Final Project of INFO-H-414

Jules Nicolas-Thouvenin

Université Libre de Bruxelles, Belgium

Abstract. Foraging tasks are common applications for swarm robotics. In this project, we develop a robot controller able to search for objects in an arena and bring them to a nest. The analysis shows that our controller is easily scalable but very little flexible. We then optimize some parameters of the controller using a fully informed PSO algorithm. This optimization however doesn't improve the performance of the manually tuned controller.

1 Introduction

The first part of this project consists in the design, implementation and testing of a robot controller, using the Simulator Argos. The robots must perform a foraging task, and bring objects located in the arena inside a nest. The controller needs to follow the guidelines of Swarm Robotics and thus only use local information to control the robots.

The second part consists in optimizing the controller developed in part one. Parameters of the controllers are selected and tuned by a fully informed Particle Swarm Optimization algorithm. The PSO parameters will be also tuned.

2 Arena

We chose to design our controller for the arena number two. A wall divides the arena in two rooms. At the center, a door in the wall allows robots to switch rooms. The objects are located at the bottom of the first room. The nest is located at the top of the other room. A light is placed at the center, just above the door. In the rest of the report, we will mention this door using the term cache area.

We chose this arena for two reasons. First, the two rooms of the arena allowed us to implement a division of labor. Some robots have to bring objects to the cache, the others have to bring the objects from the cache to the nest. Second, the light source above the cache helped guiding the robots towards this area. We thus found the types 3 and 4 very difficult to work with. Indeed, in both scenarios, the cache cannot be found simply using the light source. And we found the first scenario less interesting according to the fact there was no wall at all.

3 Design of the controller

As mentioned in the previous section, the main idea behind the controller's design is the division of labor. There are two types of robots, the hunters that explore the first room looking for objects and bringing them into the cache area, and the nesters that bring the objects from the cache area into the nest.

The first phase of the execution consists in shuffling the robots in the center of the arena. Robots reach the light source and avoid each other. This allows a fair distribution of robots in each room. This step last a few iterations (e.g. 50). After that robots will not be allowed to switch rooms.

The second phase of the execution is focused on discovering the job of each robot. If the robot is located in the object room, he is hunter. Otherwise he is nester. Each robot has a number of iterations (e.g. 300) to figure out its job. After that, the robot will keep its job. During these first iterations, the robots don't touch the objects and don't cross grey areas (e.g. cache and nest). First all the robots are nesters. If a robot sees an object, he switches to hunter. The controller doesn't support adaptive division of labor, as the robots keep the same job during the execution. We thought that the controller was complex enough, and that the results were already satisfactory.

The description of the controller reported below focuses on the main ideas to avoid polluting the explanation. More details can be found in the code.

3.1 Hunter's controller

The hunter's controller is designed as the following. Note that a hunter can never cross a grey area. This allows a proper separation between the two groups.

Exploring The robot performs a random walk with obstacle avoidance, looking for objects. If he sees an object then he walks towards it and grabs it.

Grabing The hunter has grabed an object. Now he walks towards the light. There are two ways of triggering the drop of the object. First if the robot walks into a grey area (cache). Second if the robot has been transporting the object for two long (e.g. 50 steps). When the robot needs to drop the object, the gripper is unlocked and the robot switches to a "walk away" state.

Walk Away The hunter walks away from the light source during a number of steps (e.g. 36). This ensures that the robot won't grab the same object over and over. In this state, the hunter isn't attracted to objects anymore. After the number of steps needed, the robot switches back to the "exploring" state.

3.2 Nester's controller

The nester's controller is designed as the following. Note that, excepted during the states "caching", a nester is not allowed to walk into the cache. In fact, the nester is capable of distinguishing the cache from the nest. Whereas the hunter

only sees the cache, the nester have access to both grey areas in it's room, and thus can identify which area is the nest.

Waiting The nester is waiting for objects to be brought to the cache by hunters. The robot walks randomly close the light source. If he detects an object, he walks towards it and try to grab it. If he can grab it, he locks it's gripper and switches to the state "grabing nesting". On the opposite, if he finds itself walking into the cache while targeting an object, he switches to the state "caching". Robots that have spend too much iterations waiting switches to the sate "finishing" where they walk towards the nest to see whether there are objects remaining. This helps spotting forgotten objects.

Caching and Caching back These two states allow the nester to step into the cache area and grab objects. This state is to be performed carefully because if the nester finds itself in the other room, he can create mess in the arena. Thus the robot is only allowed to walk a limited number of steps (e.g. 3) and after that limit, he walks backwards to come back to the nester's room. If he succeeds in grabing an object he will switches to the state "grabing nesting". Otherwise he switches back to the state "waiting".

Grabing nesting The nester has grabed an object. He walks backwards and drop the object if he is in this state for too long. If he walks into the nest he drops the object in it properly using a nester-specific state "unloading" which we won't describe in this report. After droping the object the nester switches to the state "walk away nesting".

Walk Away nesting The nester has recently droped an object. He isn't attracted to objects anymore. He walks towards the light source.

4 Implementation of the controller

The behavior of the robots are dictated mainly by two global variables : the job of the robot, hunter or nester, and the state (e.g. exploring, grabing...). For each pair of values of these two variables, a specific set of instructions are given to the robot. This assignment of actions is done by a router, composed of conditions on the job and the state of the robot. More information can be found in the code.

5 Results for the manual solution

The manual solution works well using only 3000 argos steps. We thus decide to fix the maximum number of iterations to 3000 for all the remaining experiments. This will fasten the optimization part of the project.

5.1 Scalability

To study the scalability of the manual controller, several swarm sizes are compared on the chosen arena (here 2). The controller is executed with swarm sizes going from 2 to 20 and sizes of 60, 100, 125, 150, ten times with a seed going from 1 to 10.

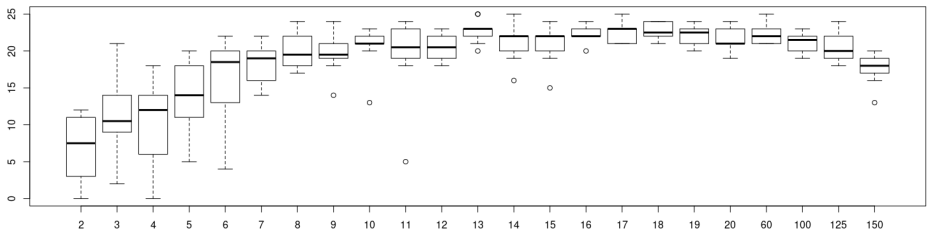

Fig. 1. Number of objects in nest depending of the number of robots

We can see that the performance scales reasonably well. The performance is strongly increasing between sizes of 2 and 13 robots. After that stage, the performance stays similar and starts to decrease when using more than 100 robots. We can explain these observations. It takes sometimes several robots to put an object to the nest. First one robot will put it at a wrong place. Then another will grab it and try putting it in the nest. If the swarm is too small, robots don't have time to do this properly for all objects. On the opposite, too large swarms create strong interferences and slow down the movements of carrying robots. Note that with more than 100 robots, the space between each one is very little. A bigger arena could possibly increase this limit.

For the rest of the experimentation, we will fix the number of robots to 13. This size allows very good results and is small enough to allow quick runs of Argos.

5.2 Flexibility

To study the flexibility of the manual controller, we run the controller ten times on the four scenarios.

The first scenario obtains similar if not better results than the second scenario. This is not surprising according to the fact that the light source, cache and nest are located at the same place in the two arenas. The absence of walls seems to facilitate the transfer of objects from one room to the other.

However the controller doesn't work on the scenarios 3 and 4. Again, this can be explained by the fact that the controller strongly depends on a central cache and light source. Adapting the controller to these scenarios would mean getting rid of the assumption that the cache is under the light, and that the arena is divided in two equal rooms. We could imagine a path finding algorithm to allow

the robots to go from the objects to the nest. Another idea could use the range and bearing sensors : robots that have just left the nest could communicate the proximity of the nest to others, and guide robots coming back with objects. These ideas are independent of the structure of the arena and thus will probably perform with more flexibility.

6 Controller optimization

The second part of this project aims to optimize the performance of the robot controller. We thus define the following optimization problem : finding the configuration of parameters for the robot controller to maximize the number of objects in the nest at iteration 3000. The number of robots has been fixed to 13 thanks to the scalability analysis. And the arena stays number two. The parameters configuration will be explained in the next section. The algorithm used for solving this optimization problem is a fully informed PSO.

7 Selection of the controller parameters

Speed The parameters "Speed" and "Speed walk-away" dictate how fast the robots move in the arena. If the robots move too slow, they won't do the task in time. On the contrary, if they move too fast, they will create mess on the arena. It is thus crucial to well tune these two parameters.

Space allocation The parameters "distance avoid robot" and "light value finishing" influence how robots move in the arena. Distance avoid robot dictates the maximum distance that can stand between two robots before they start avoiding each other. The more the parameter is high, the more the robots are far from each other. A tight and sparse swarm behave very differently and can obtain different results. That's why we chose this parameter. Light value finishing dictates the area where the nesters in state "finishing" can move. The smaller it is, the further nesters have to move from the light source. Because we can't really tune this parameter ourselves, we prefer to include it in the set of tuned parameters.

Counters limits The four remaining parameters dictates how long a robot can spend on a specific task. The first two parameters "reach light" and "start job" dictates the length of the two first phases of the execution mentioned in the controller description. "reach light" describes the end of the phase where robots are shuffled in the center, and "start job", the end of the phase where robots learn their job. If these parameters are too low, the robots don't have time to shuffle properly and learn their true job. But setting them to high might be a waste of time. "to finishing" and "to waiter" describes the maximum number of iterations that a robot can spend in the states "finishing" and "waiting". These parameters consist in a trade off between the time spent by nesters close to the cache and the time spent far from it. It is hard to tune these counters manually. We preferred tuning them using PSO.

8 Fully informed PSO

The implementation follows the algorithm of a fully informed PSO. Each particle is updated according to its position, velocity, personal best position and the personal best positions of its neighbours. We implemented three common topologies : ring, wheel and global best. In ring topology, the particles are neighbours with two particles, forming a ring. In wheel, each particle is neighbour of one central particle. In global best, all particles are neighbours to each other. A solution for the PSO algorithm is vector of 8 continuous variables, one for each controller parameter. Each variable having a lower and upper bound.

Each of the height variables have different scales. Thus we define unique bounds for each variable. The bounds for each variables are the following : speed $\in$ (50, 150), speed walk away $\in$ (50, 200), light value finishing $\in$ (0.9, 1), distance avoid robots $\in$ (50, 150), reach light $\in$ (40, 100), start job $\in$ (200,500), to finishing $\in$ (50, 200), to waiter $\in$ (50, 100).

8.1 Integration of PSO with Argos

The integration of PSO and Argos is done using subsidiary files. PSO executes Argos, Argos writes the number of objects in nest in a file, and PSO reads the file to get the evaluation.

The disposition of robots and objects in the arena can have a large impact of the controller performance. Thus one PSO solution can have various evaluations using different Argos seeds. To make the evaluation more robust and thus the PSO algorithm more efficient, the evaluation runs Argos on three different "arena seeds" and return the mean of the three different outcomes. For example for the three "arena seeds" 100, 101 and 102, one PSO solution X executed on argos returns 24,16,23. The evaluation of the solution is thus 21. This new evaluation procedure takes thus three times more computation time than a classical evaluation procedure. Thus, we had to keep the number of evaluations to 100 and limit the number of PSO runs during the experimentation.

8.2 Tuning the parameters of PSO

Due to the original optimization problem and the stochastic nature of the evaluation function, we didn't want to use existing PSO parameters. We decided to tune them using the following protocol. We tested swarm sizes 5, 10, 15 and 20 with the topologies ring, wheel and global best. Each configuration was executed two times. Wilcoxon tests weren't conclusive so we selected the configuration (swarm size = 10, topology = gbest) obtaining the best results on average. The box-plots can be found in the results/tunningPso folder.

9 Comparison of the *manual* and *optimized* solution

In order to compute the optimized solution, we execute PSO ten times. The ten solutions are then executed thirty times on Argos and we select the solution that performs the best. Results are reported below via box-plots.

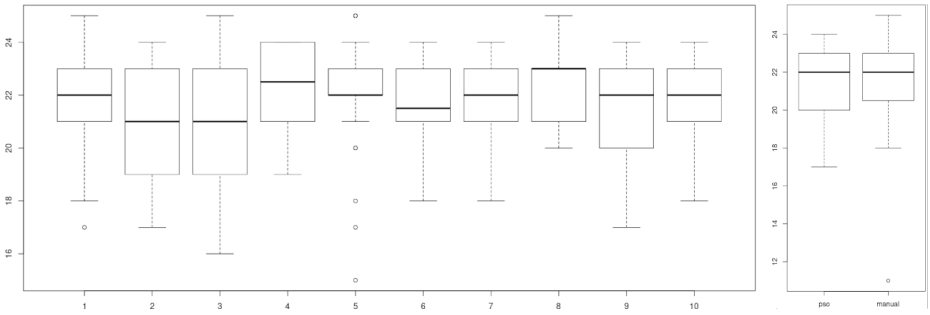

Fig. 2. Left. comparison of the ten PSO solutions, **Right.** Comparison of the pso and manual solution

The solution number 8 which has better results on average than the other nine. Wilcoxon tests comparing this solution to the others are not all conclusive but we still select this solution as our *optimized* solution.

Is reported here the two solutions compared (pso, manual) : speed = (100, 131), speed walk away = (120, 162), light value finishing = (0.9, 0.94), distance avoid robots = (75, 90), reach light = (100, 93), start job = (300,206), to finishing = (100, 95), to waiter = (100, 74).

Finally, in order to compare the optimized and the manual solution, we run the following protocol : we execute twenty times the selected optimized and manual solution. The wilcoxon test isn't conclusive, but we observe better performance on average for the manual solution.

There is multiple reasons behind this result. First it can explained by the too large search space defined by the bounds of the PSO variables. Also the fact that during the development of the controller, we tuned with attention the parameters which led to this high performance. Again, the stochastic nature of the evaluation function makes it hard for PSO to find a very good local optima. In addition, maybe the choice of the controller parameters to optimize was not optimal. Indeed it is possible that we selected the wrong parameters, or simple too many.

10 Conclusion

We developed a robot controller able to perform a foraging task on a specific arena structure. The controller which lacks flexibility as it depends partly on structural properties of the arena, possesses however a good scalability. The optimization process of the controller didn't improve the overall performance of the manually tuned controller. We currently run the controller three times to evaluate a PSO solution. We think adding more runs in this evaluation process can help the search of PSO by reducing the variance of the evaluation.