Noname manuscript No.
(will be inserted by the editor)

This is a post-peer-review, pre-copyedit version of an article published in ***Evolving Systems.*** The final authenticated version is available online at: *http://dx.doi.org/10.1007/s12530-018-9245-9*

Particle Swarm Optimization Algorithms for Autonomous Robots with Deterministic Leaders Using Space Filling Movements

Doina Logofˇatu · Gil Sobol · Christina Andersson · Daniel Stamate · Kristiyan Balabanov · Tymoteusz Cejrowski

Received: date / Accepted: date

Abstract In this work the swarm behavior principles of Craig W. Reynolds are combined with deterministic traits. This is done by using leaders with motions based on space filling curves like Peano and Hilbert. Our goal is to evaluate how the swarm of agents works with this approach, supposing the entire swarm will better explore the entire space. Therefore, we examine different combinations of Peano and Hilbert with the already known swarm algorithms and test them in a practical challenge for the harvesting of manganese nodules on the sea ground with the use of autonomous agents. We run experiments with various settings, then evaluate and describe the results. In the last section some further development ideas and thoughts for the expansion of this study are considered.

Keywords Autonomous agents · Space filling curves · Particle swarm optimization · Deterministic leaders · Application

1 Introduction

Simultaneously with the applied research of renewable resources, it is useful to find novel ways for opening up fossil ones. As example, manganese nodules can be found on the sea bottom. A considerable application field involves rust and corrosion prevention on steel [14,6]. The degradation could be reduced substantially by collecting these manganese nodules from the sea bottom using specialized robots.

Doina Logofˇatu · Christina Andersson · Kristiyan Balabanov
Computer Science Department of Frankfurt University of Applied Sciences, 1 Nibelungenplatz
60318, Frankfurt am Main, Germany
E-mail: logofatu@fb2.fra-uas.de

Gil Sobol
Industrial Engineering, Technion - Israel Institute of Technology

Daniel Stamate
Department of Computing, Goldsmiths College, University of London, London SE146NW, UK

Tymoteusz Cejrowski
Gdan´sk University of Technology, Gabriela Narutowicza 11/12, 80-233 Gdan´sk, Poland

Our focus in this work is to evaluate different ways in handling the movement of these fictional robots as autonomous agents.

The experiments can be extended to cover other collecting tasks. The base for our application is a framework for simulation and improvement of swarm behavior in changing environments [15], which we redesign and extend. It simulates the swarm behavior by using the principles of Craig W. Reynolds [12] pointed out here in Section 2.2. The main purpose of the framework regarding the application 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 gathered together, there is no more movement and the simulation ends. Naturally manganese occurs in form of nodules, thus it is distributed uniformly. For the different forms of the manganese distributions, we created several benchmarks used in the results' comparison. The next step of improvement would be the collecting procedure. The greater distance the agents move, the higher is the probability to find manganese. Consequently, we intend to reach a way for passing through a larger area. The easiest solution would be to define for each agent its own path. This would probably scatter the swarm because of the bad orientation, the changing environment and the uneven surface. Most of the research works regarding swarm behavior are inspired by nature like genetic algorithms or particle swarm optimization. These outcomes focus on fish schools or bird flocks. An alternative discussion could consider, for example, a pack of wolves. A pack of wolves means actually autonomous individuals with a specific hierarchy. Not every wolf has the same power regarding decisions for the pack. Normally there is one wolf who leads the group and the others are followers [9]. This contribution aims to study this notion more closely. We intend to set one or more leaders, who will move after a given route, but still be part of the swarm, and the rest calculate their new position, that means every iteration in consideration of all agents.

2 Previous Work

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

2.1 Framework for Adaptive Swarms Simulation and Optimization

In its core the application is based on [15]. The framework is an application that runs a simulation of agents, or robots in the context of aforementioned practical challenge [11], using various moving algorithms: Random, Square, Circle, Gauss, and Bad Centers [15]. It contains several fundamental deployment strategies used from where the moving algorithms start. The front end uses the open source framework of *processing.org* [3]. The whole visualization part is done in the *Visualization* class with support of its derived class *VisualRobot*, which serves to represent the agents as 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 autonomous agents, as well as the collection of manganese nodules.

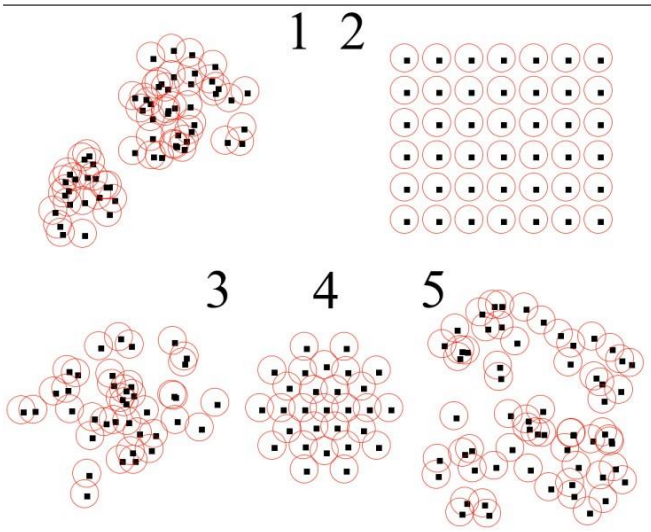


Fig. 1 Different deployment methods for the agents: Random (1), Square (2), Gaussian (3), Circular (4) and double Gaussian (5)

Manganese is located on every position in the coordinate system. The distance in walked meters of all agents together is also counted. Moreover, it is possible to set at the initial number of agents, which must be between 2 and 100.

The starting positions of the agents determine the first decisions to take and may influence the choice of the 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 dispersed, many areas may remain uncovered or clearing them would be too inefficient. In the contest, the initial methodology to distribute the agents was never specified and each simulation uses a different starting set. Therefore, it is important to have the chance to test the same moving paradigm with different deployments. Therefore, five starting configurations were implemented, following different regular patterns (square, circular) or mathematic distributions (random, Gaussian and double Gaussian).

The *Random* deployment (Fig. 1, (5)) drops every agent 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 agents too far away from the rest, which would then become isolated. An interesting line of research could study which algorithms lead to the isolation of agents. 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 agents going to be distributed, even without their exact coordinates or the distance between them. Obviously, parameters such as the deviation can be manipulated in every simulation scenario. 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 in number of items and

deviation. Both means are calculated in a way that some agents from one kernel can perceive some from 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 agents 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 in order to try and 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 agents on the circumference until this is full and a new ring can be added.

2.2 Moving Algorithms

Artificial systems are, for example, needed 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 moves forward individually while considering 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 [13]. The previous work [15] uses a simplification of the bird flock movement described by Craig W. Reynolds [12]. The idea was to develop algorithms that simulate swarm behavior inspired by flocks of birds or schools of fish. Therefore, three criteria, that every agent follows at each iteration, were set up. The contribution implemented three different algorithms that run simultaneously: cohesion, separation, alignment.

The limited communication capacity between the agents makes it too complex to determine the shape or the positions of all the other agents (a strong communicating network should be established) and it would differ too much from real life scenarios. Nature-inspired algorithms for ants [11] or bees [14] 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 [6].

The moving paradigm chosen always follows a same overall concept, 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 agent 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 from the following rules:

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

Equation 1 calculates the average position of nearby agents and tends to follow the principle of cohesion and avoid the group spreading around or a agent to travel too far away, which may result in losing contact due to its limited view. After testing, this basic concept was not resembling a real-like flow good enough, 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, a separation algorithm (see Eq. 2) is also used in the final calculation of the new movement.

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

In this case, the algorithm finds the opposite value of the cohesion formula. However, the influence of the algorithm increases only when the object is very close to the others. For this reason, the average point is calculated among the agents 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 is significantly increased when the agent is too close to others, so it would avoid an actual crashing between boids.

Finally, Eq. 3 computes the average direction in which the neighboring agents are moving. 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.

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

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

2.3 Particle Swarm Optimization

Particle Swarm Optimization (PSO) was first proposed in 1995 by J. Kennedy and R. Eberhart [5]. The idea was to build swarm behavioral algorithms for solving problems by iteratively improving a candidate's solution until termination criteria is satisfied [1]. It is similar to a genetic algorithm in terms that both algorithms are initialized with a random population, in PSO called particles. The difference is that in PSO algorithms, each particle is assigned 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.

$$x^{(i)}(n+1) = x^{(i)}(n) + v^{(i)}(n+1), \qquad n = 0,1,2,...,N-1 \tag{4}$$

Equation 4 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 is constructed with the following iteration:

$$v^{i}(n+1) = \underbrace{v^{i}(n)}_{inertia} + \underbrace{r_1^{(i)}(n)\left[x_p^{(i)}(n) - x^{(i)}(n)\right]}_{personal\ influence} + \underbrace{r_2^{(i)}(n)\left[x_g^{(i)}(n) - x^{(i)}(n)\right]}_{global\ influence} \tag{5}$$

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 focuses on two goals in every iteration. The first one is *diversity*, which means particles are scattered, traversing a large area but imprecisely. The second goal is *convergence*, i. e. the particles are close together, examining a small area very precisely. The best result can be achieved through a combination of both.

2.4 Space Filling Curves

A Space Filling Curve is a special function of calculus that fully covers a two or three dimensional space. Giuseppe Peano (1858-1932) discovered them first in 1890. He wanted to create a continuous mapping construction from the unit interval onto the unit square [1].

2.4.1 Peano Curve

Until 1890 one assumed that a constant curve with parametric function of only one variable $x = \varphi(t)$ and $y = \psi(t)$, cannot reflect surjectively the unit interval onto the unit square. The reason for this was the theorem of Eugen Natto, who showed that a bijection must be unsteady to satisfy this. However, Peano found a steady function f_p, such that $f_p(I) = 2$.

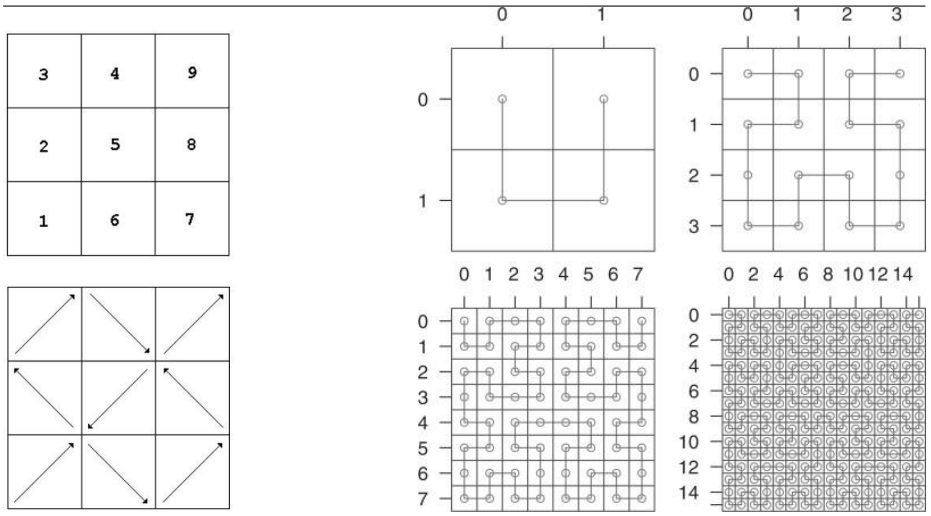
Definition 1 (Peano Curve [6]) The projection $f_p: I \to 2$ with

$$f_p(0_3.t_1t_2t_3t_4) = \begin{pmatrix} 0_3\ t_1(k^{t_2}t_3)(k^{t_2+t_4}t_5)\ \ldots \\ 0_3\ (k^{t_1}t_2)(k^{t_1+t_3}t_4)\quad \ldots \end{pmatrix}$$

and the operator $k^{t_j} = 2 - t_j (t_j = 0,1,2)$, where I is the unit interval [0,1], $0_3.t_1t_2t_3t_4$ is a ternary number with $t_j \in \{0,1,2\}$ and k^v is the v-th iteration of k, we call Peano Curve.

So according to this definition we have:

$$f_p(0_3.00t_3t_4\ldots) = \begin{pmatrix} 0_3.\ 0\xi_2\xi_3\xi_4\ \ldots \\ 0_3.\ 0\eta_2\eta_3\eta_4\ \ldots \end{pmatrix}$$



(a) (b)

Fig. 2 Construction order and orientation for the Peano Curve (a) and for the Hilbert Curve levels 1-4 (b).

To create the Peano's curve we start at point (0,0) and finish in the diagonal corner at point (1,1). The starting point of a sub square must be the endpoint of the previous sub square. Figure 2 (b) illustrates where the start- and endpoints are marked as arrows.

2.4.2 Hilbert Curve

Although it was Peano (1890) who produced the first space-filling curves, it was Hilbert (1891) who first popularized their existence and gave an insight into their generation. His approach was, if the unit interval can be mapped steadily onto the unit square, then also sub intervals can be mapped steadily onto sub squares. In the first step, Hilbert divided the unit interval into four sub intervals of the same size as well as the unit square into four equally sized sub squares, where each sub interval is mapped onto one sub square. If we repeat this arbitrary frequently, I gets divided into 2^{2n} for $n = 1,2,3,...$ congruent subs.

Definition 2 (Hilbert Curve [6]) The same as the Peano Curve: we divide the unit square 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 now connect the center of these squares in the right order, we would get unequivocal curves C_n (see Fig. 2 (a)).

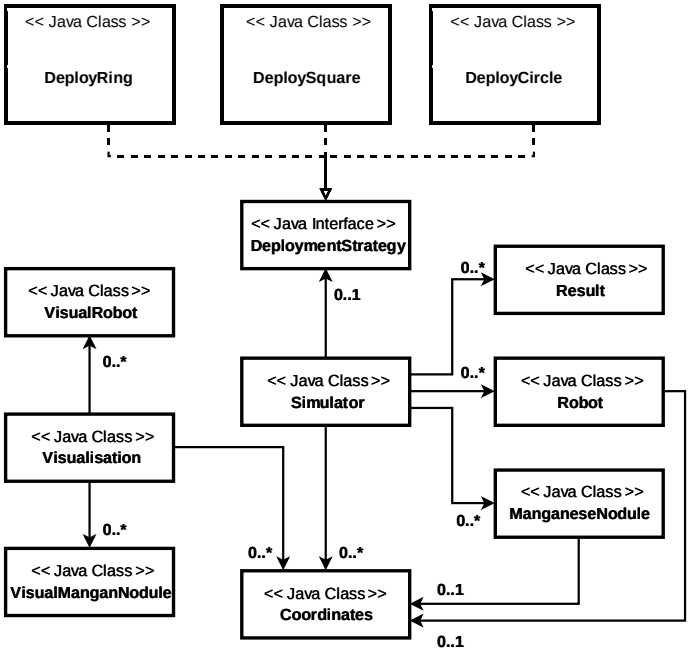


Fig. 3 Conceptual class diagram of the developed application.

3 Implementation Details

For our tests we researched for already existing libraries, frameworks and any adequate implementations that offer exactly the functionality we needed regarding space filling curves. Proprietary products were discarded as the scope of our project was too small for such an investment. We evaluated the existing *open-source* implementations according to three criteria [10]: 1) *maturity* – is the product still in the development stage, i. e. prone to bugs, or has a stable release already been deployed; 2) *longevity* – would the developers of the product continue to improve it and provide support for it; 3) *flexibility* – how hard it would be to modify the product or to integrate new functionality into it; Unfortunately, we had to resort to creating our own software, specially designed for the intended tests. The application in question was developed using Java due to the rich variety of existing libraries. This section gives a short overview of the more interesting implementation details such as the concept class diagram (see Fig. 3), the Peano and Hilbert algorithms, as well as the type of benchmarks used for the tests.

3.1 Key Classes

The classes *Visualisation* and *Simulator* together build the core of the environment, in which the tests are run. *Simulator* is the backbone class "glueing" everything else together, while *Visualisation* together with a set of smaller classes, such as *VisualRobot*, *VisualManganNodule* and *Coordinates*, act as the interface between the observing user and the conducted experimental procedures. *Robot* is the class representing the fictional robots gathering manganese, i. e. the autonomous

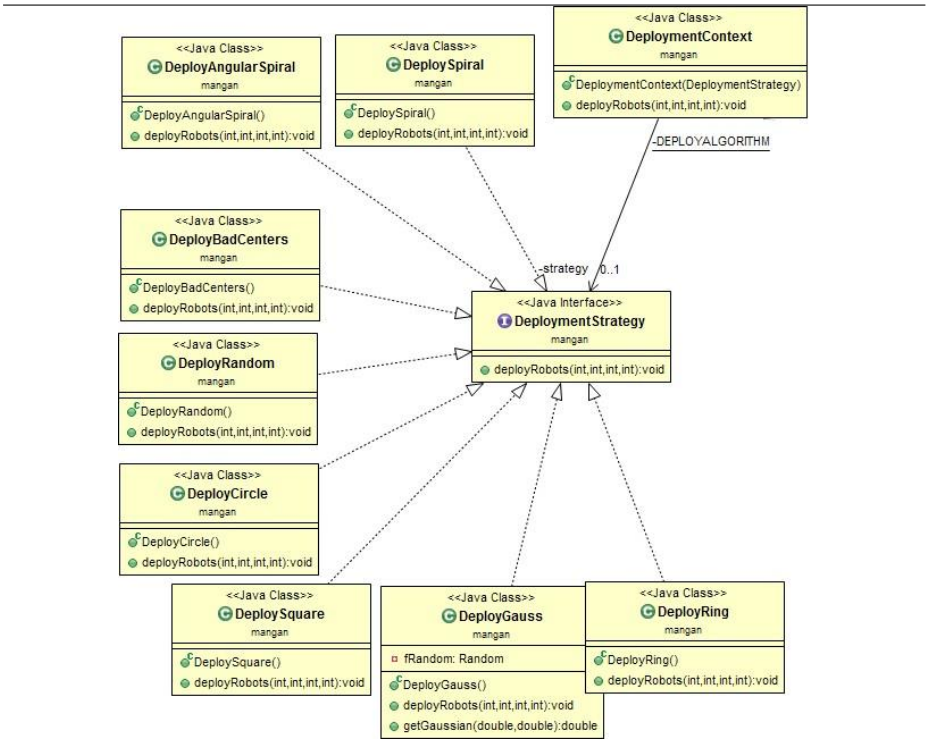
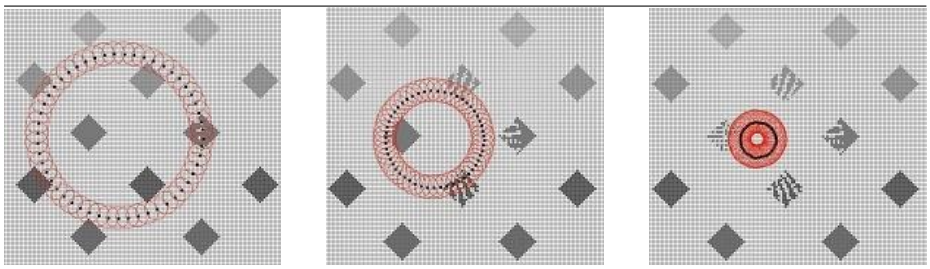


Fig. 4 Different deployment approaches encapsulated using the *Strategy design pattern*.

agents, and *ManganeseNodule* represents the manganese deposits in the backend simulation with their respective attributes and applicable actions. For instance an object of type *ManganeseNodule* has two attributes: the size of the nodule as an integer that ranges from 1 to 7; the state of the nodule, i. e. whether it is still available or it has already been collected. Each such object also contains a *Coordinates* object, which specifies the exact position of the nodule in the coordinate system.

As we experimented with a wide variety of deployment strategies an elegant approach was needed to encapsulate such functionality in a way allowing for easy modification and interchanging. Therefore, the *Strategy design pattern* described by [4] was used: a *Java interface* was created describing the basic functionality a deployment strategy should offer (see Fig. 4); The class *DeployRing* is an example for a deployment strategy. It deploys the agents in a ring shaped way and it is similar to *DeployCircle* for instance, but has a specific radius right from the beginning. A ring can cover a wide area and by shrinking towards its center, while following the rules described in Section 2.2, it is assured that no two agents would cross routes. As a result a agent would not traverse a position where another agent has already collected all the manganese. Considering the principles of MapReduce this strategy has excellent results regarding the amount of collected manganese. Finally the *Result* class is used to log the progress during the execution of an experimental procedure and the final statistics.



(a) Initial (b) Intermediate (c) Final

Fig.5 DeployRingstages.

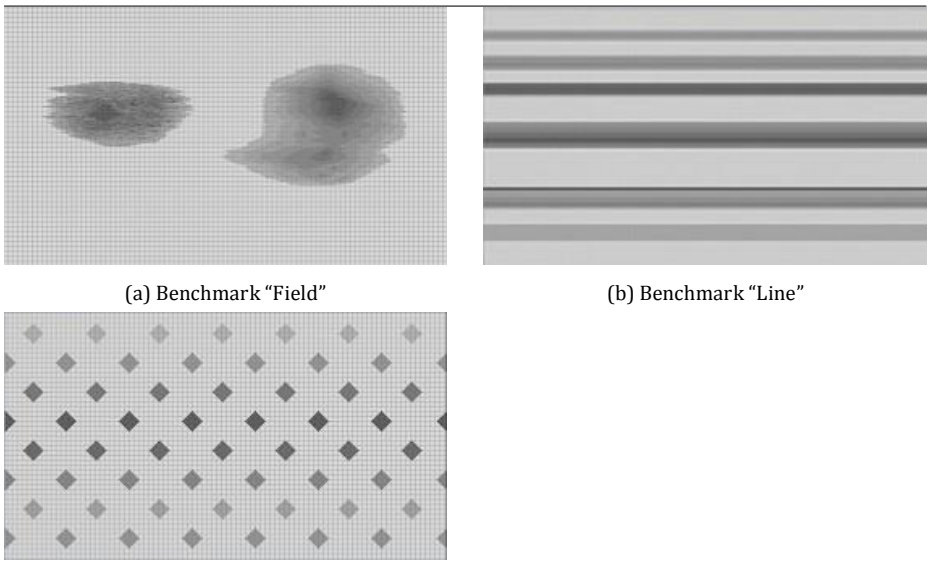
Fig.6 Nodulesizescale

3.2 General Evolution

The *getMangan()* method was devised 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 at the given position at all. If so, its respective size is returned. The *step()* method helps to create the route according to a specific space filling algorithm as described later. The method receives the step length (*double* len) and direction (*int* dir). With this information it calculates the next position and places it in the route list.

3.3 Benchmarking

To achieve a better comparison between the different algorithms and their efficiency, benchmarks were introduced. The benchmarks are provided as independent files. It was necessary to create the 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 purposes in the graphical user interface. Every *ManganeseNodule* has the following attributes: 1) *coordinates(x/y)*; 2) *size*; 3) *activate*. The coordinates help to exactly define the nodule's position on the map and can be compared to the agents' positions so that they can collect it later in the simulation. A *boolean* variable shows if the nodule is still activate or already depleted. Each active nodule has a specific size given as an integer. It ranges from 1 to 7 where seven is the biggest and one the smallest possible value. This is also shown in the graphical user interface with different gray tones (refer to Fig. 6).



(a) Benchmark "Field" (b) Benchmark "Line"

(c) aBenchmark "Diamond"

Fig. 7 Benchmarks in Order: Fields, Lines, Diamond. All three graphics represent benchmark maps. The benchmarks include manganese nodules from size 1-7. The benchmark "Fields" (a) consists of two big fields. The left hand field does have a center with manganese nodules of size 7. All manganese nodules around it do not have a specific pattern. The right hand field increases its nodule size from the outside to the inside. The benchmark "Lines" (b) consists of six bars that go through the full map width. All bars are filled with manganese nodules of different sizes, where each line is always filled with same size nodules. The benchmark "Diamond" (c) consists of 56 diamonds that are distributed uniformly over the map. Each diamond consists only of nodules of one size

The visualization part is done by the *VisualManganNodule* class. Each of its instances represents one nodule on the map. Objects of this class are created and deleted after every iteration. This class has only one function to be called: *display()*. This function creates a rectangle at the right position with the right gray tone. The maps are based on three ASCII files and can be chosen in the graphical user interface. The files reside within the project archive and are filled with ASCII characters representing the numbers between 0 and 7 inclusive (refer to Fig. 8 for an example). The row in which a character is placed represents its *y*-value and the column its *x*-value in the coordinate system of the graphical user interface. Each number corresponds to the size of the nodule at the respective position, where zero means that no nodule can be found at that position. The user can choose between three options: MAP 1, MAP 2 or MAP 3 and load them. Then the associated file is scanned and the nodules created.

3.4 Peano Algorithm

The Peano algorithm is implemented with a recursive function that follows the description in Section 2.4.1. The function is called every time the agent moves

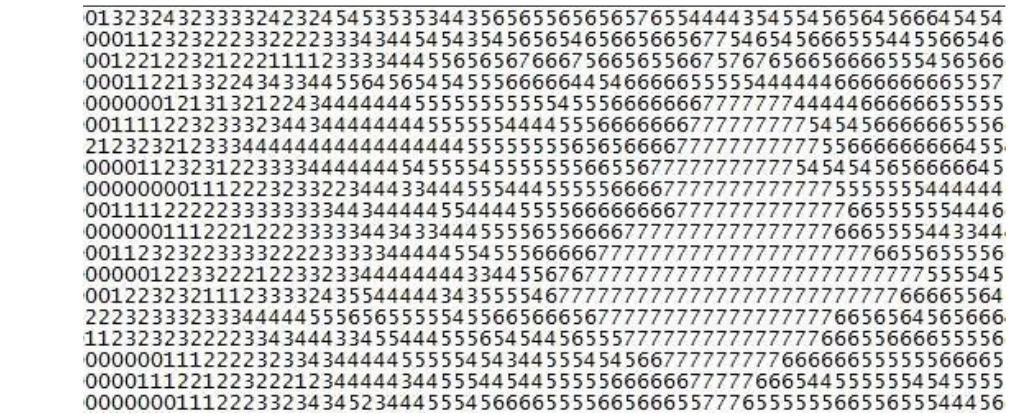


Fig. 8 An extract from the deposit map file for the benchmark Fields. A rich manganese deposit is indicated by the group of '7's to the right. The concentration of manganese (size of the nodules) gradually decreases when moving away from the enriched center.

into the next unit square. The function can be called in either clockwise or counterclockwise (negative) rotation rotation. The basic strategy of going through the 9 subsquares is fixed. The pseudo code is shown Alg. 1. 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 counterclockwise; a specific rotation degree has to be inserted
- *int* deep: determines how many levels deep the algorithm should go

3.5 Hilbert Curve

The implementation of the Hilbert algorithm (see Alg. 2) is analogous to that of the Peano algorithm. It is a recursive function that calls itself once in each sub square. Again, the basic approach for going through the sub squares is fixed.

4 Experimental Results

This section presents experimental results we achieved with the extended implementation of the application. The measured variables are the distance and the collected amount of manganese of all agents in one pass. The difference of agents *Rob Total* and the sum of *Rob Hilbert* and *Rob Peano* are agents behaving according to the principles of the moving algorithms described in Section 2.2.

4.1 Diamond, Square, Peano 0-50

In this experiment we increased the number of Peano Agents and ran 1000 iterations with every increase. This experiment runs with the benchmark Diamonds

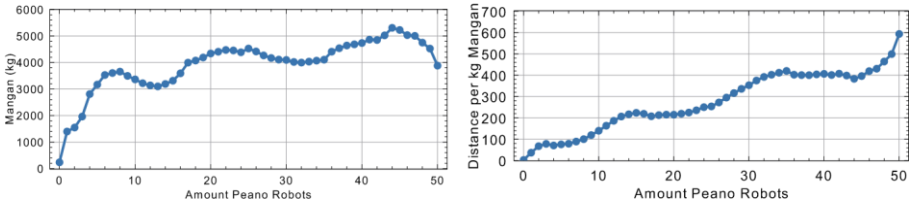
```
peanoAlgorithm(length, direction, rotation, deep){
if under lowest level then | return;
end
        peanoAlgorithm(length, direction, clockwise rotation, deep-1); step
        forward with given length and direction;
        peanoAlgorithm(length, direction, counterclockwise rotation, deep-1); step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

Algorithm 1: Pseudo Code Peano Algorithm

```
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);
}
```

Algorithm 2: Pseudo Code Hilbert Algorithm

and the deployment strategy Square. The results are listed in Table 1. With every increase in the number of Peano Agents, the covered distance of all agents increases by 30,000-50,000 m with an average increase of 46,081.46 m. The collected manganese does not increase constantly. The global maximum of 5308 kg is reached with a constellation of 44 Peano Agents (see Fig. 9, left). The biggest 15 jump is between the first and the second measurement, with an increase of 589%.



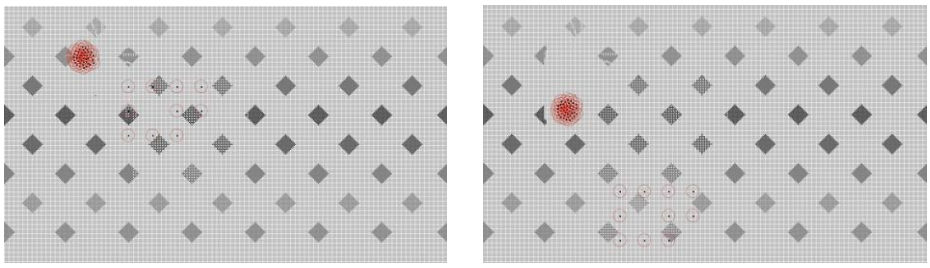
(a) (b)

Fig. 9 Analysis of the experimental procedure Diamond, Square, Peano 0-50 increase: (a) collected amount of manganese; (b) relation between the total amount of collected manganese and the distance all agents (robots in the given context) have covered.

The fewer meters a agent has to travel for the same amount of manganese, the more efficient it is. The right diagram in Fig. 9 shows this relation of average distance per kg manganese for each amount of Peano Agents. The best efficiency occurs, without any Peano agent, in the simulation with an average distance of 3 m per kg manganese. But as we can see from the other diagram (Fig. 9, left) the total amount of manganese is very little. So we want to focus on analyzing all cases where Peano agents are involved. There are a few amounts of Peano agents with a very close distance per kg manganese. This is the case with the amount of 2, 3, 4, 5 and 6 Peano agents (average absolute deviation 3.9 m), with the amount of 16 to 21 Peano agents (average absolute deviation 3 m) or with the amount of 36 to 42 Peano agents (average absolute deviation 2.1 m). Another interesting point is at the amount of 44 Peano Agents, where the total manganese maximum is. The distance per kg manganese diagram shows here a local minimum of 383 m per kg manganese. This leads to the conclusion that we have a reasonably efficient constellation.

4.2 Diamond, Square, Hilbert 0-50

This experiment is similar to the one described in Section 4.1. However, we increased the number of agents and ran 1000 iterations with every increase. This experiment runs with the benchmark Diamonds and the deployment strategy Square.



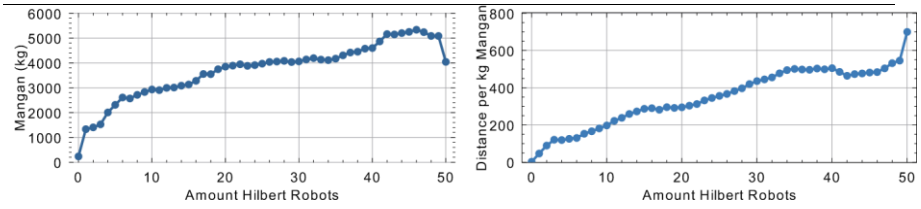
(a) (b)

Fig. 10 Visualization of the experimental procedure Diamond, Square, Peano 0-50: (a) state after 500 iterations; (b) state after 1000 iterations

Table 1 Overview of the results from the experimental procedure Diamond, Square, Peano 0-50. In every subsequent test the number of agents with movement behavior based on the Peano curve was increased.

Exp.	Benchmark	Deploy. St.	Agents	Peano	Hilbert	Mangan	Distance
0	Diamond	Square	50	0	0	237 kg	677.03 m
1	Diamond	Square	50	1	0	1398 kg	51,462.41 m
2	Diamond	Square	50	2	0	1549 kg	104,572.08 m
3	Diamond	Square	50	3	0	1962 kg	153,735.6 m
4	Diamond	Square	50	4	0	2811 kg	199,127.07 m
5	Diamond	Square	50	5	0	3169 kg	240,628.72 m
6	Diamond	Square	50	6	0	3530 kg	279,077.49 m
7	Diamond	Square	50	7	0	3605 kg	321,266.84 m
8	Diamond	Square	50	8	0	3657 kg	367,202.48 m
9	Diamond	Square	50	9	0	3496 kg	417,403.93 m
10	Diamond	Square	50	10	0	3364 kg	470,277.73 m
11	Diamond	Square	50	11	0	3220 kg	525,898.89 m
12	Diamond	Square	50	12	0	3138 kg	584,986.30 m
13	Diamond	Square	50	13	0	3100 kg	640,421.38 m
14	Diamond	Square	50	14	0	3191 kg	692,726.52 m
15	Diamond	Square	50	15	0	3315 kg	741,953.53 m
16	Diamond	Square	50	16	0	3593 kg	788,247.15 m
17	Diamond	Square	50	17	0	3996 kg	830,324.91 m
18	Diamond	Square	50	18	0	4074 kg	868,293.25 m
19	Diamond	Square	50	19	0	4190 kg	902,547.03 m
20	Diamond	Square	50	20	0	4340 kg	933,433.68 m
21	Diamond	Square	50	21	0	4407 kg	968,292.57 m
22	Diamond	Square	50	22	0	4474 kg	1,007,276.48 m
23	Diamond	Square	50	23	0	4456 kg	1,050,369.68 m
24	Diamond	Square	50	24	0	4386 kg	1,097,769.49 m
25	Diamond	Square	50	25	0	4528 kg	1,149,616.11 m
26	Diamond	Square	50	26	0	4418 kg	1,203,643.76 m
27	Diamond	Square	50	27	0	4267 kg	1,260,017.35 m
28	Diamond	Square	50	28	0	4170 kg	1,319,224.76 m
29	Diamond	Square	50	29	0	4110 kg	1,381,581.11 m
30	Diamond	Square	50	30	0	4097 kg	1,447,080.65 m
31	Diamond	Square	50	31	0	4021 kg	1,509,116.10 m
32	Diamond	Square	50	32	0	3998 kg	1,567,840.06 m
33	Diamond	Square	50	33	0	4034 kg	1,623,434.98 m
34	Diamond	Square	50	34	0	4071 kg	1,676,280.30 m
35	Diamond	Square	50	35	0	4108 kg	1,726,845.74 m
36	Diamond	Square	50	36	0	4409 kg	1,775,114.97 m

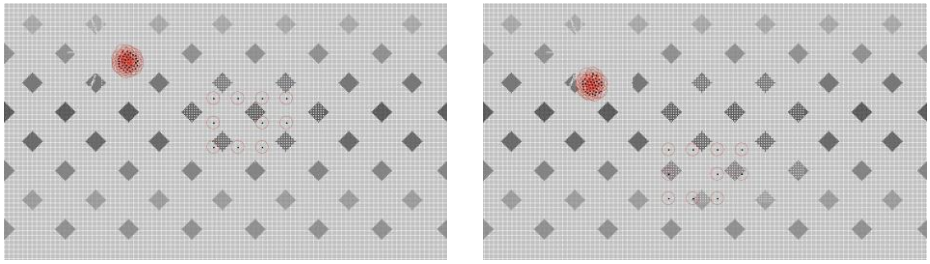
37	Diamond	Square	50	37	0	4539 kg	1,819,105.33 m
38	Diamond	Square	50	38	0	4642 kg	1,858,490.51 m
39	Diamond	Square	50	39	0	4686 kg	1,893,356.90 m
40	Diamond	Square	50	40	0	4735 kg	1,924,538.22 m
41	Diamond	Square	50	41	0	4864 kg	1,951,656.99 m
42	Diamond	Square	50	42	0	4849 kg	1,975,349.62 m
43	Diamond	Square	50	43	0	5024 kg	2,002,892.39 m
44	Diamond	Square	50	44	0	5308 kg	2,034,784.58 m
45	Diamond	Square	50	45	0	5227 kg	2,070,537.02 m
46	Diamond	Square	50	46	0	5033 kg	2,110,861.60 m
47	Diamond	Square	50	47	0	5006 kg	2,155,917.08 m
48	Diamond	Square	50	48	0	4751 kg	2,205,642.77 m
49	Diamond	Square	50	49	0	4526 kg	2,260,180.65 m
50	Diamond	Square	50	50	0	3882 kg	2,304,072.83 m



(a) (b)

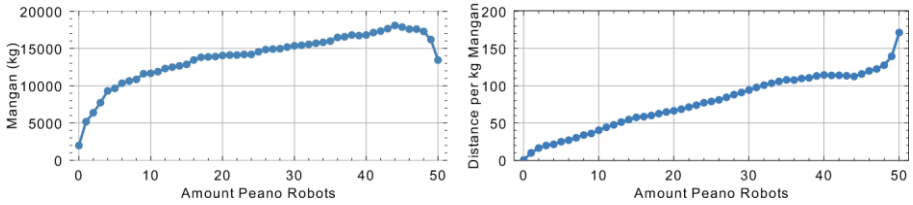
Fig. 11 Analysis of the experimental procedure Diamond, Square, Hilbert 0-50 increase: (a) collected amount of manganese; (b) relation between the total amount of collected manganese and the distance all agents (robots in the given context) have covered.

The experiment provides similar results like in Table 1. With every increase of the number of Hilbert Agents, the covered distance of all agents increases by 40,00060,000 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 Peano Agents (see Fig. 11, left). The biggest jump is between the first and the second measurement, with an increase of 562%. If we have a look at the efficiency illustrated in Fig. 11 (right) we can see a raising graph with some “flat” parts, all amounts of one “flat” part have the same efficiency, which is the case for the amount of 3-6 Hilbert Agents (average absolute deviation
3.2 m), for the amount of 15-21 Hilbert Agents (average absolute deviation 5.2 m) or 34-40 Hilbert Agents (average absolute deviation 2.9 m). This time we do not have a local minimum at the same amount as the maximum in the 18 Total Manganese diagram, like in Section 4.1. The correlating local minimum occurs with the amount of 42 Hilbert Agents and 464 m per kg manganese. Running the simulation only with Hilbert Agents brings very bad results. In this case, the total amount of manganese decreases roughly by 21% compared to the simulation run with 49 Hilbert Agents.



(a) (b)

Fig. 12 Visualization of the experimental procedure Diamond, Square, Hilbert 0-50: (a) state after 500 iterations; (b) state after 1000 iterations.

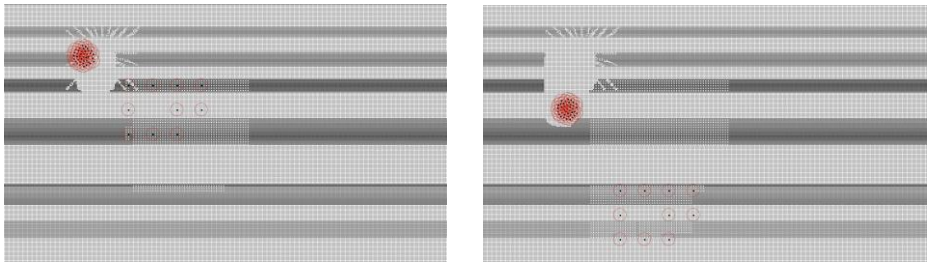


(a) (b)

Fig. 13 Analysis of the experimental procedure Lines, Square, Peano 0-50 increase: (a) collected amount of manganese; (b) relation between the total amount of collected manganese and the distance all agents (robots in the given context) have covered.

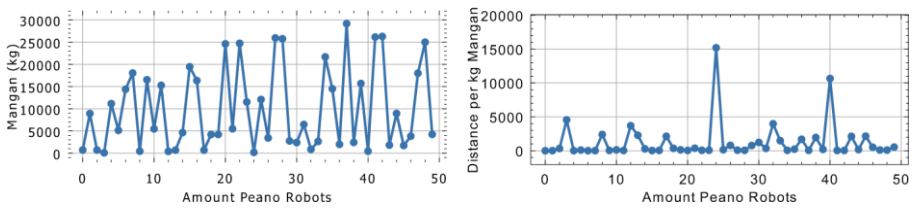
4.3 Lines, Square, Peano 0-50

In this experiment we switched our benchmark to the benchmark Lines. The deployment strategy is Square and we increase the number of Peano Agents from 0-50. This is the same like in the experiment of Section 4.1. With every increase of the number of Peano Agents, the covered distance of all agents increases by 30,000-50,000 m with an average increase of 46,081.46 m. Logically this is exactly the same as in Table 1. The collected manganese increases constantly until an amount of 38 Peano Agents. The global maximum of 18090 kg is reached with a constellation of 44 Peano Agents (see Fig. 13, left). With an amount of 6 Peano Agents we achieve a result of 10,341 kg total manganese. This is more than 50% from what we achieve with our global maximum with 44 Peano Agents. That means, we achieve half of the global maximum with an efficiency of 26.99 m per kg manganese in contrast two 112.48 meter per kg manganese. In conclusion we get 100% more manganese for 416.75% less efficiency. That is in no reasonable relation to the benefits. Overall the distance per kg manganese increases almost linearly up to the global maximum of total manganese with 44 Peano Agents and goes steeply up afterwards. It is striking that this experiment has its maximum with the same amount of Peano Agents, like the experiment of Section 4.1. The only thing that distinguishes these two experiments is the benchmark maps.



(a) (b)

Fig. 14 Visualization of the experimental procedure Lines, Square, Peano 0-50: (a) state after 500 iterations; (b) state after 1000 iterations.



(a) (b)

Fig. 15 Analysis of the experimental procedure Lines, Circle, Peano 0-49 increase: (a) collected amount of manganese; (b) relation between the total amount of collected manganese and the distance all agents (robots in the given context) have covered.

4.4 Lines, Circle, Peano 0-49

In this experiment we use a different deployment strategy for the first time in our series of experiments, namely Circle. The circle deploys on a random position in the map in contrast to the deployment strategy Square, where the square is deployed always at the top left corner of the map. The benchmark remains Lines and the procedure remains increasing the Peano Agents.

The results from this specific experiment might look quite perplexing at first glance as they do not follow any obvious pattern like in the previous test scenarios (refer to Fig. 15). If we, however, ignore the outliers in the plotted data (see Fig. 15 (a)), i. e. test cases with very low quantity of gathered manganese, we would see a slight improvement in the gathering efficiency proportional to the number of Peano agents. This can be attributed to the fact that the circle deployment is placed randomly on the benchmark map. As a result agents often disappear from the map after reaching its boundaries and figuratively speaking “fall off the edge of the simulation world”, hence no correct measuring can be done. This problem can be easily dealt with by introducing *boundary conditions* for the simulation space similar to those used in cellular automatons [2, chap. 2]. One possibility would be to set *periodic* boundary conditions by connecting opposite ends of the map and essentially eliminating the boundaries, i. e. transforming the 2-dimensional map into a 2-dimensional toroid (torus). This insight suggests that combining Moving Algorithms (Section 2.2) and space filling curves (Section 2.4) makes sense, if there is knowledge of the direction in which the manganese nodules are located prior to starting the tests or if the environment is bounded.

5 Conclusions and Future Work

As this work was focused only on the beginning in combining swarm behavior with space filling curves, there are many more things to work on 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 when 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 the maximum amount of manganese, the swarm could divide and follow different leaders. The number of agents who join a leader could vary. This could be determined by the strength of the leader which is defined by the total amount of

collected manganese in the last few iterations. 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 thing would be to combine space filling curves with genetic algorithms. One could easily imagine building populations with different amounts of Hilbert, Peano and swarm agents, like this work already did, but keep on developing the next generation after the principles of genetic algorithms. All this would be interesting to analyze in an environment with hundreds or thousands of agents.

References

1. Barnsley M. F., Fractals Everywhere, Dover Books on Mathematics, New Edition, ISBN 978-0486488707 (2012)
2. Floreano, D., Mattiussi, C.: Bio-inspired artificial intelligence : theories, methods, and technologies. MIT Press, Cambridge (2008)
3. Fry B., Reas C., Processing, https://processing.org/ [Accessed 8-May-2018]
4. Vlissides J., Johnson R., Helm R., Gamma E.: Design Patterns: Elements of Reusable Object-Oriented. Springer, Addison-Wesley Professional, Berlin (1994)
5. Kennedy J., Eberhart R., Particle swarm optimization, IEEE Conference on Neural Networks, 4, 1942–1948
6. Kim Min Jun , Kim Jung Gu, Effect of Manganese on the Corrosion Behavior of Low Carbon Steel in 10 wt.% Sulfuric Acid, Int. J. Electrochem. Sci., 6872–6885, 10 (2015)
7. Logof˘atu D., Sobol G., Stamate D., Particle Swarm Optimization Algorithms for Autonomous Robots with Leaders Using Hilbert Curves, 18*th* International Conference on Engineering Applications of Neural Networks (EANN 2017), pp. 535-543. Springer, Athen (2017)
8. Logof˘atu D., Sobol G., Stamate D., Balabanov K., A Novel Space Filling Curves Based Approach to PSO Algorithms for Autonomous Agents, 9*th* International Conference on Computational Collective Intelligence (ICCCI 2017), pp. 361-370, Springer, Nicosia (2017)
9. Muro C., Escobedo L., Spector L., Coppinger R. P., Wolf-pack (Canis lupus) hunting strategies emerge from simple rules in computational simulations, Behavioral Processes, Vol. 88, Issue 3, 192–197 (2011)
10. Norris, J. S.: Mission-critical development with open source software: lessons learned. In: IEEE Software, pp. 42–49, 21 (January), (2004)
11. Detailed requirements for the first prototype, http://docplayer.org/22922344-Informaticup-informaticup-2014-aufgabe-manganernte-einfuehrung-1-aufgabe.html [Accessed 8-May-2018]
12. Reynolds W., Boids (simulated flocking), http://www.red3d.com/cwr/boids [Accessed 8May-2018]
13. Rodriguez F., Garcia-Martinez C., An Artificial Bee Colony Algorithm for the Unrelated Parallel Machines Scheduling Problem, PPSN XII (II), 143–152, Springer, Taormina (2012)
14. Rossum J. R., Fundamentals of Metallic Corrosion in Fresh Water, http://www.roscoemoss.com/wp-content/uploads/publications/fmcf.pdf [Accessed 8-May-2018]
15. Canyameres S., Logof˘atu D., Platform for Simulation and Improvement of Swarm Behavior in Changing Environments, 10*th* International Conference Artificial Intelligence Applications and Innovations (AIAI 14), Springer LNCS, Island of Rhodes, Greece (2014)
16. Shyr W.-J., Parameters Determination for Optimum Design by Evolutionary Algorithm, Convergence and Hybrid Information Technologies,, DOI: 10.5772/9638, (2010) https://www.intechopen.com/books/convergence-and-hybrid-informationtechnologies/parameters-determination-for-optimum-design-by-evolutionaryalgorithm [Accessed 8-May-2018]