Optimziation Methods for Beacon Based Foraging Algorithms

Christopher Sanford(✉) and Jae Oh

Syracuse University, Syracuse, NY 13203, USA
{clsanfor,jcoh}@syr.edu

Abstract. Beacon-based Robotic foraging is inspired by nature's ability to create efficient explorers and gatherers, and imposes a number of constraints on how agents can interact. In decentralized models, the robots must maintain chains of communication, effectively explore areas, and start collecting from discovered targets. Previous approaches have used a beacon-based technique, which is dependent on swarm size to environment size ratios, and do not have guarantees on finding all targets. This paper outlines the issues in these approaches and offers solutions to finding targets reliably, robust task allocations, and efficient beacon network. We verify our techniques by providing metrics of successful swarm size to environment size ratios, robot congestion improvement, and target utility independent measurements for gathering.

Keywords: Multi-agents · Foraging

1 Introduction

Robotic or automated foraging has become an interesting problem recently and combines many multi-disciplinary techniques to accomplish its goals. Automated foraging is given an unknown environment (structure, layout, existing entities, and desirable resources unknown) and is required to explore this environment to find targets. The robots are to navigate, map out, and efficiently cover the environment so that target collection can be done. Robots must repeatedly deliver items from discovered targets to the home base.

This continuous foraging is not a trivial problem, especially when there are no localization techniques. When localization is absent, robots must maintain a network of communication in some form. We assume no localization method, and thus we need to make up for this lack of navigational luxury.

The second aspect of foraging, continuous item retrieval, requires orienting and allocating robots to discovered targets. Some techniques utilize robots to create static waypoints for the retrieval process for navigation. However, static waypoints may waste the robots for the retrieval task. For this paper, we assume that we can feasibly create an optimal static waypoint network for robots to navigate, thus eliminating the need for mobile networks.

© Springer International Publishing AG, part of Springer Nature 2018
M. Mouhoub et al. (Eds.): IEA/AIE 2018, LNAI 10868, pp. 22–33, 2018.
https://doi.org/10.1007/978-3-319-92058-0_3

We look at a beacon based foraging algorithm proposed by Hoff et al. [6], and its multi-target extension done by Jiao et al. [4]. The approach Hoff et al. uses two types of agents (the extension uses three) Beacon, Walker (Explorer in the extension), and Worker (in [4]). The beacons work to map out the area by creating routes in the environments with themselves. The explorers search along the currently established network to find targets, or to become beacons to extend the network, and workers specifically only gather from found targets.

The previous algorithms solves the problem rudimentally, therefore has a number of fundamental issues such as task allocation (target assignment), congestion, and poorly utilized beacons. We present two algorithms: Rostering for controlling how robots are assigned to targets for gathering, and NetOpt for better utilizing beacons for discovering targets and possible reallocating as workers.

2 Related Work

Many Robotic foraging models are inspired by biological examples such as ants. Using ants as a primary example, they work in a rather stochastic way which can coordinate when necessary to accomplish food gathering. Though stochastic models have been studied in Adler and Gordon [1], we focus on a deterministic approach using only a minimal amount of random walking.

Below we outline the progression of foraging techniques and analysis that has proceeded this work. Generally foraging has been concerned with single targets, with only recently multiple targets becoming a topic of concern.

- Svennebring and Koenig have used visual markings to identify explored terrain [7]. This has the advantage of not necessarily needing additional robots to maintain mapping information, but physical markings can be unreliable depending on the material, and the visual identification capabilities of the robot.
- O'Hara et al. use statically deployed networks in their G.N.A.T.S. system to create potential fields to guide multi-agent systems perform distributed path planning [5]. Theirs has the advantage of being able to plan out the network, but not allowing optimized searching. Specifically, if the network has not been deployed optimally, or the area has been unexplored, their method will not work well.
- Barth [2] studied the deployable network, which allows beacons to be placed as the robots explore. This method has the advantage of having a more dynamic setup with the mapping data and doesn't rely on known information about the environment.
- Hoff et al. proposed two techniques to forage for a single target: A modified virtual pheromone and a multi-state beacon algorithm [6]. In the multi-state beacon algorithm, robots can either be a Beacon, an Explorer or a Worker. Each of these 'types' has a different task assigned to it. Robots can switch between 'types' according to instructions in the algorithm to accomplish their task of finding targets and moving between their targets and the home base.

Explorers explore the area for their target and have a chance to become beacons under the right circumstances. Beacons transmit information across the network of robots, and workers move between the base and the target using the beacon network. The algorithm assumes that the robots spawn out of the base one by one over a period. The first robots to spawn quickly become Beacons marking the location of the base. As more robots spawn out of the base, they begin to explore the map, indexing themselves based on the *cardinality* of Beacon robots in their vicinity. Over time the robots will spread out and create a network of beacons, and once the robots locate a target, they can use that network of beacons to move to and from the base and targets. These beacon type robots index themselves in two ways, ascending and descending from the home base. This creates a network which can relay information to the robots and also serves to create a path for the robots to follow once they find their target (food). Once the 'food' has been found some robots transition into worker types and move up the indexed beacons until they gather the 'food' and then return it to the home base. The algorithms handle dynamic foraging as with Barth's beacon setup and allow parameter tuning with the beacon based approach. We focus on an extension to the beacon method and analyze some of its parameter tunings.

– Jiao et al. provide an extension to Hoff's beacon based algorithm to deal with multiple targets [4]. The changes create the network into a tree like structure branching to each target and allow for workers to individually select targets to work with. The algorithm suffers from problems such as congestion and low beacon utilization, also present in Hoff's, but provides an initial effort to deal with multiple targets.

The above algorithms take advantage of storing mapping information somewhere and use this information to navigate without localization. Our approach will not use localization techniques, such as GPS, either; our algorithm relies on local communication for distributing navigation information as well as coordinating resource gathering.

Both Hoff's [6] and Jiao's [4] algorithms suffer three fundamental problems which will be addressed by our optimizations.

1. Target allocation: We need to ensure that targets have an adequate number of workers allocated when discovered. Since worker allocation is done via probability parameters ($Pr(Explorer \rightarrow Worker) = wprob$), robots can often be allocated too quickly to early targets or not quickly enough and most explorers become beacons.
2. Robot congestion: There tend to be too many robots converging to a target that can impede robots motion. This happens due to no control flow or coordination done between robots assigning themselves to targets.
3. Beacon optimization: The previous algorithms can use beacons inefficiently and lack the ability to reuse beacons when not being utilized.

3 The Rostering Technique

The multi-target extension [4] does not make any guarantees about target allocation, often having workers flock to the first target found, and any subsequent targets have little or no gathering workers. The parameter tuning for this algorithm does not add any guarantees to it as well, and thus needs more control. Due to the same occurrence, too many workers flocking to one target causes traffic congestion impeding the traffic of mobile robots and slowing down the gathering process. Lastly, beacons are not necessarily allocated optimally, either not heading towards a target, or deploys themselves nearby other beacons when the network could spread out further.

We propose the rostering algorithm to help with both target allocation and robot congestion, with beacon optimization handled in a later section. To elaborate on what these issues entail, we need to see what happen when targets are found. As soon as a target is found, the cardinality information will propagate across the network and will be broadcasted to explorers. Explorers will check to become a worker at each iteration at a fixed problem probability ($P(Exp \rightarrow Worker) = wprob$), and if $wprob$ of too large most of the explorers will quickly become workers. Large probabilities ($wprob \geq 0.001$) will lead to explorers becoming workers far too quickly and prevents further exploration.

To remedy both congestion and target allocation we propose the rostering algorithm that can control the number of robots going between different targets. Taking advantage that beacons already broadcast target information, they now use a new structure to allocate robots to particular targets shown below.

```
class RosteringRobot extends Robot {
    map<target, set<Robot>> roster;
}
```

The structure assigns for each target a set of robots to work on it. When a beacon broadcasts its cardinality information, it will also broadcast roster openings, and nearby explorers can choose to join in on these rosters. As these rosters fill up, they are broadcasted throughout the network to maintain a global structure. To use this structure when an explorer hears an open roster with a given fixed probability, it will assign itself to a target on that roster. To prevent too many robots assigning themselves to a specific target every robot has a maximal roster size that it will ignore. This maximal amount is fixed before the foraging starts and is done as such since global synchronization of these decisions is costly. This process is demonstrated below.

```
function RosteringExplorer()
    Let robots ← set of nearby robots;
    Let bcasts ← set of heard broadcasts;
    Let targets ← set of nearby targets;
    Let map ← ⋃_{m∈bcasts} m;
    if (targets ≠ ∅ and ∃t ∈ targets, map[t] ≠ START) or |bcasts| = 1 then
        type ← BEACON;
    else if |keys(map)| > 0 and open_roster() then
        With the probability of p, type ← WORKER
        assignSelfToRoster();
    else
        RandomWalk();
```

One thing to note is the **assignSelfToRoster()** procedure; The explorer broadcasts a message and if a beacon hears this message the algorithm will function correctly. To see how this works we look at the modified beacon procedure below.

```
function RosterBeacon()
    Beacon();
    Broadcast roster
    Let bcasts ← set of heard broadcasts;
    Let bots ← set of nearby robots;
    for t ∈ knownTargets() do
        if t ∉ keys(roster) then
            roster[t] = {};
    for (t ↦ assigned) ∈ roster do
        roster[t] = roster[t] ∪ assigned;
```

First note that it does what the multi-target beacons do by using the **Beacon** procedure by Jiao et al. [4]. It first updates its roster list as necessary for initialization. Secondly it listens to all broadcasts; These broadcasts now contain roster information and more important roster information from workers. Within these broadcast if there is a worker assigned to a target it doesn't know about, the beacon will add the worker to the list of known workers. In practical situations this broadcast will most likely be more than just a simple procedure call for an explorer, and its possible workers can continue to broadcast this information for a short time before stopping to guarantee the information is heard entirely.

We should note that this doesn't guarantee a certain amount of workers will be allocated, usage of a high assignment probability will guarantee assignments, but the rostering may allow assignments of higher than the desired value. However, as long as network latency is low and a good probability is chosen, then the actual assignment population will be very close the desired amount.

4 Rostering Experimental Analysis

We want to analyze how the rostering technique improves performance. We measure a few different factors, such as congestion improvements, the number of targets found, and average time lost due to congestion. These metrics represent vital issues that are present in the multi-target extension; the multi-target extension suffered from the lack of ability to find all the targets in a given environment, which can be undesirable when different targets represent different needs. The multi-target extension can also suffer from the congestion issue due to the lack of control in sending robots to particular targets; Specifically, too many robots converge to the first target found.

For these experiments we will be maintaining a few fixed parameters; time, an import factor to allow robots to work, is measured in no specific units and the speed of robots is constant, relative to those units. We also fix the size of the population as 100 robots. This value was chosen through experimentation on finding the disc shape of the robots coverage area and chose an appropriate target location distribution similarly described by Jiao et al. [4]. Lastly, we provide the worker assignment bound to be 5.

Fist we will compare how we improve upon congestion; We compare performance by measuring the average time lost on each trip for each worker. This is done by computing the minimal time required to reach the target (given a known maximal speed of each robot) and find the difference between each trip time that minimal time, we then average these trip times for each robot to compute the average time loss for each robot. We compare our rostering algorithm against the multi-target *hoff* algorithm with two different probability parameters on worker conversion.

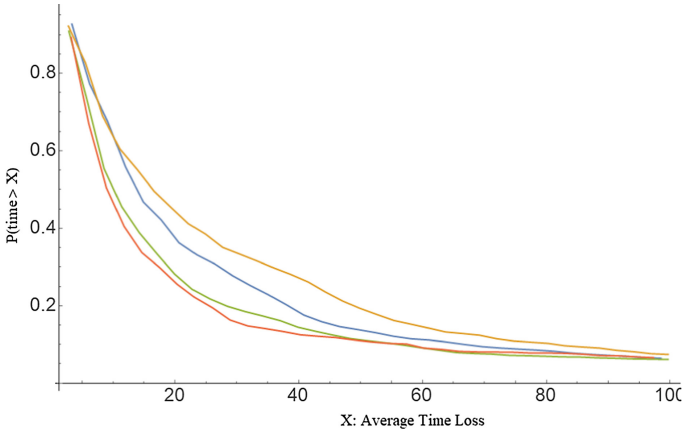


Fig. 1. Congestion metrics

Figure 1 shows the cumulative probabilities of a particular average time loss. The x-axis represents the average time loss for robots, and the y axis represents

the probability of having at least X average lost time. By definition, as $X \to 0$ $Y \to 1$ and $X \to \infty$ as $Y \to 0$. So we choose a significant section of the data to display. Our experimental setup has that the minimal amount of time required to direct move to each target is at most 20 units of time. Figure 1 shows the improvement of our algorithm compared to the original multi-target. Specifically for some chosen values, we have the following values:

	$X = 10$	$X = 20$	$X = 30$	$X = 40$
Hoff(p = 0.001)	0.6764	0.4218	0.2775	0.21286
Hoff(p = 0.01)	0.6901	0.4534	0.3508	0.2803
Roster(p = 0.001)	0.5542	0.2820	0.1985	0.1445
Roster(p = 0.01)	0.5048	0.2990	0.1632	0.1333

Fig. 2. Sample results of congestion

Figure 2 demonstrates some of the results of improvement. For $X = 10$ Not much improvement is demonstrated (at most 27%). However, as X increases we start seeing at most 52% improvements. Qualitatively we can view the simulations and check that robots are assigned correctly, but here we have better quantified demonstrations on how well the rostering approach helps deal with traveling time and improves congestion conditions.

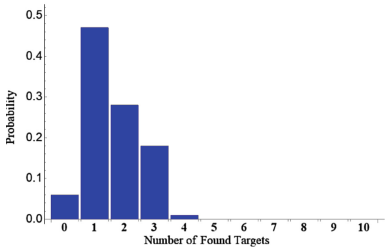
The second metric we want to look at for improvement is the number of targets found. In Jiao et al. they had looked at how many targets were found when varying the worker conversion parameter and had demonstrated that while the parameter can be tweaked to do better often, it will cripple the ability to do the retrieval. Here we look at the distribution of targets found for the hoff multi-target and the rostering approach. Our setup is similar to that of Jiao et al. [4], where the simulation will last for 600 units of time each.

Figure 3a and b shows the distribution of targets found for one of the better and worse case scenarios for the base multi-target algorithm. We note that the number of targets found does not exceed seven, even in the best-case scenario. This also is the worst case scenario for converting workers for gathering.

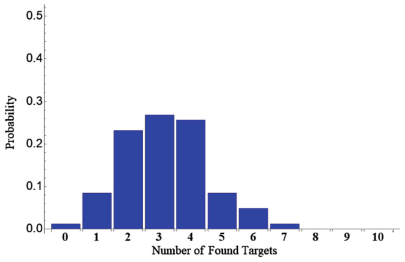
We want to see the performance increase for using this for finding new targets; Since when workers are converted very quickly in the original multi-target algorithm, the number of workers for exploration decreases dramatically and instantaneously. We then take the rostering approach with $p = 0.01$. Here we do not vary the probability, as we want to see how much the performance improves due to a fewer worker conversions. Figure 3c demonstrates the performance.

As demonstrated, the performance does not increase by a large amount, but it does so comparatively to experiments with similar parameterization. The multi-target algorithm with $p = 0.0001$ performs similarly. However, we should note that is because no gathering is being done (generally, only a few robots become workers), whereas the rostering approach almost all targets will have workers and early targets are essentially guaranteed at least five gathering workers (this is due

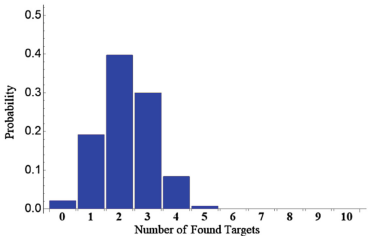
to the high probability with worker conversion). So while the target distribution has not improved by much, it does improve slightly with more guarantees on gathering.



a) Targets Distribution for Hoff($p = 0.01$)



b) Targets Distribution for Hoff($p = 0.0001$)



c) Target Distribution for Rostering($p = 0.01$)

Fig. 3. Rostering comparison

5 Network Optimizaiton

In this section, we attempt to resolve the issue of how effectively the beacons are used. There are two focus areas: Beacon placement and beacon recycling. Both in Hoff's and the multi-target extension, beacon placement is not effective: Often there are two beacons almost right next to each other due to the condition for becoming a beacon. Secondly, we design an algorithm for reusing the beacons, which is non-existent in the previous two algorithms. We assume that the robots can check the distance between beacons by using infrared detection or similar methods. For this section, we use the term **NetOpt** to denote the extended algorithm using these two methods.

First, we consider the minimal distance checking which is a minor extension to the algorithm; This check is inspired by other techniques that mimic the molecular model–a technique that seeks to find a spread equilibrium for the robots to perform a coverage of the map as in Batalin [3]. The area coverage problem is a subproblem of foraging, though in general, it is not necessary to

cover the entire environment in foraging. Batalin's algorithm creates an equilibrium by treating the robots as molecules and uses distance checks. Our algorithm does indeed use distance checks, but instead of creating an equilibrium it forces the robots to spread out farther and thus covering more region. This addition to the algorithm is straight forward and is captured in the following modification to the **Explorer** procedure from the multi-target algorithm.

---

function MinDistExplorer()
  Let $robots \leftarrow$ set of nearby robots;
  Let $beacons \leftarrow$ set of nearby beacons;
  Let $bcasts \leftarrow$ set of heard broadcasts;
  Let $targets \leftarrow$ set of nearby targets;
  Let $map \leftarrow \bigcup_{m \in bcasts} m$;
  if ($targets \neq \emptyset$ and $\exists t \in targets, map[t] \neq START$) or $|bcasts| = 1$ **then**
    Let $D = \{dist(b, this) : b \in beacons\}$
    if $\forall d \in D$, $d < MAX\_DIST$ **then**
      $type \leftarrow BEACON$;
  else if $|keys(map)| > 0$ **then**
    With the probability of p, $type \leftarrow WORKER$;
  else
    RandomWalk();

MAX_DIST is a configurable constant for all robots; We use the constant as a fixed ratio of the physical detection distance. This check maintains the desired properties of the multi-target extension and ensures that new beacons will not cluster around a target and only become beacons if they detect exactly one beacon.

Secondly, we'll look at beacon recycling beacons that are not used or used inefficiently. One problem with the previous beacon-based methods is that when beacons are placed, they will always be there, thereby creating un-utilized beacons. We want to be able to identify unused and viable beacons to be recycled. One property of the network we will use for this is that our network is a tree-like structure, and captured more succinctly in the following theorem.

Theorem 1. *Given a configuration of the beacon network N, then any beacon which does not detect a larger nest cardinality can be removed without disconnecting the network.*

This theorem tells us that beacons only need to check this condition in broadcasts to determine if they are a **leaf** beacon. More explicitly the two condition we check for are:

1. Is the beacon a leaf?
2. Is the beacon allocated as an initial beacon for a target?

The first condition is just the leaf condition; the second is to make sure that no beacon that has found a target will leave the network. When a beacon decides to leave and become an explorer, it may use the beacon network to navigate back to the nest and restart its search. The decision on whether or not to recycle oneself for each leaf beacon depends highly on the knowledge given the region; We assume the most general case where the targets can be placed in an arbitrary distribution, so our beacons wait a fixed amount of time before recycling themselves.

Care must be taken for this process; if a beacon was to leave the network when an explorer was nearby, then its possible that the explorer will become loose the sight of the network. This could happen only with the minimal distance checks algorithm. Without minimal distance checks, the leaf-time being checked against should not be incremented when there are explorers within range. Otherwise, explorers can just end up replacing that beacon immediately and can stunt the growth of the network.

6 NetOpt Experimental Analysis

NetOpt is designed to improve target finding performance, so in this section, we provide insight on the improvements that NetOpt brings on the distribution of targets found. We keep the same setup as in the rostering experiments and apply NetOpt with rostering simultaneously. Figure 4a shows the distribution of targets found using the NetOpt addition to the algorithm.

As can be seen, NetOpt provide a large improvement, reaching targets found that the previous two algorithms could not achieve, as well as being able to maintain a centered distribution about more than 70% of targets found. While this shows a vast improvement given some of the resources available, one might question the performance should requirements change in robot population and gathering necessities.

Figure 4d shows the performance with only 50 robots and five workers. The distribution is less spread but has much better performance than the original. Next, we take the previous setup and decrease the worker factor to show how much performance can increase by reducing the worker count. Figure 4b shows the distribution by reducing the worker bound to 2, thereby allowing for more explorers. A large percentage still find only four targets, but vastly less find less. Shifting much of the distribution towards five and six targets found. Though neither of these still find all ten targets.

On the other side of the spectrum, we provide a better scenario for our robots by increasing the population to 125. Figure 4c shows the results with a worker bound of 5. With only a 25% increase in population the performance increases drastically from the 100-5 configuration. Though ten targets are not guaranteed, the performance increase is noticeable.

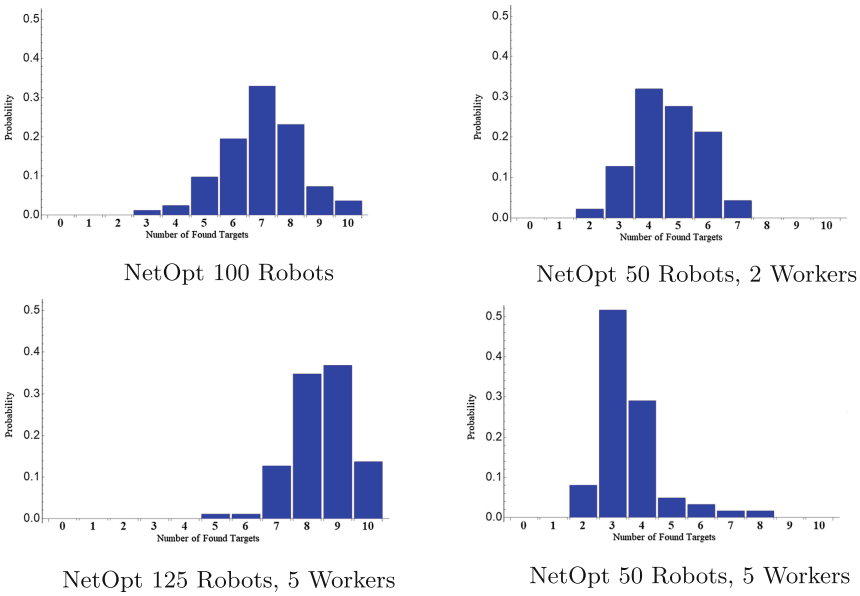


Fig. 4. NetOpt Comparisons

7 Future Work

Our algorithm does demonstrate nice improvements on the old ones but still only works under certain assumptions. To reiterate some of these assumptions:

1. Local communication for global knowledge;
2. Random walking is sufficient to eventually reach all targets;
3. Quantity of robots is enough to fulfill the previous.

Having no localization is the requirement that forces to have a beacon network; Though the beacon network is not necessary, local communication and connectivity are important in general. The other two requirements are what allows to make the simplifying tests of allowing beacons to remain static once they have positioned themselves. Future work would be to address these constraints which could allow for more dynamic beacons. Our NetOpt algorithm handles recycling beacons once they have found themselves to be not helpful, but if targets existed much farther past the random walk trajectory, then even the NetOpt algorithm would do very poorly.

The other issues are related to how exploration is done; since our algorithm uses random walking, it may travel to already explored regions. Since a global localization is not employed there would need to be a way to maintain relative regions of exploration to each beacon, and communicate desired regions of unexplored areas.

8 Conclusions

Foraging algorithms have been explored by many researchers and have taken various approaches. Statistical models have been developed through the inspiration from biological foragers such as ants. More deterministic approaches have been developed assuming both localization and lack of localization methods. These older methods focused more on single target foraging, whereas the more recent work considers a multi-target extension.

Our work solves a number of problems with the multi-target extension. The older algorithms suffered from many defects, specifically the naive probability parameterization leading to the lack of ability to effectively control exploration and target item delivery, congestion due to uncontrolled worker conversion, and lack of guarantees on gathering properties.

We demonstrate the efficacy of our work in a few ways. We develop a way of measuring congestion which was absent in previous work. We continue the measurements from older work that computes the distribution of targets found, and demonstrate the improvements from older work. We also outline the assumptions that facilitate these metrics as well as allow the algorithm to perform well. We presented an algorithm to utilize beacon robots more effectively, consequently improving the exploration capability. Lastly outlining what needs to be accomplished to break away from these assumptions for future work.

References

1. Adler, F.R., Gordon, D.M.: Information collection and spread by networks of patrolling ants. Am. Nat. **140**(3), 373–400 (1992)
2. Barth, E.J.: A dynamic programming approach to robotic swarm navigation using relay markers. In: Proceedings of the American Control Conference, vol. 6, pp. 5264–5269, July 2003
3. Batalin, M.A., Sukhatme, G.S.: Spreading out: a local approach to multi-robot coverage. In: Asama, H., Arai, T., Fukuda, T., Hasegawa, T. (eds.) Distributed Autonomous Robotic Systems, vol. 5, pp. 373–382. Springer, Tokyo (2002). https://doi.org/10.1007/978-4-431-65941-9_37
4. Jiao, Z., Sanford, C., Oh, J.: Multi-target extension for beacon based foraging methods. Technical report, Syracuse University, November 2017. https://surface.syr.edu/eecs/250/
5. O'Hara, K.J., Walker, D.B., Balch, T.R.: The GNATS — low-cost embedded networks for supporting mobile robots. In: Parker, L.E., Schneider, F.E., Schultz, A.C. (eds.) Multi-robot Systems. From Swarms to Intelligent Automata Volume III, pp. 277–282. Springer, Dordrecht (2005). https://doi.org/10.1007/1-4020-3389-3_24
6. Hoff, N.R., Sagoff, A., Wood, R.J., Nagpal, R.: Two foraging algorithms for robot swarms using only local communication. In: 2010 IEEE International Conference on Robotics and Biomimetics, ROBIO 2010, pp. 123–130, January 2011
7. Svennebring, J., Koenig, S.: Building terrain-covering ant robots: a feasibility study. Auton. Robots **16**(3), 313–332 (2004)