11-30-2017

# Multi-target Extension for Beacon Foraging Methods

Christopher Sanford
*Syracuse University*

Ziong Jiao
*Syracuse University*

Jae Oh
*Syracuse University*

11-30-2017

# Multi-target Extension for Beacon Based Foraging Methods

Christopher Sanford

Zilong Jiao

Jae Oh

# Multi-target Extension for Beacon Based Foraging Methods

Zilong Jiao Christopher Sanford Jae Oh

November 20, 2017

**Abstract**

Robotic foraging is a complex problem that encompasses both the problem of exploring an area and retrieval of targets. To solve this, biologically inspired algorithms have been proposed that handle the scenario when only one target exists, but requires extension to multiple targets. In both scenarios there exists problems of robot allocation and congestion, and we analyze ways of optimizing both allocation and minimizing congestion for our algorithms. We demonstrate the results through parameterized metrics and compare the improvements in each scenario

## 1 Introduction

Multi-robot foraging has become an important problem in automated exploration and retrieval, and combines multiple disciplines to achieve its goals. Automated foraging has two primary goals: exploration of an unknown area (i.e. the entities/objects are unknown), and within that area find desirable resources (e.g. food) so that they may be returned to the base. This problem and solutions have been primarily inspired by biological examples, such as ant colonies. Ants demonstrate a remarkable array of ability for exploration and retrieval; They are able to separate in a chaotic manner to explore an area, and then coordinate once they have found something of interest. This ability to quickly communicate, and coordinate is a spectacular example to study to accomplish our goal.

## 2 Related Work

Automated foraging has been studied by researchers in the past, and have developed a number of techniques to deal with different aspects of the problem. Many of these techniques have been summarized by Hoff et al., and use a variety of techniques, but have some shortcomings. Most techniques focusing on maintain pathing data in some manner; Many of of such are inspired from ant-like structures.

- Physical markings are a simple way to navigate a terrain, and is a common way we navigate ourselves (e.g. road signs). This technique has been used by Svennebring and Koenig wherein they use visual marking to identify explored terrain [**?**]
- Statically deployed networks can also be used as a landmarking technique to guide robotic foragers in their exploration phase. O'hara et al. use this technique in their G.N.A.T.S. system to create potential fields to guide multi-agent systems perform distributed path planning [**?**].
- Deployable network have also been explored by Eric Barth [**?**], which allows deployable beacons to be placed as the robots explore. This has the advantage of not requiring knowledge of the area beforehand, allows for more dynamic networks, but only searches for one target.
- Hoff et al. proposed two techniques to forage for a single target: A modified virtual pheromone and a multi-state beacon algorithm [**?**]. These two algorithms take advantage of the ability of being able to deploy beacons or pheromones to create a network in unknown area, and doesn't require the exploring robots to carry objects to act as beacons.

The above algorithm describe the progression of the solutions to *single-target foraging*, however none of these algorithms can be used to find multiple targets in an efficient manner. While one can always retrieve one target at a time then redeploy each of these algorithm, this tactics becomes infeasible for two reasons: Lack of efficiency for reployment and they cannot in a decentralized manner choose which targets to ignore and only search for one at a time. The first problem is something that can be addressed individually for algorithms for dealing with redeployment, but the second issue requires more work. Supposing there is more than one target to find, the system must agree upon which target to ignore for the algorithms to work reliably. In this paper we propose an extension to the cardinality algorithm proposed by Hoff et al. to handle multiple targets.

# 3 Algorithm Description

Our algorithm is an extension of Hoff et al.'s cardinality algorithm algorithm which allows for specific allocation to different targets. The original algorithm method of storing cardinalities can be extended so that for each target there is a different cardinality. This will create a tree-like structure reaching out to the different targets, and handles the situations that Hoff et al.'s cardinality algorithm could not when given multiple targets. The below figure shows the outline of the structure for our robot. The important aspects are the **assignment** and **cardinalities** fields. Robots will generally be deployed one at a time in a sequential order, and we will assume that is how this will be done in this paper, but with some minor modifications robots can be activated simultaneously.

```
class Robot {
    bool has_food;
    target assignment;
    robot_state state;
    map<target, index> cardinalities
}
```

```
function RobotAtEachIteration()
    if type = beacon then
        Beacon();
    else if type = explorer then
        Explorer();
    else if type = worker then
        Worker();
```

The algorithm is organized as an iterative, state-based process where a given state determines which functions shall be used. The three states are **beacon**, **explorer**, and **worker**. Explorers are the initial state for the robots and are the robot type which scour the environment to find targets and begin forming the network.

The goal of the explorer is two fold: Expand the network and find targets. The first robot to be deployed will become the first beacon; from there explorers will only change their state if they move out of range of the center of the network. This condition is detected when only one beacon is within range, similar to Hoff et al's walker state. The other condition for becoming a beacon is when a target is detected and that target does not have a beacon dedicated to guiding workers to that target.

This state is a factored version of Hoff et al. which only handles the exploration part with food retrieval factored out into the **worker** state. To change into being a **worker**, the explorer will detect the cardinality broadcasts, and with a fixed probability, decided a priori, will become a worker. This probability check happens each iteration, so high probability will force explorers to gather very quickly, whereas low probability will allow explorers to continue exploring for longer.

```
function EXPLORER()
    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 then
        With the probability of p, type ← WORKER;
    else
        RandomWalk();
```

The beacon routine is the next possible state our robots can take during a foraging run. Beacons maintain the network of communication, are an analogous map of the world, and are responsible for making sure that workers correctly reach their targets. Beacons, unlike Hoff's algorithm, will never change to another state. Our conditions guarantee that no two start beacons will be within communication range of each other. This helps prevent beacon congestion around a target, and allows beacons to remain static. This lack of a state change also prevent network disconnection; The condition that Hoff's cardinality algorithm allows beacons to leave if they hear too many beacons; This condition isn't sufficient to guarantee a path back to the nest should a beacon leave.

```
function BEACON()
    Broadcast cardinalities
    Let bcasts ← set of heard broadcasts;
    Let bots ← set of nearby robots;
    Let targets ← set of nearby targets;
    Let map ← ⋃_{m∈bcasts} m;
    if targets ≠ ∅ and map[t] ≠ START for some t ∈ targets then
        t ← choose_one(targets) such that map[t] ≠ START;
        cardinalities[t] ← START;
    else
        for (t → c) ∈ map do
            if t ∉ keys(cardinalities) then
                cardinalities[t] ← c + 1;
```

Last we have the **worker** state; This state handles traveling back and forth between the nest. This state is only activated when a worker hears a beacon with target information. Upon changing to this state it will randomly select a broadcasted target and begin its search towards that target. This state takes advantage of several abstract functions: **nest_search** and **target_search**. Both work as their names suggest, but are dependent upon implementation (virtual,

hardware), and assumed structure of the robots. Though we omit a specific implementation they work by going to the next beacon in the chain; There is always a next beacon to go towards, and generally one would choose the beacon of lowest cardinality towards whichever target they wish to move towards.

For dealing with some of the robots' structure, such as detection distance and obstacle avoidance, there are a couple details that will be generally need to be dealt with. First to detect other robots while searching the distance between beacons should be small enough so that the detection distance can detect the next beacon as it arrives to its current one. Secondly for object avoidance, since there is a requirement for following beacons object avoidance should allow robots to orbit around beacons.

```
function WORKER()
    if stuck() then
        type ← explorer;
    else
        Let bcasts ← set of heard broadcasts;
        Let map ← ⋃_{m∈bcasts} m;
        if assignment is None then
            target ← choose_random(keys(map))
        if !has_food and on_target(assignment) then
            collect();                    ▷ Routine for collection and deposition
        else
            if has_food then
                nest_search();
                attempt_food_drop();
            else
                target_search(assignment);
```

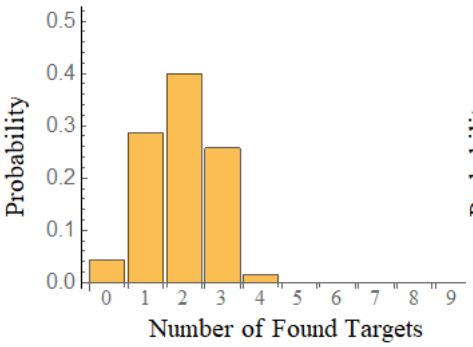
# 4 Experimental Analysis

In this section we outline the setup for analyzing the performance and capabilities of our algorithm. We wish to provide an analysis on how the parameters of the algorithm affect the performance by choosing a sufficiently reasonable fixed set of world parameters. Since both Hoff's algorithm and ours uses a *randomWalk()* procedure, both algorithm will have explorers generating a disc shaped network coverage on average, and thus we'll be using a target distribution that maintains target locations in a general area on the periphery. This means that given the known shape of the disc-like coverage, choose a distribution so that the targets like in a range that sits near the edge of that disc (both within and without that disc).

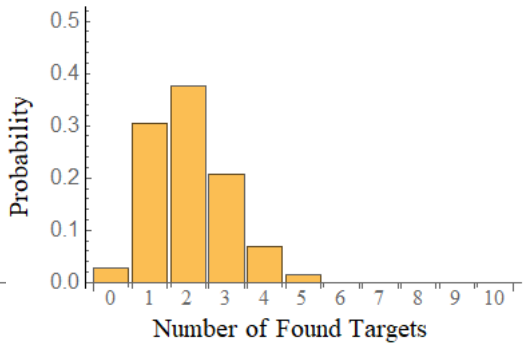
We use a population of 100 robots for each test, as well as 10 targets. Our

goal is to see how many targets will be found given by the worker assignment probability. We choose this due to how it can decide whether or not multiple targets will be found or will many robots decide quickly to become workers. We choose three different probabilities and measure the distribution of targets found over numerous different environments given the same target distribution parameters.

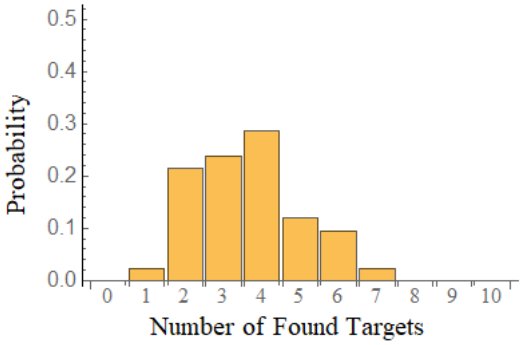
It is hypothesized that as $p \to 0$ that more targets will be, though we wish to to understand to what extent these values increase performance of targets found. Figure 1a shows the distributions handled by each different worker probability parameters. None of the particular probabilities perform to maximum performance in this metric, but it does show that that the best performance we have observed occurred at $p = 0.0001$.




(a) Targets Distribution for $p = 0.01$




(b) Targets Distribution for $p = 0.001$




(c) Targets Distribution for $p = 0.0001$

Figure 1: Target Distribution for $p = 0.001, 0.01, 0.0001$

We could go further and lower the probability, however it was observed that while we do find more targets, we often have almost no workers ($\approx 2 - 3$) each

run. So the foraging aspect of the algorithm does fall short in the more optimally parameterized exploration phase. This has to do with how exploration is done; Since the shape of the region covered will on average be a disc, targets which fall outside of that region will unlikely be found. Thus to find them more workers are sacrificed to make the disc larger.

Another issue is to consider that, while less targets are found for $p = 0.01$ and more foraging is being done, the foraging tends to be to the first target found, since rapid probability checks tend to converge to a guaranteed choice. This leaves an issue of dealing with foraging viability and needs to be address in future work.

# 5 Conclusion

In this paper we've proposed an extension of a beacon based foraging algorithm to handle multiple targets. Our algorithm solves the issues that the single target algorithm have when encountering an environment with multiple targets. Our algorithm factors out the states of the robots for a cleaner abstraction for the work flow of the machines. Though while our algorithm is correct, there exists issues to be resolved; Task allocation is handled in a probabilistic manner which doesn't allow fine-grained control of how robots are allocated. While the parameter can be tuned for particular allocations, there are no guarantees for any particular instance of the problem, and gives a wide distribution for the number of targets that are found.

Furthermore, while we can measure performance in terms of targets found, the performance in terms of foraging rate is either poor or ill-defined. When more targets are found, the algorithm simply sacrifices foraging ability to do so, and thus only accomplishes one of the tasks. Secondly, while we can measure the rate at which targets are being collected, simply finding the near target will yield biased results and not consider fairness; In future work this needs to be addressed with possible solutions of utility-free metrics, or require that targets have utility associated with them.