Universiteit Leiden
The Netherlands

Opleiding Informatica

The effect of social reward
on swarm robots in a foraging task

Thomas Coret

Supervisors:
Joost Broekens & Mike Preuss

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

18/07/2020

Abstract

In this paper we report on research on social behaviour in swarm robots that perform the task of item foraging, i.e. collaborative collection of objects. The main research question is: what kind of social reward is favorable to the collaborative item collection, and does the ability to perceive other robots influence this. To study this we developed an experimental setup. In this setup, robots have the task of collecting food. In order to learn the desired behavior, the robots are trained with an elitist evolutionary algorithm that adapts neural network weights based on a fitness of received reward (food collected). First we study the learning process of the robots by searching for the best metaparameters to train the robots. Then we use these metaparameters to train the robots with variations of social rewards, where a social reward is defined as an additional individual reward received when other agents collect food. Our results saw that although the effect is small, social reward combined with perception of other robots has a positive impact on the performance of the robots.

1 Introduction

Swarm robotics is the field of studying multiple simple robots working together without centralized control [2]. The name swarm robotics comes from the term "swarm intelligence" which means: the collective behaviour that comes from a group of simple autonomous individuals interacting with each other and the environment around them . In swarm robotics the goal is to get this behaviour to lead to the completion of a single task [3]. Swarm robots can either perform tasks that are impossible to do by a single (larger/ more intelligent) robot, or tasks that are simply more efficient to do with swarm robots. Characteristics of swarm robots are [3]:

- The robots are autonomous, i.e. make decisions based on their own inputs.
- The robots work together to complete a single task.
- The robots do not have global knowledge.
- The robots can only sense or communicate their direct surrounding.

When it comes to the robots there are three clear advantages of using swarm robots [3] . Firstly the amount of robots is easily scalable, adding or taking away robots is easy since they work autonomously. Secondly, swarm robots come with robustness, a robot can stop functioning without risk of not completing the task. This is because the robots are generally not dependent on each other. Finally it also gives flexibility, this comes from redundancy, simplicity of the behaviors and mechanisms such as task allocation [3]. In this research we will be looking at the task of item foraging. Item foraging finds its origin in nature. A common used example is ants who gather food around their base and bring it back, or bees searching for flowers to gather pollen. In this research however we will only focus on finding the food not bringing it back. This is definitely not a trivial task when it comes to the application of swarm robots [7] and many examples could be thought of where swarm robots have to do some form of foraging. For example robots collecting trash that has been scattered around an area. The robot we will use for this research will use a neural network to control their movement based on inputs from its vision. To train the robot for this task an evolutionary strategy is used. This algorithm uses a fitness score to decide which robots can pass on their weights to the next generation. This algorithm naturally introduces a form of competition between robots, only the best robots pass on their weights. This goes against the principle of swarm robots, working together to complete their task. To investigate if collective behavior can emerge based on a competitive setting, we introduce the concept of the social reward. This gives robots a reward when other robots collect food. In this paper we will research the effect of this social reward and if it can lead to social behaviour in a competitive environment.

1.1 Related work

In this section we will briefly review other relevant studies done on the topic of item foraging with swarm robots. Alan Winfield first proposed the task of item foraging as an engineering science, proving its importance as a benchmark problem for robotics [7]. Foraging with swarm robots has classically been done by robots that use state-transition diagrams. Wenguo Liu et al. presented a adaptation mechanism to automatically adjust the ratio of foragers to resters in a swarm of foraging robots [5]. In this paper also state-transition diagrams were used for the decisions made by the robots. However, the use of a neural network as a controller for swarm robot foraging has recently also been thoroughly researched. In their paper, Timmes et al. [6] proposed a neural-endocrine system combined

with a feed-forward back-propagation layered perceptron neural network for the swarm controller. John Ericksen et al. [4] used Neuroevolution of Augmented Topologies (NEAT) to design a neural network controller for a swarm of homogeneous robots. Both these papers show promising results on the use of neural networks for swarm foraging.

2 Research question

Our research question is: what kind of social reward is favorable to the collaborative item collection, and does the ability to perceive other robots influence this.

With this research question we will be able to study whether collaboration between swarm robots can be stimulated through rewards. This is important because collaboration between swarm robots can be more effective for certain tasks.

2.1 Hypotheses

Using a social reward, i.e. a bonus to fitness when other robots collect food, will lead to increased performance of the swarm robots when they work as a team, as it encourages the robots to work together. Using a social reward that is equal to the reward for collecting food will lead to the best performance as the incentive to collect food is as big as letting other robots collect food. Adding the ability to perceive other robots will lead to an increase in performance with the social reward settings as robots are able to give each other space to search different areas.

3 Method

3.1 The simulation

Performing this experiment will be done through a simulation, which is entirely written in C++.

3.1.1 The robot

The first and most important part of the simulation is the robot. The robots contain a location in coordinates, a speed and a rotation. This is how they move around in the world. The robot is able to see the world around him through casting rays. These rays stop after hitting either food, a robot, a wall or at the max length of the rays. To turn, the robot can change his rotation with an amount of degrees within the maximum turnspeed. This turn speed is the maximum amount of degrees a robot can rotate per step. The maximum turnspeed for the robot is 90 degrees. To move, the robot can choose any speed between zero (standing still) and its maximum speed. The max speed is set to 1 unit of distance per step. To decide how to move and turn, the robot uses a neural network. This neural network consists of an input layer, a single hidden layer and an output layer.
The input layer consists of nine inputs and a bias (see figure 1). These inputs represent the three different observations the robots make: whether it sees food, whether it sees a wall and whether it sees a robot. Each type of observation has three inputs. Each input represents a part of the field of view. The field of view is therefore equally distributed in three parts. For each part the input represents the amount of food in his field of view. The robot can see in 180 degrees in the direction of his current rotation. This means that for the robot and food inputs each input represents sixty degrees of his field of view, the first sixty degrees, the second sixty degrees and the last sixty degrees. These sixty degrees are then split up into twenty rays equally separated. Each of these rays either returns 0 or 1 when it hits food or a robot, depending on which input we are currently calculating. The final input that goes in the neural net is calculated using the following formula:

$$inputfood_i = \frac{\texttt{amount of rays that hit food in section i}}{\texttt{amount of rays}} \quad i = 1, 2, 3 \tag{1}$$

$$inputrobot_i = \frac{\texttt{amount of rays that hit a robot in section i}}{\texttt{amount of rays}} \quad i = 1, 2, 3 \tag{2}$$

When it comes to the input for the wall the robot only needs to know the direction of the wall straight ahead (0 degrees), to its left (-90 degrees) and to its right (90 degrees). However, since we want the robot to be able to react

to coming to close to the wall, we flip the distance by subtracting the distance from the maximum length of a ray. This means that when the wall is not visible, i.e. the distance to the wall is equal to the max ray length, since the robot cannot see further than that, the input will be 0. (raylength - raylength = 0). However when the robot is next to a wall the input will be the same as the raylength. (raylength - 0 = raylength). Finally we normalize this between 0 and 1 by dividing by the raylength. This means that when a robot is close to a wall its input will be higher. Note that we do not need to invert the inputs for food or robots since they are not distance based. These nine inputs and the bias are then fed into the single hidden layer. This hidden layer consists of four hidden nodes and again a bias. We chose to use four hidden nodes because we wanted to keep the robot simplistic but give it enough power to effectively solve the task at hand. The same applies to the fact that we only use one hidden layer.
Finally these hidden nodes lead us to our output layer. The output layer consists of two outputs between 0 and 1. The first number determines the turning of the robot. When this output is 0 the robot turns to the left as far as possible and when the output is 1 the robot turns as far right as possible. When the output is 0.5 the robot will not turn. The second output determines the speed of the robot. When this output is 0 the robot does not move (however it can still rotate) and when the output is 1 the robot moves at max speed. The values for both the turning and speed the values are linear, i.e. 0.5 is half speed and 0.25 is half of a max rotation to the left. The activation function for the hidden and output nodes is the Sigmoid function. We chose this function because the outputs are normalized, this makes it easy to decide whether to turn right or left with the output. The Sigmoid function works by taking the input x and performing the following translation:

$$output = \frac{1}{1 + e^{-1 \cdot x}} \tag{3}$$

3.1.2 The world

Finally, the simulation consists of a world, which contains the robot(s), food and the surrounding walls. The world also has a width and a height. At the start of the simulation the simulation chooses random locations within the width and height for all the robots and food. The world is simulated in steps, every step the robots can move a small amount and rotate.

3.1.3 The food

Food consists of a location in coordinates, x y and a width. The width is necessary for the robots vision, to determine whether the rays hit the food or not. The width for the food in the experiments will be 2.

3.1.4 The evolutionary strategy

For the robots to adapt its behavior and learn we use an evolutionary strategy. Every generation consists of a group of n individual robots. Each individual robot runs his own instance of a world to collect the food in. This means that there is no competition between the robots as they all have their own world. At the end of every generation we use elitism to pick the robot from the group who passes on his weights [1]. This robot passes on the neural network weights to the robots of the next generation with a mutation. This mutation is based on the learning-rate of the robot. The formula of the mutation is as follows:

$$newweight = inputweigth + learningrate * random(-1, 1). \tag{4}$$

Where random(-1,1) is a random number between -1 and 1. This mutation is applied to all weights of the robot. The best performing robot moves on to the next generation without any mutations. This is a classic idea in the elitist approach to the evolutionary strategy [1]. If we do not bring over the best robot to the next generation we might lose the strategy this robot gives through the mutations that are added. This could lead to a decrease in performance. To measure the performance of the robots we keep track of each individual robot's fitness. The fitness of the robot needs to stimulate the robot to perform the task we set out for the robot. In this case the task is foraging, so we need to stimulate the robot to gather the food. To do so we give each robot 100 fitness for each food it manages to gather. We also give each robot a bonus for the distance it has traveled, 0.1 per distance. This encourages the robots to move around which increases their chance of finding food. The fitness will remain the same for all the coming experiments.

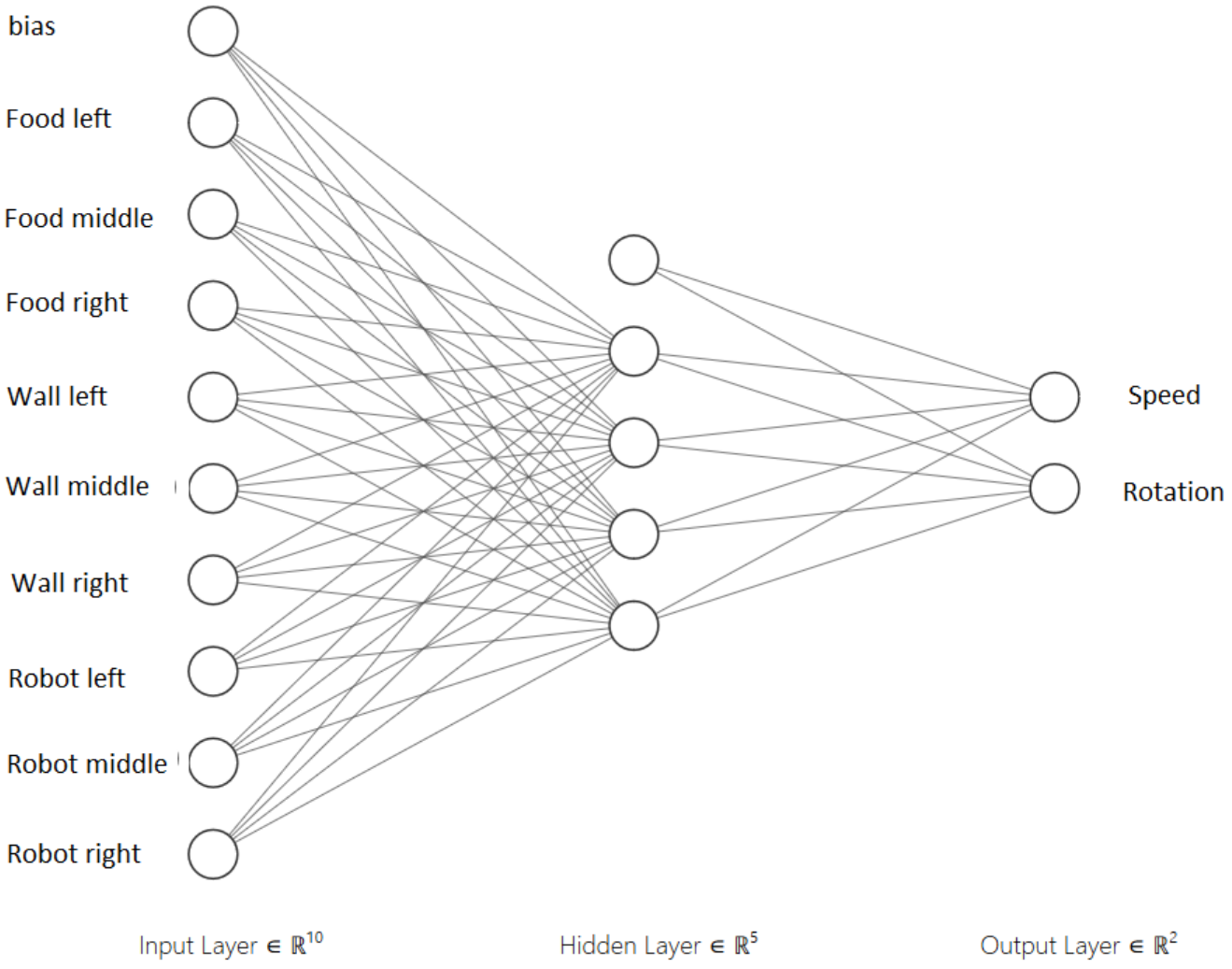

Figure 1: Diagram of the Neural Network used by the robots.

3.1.5 Definition of a run

A single run of the simulation exists of an amount of iterations and an amount of generations. Generations is the amount of times the evolutionary strategy is performed. Iterations are the amount of steps a generation consists of. As mentioned every step a robot can move and rotate. The more iterations there are the more distance the robots can travel to collect the food.

3.2 Experimental setup / approach

3.2.1 Experiment 1

The goal of the first experiment is to determine the best metaparameters to training the robot. To do this we need to start with a baseline, *setting1*. In table 1 the parameters of *setting1* are shown.

In this table we introduce some new terms we need to explain before we move on. Static learning-rate means that the learning-rate does not change over the generations. One of the settings we will be testing uses a dynamic learning-rate which means it gets lower as the generations go on. This can be useful for specializing the robots and

Number of generations	200
Iterations per generation	500
World width	100
World height	100
Number of food	10
Number of worlds	20
Robots per world	1
Learning-rate	0.1
Hidden nodes	4
Static learning-rate	true
Relative learning-rate	false
Ray-length	20
Can see other robots	false

Table 1: Parameters of the baseline.

maybe lead to an increase in performance. Relative learning-rate means whether we take the current weight of the robot in account when updating it. This means that if the weight is smaller the mutation will be smaller and if the weight is bigger the mutation will be bigger. We also mention the ray-length of the robot, this refers to the ray-casting the robot does to see its surrounding. In this case the robot can see 20 units far which is one fifth of the entire world. Finally can_see_other_robots means whether we use six inputs (without the inputs for seeing other robot) or nine inputs. It is important to note that not seeing other robots reduces the amount of weights needed to train the robot.

3.2.2 Experiment 2

The goal for experiment 2 is to test the effects of adding a social fitness to the fitness of the robots. This means that the robots gets an extra amount of fitness when another robot collects food when both robots are in the same world. For this we will use eight settings. Four of the settings contain robots that are able to see each other and the other four contain robots that are not able to see other robots. The following four settings are the same for both the settings with and without robot vision. The first of these four is the default setting without social reward. The second setting has a social reward which is half of the normal reward for eating food. The third setting has a social reward which is equal to the reward for eating food. The final setting has a social rewards which is double the reward for eating food. For each of these settings we will use the best performing setup from experiment 1. Furthermore, the settings will contain three robots. You can see all eight settings summed up in Table 2.

Without robot vision	With robot vision	name
0 social reward	0 social reward	no social reward
50 social reward	50 social reward	half social reward
100 social reward	100 social reward	equal social reward
200 social reward	200 social reward	double social reward

Table 2: The eight different settings we tested in experiment 2. The reward for collecting food is 100 fitness. The far right column shows the names we will be using for these settings in the paper.

3.3 Measures

To compare the performance of the robots we will use the fitness of the single best performing robot of each generation. Because of the mutations each generation comparing the averages could give a false perspective since the mutations could reduce the performance of the robots. Therefore we look only at max performance per generation. Per setting we use a graph containing the max fitness per generation. These graphs will show the learning curve per setting. We will compare the curves to draw our conclusions. To make our data less sensitive to outliers we take the average of 10 runs for each setting we test.

4 Results

4.1 Experiment 1

Figure 2: Results of experiment 1

Setting 1	Baseline
Setting 2	See other robots = true
Setting 3	learning rate = 0.2
Setting 4	relative learning rate = true
Setting 5	dynamic learning rate 0.1-0.01
Setting 6	150x150 world

Table 3: Settings of experiment 1.

In Figure 2 we can see the results of the first experiment. In Table 3 we summarise the different settings. The first thing we notice when we look at the graphs, is the big variety in the starting fitness. The highest starting setting already starts at a fitness of 500. This is equivalent to an average of five food collected per world on average. This is a really big difference from the lowest average, 250, which equals an average of two to three food collected per world. In the starting position everything is random, the robots weights, position and the position of the food. This tells us that the random starting position of the robots and their random neural network can have a lot of influence on the initial fitness. This means that we sometimes cannot directly compare the absolute fitness between the different settings when the difference in starting fitness is too large. What we can compare is the learning curve of the settings and the relative increase of fitness per setting. Looking at the setting that reached the highest fitness we see that *Setting1*, the baseline, has the highest ending curve. Starting off at an average fitness of 380, after 200 generations this increased to an average of 700. This is increase of almost 200%. We can compare this directly to *Setting 4* whose curve also began at 380. The curve of *Setting 4* however ends at an average of 530. When we also compare the curve of *Setting 4* to setting1 we see that the curve of *Setting 1* goes up very quickly, where the curve of *Setting 4* only starts improving after 100 generations. Therefore we can rule out *Setting 4* as being the best setting for training the robots as *Setting 1* is performing better. Comparing *Setting 1* to *Setting 5*. We can see the the relative increase of *Setting 1* is higher. This is obvious because it starts at a lower fitness and ends at a higher fitness. However we can not yet dismiss *Setting 5*, because for a long time *Setting 5* follow the same trajectory as *Setting 1*. But in the end

Setting 5 appears to lose out on *Setting 1*. This can be explained by the dynamic learning rate of *Setting 5*, because as the generations go on the learning rate becomes smaller. This means that in the final generations the performance increase will get smaller. So in conclusion instead of helping the performance by specializing the robots with a smaller learning rate in the end, *Setting 5* actually loses performance by not being finished learning before the learning rate gets too low. When we look at *Setting3* we can see that its initial performance is very low, 180. This means there is a lot of room for improvement. In the end the average performance gets to 480. This is more than double the initial performance. This is better than the performance gain of *Setting 1*. However, when we look at *Setting 1*, even though its initial performance was already high, the curve is still very steep for the first 40 generations. Where the curve for *Setting 3* is more linear. Based on the shape of the curve we choose that the performance of *Setting 1* is actually better. Comparing *Setting 1* against *Setting 2* appears to be trivial. The performance of *Setting 2* has not increased over 200 generations. *Setting 6* has a bigger world than *Setting 1*. Therefore comparing the fitness is not fair as food is harder to come by. The point of *Setting 6* is to see the performance of the robots on a bigger world. Looking at the curve of *Setting 6* we can see that the robots are not able to come close to collecting all food, therefore we decide that the problem is too hard for the purpose of training the robots.
One thing that is very noticeable in all the curves, is that, except for *Setting 3*, they all go horizontal after around 50 generations. Because we want the robots to be able to solve the problem, consistently collecting all, or almost all food, we introduce a new setting, *Setting 7*. Looking closely at the generational algorithm we discover a bit of a problem: the first generation of random robots the algorithm chooses the best performing one to give his weights to the next generation, therefore all future generations will be mutations of that robot. When this robot has a strategy that cannot easily be adapted to make progress in collecting the food this becomes a problem . *Setting 7* is set to reduce this problem, *Setting 7* will not use the top 1 robot per generation, but the top 5 robots per generation. Mutations of these robots will be evenly spread throughout the next generation. The single best robot still moves on to the next generation. This means that randomly one of the top 5 best robot has one less offspring in the next generation to make place for this best robot.

Figure 3: Results of experiment 1 with setting 7

In Figure 3 We can see that the performance of *Setting 7* is much better than the other settings. Comparing this to *Setting1*, our previously decided best setting, we firstly see that the begin performance is about equal. However, the end performance is almost 100 fitness higher. Looking at the curve we can also see that after generation 50 there is still a big improvement in fitness.
Increasing the amount of robots we choose per generation is not the only adaptation we can make to our algorithm. As we mentioned the problem before being stuck with only the best robots out of the first 20 can be limiting to the performance. Therefore we introduce one final addition to solve this problem: each generation we add in a random

robot. This means that we do not have to commit to the first 20 random robots but have more robots to chose from in the very important first generations. We will add this to *Setting 7* and compare the performance for different parameters.

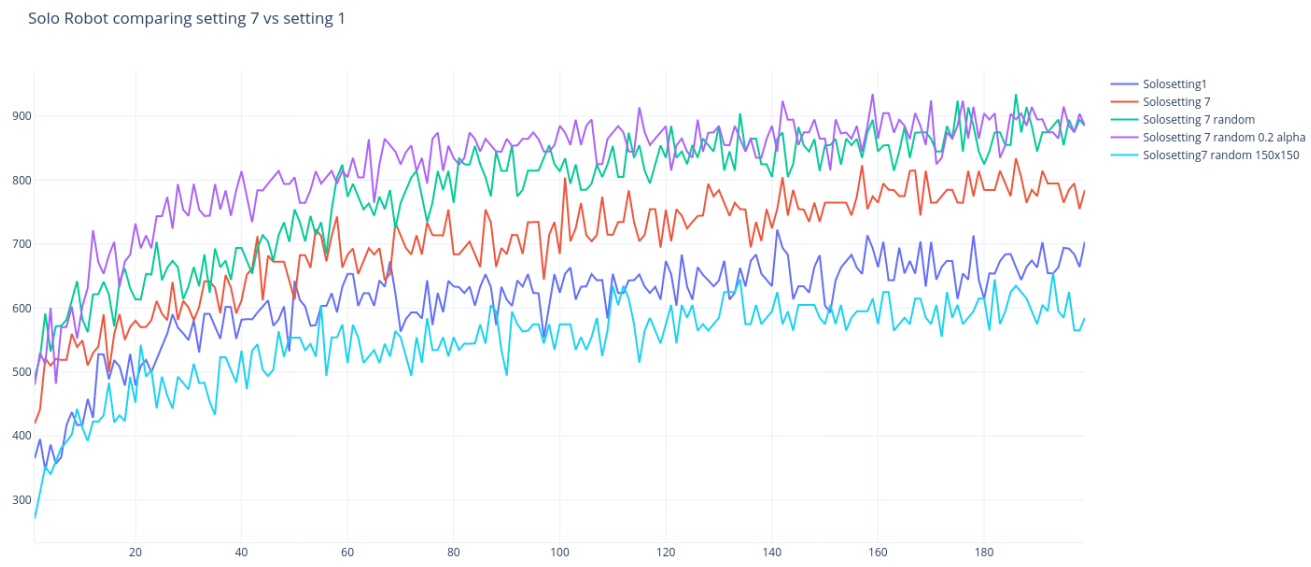

Figure 4: Comparing setting 1 vs setting 7 vs setting 7 random

Looking at the results of figure 4 We can see two settings scoring the highest fitness, *Setting 7* random and *Setting 7* random with 0.2 alpha. Both settings at the end of the curve when they are optimal are able to consistently score 100 fitness more than *Setting 7* without random robots. The final decision is between a learning rate of 0.1 or a learning rate of 0.2. Looking at both curves there is one clear difference, *Setting 7* with a learning rate of 0.2 has a steeper learning curve within the first. Because the performance is otherwise similar we will finally choose *Setting 7* with a random robot and 0.2 learning rate to be the optimal way of training the robots in this situation out of all methods we have tested. One thing that is also remarkable about these results is the performance of *Setting 7* with a random robot on the 150x150 world. The performance is almost as high as setting1 on a 100x100 world. In Table 4 the setting we will be using for experiment 2 is summed up.

Final Setting 7
Baseline
5 best robots
learning rate = 0.2
Random robot each gen
150x150
3 robots per world

Table 4: Outline of the setting we used in experiment 2.

4.1.1 Experiment 2

As described in the experiment 2 subsection of the measures section we will be exploring the effects of adding a social reward to the fitness of the robots. For this social reward to take an effect we need more robots in the same world, otherwise there are no other robots to collect the food for the social reward. To do so we will from now on have three robots per world. However, this causes a bit of a problem. As we have seen one robot on its own is able to consistently collect nearly all food. When we add two robots to the world we simplify the task. One robot will

now on average only need to collect three to four food. Therefore we will have to make the task harder. We do so by making the world bigger and from now on will use a world size of 150x150.
One problem that comes from adding more robots to a world stems from the way we pick the 'best' robot. Now we have three robots in one world, which do we pick to pass on their weights? Our solution to this is using identical robots. This means that each world only has one pair of weights to pass on to the next generation. One reason for this is that, if we use non-identical robots there is not one of the three robots limiting the performance of the other robots since they are all equal.
When we want to compare the performance of the robots we previously looked at the maximum fitness per generation. Now that we have multiple robots in the world who are working together we do not want to look at the fitness of just one robot but we want to look at their collaboration. In the end, the goal of the task is to collect the food, so group performance should be based upon that. Therefore we introduce two new measures, maximum food collected per generation and average food collected per generation. Note that because we are altering the amount of fitness gained per food by adding the social rewards it becomes challenging to compare the amount of fitness as we used to.
For the evolutionary strategy we will now look at the best robot of each world as candidates to reproduce. This means that only one robot per world can be chosen to pass on the weights to the next generation. This is, however, not a problem since the robots are identical, therefore no genes of well performing robots will be lost.
The final change we will be making compared to experiment 1 is that we look at 120 generations instead of 200. From the graphs of figure 4 we can clearly see that after generation 120 there is no significant progress if any at all.

4.1.2 Results without vision of other robots

First we will look at the results of the experiment without vision of other robots. In figure 5 we first look at the baseline of the results, no social reward. Looking at the curve for the max food collected we see that for the first 40 generations the progress is linear, starting at 3 max food collected going up to 8. After this there is no further progression as the average fluctuates around between 8 and 9 max food gathered. For the average food collected line we see the same progression as the max food collected. Firstly comparing this to the setting with half social reward, we immediately notice a much more rapid progression in the first generations. Starting off at three again the half social reward is up to 6 max food collected within 5 generations, where it took the no social reward robots 15 generations to reach 6 max food collected. After this rapid incline of the graph we see a linear increase also until generation 40 where the robots reach a max of between 8 and 9 food collected. Looking at the setting with equal social reward we see a similar trend as we see with the half social reward setting. There is also a rapid increase in the first 5 generations quickly reaching an average of 6 max food collected and the same linear increase afterwards. However this setting reaches an average of 8 max food collected after 20 generations instead of 40. Finally looking at the double social reward we notice a trend more similar to the no social reward. Looking at the progress after 40 generations we see that the average is still only around 7 max food collected, where the other graphs all already had averages around 8 max food collected. Only after generation 80 the double social reward setting consistently goes over 8 max food collected. This shows that this setting actually has a harder time learning the item foraging task than the other settings with or without a social reward. From these graphs we draw the following conclusions. First of all adding a social reward that is equal to half or to same as the reward for collecting food increases the speed in which the robots learn. Moreover, adding an equal social reward leads to the fastest learning out of all. We can explain this result in theory when we look at the effect of the equal social reward setting compared to the no social reward setting. In the equal social reward setting, because the robots get an equal bonus whether you are the one to collect the food or whether another robot collects the food, the robots all have the same fitness. This fitness is then equal to the total amount of food collected. This means that we do not pick the robots based on their individual performance but the performance of the group of robots. In this situation to collect the most food, the best strategy is to work together which is therefore encouraged by the equal social reward, by changing the way we pick robots from a individual basis to a group basis. Finally adding the double social reward does not increase performance but leads to a decrease in performance by slowing down the speed the robots learn. These results we expected and we can also explain them. Adding the double reward introduces a dilemma for the robots that they have a hard time solving: The robots want to collect food but they also want the other robots to collect food since that gains them more fitness. However, since the robots are clones they cannot develop a strategy where they let one robot, for example, find all the food. Since the robots are clones the same robot wants to collect food but at the same time also not collect food.

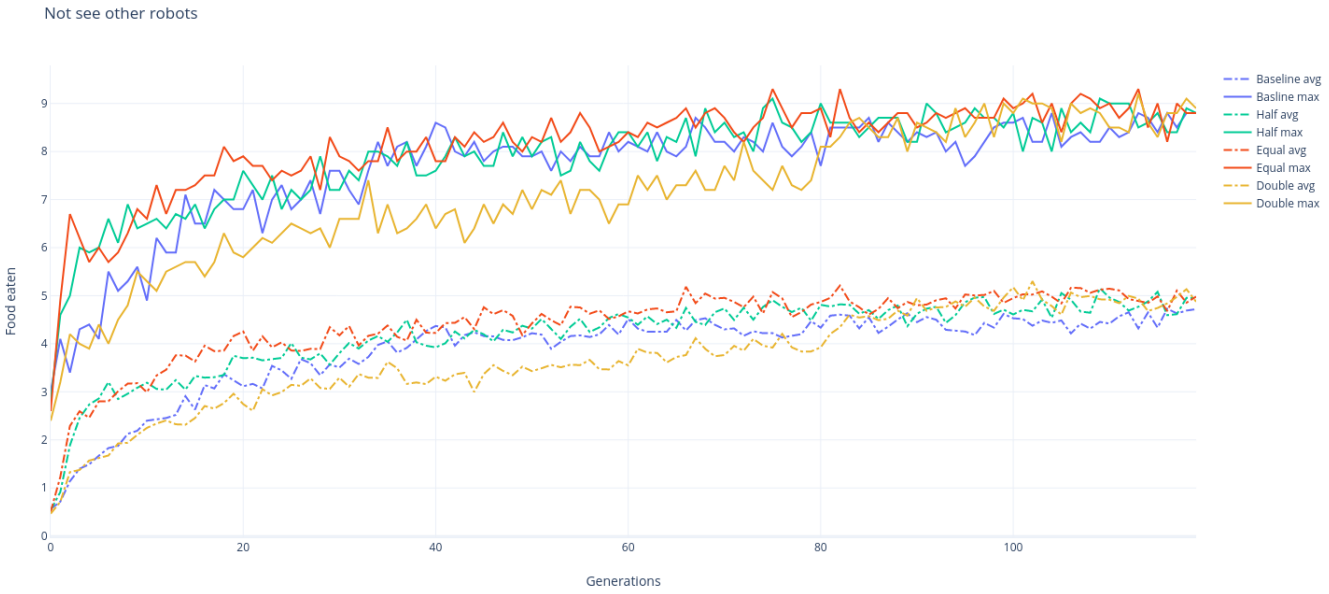

Figure 5: Maximum food collected and average food collected per generation, without robots being able to see eachother.

4.1.3 Results with vision

Now we take a look at the results of the experiment where the robots are able to see each other. Adding this form of 'communication' between the robots we expect the results to be slightly better than the results without vision. However, we did already see that the robots without vision were almost able to completely solve the problem, so there is not much room for improvement left. At the very least these robots should be performing equally to the robots without vision. First we will look at our baseline, no social reward. We can compare this to the baseline without vision in figure 7. In this graph we can see all the max fitnesses compared to each other giving us an idea how the settings without robot vision compare to the settings with robot vision. Here we can see that the baseline's have a similar learning curve for the first 40 generations. After the robots performance stops increasing the baseline with robot vision reaches a maximum of 8 max food collected. This is slightly less than the peak performance of the baseline without vision. This shows that the performance of the baseline slightly decreased when we added the ability to see other robots. When we compare the baseline with robot vision to the half social reward setting with robot vision, we see the same steep learning curve. However when the robot is done learning it is able to consistently have a max food collected of almost 9. This is an increase of almost 1 food collected over the baseline. Looking at the equal social fitness setting, we see the same curve as with the half social fitness setting. Finally looking at the double social reward setting with vision we see an improvement on the double social reward setting without vision in figure 5. In this figure we see the same curve we saw in the other settings where the robots were able to see each other, whereas in the setting where they were not able to see each other we saw a worse curve than the other settings. This can also be seen in figure 7. We see the same effect to a slightly lesser extend with the equal social reward setting and the half social reward setting where the settings with robot vision are performing slightly better than the versions without robot vision. This can be seen in figure 7 by the dotted lines being above the solid lines of the same colour.
The first conclusion we draw from these results is that adding vision of other robots has made the problem easier. We conclude this based on the performance of the double social reward setting. Without vision the double social reward setting has a lower curve then the other settings as we observed. This means that the double social reward setting has a harder time learning the problem without seeing other robots. However, when we added the ability to see other robots, the double social reward setting was not performing worse than the other settings anymore. One thing we can conclude now, that we also observed in the results without vision, is that the settings that had the social reward bonus were able to reach a slightly higher max food collected than the baseline without the social reward.

We can clearly see this in figure 6, where the blue line which represents the baseline is the lowest line out of all. From this we conclude that adding vision in combination with a social reward can lead to better results than only adding vision. Now that we have all the results with vision and without vision we can look at the best value for the social reward. Looking at both the results with and without vision we see that the equal social reward setting has the steepest learning curve. Furthermore, looking at the results after the robots are done learning, we see that the equal social reward has the best performance of all the settings, being the highest line. For this reason we consider a social reward that is equal to the reward of collecting food to be the best value for the social reward. It performed the best for the settings with and without vision. Half social reward is not far behind, however in the setting with vision the learning curve is lower than the equal social reward. When the robots are done learning, however, the results are the same as the equal social reward. Therefore we consider this the second best option. When it comes to using a double social reward this setting can only be used in combination with vision. If the robots do not have vision this social reward will lead to a dilemma for the robots, slowing the learning rate of the robots.

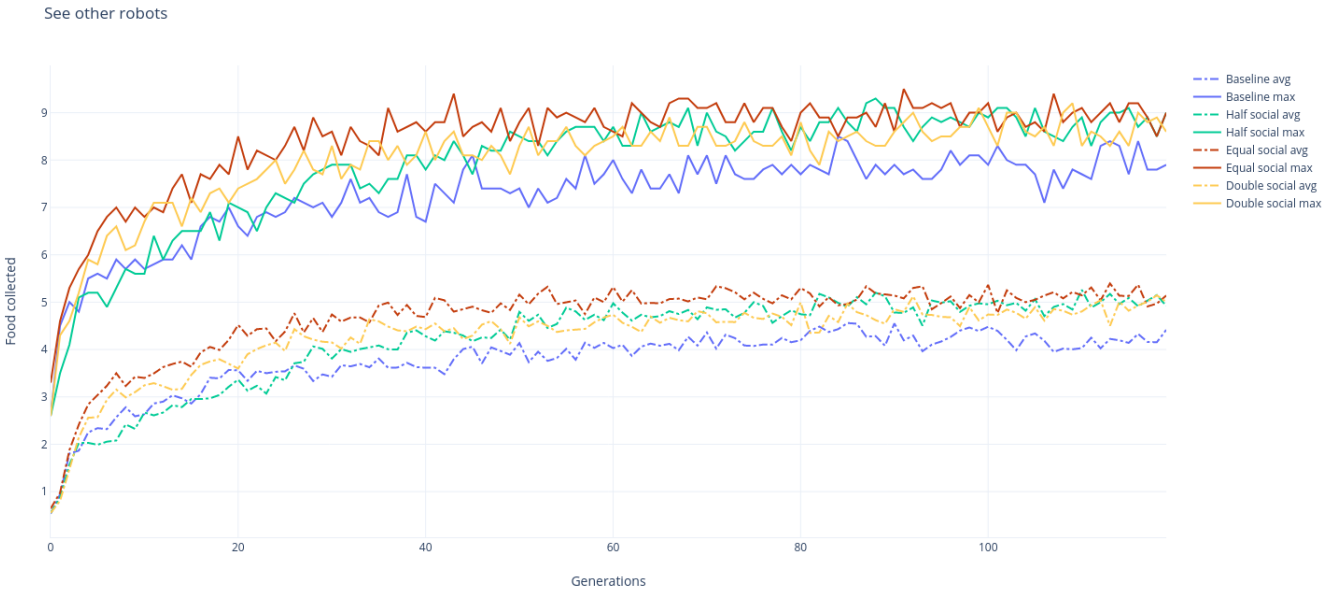

Figure 6: Maximum food collected and average food collected per generation, with robots being able to see each other.

5 Conclusion

In this paper we have looked at the effect of a social reward on the performance of swarm robots in the context of item foraging. From the results we conclude that there is a small benefit to adding this social reward as it can slightly increase the speed in which robots learn the problem. We also conclude that the best value for the social reward is equal to the reward for collecting food. From the results from experiment 2 we can also conclude that adding vision in combination with a social reward leads to better results than just adding vision. From this we conclude that adding a social reward to swarm robots performing the item foraging task can lead to increased performance through the robots evolving to collaborate with each other. We also saw that for this collaboration to form we need some sort of communication between the robots, in this case being vision. We also saw that giving the robots vision of the other robots decreases the difficulty of the item foraging task. When it comes to the training of the robots we observed that giving the evolutionary algorithm more options to pick robots, by using more robots to reproduce and adding a random robot each generation, results in the best chance of the robots being trained optimally. We also concluded that the best way to use the learning rate is to keep it static, i.e. not decreasing over generations, and not have the mutations be relative to the current size of the weights, i.e. bigger weights get bigger mutations and vice versa.

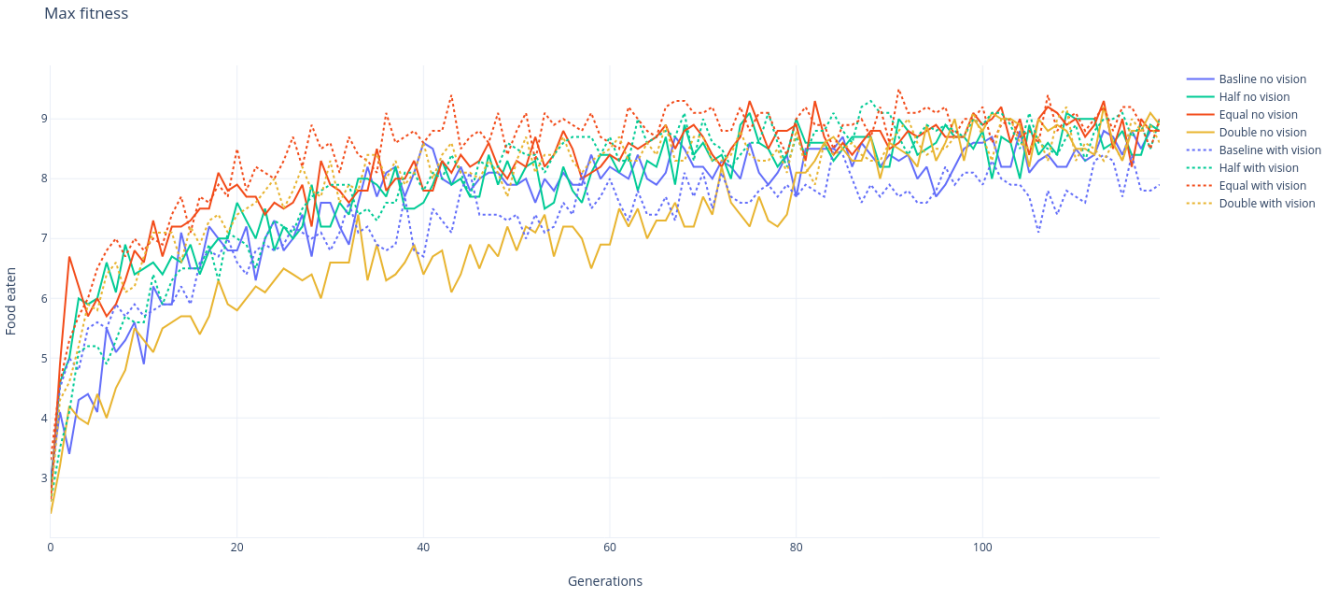

Figure 7: Comparing max food collected between robots with and without the ability to see other robots

6 Future work

When it comes to future work on the effect of social reward on the task of foraging with swarm robots a few new options can be considered. First of all we think the most interesting change that could be made is using a single large world instead of multiple worlds like we used. This could lead to more cooperation between the robots and a bigger effect of the social reward. In such a setting also research could be done on comparing a local to a global social reward. I.e a local social reward would mean closer robots would get a bigger social reward than distant robots. It can be interesting to see if different forms of collaborations stem from the difference in local vs global social reward. In the setting we used in this experiment also research could be done with the use of individual robots instead of identical robots that we used in this paper. This could lead to the robots creating interesting new strategies to work together. However as discussed in the paper, the way to use the evolutionary algorithm on these robots is not so straightforward. When it comes to the topology of the Neural Network used there is also a lot that can be explored. For example, the amount of hidden layers and hidden nodes. Also it could be interesting to use a neurocontroller created by NEAT as done by in the reserach of John Ericksen et al. [4]. Finally also the shape of the world can be changed or obstacles can be added to the worlds to give the robots a different challenge and perhaps increase the difficulty of the task.

References

[1] Chang Wook Ahn and Rudrapatna S Ramakrishna. Elitism-based compact genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 7(4):367–385, 2003.

[2] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. Swarm intelligence : From natural to artificial systems / e. bonabeau, m. dorigo, g. theraulaz. 01 2001.

[3] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, 2013.

[4] John Ericksen, Melanie Moses, and Stephanie Forrest. Automatically evolving a general controller for robot swarms. In *2017 IEEE symposium series on computational intelligence (SSCI)*, pages 1–8. IEEE, 2017.

[5] Wenguo Liu, Alan FT Winfield, Jin Sa, Jie Chen, and Lihua Dou. Towards energy optimization: Emergent task allocation in a swarm of foraging robots. *Adaptive behavior*, 15(3):289–305, 2007.

[6] Jon Timmis, Lachlan Murray, and Mark Neal. A neural-endocrine architecture for foraging in swarm robotic systems. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, pages 319–330. Springer, 2010.

[7] Alan FT Winfield. Towards an engineering science of robot foraging. In *Distributed Autonomous Robotic Systems 8*, pages 185–192. Springer, 2009.