Platform for Simulation and Improvement of Swarm Behavior in Changing Environments

Sergi Canyameres, Doina Logofătu

Platform for Simulation and Improvement of Swarm Behavior in Changing Environments

Sergi Canyameres[1], Doina Logofătu[2]

[12]Computer Science Department of Frankfurt University of Applied Sciences 1 Nibelungenplatz 60318, Frankfurt am Main, Germany
logofatu@fb2.fh-frankfurt.de

Abstract

Simulation of particle movement plays an important role when understanding the behavior of natural entities, which can be adopted by many other research fields. The complexity of every environment can cause lots of difficulties to ensure accurate und faithful results, leading to extensive researches but always in very specific conditions. This paper describes a framework which allows wide flexibility on the moving algorithms of the swarm particles as well as an easy modification of the surrounding constrictions. A simple, intuitive interface is implemented and a swarm paradigm is already used. The aim of this tool is to obtain acceptable results in any upcoming test case without the need to run an exhaustive and expensive research.

1 Introduction and Motivation

This paper explains the natural extension of our previous participation in the InformatiCUP challenge [1], a German competition arranged by the "Gesellschaft für Informatik" [2], an organization promoting the transmission of computer science knowledge and research. The goal in this 2014 edition [3] was to implement an ideal simulation of how a set of manganese-collecting machines had to move above a water surface without remote help on how to determine the other's position and direction. Only a small area is reached by radar that provides them information about the relative position of the nearest bots within this range. Although the purpose was to collect as much manganese as possible, the heuristics to determine which factors were to be taken into consideration were not clear. This brought to different paradigm possibilities to develop, and made us think of the need of a general purpose application instead of the strictly constricted projects existing so far. Hence, after focusing on the idea of allowing changing specific requirements, we started to work on the versatile framework that is presented in this paper.

This paper is the basis of a platform which breaks the tendency of thoroughly focusing on carefully fixed environments as done so far [4][5][6]. Here several solutions for different contexts related to the mentioned contest are purposed, even though any other cases are meant to be adopted. The robots can start in several formations, which may be a key fact when determining the movement patterns to be followed. Afterwards they start to recalculate their velocity and direction following one of the most extended swarm paradigms, studied by C. W. Reynolds [7] and consisting of three main rules. During the simulation any algorithm may be activated

or deactivated as well as have their inner parameters modified. Different final events can be handled to conclude the simulation in different ways depending on previously established criteria.
In many occasions it is necessary to know a first approach of the simulation's results, even when the first drafts are not yet designed. For new research fields, gaming industry or any nature-inspired applications it can be interesting to have a tool to find good preview results for simulation and optimization in a dynamic and low cost system without the need to build full applications for that purpose. Many improvements and complements are likely to be developed in the near future, such as distributed computing, more accurate methods for the optimization of the parameters, real-time comparison between two simulations, etc.
Section 2 thoroughly describes the implementation of the main blocks of the application, from the structure of the program until the function to search for the best parameters. Section 3 proposes the possibility of the further parallelization of the optimization function. In section 4 we show and explain some of the first results. Finally, an evaluation of the work done as well as a brief proposal for future research can be found in section 5.

2 Simulation and Optimization Functionalities

Each one of our units is an instance thinking independently and interacting with the simulator class just enough to represent real-life perception of the individual. A coordinate's class is implemented in such a way that all position- and velocity-related data is embedded and can be easily transferred and recalculated. The central processing class organizes the tasks and handles the list of robots, coordinates, parameters and rules to follow, but are the single instances the responsible for taking their own decisions with the limited information provided. The surface where the robots are going to move should be unlimited, but for a comfortable visualization in the GUI this area is constricted to a canvas of 500 by 500 positions or pixels (in this case, representing square meters).

2.1 Deployments

The starting positions of the robots determine the first decisions to take and may condition the overall algorithms to follow. If they start very close to each other, they need to spread in order to cover more area. On the other hand, if they are too distant, many areas may remain uncovered or clearing them would be too inefficient. In the contest, the initial methodology to distribute the robots was never specified and each simulation uses a different starting set. Therefore, it is important to have the chance to test a same moving paradigm after different deployments. For this, five starting configurations are now implemented, following different regular patterns (square, circular) or mathematic distributions (random, Gaussian and double Gaussian).

The **Random** (Fig. 1.5) deployment drops every robot in a completely random position within the sea limits established in order to focus the simulation in a constricted area. In further simulations much larger limits may be used, taking the risk of dropping robots too far away from the rest and becoming isolated. An interesting line of research could study which algorithms should isolated robots follow. However,

a purely random starting set does not bring many possibilities to study constricted behaviors, so more complex irregular deployments are developed.

The **Gaussian** (Fig. 1.1) and ***Double Gaussian*** (Fig. 1.3) configurations are used to guarantee a certain pattern within the randomness of a non-uniform distribution. With them we can know *a priori* how are the robots going to be distributed, even not their exact coordinates or the distance between them. Obviously, parameters such as the deviation can be manipulated in the every simulating case. In the case of the *Double Gaussian*, this is the name that an evolved deployment received. It consists of two Gaussian deployments, which are independent on number of items and deviation. Both means are calculated in a way that some robots from one kernel can perceive some of the other one, but never all of them. With this, we try to see the strength limits of the algorithms, as both Gaussians will tend to merge and join their own nucleus but at the same time some robots may push towards the other group.

Finally, regular patterns are also implemented to test uniform movements or simulate situations where the initial set can be regular. In this first version, uniformly fulfilled **Square** (Fig. 1.2) and a **Circular** (Fig. 1.4) patterns are available. Due to the variable amount of bots, the shape might slightly vary to try to keep the regularity of the distribution. The square will become a rectangle to ensure equal distance between all the boids, whereas the circle will deploy equally-distant robots on the circumference until this is full and a new ring can be added.

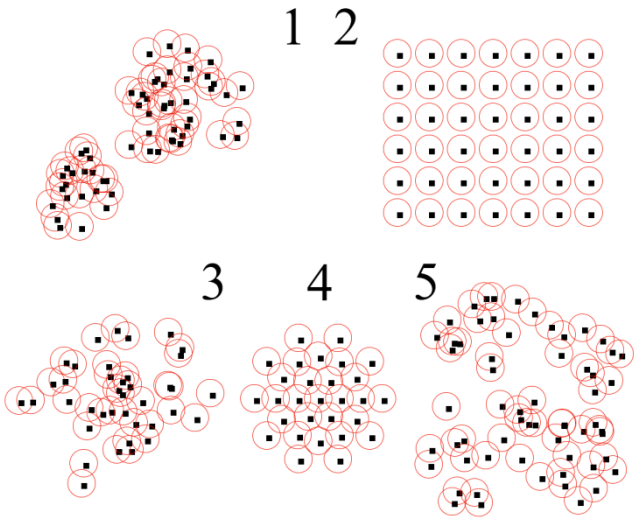


Fig. 1. Captions of the robots with the GUI representation of the different deployment methods: Random (1), Square (2), Gaussian (3), Circular (4) and double Gaussian (5).

2.2 Swarm Behavior: Equations and Weights

The limited communication capacity between the robots makes it too complex to determine the shape or the positions of all the other robots (a strong communicating network should be established) and it would escape from realistic situations. Nature-inspired algorithms for ants [8] or bees [9] are not based on sharing such big amount of data, but on generating a collective behavior out of the minor parts of information shared by each individual [10].

The moving paradigm chosen always follows a same overall concept [Fig.2][7] but at the same time we introduced some regulators which make the decisions experience small variations in order to adapt to the specific situation of each individual within the group. Every single robot analyzes the relative position p_i of all its n surrounding bots and calculates a variation of speed and direction after a weighted average of the values obtained through the main following rules:

$$\text{cohesion}(surrounding \quad 1...n) = \frac{1}{n}\sum_{i=1}^{n}(p_{ix}, p_{iy}) \tag{1}$$

Formula (1) calculates the average position of the nearby robots and tends to move there to follow the principle of cohesion and avoid the group spreading around or a robot to travel too far away, which may result in losing contact due to its limited view. After testing, this basic concept was not acting enough as a real-like flow, so a small correction was made to empower the weight of this algorithm when the boid is very far away from the others, and to decrease it when it is too close. To avoid direct contact between the elements, the separation algorithm (2) is also introduced in the final calculation of the new movement.

$$separation(surrounding \quad 1...n) = -\frac{1}{n}\sum_{i=1}^{n}(p_{ix}, p_{iy}) \tag{2}$$

In this case, the algorithm finds the opposite value of the cohesion formula. However, only when the object is very close to the others is when this algorithm should be more important. For this reason, the average point is calculated among the robots positioned in a smaller range than the viewing area used normally. In a similar way as in the first algorithm, this final value is not proportional with distance and gets strongly increased when the robot is too close to others, so it would avoid a final crashing between boids.

Finally, a third formula (3) computes the average direction where the neighboring robots are moving to. This alignment offers a smoother and more realistic movement and provides a non-static understanding of the environment, as it uses the actual velocity instead of the positions.

$$\text{a}lignment(velocities \quad 1...n) = \frac{1}{n}\sum_{i=1}^{n}(v_{ix}, v_{iy}) \tag{3}$$

The modularity of these implementations allows a full addition or modification of the algorithms to follow. During each iteration of the simulator, which could be understood as a second or as any other time unit, all the robots obtain the list of neighboring boids as well as the algorithms to handle. With this, they decide which algorithms and with weights use, independently from the simulator class. Only once they all have finished the calculation of their decision, is when the actual movement takes place. This avoids that the last robots in the list take decisions with later information about already updated positions, which would not be realistic and may cause synchronization problems as well as the possibility of looped situations.

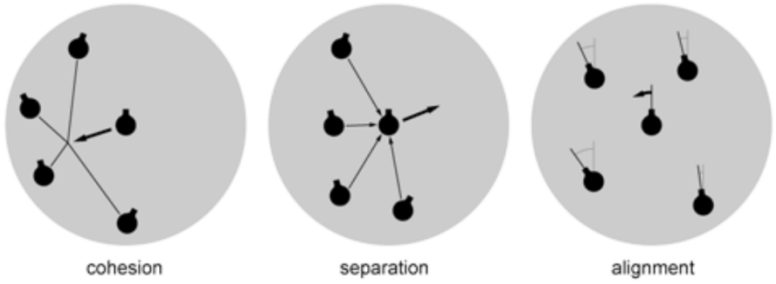


Fig. 2. Visual representation of the principles of cohesion (1), separation (2) and alignment (3) described by Craig W. Reynolds [7].

After the calculation of the final movement, a limiter controls an excessive value of the speed and reduces it to a established maximum, which can be common for all boids or specified for each individual.

Bigger dynamic paradigm modifications are also necessary for the proper simulation of realistic environments. Although the (1) and (2) criteria have small variations for specific situations, sometimes these are not enough and a new value for the algorithm's weight or even a completely new behavior must be done. In this version of the application, a practical modification of the weights allows the creation of the *Recollection* command, which consists on make all the robots meet in a central point. The original instructions from the challenge asked the boids to be collected at a certain point by a mother ship. In any case, the goal was to make them gather, even no optimization method was asked again. A good choice would be to reduce this time to the minimum by meeting them in the central point between the extreme boids. However, our choice was to optimize the fuel consumption so the parameter minimized was the total track covered by all the robots until they reach this point. For this, a signal can command the separation (2) and alignment (3) weights to become 0, so the full decision of the movement relies on the cohesion algorithm (1) and the boids are set to move at full speed. This is represents a realistic situation where the robots are called from the central control of the mother ship, or, with a set of timers, they have the command to gather at a certain execution moment, or after a threshold of inactivity due to the balance of all the robots when they are all close to each other.

2.3 Optimization Function

The aim of this research is to develop a tool which offers different possibilities of simulation, measurement, understanding and comparison, but maybe the most interesting utility is the optimization module. This function can still be improved by using more sophisticated methods such as gradient ascents, neural networks, etc. but the current version already offers a good approximation of the best parameters to use in order to reach a specified goal. Manganese collected, total or unique track covered or time to gather could be some of the goals in our goal to continue with the InformatiCUP context. The goal pursued in all the tests shown in section 4 is the total amount of manganese collected by all the robots combined.

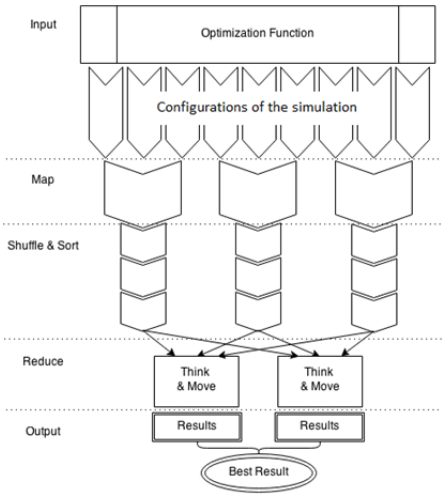
The idea is very simple and it consists on testing a specific configuration until the robots reach a balanced position, or if not, until they reach a time or distance travelled limit (as if it was fuel). The simulation accepts the introduction of a set of values chosen by the user to test, if some previous stipulation has been done because those parameters are more likely to be good. With small variations, the iteration can skip useless values as well as reduce the step between them when testing. Despite of having different starting sets, the current version asks for a fixed deployment to test with, as they are considered fully different experiments in separate contexts. Many different amounts of robots, instead, can be added in the iteration system and different weights can be tested on different number of robots for the given deployment method. In case of choosing non-uniform deployments, the positions of the robots are calculated once and saved for further simulations, in order to guarantee that the results correspond to equal testing conditions.

At the end of each simulation a .txt is generated with a header indicating which range of configurations was used, tested number of robots, and the values which offer the best result obtained. With the purpose of a better understanding of the behavior seen during the variation of the values, a list of all the iterations and the desired results for each combination is now included. A Matlab script processes these outputs and prints a 3D representation of the results, which would be too complex to understand without data processing to summarize them. This graphic can only use two of the parameters simultaneously, as the third axe is always representing the target value. Fortunately, the algorithms used so far are conceptually easy so it is possible to understand the results with the use of two graphs. Further applications will need new solutions for the improvement of these graphs and the output interpretation, but it depends on the algorithms used so no further preparation could be done now.

3 Parallel Computing with MapReduce

The longest simulations iterating over all the parameters can run for some minutes, in the most complex cases with small steps between iterations and a high number of robots, even almost an hour when using a 2GHz i5 laptop. Future applications may need more constrictions, new algorithms and even higher number of individuals, so the computation time can increase exponentially. To ensure an acceptable simulation time it is going to be essential to focus on a big improvement in this aspect.

The number of shared data and parallel applications worldwide is increasing very fast, so we would like to use some of these techniques as a solution to the excessive computation time. Apache offers an open-source implementation of the MapReduce [11] called Hadoop [12]. MapReduce splits the input in the map tasks so the data can be processed in different cores simultaneously. Afterwards, the Reduce tasks collects all the partial results, sorts them and, in our case, search the best configuration among all the simulations run. This model is a good candidate to be implemented in this kind of projects, as has already been done in similar applications [13].




```
Initialize(Deployment)
def MR_MAP(List of Array allParameters) = {
  while(0<combinations--)
    settings[i] ← generateConfiguration();
  end while }
def MR_REDUCE (List of <Configurations>
settings) = {
  while( i ← 1; i ≤ settings.size(); i++)
    for( i ←1; i ≤ settings[i].iterations; i++)
      neighborsList[i] ← getNeighbors(robots,
          coordinates);
      coordinates.update( robot.move (
          neighborsList[i] ) );
    end for
  end while
}
return bestParameters;
```

Fig. 3. Scheme and general pseudo code for one of the next steps in this project, the parallel computation of the simulation. Both diagram and code show how the *Map* splits all the configurations' processes into different chunks which are separately calculated and reunited by the *Reduce* to find the best result.

4 First Experimental Results

The application was first tested by executing simple, understandable parameters with uniform deployments, to see how the system behaved. Balance was easily reached with circular deployment and equal cohesion (1) and separation (2) weights. Once the testing phase proved the correct performance of the simulations through the GUI interface, the first full executions were done.

As explained previously, the need for variations in the algorithms as well as the more detailed output was solved, and finally some applicable results were obtained and shown (Fig. 4) with the Matlab script described in Section 2.3.

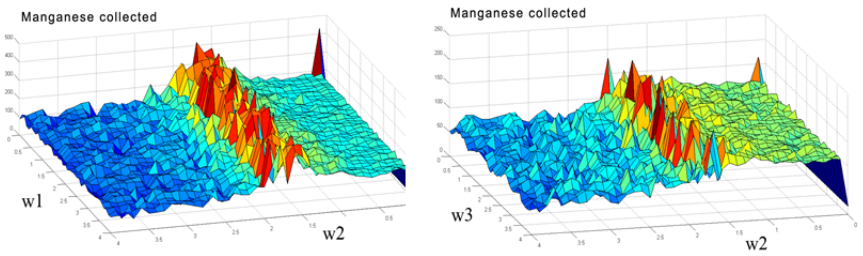


Fig. 4. Two graphics showing the total Manganese obtained (vertical axe) depending on the parameters of cohesion (w1) and separation (w2) on the left, and alignment (w3) and separation (w2) on the right. Both cases have an irregularity at the beginning of the performance.

The results obtained so far show a logical evolution of the manganese recollection. If the separation value (w2) is too small or too big, the movement is affected in excess, due to the variations implemented which empower its value when a robot is too close to the others. On the other hand, the similarity of the two graphics for cohesion (w1) or alignment (w3) suggest that these two algorithms are less decisive in comparison with the separation one. This makes the further work focus on the study of a better weight balance for this specific situation. However, the purpose of this first use of the framework is to observe and analyze its possibilities, which we consider successful. The algorithms used work properly and the graphical results are encouraging.

5 Conclusions Future Work

Being satisfied with the performance of the platform, our next research will focus mostly on the improvement of the general behavior of the simulations, looking for a more efficient execution. The implementation of new functionalities is imminent, for example the possibility of run two different simulations (with different number of robots, deployment method and/or algorithms) at the same time using the visual interface. This can help to visually compare two behaviors at real time, hopefully avoiding the need of result post-processing. Last but not least, as a mid-long term task to do, the parallelization method explained in section 3 to ensure acceptable execution times with larger and demanding simulations.

References

[1] InformatiCUP. http://informaticup.gi.de

[2] Gesellschaft für Informatik. http://www.gi.de/wir-ueber-uns.html

[3] Detailed requirements for the first prototype. http://informaticup.gi.de/fileadmin/redaktion/Informatiktage/studwett/Aufgabe_Manganernte_.pdf

[4] Fernandes, C.M., Merelo, J.J., Rosa, A.C.: Controlling the Parameters of the Particle Swarm Optimization with a Self-Organized Criticality Model. PPSN XII (II) pp. 153-164. Springer, Taormina (2012)

[5] Bim, J., Karafotias, G., Smit, S.K., Eiben, A.E., Haasdijk, E,.: It's Fate: A Self-Organising Evolutionary Algorithm. PPSN XII (II) pp. 185-194. Springer, Taormina (2012)

[6] McNabb A., Seppi, K.: The Apiary Topology: Emergent Behavior in Communities of Particle Swarms. PPSN XII (II) pp. 164-173. Springer, Taormina (2012)

[7] Reynolds, Craig W. (1987). "Flocks, herds, and schools: A distributed behavioral model" (SIGGRAPH'87) (ACM). doi:10.1145/37401.37406

[8] F. Moyson, B. Manderick, The collective behavior of Ants : an Example of Self-Organization in Massive Parallelism, Actes de AAAI Spring Symposium on Parallel Models of Intelligence, Stanford, California, 1988.

[9] Rodriguez, F.J., García-Martínez, C.: An Artificial Bee Colony Algorithm for the Unrelated Parallel Machines Scheduling Problem. PPSN XII (II) pp. 143-152. Springer, Taormina (2012).

[10] Tereshko V., Loengarov A., (2005) Collective Decision-Making in Honey Bee Foraging Dynamics. Journal of Computing and Information Systems, 9(3), 1-7.

[11] Dean, J., Ghemawat, S: MapReduce: simplified data processing on large clusters, In: Communications of the ACM, Vol. 51, Nr. 1, pp. 107-113, ACM New York, USA (2008)

[12] Apache Hadoop open-source implementation. http://hadoop.apache.org/

[13] Logofătu, D., Dumitrescu, D.: Parallel Evolutionary Approach of Compaction Problem Using MapReduce, In: Proceedings of 11th International Conference on Parallel Problem Solving from Nature (PPSN(2) 2010), LNCS 6239, pp. 361-370 (2010)