DOI: http://dx.doi.org/10.7551/978-0-262-32621-6-ch155

Framework for Adaptive Swarms Simulation and Optimization using MapReduce

Sergi Canyameres[1], Doina Logofătu[2]

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

Downloaded from http://direct.mit.edu/isal/proceedings-pdf/alife2014/26/957/1901661/978-0-262-32621-6-ch155.pdf by guest on 06 February 2023

Abstract

Natural swarms can have multiple structures and follow many different patterns or laws. Most recent studies try to obtain the best solutions in very specific and sophisticated contexts. In this research, instead, an application to find a reasonable approximation to the behaviour of any possible incoming environment is developed. This paper describes a basic framework, GUI and algorithms established in a way that can be easily modified and adapted to desired paradigms, parameters and rules. In experimental research, this will be able to provide help for many applications where natural swarm patterns are followed but no deep, expert simulator has yet been developed. To deal with the complexity of the biggest applications, the simulations are to be run in parallel computing using the MapReduce framework.

Background and Motivation

We were asked to develop a software which would fit the requirements for taking part in the InformatiCUP [1]. In this year's edition the participants were asked to find the best algorithms some robots should follow under a specific scenario [2]. These are deployed on the ocean's surface and can only communicate between other nearby boids, knowing not more than their relative positions. The aim is to move around the ocean's surface collecting manganese and gather together after a certain time or command. Many spreading or path algorithms could be appropriate, but there was a lack of information about what to optimize (time, fuel, unique track covered ...) so the goals were too ambiguous. None of the existing swarm algorithms that we found satisfied our expectations. As seen in previous studies [3][4][5], it's common to focus research in carefully delimited conditions, but no adaptable platform to use with our constrictions was found.

The goal of our project switched to cover this lack of resources and try to set the basis for a lot of possible future developments and applications in experimental research. Different states and phases were introduced to modify the different algorithms' weights according to the desired behaviour of the robots at every moment.

Section 2 thoroughly explains the context of the project and the first steps taken, describing how starting with different deployment patterns causes a need of very different movement rules, so regular shapes -square, circle- and irregular deployments -random, Gaussian, combined- were created to study all the possibilities. In section 3, we show how the application itself works, and how the iteration over the desired parameters can provide some acceptable results and how should they be understood for future applications. Section 4 introduces the further need of using parallel computing for the execution of the application. In section 5 the results are showed and commented to extract some conclusions leading to the future work purposed in section 6.

Requirements and Basis Settled

The requirements for the InformatiCUP were constrictive, especially when establishing which communications could take place between the robots. The uncertainty caused by the incomprehensibly ambiguous formulation of the task gave us total freedom to focus on the creation of a new versatile framework and leave constrictions behind. The required small viewing range and limited knowledge lead to the idea of applying swarm particle algorithms. Any other paradigm followed would be by intuition, and the time was limited to a few weeks which could not be spent on trying unfunded ideas.

To keep the difference between what the robots could know and what not, a Simulator class was created to keep track and manage most of the information and share it properly with the other classes. The robot instances are stored in a *HashMap* also containing information about position (classes Robot and Coordinates). Simultaneously, a first visualisation of a "sea" of 500 by 500 positions was

implemented, so it was time to start with the deployment configurations.

Optimising the gathering time is different to optimising the unique surface travelled or the fuel used after they start. Moreover, if the initial set of robots is positioned in different way, it becomes a key fact when it matters of which algorithm to follow in order to run over the surface following some criteria.. Hence, many deployment methods are available:

DeployRandom(). Deploys an amount of robots in random positions within the given sea limits. Initially this algorithm was implemented to verify and debug the need of creating a new robot only within the viewing range from at least one other robot. Later in the project, this distribution stayed in the simulator for further testing of new features (new visualisation, determine movement behaviours) and as one of the starting point algorithms for the InformatiCUP. As it is a very inefficient starting set the challenge was more ambitious.

DeploySquare() and DeployCircle(). Both deploy an amount of robots following uniformly filled squared or circle patterns. The first robot is one of the four central points of the figure, which grows in a spiral or surrounding shape from the centre point.

DeployGauss() and DeployBadCenters(). Given the initial random centre (mean) and variance, deploy the robots following a Gaussian distribution, or in two unequal Gaussian distributions. This offers an interesting and challenging beginning, where two unequal groups are deployed close enough to see each other but far enough to, following the main algorithms seen later, tend to split up into two groups of boids. This is very useful when extreme conditions are going to be tested.

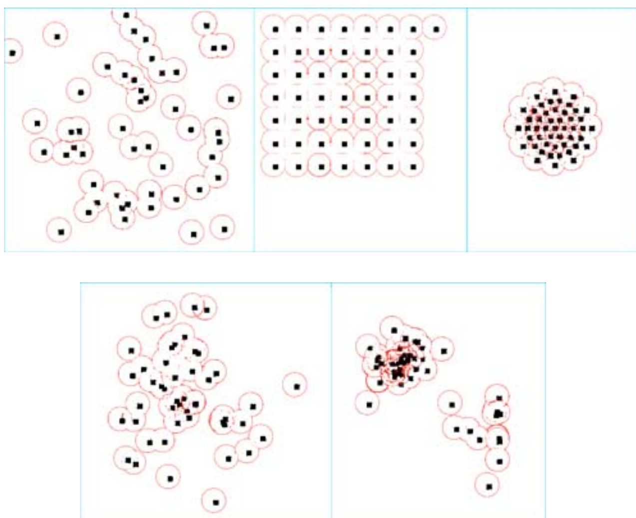

Fig. 1. In order, Random, Square, Circle, Gauss and BadCenters deployments shown in the GUI.

Previous Work

3.1 Interaction Simulator-Robot

The Simulator coordinates the evolution of the steps taken by the robots while interacting with them in order to provide the exclusive data that they are allowed to know, in the real way robots would perceive their environment. For example, the absolute coordinates are translated to relative coordinates for every robot prior to being given. Therefore, the boids are just computing machines which translate the given inputs into new positions, given a set of rules or behaviour algorithms which can be easily exchanged.

Iteration can be understood as a real-time second, fraction, etc. and the maximum speed can be changed as a consequence of this. In every step, the movement for all the robots is calculated, and only once they have found their new position they will actually move, the Simulator being responsible to iterate on the robots asking for the displacement vector. With this, both new positions and speed can be stored in order to be used in the following iterations as well as displayed through a friendly GUI.

This way of interacting is possibly not as easy to modify as the algorithms that the robots are going to use, as it is something quite specific for our project. Due to the expected versatility for the future uses, the complete code is carefully documented with Javadoc in order to make it more understandable and accessible.

3.2 Used Algorithms

Many different algorithms for moving objects or swarm behaviour could be implemented [6][7]. As said before, the main intention is not to simulate some specific scenario, but to create a platform which allows an easy transformation of the characteristics to simulate. For this reason the algorithms to be followed are not required to be very complex, only implemented in a strict modular way. This allows the addition of other algorithms or simply the modification of the existing ones.

The current application runs under a simplification of the bird flock movement described by Craig W. Reynolds [8]. This behaviour paradigm consists of three criteria every robot follows at each iteration, which can be understood as three different algorithms simultaneously working.

Cohesion: Every robot moves toward the centre of mass of the neighbouring ones (1). We assume the neighbouring group as the set of robots given by the simulator. This means that these neighbours are only those close enough (*view range* limit) to be detected by the robot. Although (1) is the concept simplification, experience made us add a

Downloaded from http://direct.mit.edu/isal/proceedings-pdf/alife2014/26/957/1901661/978-0-262-32621-6-ch155.pdf by guest on 06 February 2023

regulator which increases the value of this result if a robot is too far away from the others.

$$\mathrm{alg1}(neighbours\ \ 1\ldots n) = \frac{1}{n}\sum_{i=1}^{n}(r_{ix}, r_{iy}) \qquad (1)$$

Separation: Every robot tries to keep a minimum distance with the closest robots (2). In the same way as the first algorithm, the average position of the robots plays a role in this calculation, but a little variation gives an extra value to this result if the robot is too close from another one. The main difference is that only the really close robots are taken into consideration in this calculation. Otherwise, it would be complementary, opposite to (1) and it would not make sense to keep two algorithms.

$$\mathrm{alg2}(neighbours\ \ 1\ldots n) = -\frac{1}{n}\sum_{i=1}^{n}(r_{ix}, r_{iy}) \qquad (2)$$

Alignment: Every robot changes direction to the average position where the other robots are trying to steer to (3). This is similar to an inertial force influencing all the set of boids, as it does not work with the positions, but with the velocities.

$$\mathrm{alg3}(velocities\ \ 1\ldots n) = \frac{1}{n}\sum_{i=1}^{n}(v_{ix}, v_{iy}) \qquad (3)$$

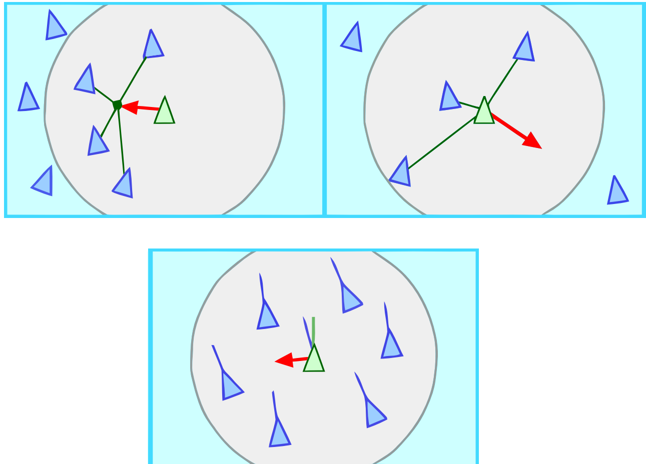

Fig. 2. Representation of algorithms (1), (2) and (3) [8].

3.3 Weights and Gathering mode

The final decision of every robot is conditioned by different weights given to every algorithm implemented (4). This allows activation, modification and deactivation of any existing algorithms without spending time programming or changing the settings. In fact, the GUI itself provides glide bars which allow the direct setting of the parameters within a given range. The current version only applies the changes prior to the deployment, but in the future can easily be changed.

$$\bar{v}(algorithms\ \ 1\ldots n) = \frac{1}{n}\sum_{i=1}^{n} w_i\ alg_i(x, y) \qquad (4)$$

In case of exceeding the maximum speed allowed by the requirements, both x and y components can be normalized so that the vector speed fits the specifications:

$$\bar{v}[x,\ y] = maxSpeed\ * \left[\left(\frac{x}{\sqrt{x^2+y^2}}\right),\left(\frac{y}{\sqrt{x^2+y^2}}\right)\right] \qquad (5)$$

The original requirements asked for a final behaviour in which all the robots should stop focusing on collecting manganese and moving towards some common point where a mother ship could collect them. Again, the criteria to look for this point was not specified, and many possibilities could be optimized: time, fuel used... The best point if the desired parameter to minimize is the total fuel consumed (or simply the total distance travelled), this should be the average position of all the robots. Hence, the existent weights used for the algorithms are adequately used for this final situation, and all of them are set to zero except the cohesion (1) weight, which is set to the maximum value. In the basic performance, a maximum of iterations can be set in order to activate Gathering Mode, simulating an eventual call from the mother ship or simply a time limit which could be integrated in the boids. Another countdown is carried on by checking if the robots moved more than a specified threshold. After a certain number of iterations (consecutive or not) when the robots are moving less than this limit set, the Gathering Mode can also be activated. Once the meeting of all the robots has taken place, the simulation stops after a specified timer and a reset of all the variables and parameters is done, in order to execute a new simulation. In case of looking for the best parameters (see next paragraph) the initial deployment positions are loaded again.

3.4 Find Best Parameters

The most important function is responsible for simulating different executions by using as many parameters modifications as possible in order to find the best configuration. The results can be understood in many ways, as there are many output values which could be optimised (manganese collected, distance travelled, ratio between manganese and distance, iterations to join, etc.). In our case the chosen one is simply the maximum amount of minerals collected.

The most logical parameters to iterate on are the weights. However, also the number of deployed robots, the deployment method or the iterations until gathering plays an important role on the results. In any case, the simulation

Downloaded from http://direct.mit.edu/isal/proceedings-pdf/alife2014/26/957/1901661/978-0-262-32621-6-ch155.pdf by guest on 06 February 2023

ends up with a generated data file containing a header with general information about the simulations; static parameters or simply sequence of parameters used, e.g. simulation done with 5, 10, 50 and 100 robots. All the results follow the header, with particular info about the set of variables for each simulation, and finally the results for manganese collected and distance travelled.

Initially only the best configuration for the same set of parameters was registered. However, in order to understand the effects and the evolution of the values better, the output file now includes the result for all the simulations. Its manipulation and understanding would be too tedious to be done manually, so a script in Matlab is created to read and plot the results in a 3D graphic, which is very helpful to see the behaviour of the simulations. As some simulations may have more dimensions, unable to be shown in 3D, some adaptations may be done. For example, (1) and (2) could be understood as complementary values (the real difference is explained in section 3.2). Instead of using an axis for each, a new parameter can be shown as the ratio between these two values given a closed range:

$$w_1, w_2 \in \left[0, 4\right] \Rightarrow w_{12} = \frac{w_1}{w_2} \tag{6}$$

3.5 Double Deployment Comparison

When finding the best parameters for a given configuration, the actual best weights might not follow an intuitive pattern which allows easy understanding. The analysis of the simulations so far was mostly a posteriori, by studying and searching a proper interpretation of the results given in the output files or the graphics in Matlab. The visualisation of the simulation for a specific set of parameters was implemented since the very beginning, but the weights were not modifiable and the proper analysis was not comfortable. Hence, we wanted to offer a functionality which could allow the user to observe and compare the behaviour of the flock in a practical, real-time way. For that, a second deployment can take place simultaneously, shown in another colour. The parameters, number of robots and the starting distribution can be different between them, offering the possibility to see in real time the development of both simulations in parallel. Fig.3 shows the visualisation offered by the GUI. On the left, a first deployment in red follows the double-Gaussian pattern, while the blue one is purely random. On the right side, a fulfilled circle is drawn by the red robots, whereas a different amount of robots, in blue, create a square.

Both double simulations perform a similar evolution, although the uniform deployments experiment smoother modifications due to the regular distance between the boids. On the other hand, the randomly positioned robots need more iterations to find a balance, which still look more irregular than in the images on the right. It was interesting to add this little option in order to explore the capabilities of the platform in the gaming world, if we understand the both flocks as a competition to see who gets a better performance. In the future, other applications might be interested in using this second deployment with newer functionalities, such as direct modification of the parameters, commands on splitting or merging the flocks, and countless other possibilities which can be added.

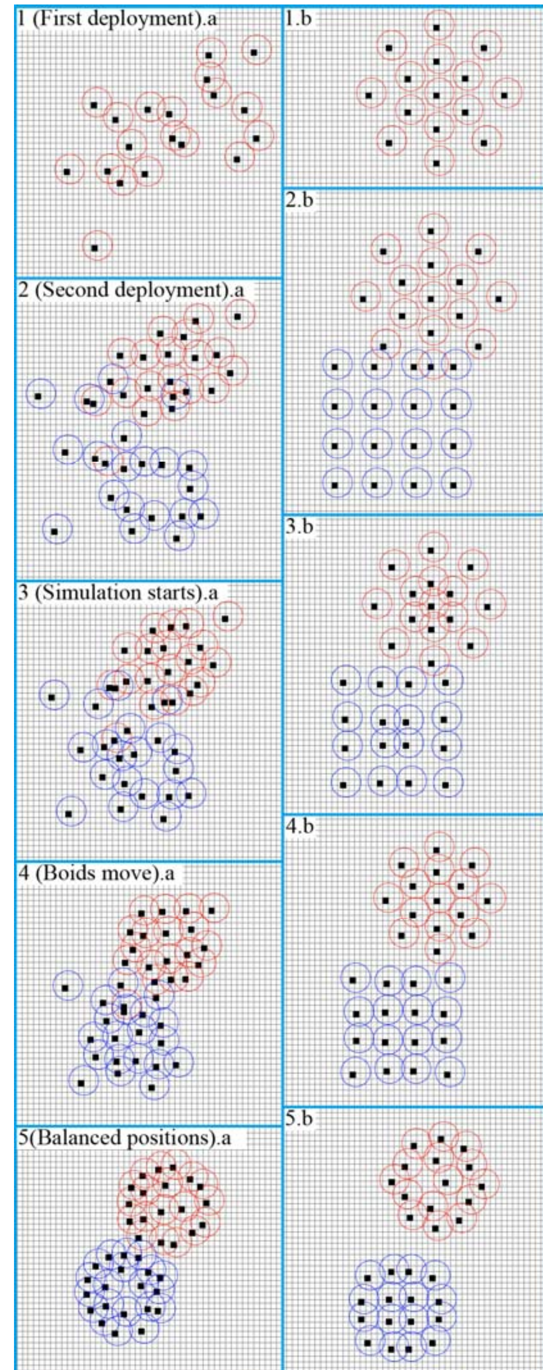

Fig. 3. Visualisation of two pairs of parallel simulations

Downloaded from http://direct.mit.edu/isal/proceedings-pdf/alife2014/26/957/1901661/978-0-262-32621-6-ch155.pdf by guest on 06 February 2023

Distributed Algorithm Using MapReduce

Nowadays it is quite strange to find big simulations being run in simple computers, and this project may not be an exception. Most of the simulations done so far by us have used low amount of robots and iterations, as well as some simplifications to just confirm the operative power of the system. In order to execute these simulations in a current generation pc, the algorithms can be sophisticated but the parameters must be simplified when finding the best configurations. Otherwise too much memory usage is needed and the program's behaviour can be irregular due to exceed of the memory space.

For all this, an implementation of algorithm parallelization is required to be the next step in the project. The Hadoop [9] open-source implementation of the MapReduce [10] model is likely to be successfully used, like in other evolutionary applications [11]. This is the OpenSource MapReduce framework implementation from Apache, a batch data processing system for running applications, which process vast amounts of data in parallel, in a reliable and fault-tolerant manner on large clusters of compute nodes, usually running on commodity hardware. It comes with status and monitoring tools and offers a clean abstraction model for programming, supporting automatic parallelization and distribution. Hadoop comes with a distributed file system (HDFS) that creates multiple replicas of data blocks and distributes them on compute nodes throughout the cluster to enable reliable, extremely rapid computations.

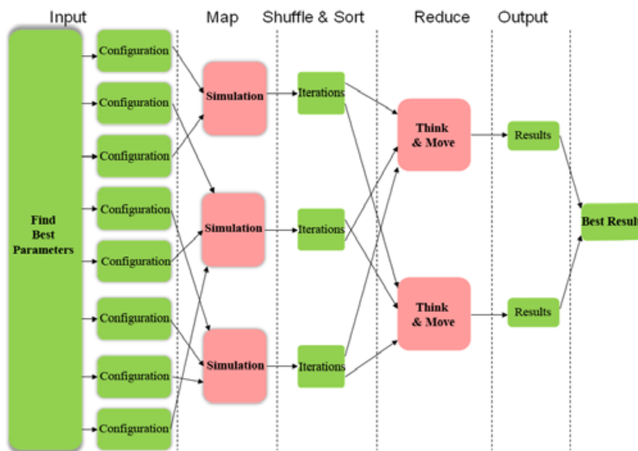

Fig. 4. MapReduce Dataflow

```
Initialize(Deployment)

def MR_MAP(List of Array possibleParameters)
= {
    for( i ← 1; i ≤ combinations; step 1)
        settings[i] ← generateConfiguration();
  end for }
```

```
def MR_REDUCE (List of <Configurations>
settings) = {
    for( i ← 1; i ≤ settings.size(); step 1)
      for( i ←1; i ≤ settings[i].iterations; step 1)
  neighboursList[i] ← getNeighbours(robots,
    coordinates);
  nextMove ← robot.move(neighboursList[i]);
  coordinates.update(nextMove);
        end for
    end for
}
return bestWeights;
```

Fig. 5. Pseudocode for MR_SW_OPT

The compute and storage nodes are typically the same. This allows the framework to schedule tasks effectively on the nodes where data is already present, resulting in very high aggregate rate across the cluster. The framework consists of a single master `JobTracker` and one slave `TaskTracker` per compute node. The master is responsible for scheduling the tasks for the map- and reduce-operations on the slaves, monitoring them and rerun the failed tasks. The slaves execute the tasks, as directed by the master. The applications specify the input/output locations, supply map and reduce functions and possibly invariant (contextual) data. These comprise the job configuration. The Hadoop job client then submits the job (Java byte code packed in a jar-archive) and configuration to the `JobTracker`, which then distributes them to the slaves, schedules the map-/reduce- tasks, and monitors them, providing status and diagnostic information to the job client.

A MapReduce job splits the input data into independent chunks (splits), which are then processed by the map tasks in a completely parallel manner. The framework sorts the maps' outputs and forwards them as input to the reduce tasks.

Experimental Results

Assuming the limitations of executing the application in a single domestic computer, different simulations were done. The initial goals were just to check the correct behaviour of the application. For this, logical results were pursuit by using understandable and basic parameters. Some combinations hold useless values. A perfect circle consisting on a regular amount of robots may not even experience any variation of the positions, as the balance is achieved since the deployment itself. Similar behaviours may occur with a perfect squared pattern, where the inner robots are balanced and only the surrounding ones experience some little repositioning. In any case, the events seen were correct. The actual implementation of the described project can be found at [12]. Once this was verified more extensive calculations took place by

Downloaded from http://direct.mit.edu/isal/proceedings-pdf/alife2014/26/957/1901661/978-0-262-32621-6-ch155.pdf by guest on 06 February 2023

increasing the range of possible values. More real-like weights were introduced. Also longer simulations could be held by deploying more robots, as well as trying all the possible deployment methods.

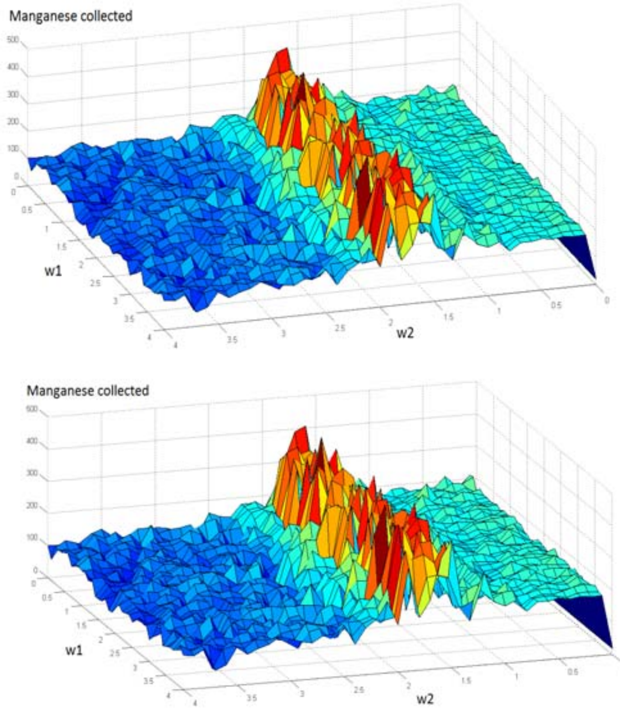

Fig. 6. Manganese depending on w_1 and w_2 after Random (above) and Square (bottom) deployments.

As the figures show, the evolution of the Manganese recollection follows a logical and acceptable evolution through the values for the main weights for algorithms (1) and (2) after 25 and 100 iterations respectively. These two graphics corresponds to two very different deployments such as random and squared-pattern. Despite the initial variations both of them look very similar. This can suggest that the robots act individually, without noticing a big influence from the surroundings. In the same way, circular and random deployments offer very similar outputs.

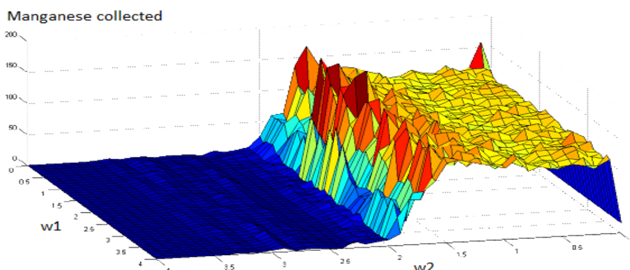

Fig. 7. Bigger simulations need more memory.

However, more detailed and long simulations should still be done. The precision is increased and the parameters are tested with smaller steps between simulations, causing a big increase of the data generated needed to be stored. Unfortunately, a standard 6Gb computer experiences difficulties to keep such volumes of information, leading into unstable behaviour of the program as seen in figure 6. It is to be understood that the output is suddenly decreased until 0, which is a senseless value because the robots gather some manganese even if they are static. In any case, the performance seen so far confirms the viability of using the framework to experiment and research with further contexts, using new algorithms. In conclusion, there are good expectations for the future.

References

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

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

[3] 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)

[4] 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)

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

[6] 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)

[7] Montes de Oca, M.A.: Particle Swarm Optimization - Introduction. http://iridia.ulb.ac.be/~mmontes/slidesCIL/slides.pdf

[8] Reynolds, C.: Boids (simulated flocking). http://www.red3d.com/cwr/boids/

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

[10] 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)

[11] 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)

[12] Implementation Adaptive Swarm Optimization (Robots): http://en.file-upload.net/download-8770063/Simulator.jar.htm

Downloaded from http://direct.mit.edu/isal/proceedings-pdf/alife2014/26/957/1901661/978-0-262-32621-6-ch155.pdf by guest on 06 February 2023