 7: The effect of P_e on HHTA completion time

4.5 Effects of d_p on PROP Completion Time for Varying Task Density

We explored the effects of varying d_p, the maximum propagation radius, on completion time for varying T. We ran 20 trials for each unique pair of $d_p \in \{0, 5, 10, 15, 20, 25, 30, 40, 50, 60, 50\sqrt{2}\}$ (note that $d_p = 50\sqrt{2}$ means all tasks' information can be propagated over the entire grid space) and $T \in \{4, 10, 16, 50\}$, using $r_p = 3$. The results can be seen in Figure 8.

For reasonably sparse task setups ($T \in \{4, 10, 16\}$), larger maximum propagation radii correlate with faster runtimes (Welch's T-test, p=0.05) but after a certain point, completion time is mostly unchanged. In contrast, for very dense task setups ($T = 50$), besides a slight improvement in completion time moving from around $d_p = 0$ to $d_p = 10$, larger maximum propagation radii result in slower completion times (Welch's T-test, p=0.05). Increasing d_p results in more propagator agents having more task information, which allows (1) for follower agents to find tasks even if they are far way and (2) for follower agents to leverage this extra task information to prioritize tasks with higher demand. This causes the initial decline in completion time for increasing d_p values. However, if d_p is too large, there is too much information being propagated, diluting the agents' strategy. This adverse effect is likely not seen with sparser environments because even with every single propagator having information about every single task, this info is bounded in size. We also infer that the turning point in each plot's trend (when runtime either starts to flatten or increase) is related to the d_p value at which every propagator agent receives *some* task information.

4.6 Effects of r_p on PROP Completion Time for Varying Task Density

We explored the effects of varying r_p, the number of rounds a propagator must wait before sharing new task information with its neighbors, on completion time for varying T. We ran 20 trials for each unique pair of $r_p \in \{1, 2, 3, 5, 10, 15, 20\}$ and $T \in \{4, 10, 16\}$, using $d_p = 25$. The results can be seen in Figure 9.

There is a clear, mostly linear trend, where increasing r_p results in increasing completion times. The trend is consistent across all distinct task densities tested. This relationship between r_p and completion time is to be expected, as smaller r_p means that task information is moved about the environment more quickly, causing

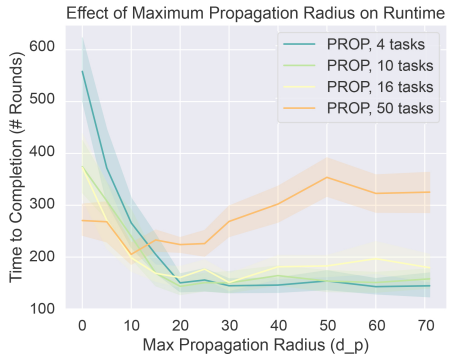

Figure 8: The effect of maximum propagation radius (d_p) on PROP completion time

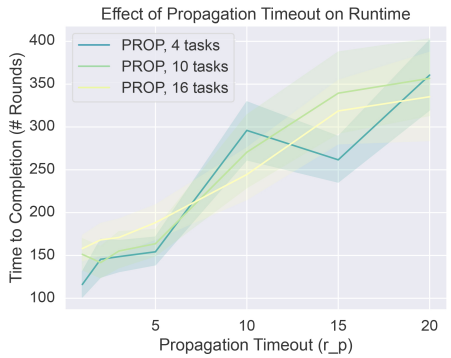

Figure 9: The effect of integer propagation timeout (r_p) on PROP completion time

the information that is used by follower agents to decide which task to move towards to be more up-to-date. Besides at the very beginning, r_p has no effect on the locations of task information, so the phenomenon we saw in which there is "too much" propagated information does not occur when varying r_p. It is the same task information, simply better when when the timeout is smaller. However, smaller values of r_p involve more message passing.

5 DISCUSSION

Our results demonstrate for both HHTA and PROP that when the total demand for agents is held fixed, task density significantly affects algorithm performance. HHTA performs better than RW when tasks are very sparse, and worse when the number of tasks is high because communicating about individual tasks matters less when there are many of them. RW performs very poorly with sparse tasks because it becomes harder over time for the remaining agents to find tasks. PROP also performs better than RW until the number of tasks is very high, as agents struggle to arrive at tasks when too much task information is being propagated. Though it outperforms RW for sparse tasks, PROP's completion time increases for very sparse tasks ($T \leq 6$). PROP also has a faster completion time and is more distributed than HHTA, but is much more resource and communication intensive, as it requires a propagator agent at every grid cell in order to spread information.

In relevant task allocation problems such as search-and-rescue or mine detection, the number of tasks in the environment is expected to be sparse, so both algorithms provide a speed-up in completion time compared to the Levy walk. HHTA provides a less agent intensive and less communication intensive approach but requires a central communication location. Contrarily, PROP provides a quicker and more distributed approach for sparse and mid-density environments but is more resource-intensive. Since the Levy flight has been shown to optimize search efficiency and can be observed in many species in nature, it makes sense that for very dense task environments with a low demand per task, the Levy flight outperforms both algorithms. Such environments are a very similar problem to foraging itself. On the other hand, environments with fewer tasks that require more agents benefit more from the coordination and communication of more advanced algorithms.

We also analyzed both algorithms' mechanics individually, showing the importance of recruitment in HHTA as well as the importance of an even balance of Exploring vs. Home agents. For PROP, we showed as expected that generally, higher d_p leads to better performance, though it is more communication-intensive. We also showed that as propagation timeout increases, time to completion increases, since task demands are stale for longer periods of time.

We also note that in extreme parameter settings, HHTA completion time was similar regardless of task density while varying algorithm parameters like P_c and P_e. However, this is untrue for PROP, which had a higher completion time for sparser environments at low d_p, and a higher completion time for denser environments at high d_p. This behavior makes sense because as d_p approaches 0, PROP reduces to RW, which is similarly affected with a higher completion time for sparse tasks.

6 FUTURE WORK

Future work could explore experiments in a dynamic setting, where new tasks can appear over time and agents can search for a new task after their existing task is finished. It could also evaluate other parameters, such as the ratio of total task demand to total number of agents. A larger such ratio would make the task allocation problem harder to solve, as there are less and less extra agents available to communicate. Another parameter left to be analyzed is swarm density; that is, the ratio of total number of agents to grid size .

Future work could also combine the strengths of the PROP and HHTA algorithms, where one agent for each task is assigned to propagate by leaving information in the vertex state of nearby vertices or communicating directly to any nearby agents like HHTA does. This algorithm would have a much smaller agent cost than the PROP algorithm while still being able to propagate task information. It would also not require a central home nest like HHTA does, instead opting to induce agent communication all around the arena.

Future work could also aim for analytical bounds on the expected completion time of our two algorithms. Because of the algorithms' relative simplicity, high probability bounds may be possible to obtain. Lastly, future work could extend to the continuous 2D as well as 3D (discrete and continuous) settings, adaptations which our presented theoretical framework is amenable to.

ACKNOWLEDGMENTS

Special thanks to Andrea Richa for her insights on the task allocation problem. This work was supported by NSF Award CCF-2003830.

REFERENCES

[1] Athula B Attygalle and E David Morgan. 1985. Ant trail pheromones. In *Advances in insect physiology*. Vol. 18. Elsevier, 1–30.
[2] Eric J Barth. 2003. A dynamic programming approach to robotic swarm navigation using relay markers. In *Proceedings of the 2003 American Control Conference, 2003.*, Vol. 6. IEEE, 5264–5269.
[3] Spring Berman, Adám Halász, M Ani Hsieh, and Vijay Kumar. 2009. Optimized stochastic policies for task allocation in swarms of robots. *IEEE transactions on robotics* 25, 4 (2009), 927–937.
[4] Mickaël Bettinelli, Michel Occello, and Damien Genthial. 2020. Coalition formation problem: a group dynamics inspired swarming method. In *International Conference on Swarm Intelligence*. Springer, 282–289.
[5] Arne Brutschy, Giovanni Pini, Carlo Pinciroli, Mauro Birattari, and Marco Dorigo. 2014. Self-organized task allocation to sequentially interdependent tasks in swarm robotics. *Autonomous Agents and Multi-Agent Systems* 28 (2014), 101–125.
[6] Grace Cai, Noble Harasha, and Nancy Lynch. 2022. A Comparison of New Swarm Task Allocation Algorithms in Unknown Environments with Varying Task Density. https://doi.org/10.48550/ARXIV.2212.00844
[7] Grace Cai and Don Sofge. 2019. An Urgency-Dependent Quorum Sensing Algorithm for N-Site Selection in Autonomous Swarms.. In *AAMAS*. 1853–1855.
[8] Jason R Cody and Julie A Adams. 2017. An evaluation of quorum sensing mechanisms in collective value-sensitive site selection. In *2017 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*. IEEE, 40–47.
[9] Micael Santos Couceiro. 2017. An overview of swarm robotics for search and rescue applications. *Artificial Intelligence: Concepts, Methodologies, Tools, and Applications* (2017), 1522–1561.
[10] Miguel Duarte, Jorge Gomes, Vasco Costa, Tiago Rodrigues, Fernando Silva, Víctor Lobo, Mario Monteiro Marques, Sancho Moura Oliveira, and Anders Lyhne Christensen. 2016. Application of swarm robotics systems to marine environmental monitoring. In *OCEANS 2016-Shanghai*. IEEE, 1–8.
[11] Xumei Fan, William Sayers, Shujun Zhang, Zhiwu Han, Luquan Ren, and Hassan Chizari. 2020. Review and classification of bio-inspired algorithms and their applications. *Journal of Bionic Engineering* 17, 3 (2020), 611–631.
[12] Ryusuke Fujisawa and Shigeto Dobata. 2013. Lévy walk enhances efficiency of group foraging in pheromone-communicating swarm robots. In *Proceedings of the 2013 IEEE/SICE International Symposium on System Integration*. 808–813. https://doi.org/10.1109/SII.2013.6776760
[13] Brian P Gerkey and Maja J Matarić. 2004. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International journal of robotics research* 23, 9 (2004), 939–954.
[14] Adám Halász, M Ani Hsieh, Spring Berman, and Vijay Kumar. 2007. Dynamic redistribution of a swarm of robots among multiple sites. In *2007 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2320–2325.
[15] Noble Harasha and Grace Cai. 2023. *Geometric Swarm Model*. https://doi.org/10.5281/zenodo.7570294
[16] John Harwell and Maria Gini. 2018. Broadening applicability of swarm-robotic foraging through constraint relaxation. In *2018 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*. IEEE, 116–122.
[17] Matthew Hoeing, Prithviraj Dasgupta, Plamen Petrov, and Stephen O'Hara. 2007. Auction-based multi-robot task allocation in comstar. In *Proceedings of the 6th international joint conference on autonomous agents and multiagent systems*. 1–8.
[18] Nicholas R Hoff, Amelia Sagoff, Robert J Wood, and Radhika Nagpal. 2010. Two foraging algorithms for robot swarms using only local communication. In *2010 IEEE International Conference on Robotics and Biomimetics*. IEEE, 123–130.
[19] M Ani Hsieh, Ádám Halász, Spring Berman, and Vijay Kumar. 2008. Biologically inspired redistribution of a swarm of robots among multiple sites. *Swarm Intelligence* 2, 2 (2008), 121–141.
[20] Dervis Karaboga and Bahriye Akay. 2009. A survey: algorithms simulating bee swarm intelligence. *Artificial intelligence review* 31, 1 (2009), 61–85.
[21] Shreeya Khurana and Donald Sofge. 2020. Quorum Sensing Re-evaluation Algorithm for N-Site Selection in Autonomous Swarms.. In *ICAART (1)*. 193–198.
[22] Qi Lu, Joshua P Hecker, and Melanie E Moses. 2018. Multiple-place swarm foraging with dynamic depots. *Autonomous Robots* 42 (2018), 909–926.
[23] Keith J O'hara, Daniel B Walker, and Tucker R Balch. 2005. The GNATS—low-cost embedded networks for supporting mobile robots. In *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*. Springer, 277–282.
[24] H Van Parunak, Michael Purcell, and Robert O'Connell. 2002. Digital pheromones for autonomous coordination of swarming UAV's. In *1st UAV Conference*. 3446.
[25] Riccardo Poli, James Kennedy, and Tim Blackwell. 2007. Particle swarm optimization. *Swarm intelligence* 1, 1 (2007), 33–57.
[26] Stephen C Pratt. 2005. Behavioral mechanisms of collective nest-site choice by the ant Temnothorax curvispinosus. *Insectes Sociaux* 52, 4 (2005), 383–392.
[27] Stephen C Pratt. 2005. Quorum sensing by encounter rates in the ant Temnothorax albipennis. *Behavioral Ecology* 16, 2 (2005), 488–496.
[28] Andreagiovanni Reina, Gabriele Valentini, Cristian Fernández-Oto, Marco Dorigo, and Vito Trianni. 2015. A design pattern for decentralised decision making. *PloS one* 10, 10 (2015), e0140950.
[29] A. M. Reynolds and C. J. Rhodes. 2009. The Lévy Flight Paradigm: Random Search Patterns and Mechanisms. *Ecology* 90, 4 (2009), 877–887. http://www.jstor.org/stable/25592573
[30] Craig W Reynolds. 1987. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. 25–34.
[31] Katherine Russell, Michael Schader, Kevin Andrea, and Sean Luke. 2015. Swarm robot foraging with wireless sensor motes. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. Citeseer, 287–295.
[32] F Sahin et al. 2002. A swarm intelligence based approach to the mine detection problem. In *IEEE International Conference on Systems, Man and Cybernetics*, Vol. 3. IEEE, 6–pp.
[33] Justin Solomon. 2018. Optimal transport on discrete domains. *AMS Short Course on Discrete Differential Geometry* (2018).
[34] Gabriele Valentini, Eliseo Ferrante, and Marco Dorigo. 2017. The best-of-n problem in robot swarms: Formalization, state of the art, and novel perspectives. *Frontiers in Robotics and AI* 4 (2017), 9.
[35] Bo Xu, Zhaofeng Yang, Yu Ge, and Zhiping Peng. 2015. Coalition formation in multi-agent systems based on improved particle swarm optimization algorithm. *International Journal of Hybrid Information Technology* 8, 3 (2015), 1–8.
[36] Dandan Zhang, Guangming Xie, Junzhi Yu, and Long Wang. 2007. Adaptive task assignment for multiple mobile robots via swarm intelligence approach. *Robotics and Autonomous Systems* 55, 7 (2007), 572–588.