Particle Swarm Optimization Algorithms for Autonomous Robots with Leaders Using Hilbert Curves

Doina Logofatu[1], Gil Sobol[2], Daniel Stamate[3]

[12]Computer Science Department of Frankfurt University of Applied Sciences 1 Nibelungenplatz 60318, Frankfurt am Main, Germany
[3]Department of Computing, Goldsmiths College, University of London, London SE146NW, UK

Abstract. The approaches in this work combine the swarm behavior principles of Craig W. Reynolds with space filling curves movements. We intend to evaluate how the entire swarm moves by including a deterministic *leader* behavior for some agents. Therefore, we examine different combinations of Hilbert Curves with the classical swarm algorithms. We introduce a practical problem, the collection of manganese nodules on the sea ground by using autonomous agents. Some relevant experiments, combining different parameters for the leaders were run and the results are evaluated and described. Finally, we propose further developments and ideas to continue this research.

Keywords: particle swarm optimization, changing environments, autonomous agents, Hilbert Curves, leaders, application.

1 Introduction

At the same time with the research of renewable resources, it would be useful to find new ways in order to open up fossil ones. For example, manganese can be found on the sea bottom in form of nodules. Actually, this is a chemical element with the periodic table symbol M_n and atomic number 25. The biggest application area is for rust and corrosion prevention on steel [9]. Degradation can be prevented by collecting these manganese nodules from the sea ground using specialized robots. It is necessary to find appropriate ways how to handle the action of these agents. The results can be generalized and cover other collecting tasks as well. Consequently, this work focuses on optimizing the swarm behavior of autonomous agents. The solution can be found by either improving the achievements or reducing the effort by keeping the same size of achievements. The background for our approach is a framework for simulation and improvement of swarm behavior in changing environments [1]. It simulates the swarm behavior after the principles of Craig W. Reynolds [2] later pointed out in

section 2.2. The main issue the associated application to the framework does, is to deploy agents with a specific strategy and then to gather them. While gathering, the agents are collecting the manganese which is distributed on every position in the coordinate system. Once they are gathered together, there is no more movement and the simulation ends. The intention of our work is to redesign and extend this framework. Manganese occurs in form of nodules, so it is not really realistic that they are distributed uniformly. Therefore, it is reasonable to implement a *Manganese-Nodule-Model*, where each nodule is represented stand-alone. It should also be considered that not all nodules have the same size (value). For this new design of the manganese distributions, benchmarks have to be created for having the opportunity to compare the results. The next issue is to improve the collecting procedure itself. The more meters the agents pass, the higher is the chance to find manganese. Consequently, we try to find a way to pass through a bigger area. The less complex solution would be to give each agent his own route. This would probably scatter the swarm because of the bad orientation and the uneven surface. Most of the research activities about swarm behavior are inspired by nature like genetic algorithms or particle swarm optimization. These researches focus on bird flocks or fish schools. An alternative discussion could be focusing on a pack of wolves, for example. A pack of wolves consists of autonomous individuals with a specific hierarchy. Not every wolf has the same power of decision for the pack. Normally there is one wolf who leads the group and the others are followers [11]. This work aims to study this concept more closely. We want to have one or more leaders who will move after a specific route, but still being part of the swarm and the rest calculates its new position, that means every iteration in consideration of all agents. We imagine an area where we know that there should be a big amount of manganese nodules. We need to find proper methods to explore systematically and carefully through a given area. One of the first things that comes to one's mind is the specialist mathematical field of space filling curves. Summarized, a space filling curve is a curve that covers recursively an entire 2-dimensional square. We are focusing on the one of the most famous space filling curves developed by David Hilbert (section 2.4).

2 Background

This section describes the previous work the application is based on. It includes three main topics: Moving Algorithms, Particle Swarm Optimization, Hilbert Curves.

2.1 Framework for Adaptive Swarms Simulation and Optimization

The starting application is based on [1]. The framework is an application that runs a simulation of agents using moving algorithms Random, Square, Circle, Gauss, and Bad Centers [1]. It has different deployment strategies implemented from where the moving algorithms start. The frontend is based on the open source framework of *processing.org* [4]. The whole visualization part is done in the *Visualization* class with support of its derived class *VisualRobot*, which helps to represent the robots in the visualization. The whole simulation part is managed by a class with the same name. It

creates the chosen deployment strategies and calculates the movement of the robots, as well as the collection of manganese. Manganese is located on every position in the coordinate system. In addition, it also counts the distance in walked meters of all agents together. It is also possible to set at the beginning the number of agents. This number must be between 2 and 100.

2.2 Moving Algorithms

Artificial systems are, for example, needed when it is wanted to solve problems which are beyond the capabilities of a single individual. In our case it is actually required to build a swarm of agents, where each agent individually moves forward with consideration of the other agents of the swarm. There are several efficient algorithms for swarm behavior and movement of agents that could be implemented in the application [4]. The previous work [1] uses a simplification of the bird flock movement described by Craig W. Reynolds [2]. The idea was to develop algorithms that simulate swarm behavior inspired by flocks of birds or schools of fish. Therefore, three criteria every robot follows at each iteration were settled up. The contribution implemented three different algorithms that run simultaneously: cohesion, separation, alignment.

2.3 Particle Swarm Optimization

Particle Swarm Optimization (PSO) was first proposed in 1995 by J. Kennedy and R. Eberhart [6]. The idea was to build swarm behavioral algorithms for solving problems by iteratively improving a candidate's solution until termination criteria is satisfied [7]. It is similar to a genetic algorithm regarding that both algorithms are initialized with a random population, in PSO called particles. The difference is that in PSO algorithms, each particle is assigned to a randomized velocity and the particles move through hyperspace. Each particle consists of its position, its velocity, its current objective value and its personal best value of all time. PSO also keeps track of the global best value that is the best objective value of all particles and also the corresponding position.

$$x^{(i)}(n+1) = x^{(i)}(n) + v^{(i)}(n+1), \qquad n = 0, 1, 2, ..., N-1 \tag{2.1}$$

The formula above describes a classical iteration for particle movement. The next position $x^{(i)}$ (n+ 1) is made from the current position $x^{(i)}(n)$ and the velocity vector $v^{(i)}$ (n+ 1) of a specific particle i. The velocity vector gets created by the following iteration:

$$v^{(i)}(n+1) = \underbrace{v^{(i)}(n)}_{inertia} + \underbrace{r_1^{(i)}(n)[x_p^{(i)}(n) - x^{(i)}(n)]}_{personal\ \ influence} + \underbrace{r_2^{(i)}(n)[x_g^{(i)}(n) - x^{(i)}(n)]}_{global\ \ influence} \tag{2.2}$$

$$n = 0, 1, 2, ..., N-1,$$

where x_p represents the individual and x_g the global best position. $[x_p^{(i)}(n) - x^{(i)}(n)]$ calculates a vector towards the personal best which is influenced by the random vector $r_1^{(i)}(n)$, that contains values uniformly distributed between 0 and 1.
$[x_g^{(i)}(n) - x^{(i)}(n)]$ calculates a vector towards the global best which is also influenced by some randomness $r_2^{(i)}(n)$. PSO has two options to focus on every iteration. The first option is *diversity*, that means particles are scattered, searching a large area but imprecise. The second option is *convergence* that means particles are close together, searching a small area very precise. The best result can be achieved through a combination of both.

2.4 Hilbert Curves

A Space Filling Curve is a special line of the mathematical calculus that fully covers a two or three dimensional area. Giuseppe Peano (1858-1932) was the first to discover them in 1890. He wanted to create a continuous mapping construction from the unit interval onto the unit square [7]. Space Filling Curves have a wide field of purpose in computer science. They are used specially to linearize multidimensional data, e.g. matrices, images and tables. With their help it is possible to simplify data operations like load-store operations, matrix multiplications and updating and partitioning of data sets by finding an efficient way to go through the data.

Definition 2.2. *Hilbert Curve* [10]. The unit square is divided into congruent sub squares $Q_n^{(k)}$ with side length 2^n. The only condition is, that neighboring sub intervals are mapped onto neighboring sub squares, whereby the square that is next to the zero position is always the first and the one that is next to the point (1, 0) is always the last. If we are now connecting the center of these squares in the right order, we get unequivocal curves C_n (Fig. 1).

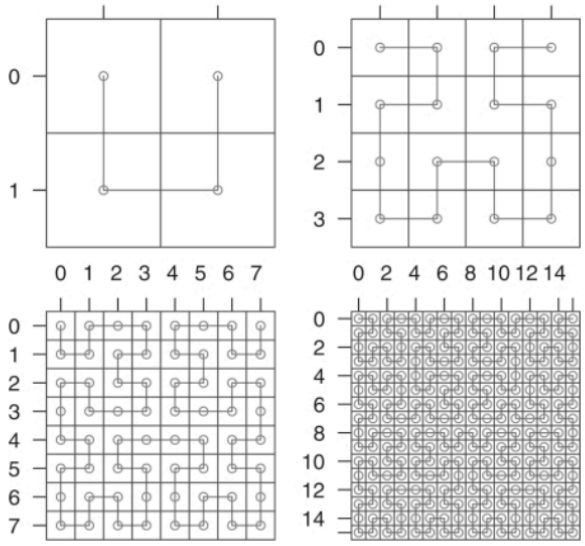


Fig. 1. Level 1-4 of the Hilbert Curve

3 Implementation Details

This section shows the practical changes and extensions that were necessary to implement for the experimental procedure. At first, some new classes had to be implemented to lay the basis for the new *Manganese-Nodule-Model*. These new classes help us to represent the nodules on the map as well as for the calculations in the back. In section 3.3 is described how these new classes get connected to the existing simulation and visualization.

3.1 New Classes

The class *DeployRing* deploys the robots in a ring shaped way. It is part of the *DeploymentStrategy* interface. The deployment algorithm is similar to the *DeployCircle*, but has a specific radius right from the beginning. Objects of type *ManganeseNodule* represent the manganese nodules in the backend simulation. Each *ManganeseNodule* object also contains a *Coordiantes* Object, which specifies the exact position of the nodule in the coordinate system. The class *VisualManganModule* helps to visualize the manganese nodule in the simulation. For each available *ManganeseNodule* object in the corresponding list, a new *VisualManganNodule* object gets created every iteration.

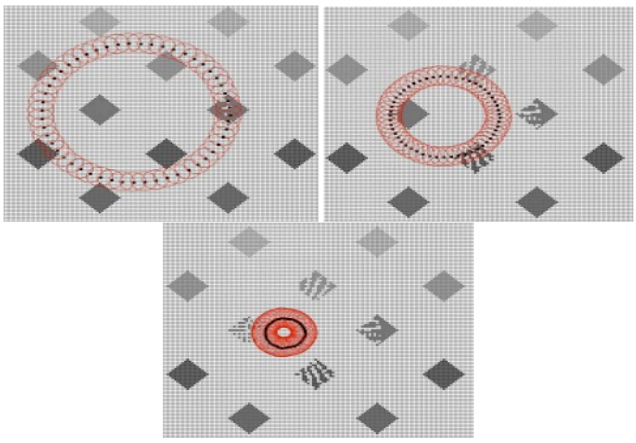


Fig. 2. DeployRing; Top Left initial deploy.

Objects of type *ManganeseNodule* represent the manganese nodules in the backend simulation. They have two attributes. One is the size of the nodule that ranges from 1-7 as an integer. The other one is the status if the nodule is available or already collected. Each *ManganeseNodule* Object also contains a *Coordiantes* Object, which specifies the exact position of the nodule in the coordinate system.

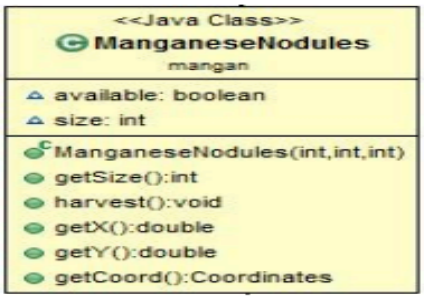


Fig. 3. Visual Class Diagram *ManganeseNodule*.

The class *VisualManganModule* helps to visualize the manganese nodule in the simulation. For each available *ManganeseNodule* object in the corresponding list, a new *VisualManganNodule* object gets created every iteration. This class includes the conversion from the size as an integer to the corresponding grey tone for placing it into the map in the application.

3.2 General Evolution

The *getMangan*() method was advised due to the new structuring of the benchmarks. As there is not manganese on every position anymore, this function needs to check if there is a manganese nodule on this position. If so, the size of it is also returned. The *step*() method helps creating the route to an specific space filling algorithm described later.

3.3 Benchmarking

The benchmarks are provided as independent files. It was necessary to create new classes *ManganeseNodule* and *VisualManganNodule*. These two classes help us to simulate the collection of manganese by our autonomous agents. The class *ManganesNodule* is thereby necessary for all backend happenings and the class *VisualManganNodule* is necessary for visualization in the graphical user interface. The visualization part is done by the *VisualManganNodule* class. The files are deposited in the project archive. Each line represents a *y*-value and each char represents a *x*-value in the coordinate system of the graphical user interface. The lines are filled with numbers from 0 to 7 in accordance with the size of the nodule, where zero means that no nodule can be find on this position. The user can choose between three options: MAP 1, MAP 2 or MAP 3 and load them. Then the associated file gets scanned and the nodules created.

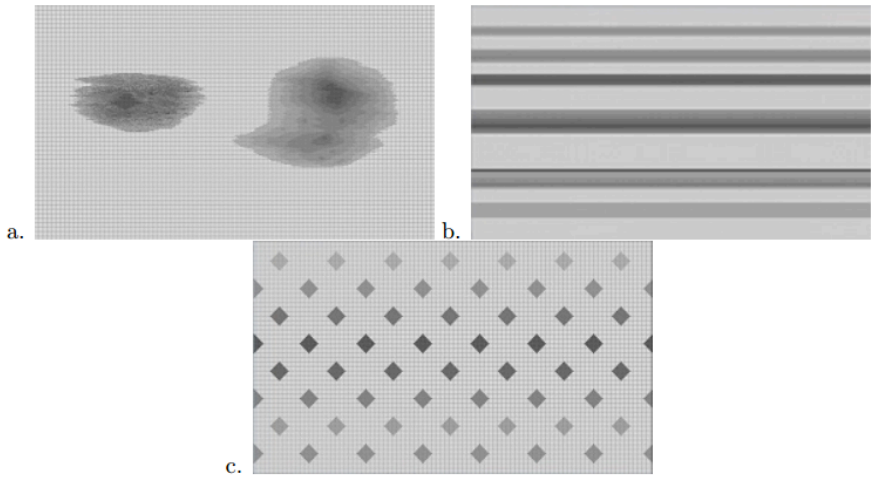


Fig. 4. Benchmarks: Fields (a), Lines (b), Diamond (c). All three graphics represent benchmark maps. The benchmarks include manganese nodules from size 1-7.

3.4. Hilbert Algorithm

The Hilbert algorithm is implemented with a recursive function which follows the description of section 2.4. The function is called every time when the agent moves into the next unit square. The function calls varying from clockwise rotation to negative rotation which means counterclockwise. The ground structure of how going through the 9 sub-squares is fixed implemented. The algorithm function receives four parameters:

double len:	initial step length.
int direction:	specifies the starting direction on the coordinate system in degree.
int rot:	indicates whether the curve should run clockwise or not.
int deep:	determines how many levels deep the algorithm should go.

```
hilbertAlgorithm(length, direction, rotation, deep){
if under lowest level then
    return;
end
    hilbertAlgorithm(length, direction, clockwise rotation, deep-1);
    step forward with given length and direction;
    direction turn clockwise with given rotation degree;
    hilbertAlgorithm(length, direction, counterclockwise rotation, deep-1);
    step forward with given length and direction;
    direction turn clockwise with given rotation degree;
    hilbertAlgorithm(length, direction, clockwise rotation, deep-1);
    step forward with given length and direction;
    hilbertAlgorithm(length, direction, counterclockwise rotation, deep-1);
```

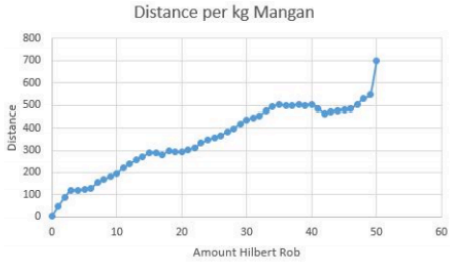
Fig. 5. Hilbert Algorithm

4 Experimental Results

The measured variables are the distance and the collected amount of manganese of all robots in one pass. The difference of robots between *Rob Total* and *Rob Hilbert* are robots behaving after the principles of Moving Algorithms in section 2.2.

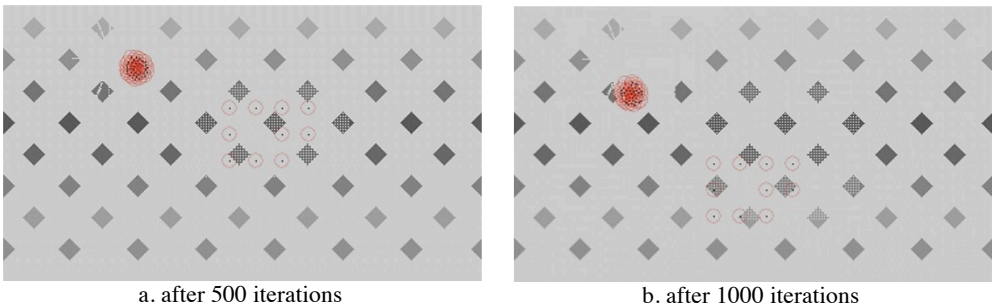
We increase successively the number of Hilbert Robots and ran 1000 iterations with every increase. It runs with the benchmark Diamonds and the deployment strategy Square. This experiment runs with the benchmark Diamonds and the deployment strategy Square. With every increase of the number of Hilbert Robots, the covered distance of all robots increases by 40,000-60,000 m with an average increase of 56,569.85 m. The collected manganese does not increase constantly as well. The global maximum of 5335 kg is reached with a constellation of 46 Hilbert Robots (see Fig. 6). The biggest jump is between the first and the second measurement, with an increase of 562%. If we have a look at the efficiency in fig. 6 (left diagram) we see a raising graph with some flat parts, all amounts of one flat part have the same efficiency, that is the case for the amount of 3-6 Hilbert Robots (average absolute deviation 3.2 m), for the amount of 15-21 Hilbert Robots (average absolute deviation 5.2 m) or 34-40 Hilbert Robots (average absolute deviation 2.9 m).

a. Collected amount of manganese of Diamond, Square, Hilbert 0-50.



b. Relation between the total amount of collected manganese and the distance all robots have covered, for the experiment Diamond, Square, Hilbert 0-50.

Fig. 6. Analysis Hilbert 0-50 increase.



a. after 500 iterations b. after 1000 iterations

Fig. 7. Diamond, Square, Hilbert 0-50: Screenshots experimental procedure.

The correlating local minimum occurs with the amount of 42 Hilbert Robots and 464 m per kg manganese. In this case, the total amount of manganese breaks down roughly 21% compared to the simulation run with 49 Hilbert Robots.

5 Conclusion and Future Work

As this work was focused only on the beginning in combining swarm behavior with specific space filling curves, there are further things to develop in order to get deeper into this topic. It would be conceivable to think of different leaders with different weightings. For example, the leader who collected the most manganese in the last 10 iterations could get the highest weight in calculating the next position of each agent of the swarm. In addition to that, a distributed system could be implemented and thereby the communication between the agents would be extended. To really get a maximum amount of manganese, the swarm could divide and follow different leaders. The number of agents who join a specific leader could vary. If the leader loses strength, more and more agents join another swarm. The leader stays on his route; this keeps the chance high to find new manganese nodule fields. If a leader does not collect any manganese for a longer period, he may become a follower and joins a swarm. This could also be possible the other way around. If there is a big swarm, new leaders could be chosen to search in a specific direction. Another interesting direction would be to combine space filling curves with genetic algorithms. It would be imaginable to build populations with different amounts of deterministic and swarm agents, like this work already did, but keep on developing the next generation after the principles of genetic algorithms. As extension, we can consider a changing environment with hundreds or thousands of agents.

References

1. S. Canyameres, D. Logofatu, Platform for Simulation and Improvement of Swarm Behavior in Changing Environments, 10th International Conference Artificial Intelligence Applications and Innovations, AIAI 14, Springer LNCS, Island of Rhodes, Greece, 2014.
2. W. Reynolds, Boids (simulated flocking), http://www.red3d.com/cwr/boids [Accessed 10 June 2017].
3. W.-J. Shyr, "Parameters Determination for Optimum Design by Evolutionary Algorithm", DOI: 10.5772/9638 [Accessed 10 June 2017].
4. B. Fry and C. Reas, "Processing." https://processing.org/ [Accessed 10 June 2017].
5. F. Rodriguez and C. Garcia-Martınez, "An Artificial Bee Colony Algorithm for the Unrelated Parallel Machines Scheduling Problem", PPSN XII (II) pp. 143-152. Springer, Taormina (2012).
6. J. Kennedy and R. Eberhart, Particle swarm optimization, IEEE Conference on Neural Networks, 4:1942-1948.
7. M. F. Barnsley, Fractals Everywhere, Dover Books on Mathematics, New Edition, ISBN 978-0486488707 (2012).
8. Detailed requirements for the first prototype. http://informaticup.gi.de/fileadmin/redaktion/Informatiktage/studwett/Aufgabe_Manganernte_.pdf [Accessed 10 June 2017].
9. J. R. Rossum, Fundamentals of Metallic Corrosion in Fresh Water, http://www.roscoemoss.com/wp-content/uploads/publications/fmcf.pdf [Accessed 10 June 2017].
10. Min Jun Kim Jung Gu Kim, Effect of Manganese on the Corrosion Behavior of Low Carbon Steel in 10 wt.% Sulfuric Acid, Int. J. Electrochem. Sci., 10 (2015) 6872 – 6885.
11. C. Muro, L. Escobedo, L. Spector, R.P. Coppinger, Wolf-pack (Canis lupus) hunting strategies emerge from simple rules in computational simulations, Behavioral Processes, Vol. 88, Issue 3, pp. 192-197 (2011).