NTNU

Norwegian University of Science and Technology

Artificial intelligence

IE502014

Swarm intelligence

Assignment 4

Author:
Nicolò Pinciroli

April 3, 2020

Introduction

In this assignment, I have dealt with Particle Swarm Optimization. I have used Python in order to develop the examples, since I have found a library - vpython - that allows to have tridimensional representations quite easily.

I have decided not to use Unity3D because of its slowness on less powerful computers. Python, on the other side, does not contain some features, such as automatic collision detection and physics. Moreover, it is more difficult to visualize the objects, but since the purpose of the proposed tasks wasn't to create graphically pleasant objects, but rather to visualize the agent behaviour, it hasn't been necessary to implement any complex graphical property.

As for the last task, I have thought to implement a problem related to astrophysics, that is, given a set of n celestial objects, find the point such that the total gravitational attraction is zero, as explained in more detail later in this report.

1 Task 1: Target finder

1.1 Problem description

In this task, I have developed the PSO algorithm as shown in the presented paper. In particular, I have applied the following formula:

$$v(k+1) = \omega \cdot v(k) + c_1 \cdot r_1 \cdot [p_{lb} - p(k)] + c_2 \cdot r_2 \cdot [p_{gb} - p(k)]$$

In this formula, $v(k)$ is the vector of the velocities, that could be a 2-vector or a 3-vector. My implementation would allow to have, in general, an m-vector, with $m \in N^+$, since PSO could also be used with more than 3 parameters. The agents, however, are represented only if there are two or three parameters, since the euclidean space is tridimensional.

$p(k)$ is also a vector and it is used to represent the current position. As for the velocity, the position could be defined as a 3-vector, for instance, but my implementation would allow an m-vector.

An agent also remembers its best position (considering its whole history), that is denoted by p_{lb} and the best global position, p_{gb}, where global refers to a cluster of agents, and not necessarily to all the agents involved. In particular, I consider the k closest agents, where $k \leq n$, if n is the total number of agents involved. This means that a certain agent may belong to more than one cluster at a given moment.

Moreover, there are the parameters c_1 and c_2, that are the weights associated with the local best and the global best respectively. Higher value are directly related to an higher amount of that kind of exploration.

The parameters r_1 and r_2, instead, are random numbers in the range between 0 and 1, multiplied by the weights to generate a less deterministic evolution, that could bring to a better spatial exploration.

Finally, ω is a parameter to determine the contribution of the velocity at the previous timestep. This parameter could be determined in several ways. The proposed approach consists in a weight that changes linearly:

$$\omega(k) = \omega_0 - \frac{\omega_0 - \omega_f}{n_i} k$$

where ω_0 is the initial weight, ω_f is the final weight, n_i is the total number of iterations of the algorithm and k is the current iteration index.

For instance, if $\omega_0 > \omega_f$, then the weight will decrease linearly through time. The practical effect is that in this way the exploration will be reduced as the number of iterations increases. This makes sense in some applications, since it is expected that the swarm will converge toward a solution after some time.

1.2 Parameters variation

I have tried to observe the changes in the results on the basis of the following parameters:

- Number of agents
- Number of iterations
- Initial weight (w0)
- Final weight (wf)
- Local search factor
- Global search factor
- Number of agents in a cluster

For each case, I present the evolution in terms of the distance from the target and the final configurations of the agents. The distance shown in the graph corresponds to the one of the best agent. This means that some other agents may have not converged yet to the target point, since in some cases they have formed separate clusters, depending on the parameters.

The target is always shown in red, while the agents are always shown in white.

1.2.1 Case 1

- Number of agents: 100
- Number of iterations: 100
- Initial weight (w0): 0.9
- Final weight (wf): 0.2
- Local search factor: 1
- Global search factor: 1
- Number of agents in a cluster: 20

In this case, it is important to observe that the weight decreases to a low value (0.2), therefore at the end of the evolution the particles that had gathered in clusters remain there without exploring the space extensively. In the next case I will try to change the final weight to see if there will be any change.

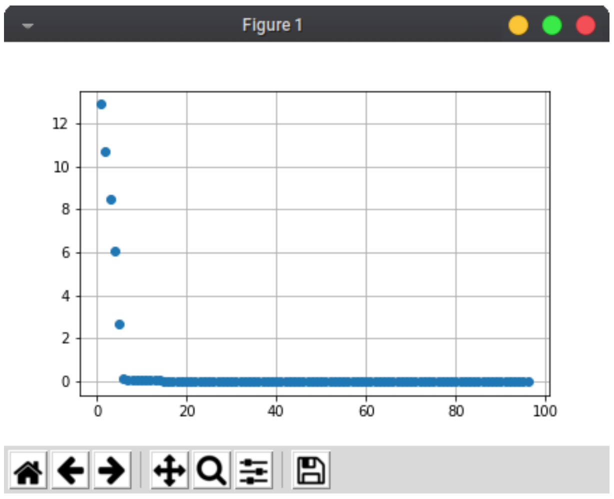

Figure 1: Case 1 - distance from the target

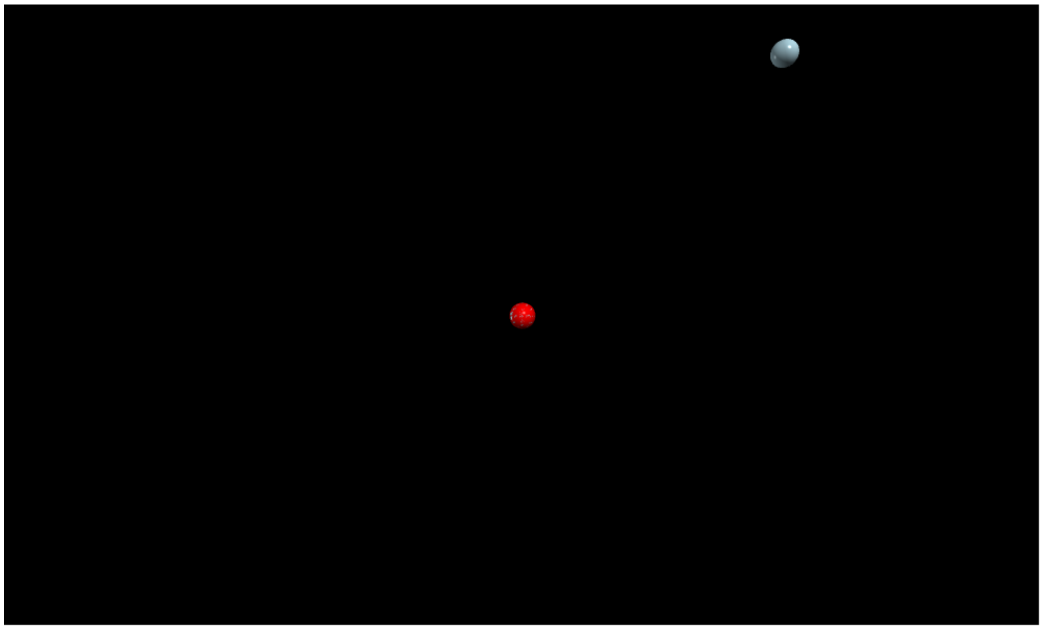

Figure 2: Case 1 - situation at the end of the evolution: most of the particles have reached the target and overlap with it

1.2.2 Case 2

- Number of agents: 100
- Number of iterations: 100
- Initial weight (w0): 0.9
- Final weight (wf): 0.9
- Local search factor: 1
- Global search factor: 1
- Number of agents in a cluster: 20

In this case, that is almost equal to the previous one, the value for ω remains the same, since $w_0 = w_f = 0.9$. This allows to find the target as fast as in the previous case, but since ω remains high the agents keep exploring until the end, as shown. However, the particles tend to go to the center, where the target is, anyway.

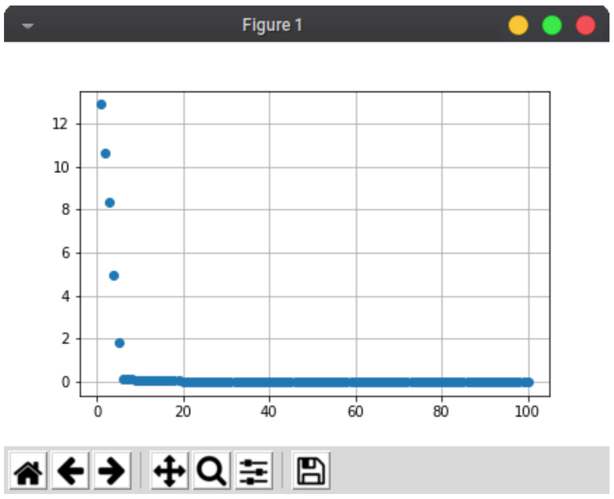

Figure 3: Case 2 - distance from the target

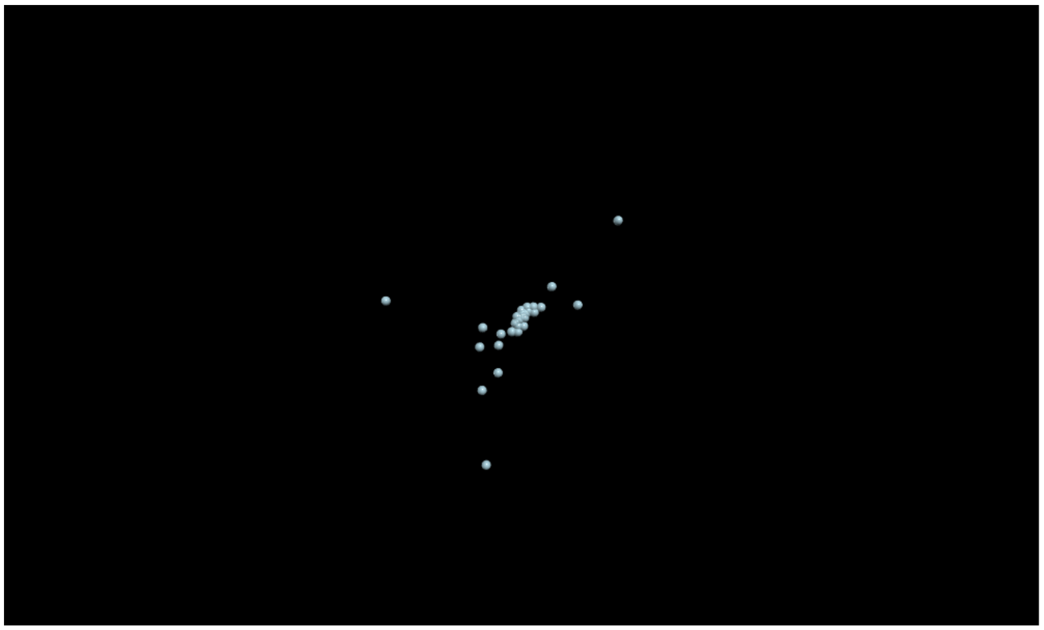

Figure 4: Case 2 - situation at the end of the evolution: the particles keep exploring the space although they are close to the target

1.2.3 Case 3

- Number of agents: 100
- Number of iterations: 100
- Initial weight (w0): 0.2
- Final weight (wf): 0.2
- Local search factor: 1
- Global search factor: 1
- Number of agents in a cluster: 20

In this case, that is almost equal to the previous one, the value for ω remains the same, since $w_0 = w_f = 0.2$. This low value is the reason why the agents move very slowly, and the distance value converges toward 13.4, differently from the previous simulations.

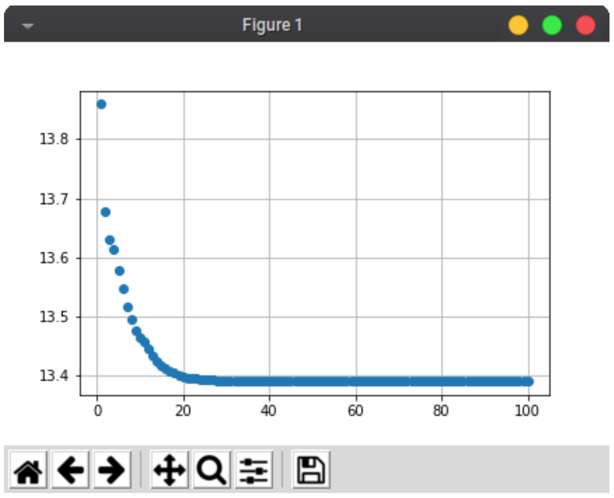

Figure 5: Case 3 - distance from the target

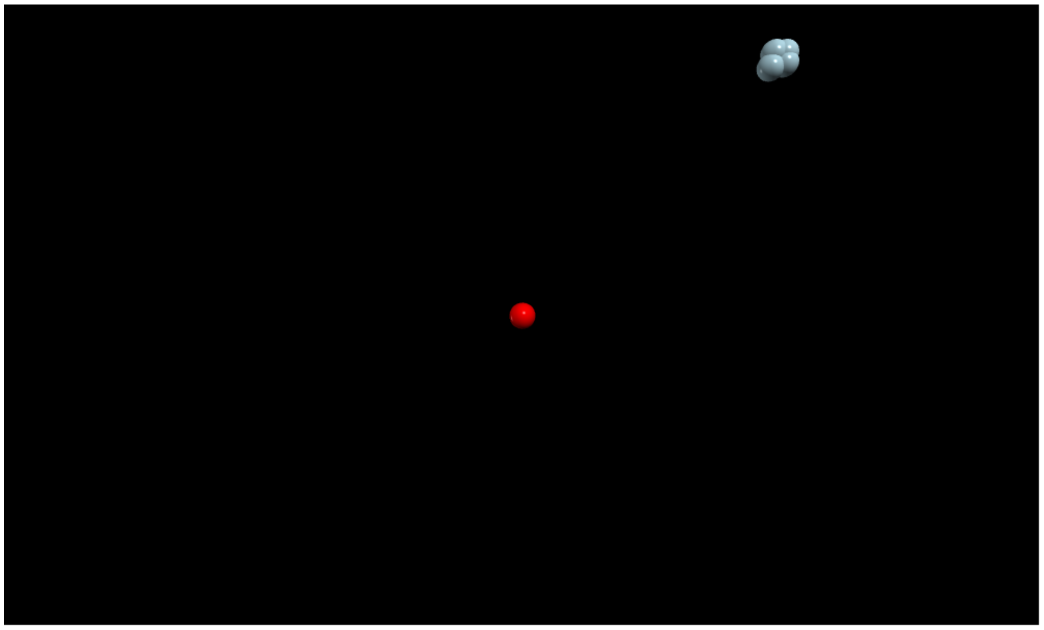

Figure 6: Case 3 - situation at the end of the evolution: no particle has reached the target

1.2.4 Case 4

- Number of agents: 10
- Number of iterations: 100
- Initial weight (w0): 0.9
- Final weight (wf): 0.5
- Local search factor: 1
- Global search factor: 1
- Number of agents in a cluster: 5

The settings here are more balanced in terms of weights, but there are only 10 agents. The agents converge to the target as well, as shown below. The improvement with respect to the other cases is that, since there are less agents, the simulation time is much lower than before, even though it takes more iterations to reach a solution.

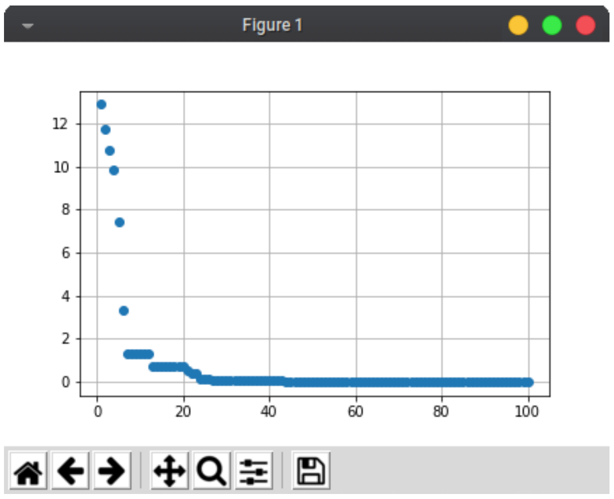

Figure 7: Case 4 - distance from the target

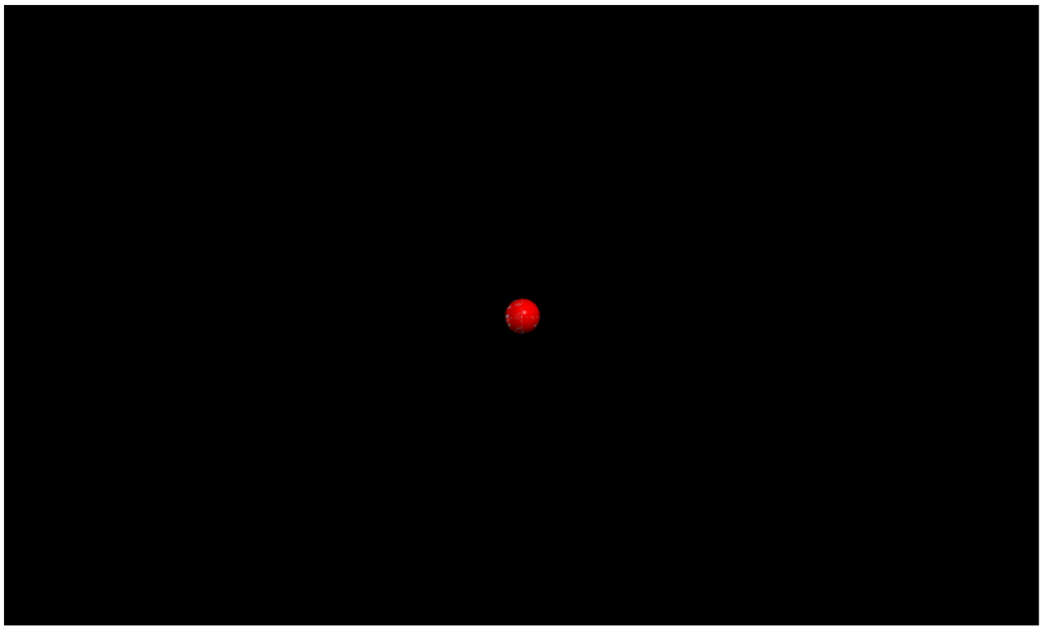

Figure 8: Case 4 - all the particles have reached the target and, since none of them is visible, the precision is quite high

1.2.5 Case 5

- Number of agents: 100
- Number of iterations: 100
- Initial weight (w0): 0.9
- Final weight (wf): 0.5
- Local search factor: 1
- Global search factor: 2
- Number of agents in a cluster: 50

In this case I have incremented the global search factor. During the simulation, it is possible to observe that the agents explore more space if compared to the previous cases, and they arrive close to the target in a short time. However, the final position is less precise than in other cases (this can be seen graphically, since the target is "covered" by the agents.

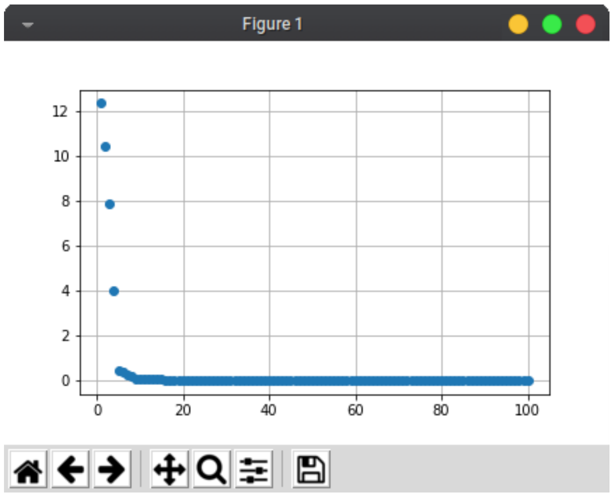

Figure 9: Case 5 - distance from the target

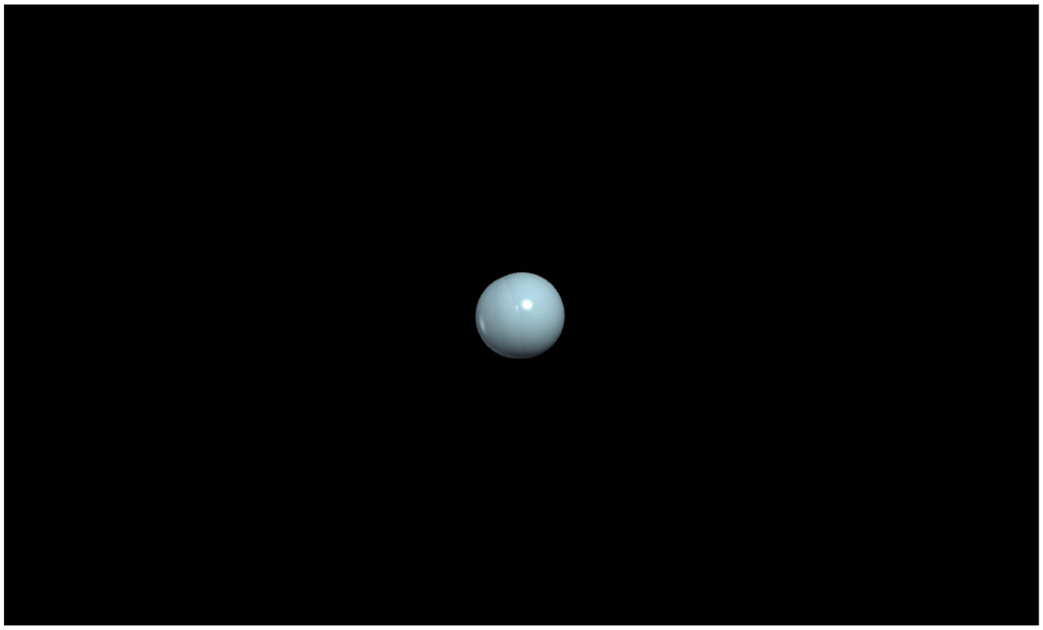

Figure 10: Case 5 - all the particles have reached the target, but less precisely than in other cases

1.2.6 Case 6

- Number of agents: 100
- Number of iterations: 100
- Initial weight (w0): 0.9
- Final weight (wf): 0.5
- Local search factor: 2
- Global search factor: 1
- Number of agents in a cluster: 50

In this case I have incremented the local search factor.I haven't observed relevant changes, with the exception that the agents explore more space than in other solutions, and tend to form some clusters further away from the target. The target is reached anyway, and faster than in the previous case.

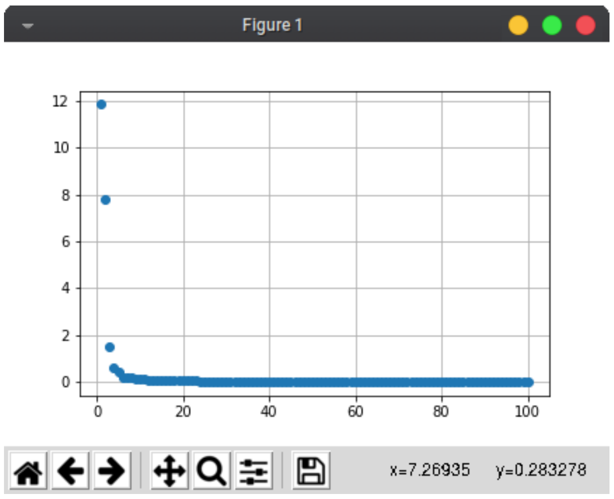

Figure 11: Case 6 - distance from the target

Figure 12: Case 6 - all the particles have reached the target, but less precisely than in other cases

2 Task 2: Path planning

The problem to be solved consists in finding the best path from an initial position to a target. The difference with respect to the previous problem consists in the fact that in this case the goal does not consist in understanding where is the target, but rather in finding how to arrive there. Initially, the problem has been solved for a simple landscape without obstacle, then some obstacles have been added.

This solution would work in three dimensions, but for the sake of simplicity I have created an environment that evolves on only one plane. In order to have a three-dimensional scene, it would be necessary to define the position and the velocity of an agent as a 3-vector in the agent generation phase. I have implemented this algorithm using different approaches, that are shown in different files.

The first approach consists in ignoring any obstacles and in going to the target. This means that the expected shortest path would be just a straight line, in two dimensions. Using PSO, the approach consists, in practice, in a greedy search:

- All the agents start from the same position and swarm for one iteration
- Only the best agent is kept
- Another swarm of agents, in which the number of agents is the same as the initial swarm size, is generated from this new position

This approach does not guarantee an optimal path, since there is always the possibility that no agent is close to the straight line path. However, an increment in the number of agents makes this possibility unlikely and negligible.

This approach doesn't work in case there are obstacles. I have implemented static obstacles as cubes that have the same z-coordinate ($z = 0$) in order to keep the problem in two dimensions. Those cubes, therefore, can be seen as squares for this particular case.

The idea behind cubes was to replicate the concept of pixel in three dimensions (creating, in practice, a sort of voxel), since sets of very small cubes could resemble a curved figure, such as a sphere. Another possibility would have been to create triangles and to design obstacles by putting them together, but the code would have become much less readable and probably too much focused on topics not strictly related to swarm intelligence.

It is important, then, to see if there is an overlap between a sphere (representing an agent) and a cube (representing an obstacle). It is possible to see if such a collision happens by considering the centers of the cube and of the sphere and their side and radius. Moreover, it is also important to see if the sphere intersects the cube during the path, and not only in correspondence of the starting and arrival point.

The cube-segment intersection could be calculated precisely or approximately. I have used a library (sympy) to calculate if a path and a cube intersect, but this implementation slows down the simulation. For this reason, I have also proposed an approximated version, that is much faster, that allows to calculate if there is an overlap for some of the points on the line describing the path.

This algorithm is in general quite similar to the one implemented in the previous task. A new feature consists in setting the maximum velocity as a parameter.

Moreover, I have considered two variants of the problem. The first variant consists in the generation of the swarm, followed by the movement the agents according to the parameters, the selection of the best agent and the generation of a new swarm from there. This approach, however, tends to get stuck right before the obstacles, without going around them. The second approach, therefore, consists in iterating several times before re-generating the swarm. In practice, the second variant consists of a nested particle swarm optimization. None of those approaches, however, can guarantee an optimal solution, since a greedy component remains when the new agents are re-generated.

2.1 Without intermediate paths

The figures presented in his section have been generated using the following parameters:

- Number of agents: 100
- Number of iterations: 100
- Initial weight (w0): 0.9

- Final weight (wf): 0.2
- Local search factor: 1.5
- Global search factor: 1
- Number of agents in a cluster: 25
- Maximum velocity: 5

The graph that reports the positions reports only the positions obtained in the main loop (that is, every time a new swarm is generated), while all the intermediate positions are shown, in yellow, in the visualization.

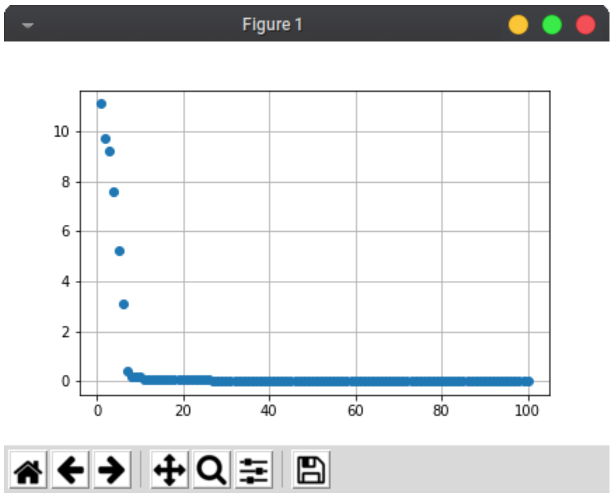

Figure 13: Task 2 with obstacles and no intermediate paths: the target has been reached

2.2 With intermediate paths

The figures presented in his section have been generated using the following parameters:

- Number of agents: 100
- Number of iterations: 100

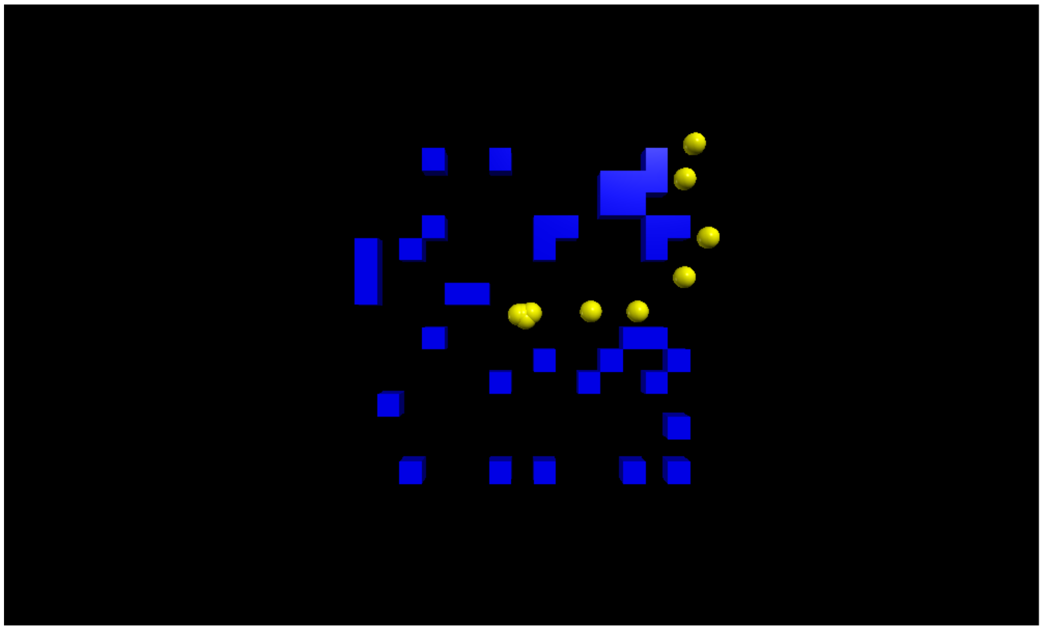

Figure 14: Task 2 with obstacles and no intermediate paths: the yellow spheres represent the path found so far and the blue cubes are the obstacles. The target is hidden behind the yellow spheres in the center.

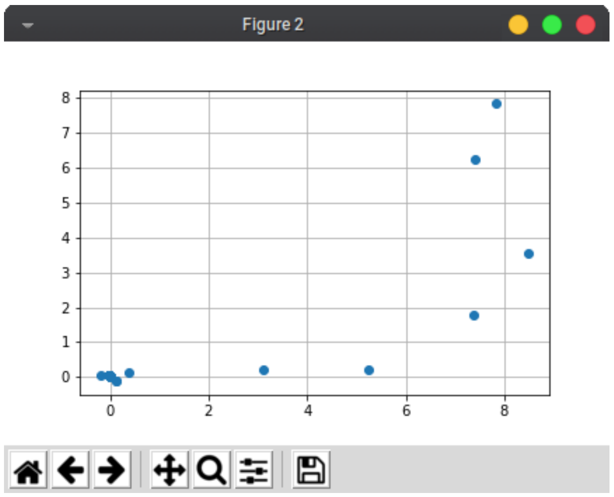

Figure 15: Task 2 with obstacles and no intermediate paths: the path containing only the position in which the swarm is re-generated

- Initial weight (w0): 0.9
- Final weight (wf): 0.2
- Local search factor: 1.5
- Global search factor: 1
- Number of agents in a cluster: 25
- Maximum velocity: 5
- Number of steps for sub-path: 10

The graph that reports the positions reports only the positions obtained in the main loop (that is, every time a new swarm is generated), while all the intermediate positions are shown, in yellow, in the visualization. The semi-transparent spheres shown during the simulation represent the rest of the swarm at a given moment. *Note: I show only the first part of the evolution, since the target has been reached before $k = 100$.*

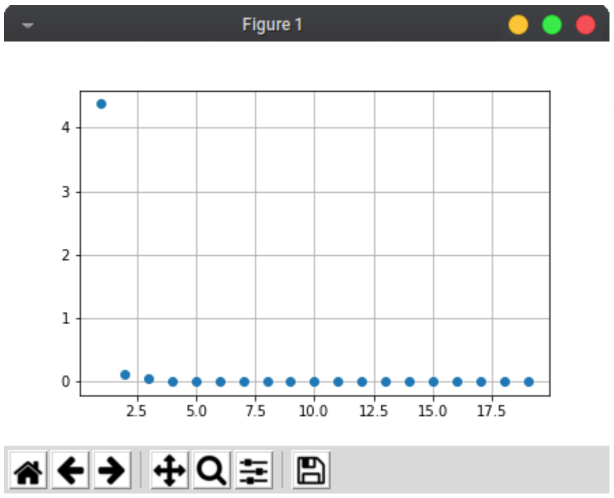

Figure 16: Task 2 with obstacles and intermediate paths: the target has been reached

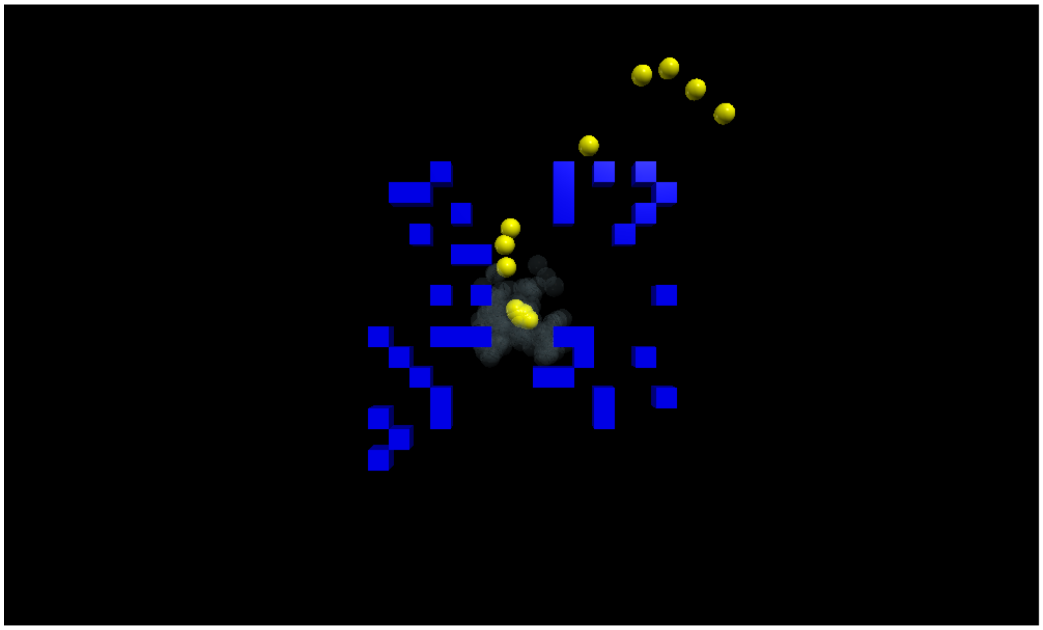

Figure 17: Task 2 with obstacles and intermediate paths: the yellow spheres represent the path found so far and the blue cubes are the obstacles. The target is hidden behind the yellow spheres in the center.

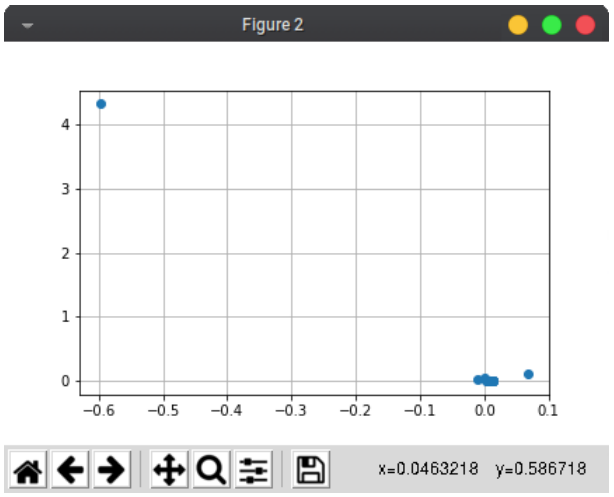

Figure 18: Task 2 with obstacles and intermediate paths: the path containing only the position in which the swarm is re-generated

2.3 Further improvements for a dynamic environment

In a dynamic environment, finding an optimal solution is more complicated. Some possibilities, in my opinion, could be:

- If the motion of the obstacle is known a priori, then the problem can be solved by considering the time as a relevant dimension for optimization. This would mean that, for instance, it would be possible to define a fixed interval for the agent movement, so that it is possible to determine exactly in which position the obstacle will be at a given iteration in the future
- If the movement of the obstacle is not known, it could be possible to use machine learning and pattern recognition to try to obtain it after an observation phase. After the observation, the agent could behave as in the previous case.
- It could be possible to create a swarm that evolves and such that when an agent is hit, it does not survive and the next best agent takes its place.
- Another possibility could be reinforcement learning: some agents are sent to the dynamic obstacle and they learn that avoiding it is better, since they could receive a reward.
- The agent proceeds as usual and when it sees something that moves, it waits until it has cleared the way and then goes. If after a while the way has not been cleared, the agent could change its position to look for another passage.

3 Task 3: Finding the point with a gravity of zero

It is possible to consider, for instance, a system formed by a star and a planet, such as the Sun-Earth system. The Sun and the Earth exert a force on each other, according to Newton's law of universal gravitation. The same happens when charges are considered: an electron and a proton, for instance, are attracted according to Couloumb law.

Differently from electric forces, gravity has only been observed to be attractive, and never repulsive. This means that, given two bodies, there is only one point in which the total gravitational force is zero. Moreover, this point has to lie inside the parallelepiped[1] in which all the bodies are contained.

At an infinite distance from the masses the gravitational force also tends to zero. For the sake of simplicity, I am assuming that the gravitational force travels at an infinite speed, which is not true, since it travels at most at the light speed, as a consequence of the existence of gravitational waves. However, for practical purposes, the most interesting point is the one that lies inside the parallelepiped defined above.

In a two-body system on which no force but gravitational force is exerted, this point is one of the Lagrangian points, and specifically L_1. Lagrangian points are used, for instance, to determine the dynamic of artificial satellites. In particular, L_1 could be as an observation point for the Eart-Sun system.

My solution doesn't find exactly L_1, since it doesn't take into account the centripetal forces. The reason why I haven't inserted any centripetal force is that it would have been more difficult to visualize and more expensive from a computational point of view, since the centripetal forces had to be generated by performing an n-body simulation, that, unless some approximations are introduced, has a quadratic complexity.

My solution, instead, assumes that there are only gravitational forces and that the purpose is to find the point in which the total gravitational force is equal to zero for a system with n bodies.

The proposed problem, therefore, has been designed as a static problem, since it is possible to run it for several subsequent positions of the bodies still obtaining an optimal solution. It is also important to define the constraint that the point has to be contained into a region delimited by the bodies under consideration. It is more expensive, from a computational point of view,

[1]Or, more precisely, a polyhedron whose vertices are the bodies' centers

to calculate the exact polyhedron that passes though all the centers of the bodies, and this is the reason why I have decided to propose a parallelepiped. Its purpose is to avoid that the swarm goes in the outer space, since it would keep going there, since the total force decreases even when going away from the bodies.

In order to compare the two versions (with and without the boundary control), both the codes are attached. In the solution that checks whether an agent is inside the boundaries, if an agent tries to escape, it is placed in a random position inside the boundaries (and in this situation having a parallelepiped simplifies the code and the computational complexity).

3.1 Gravitational force definition

The gravitational force between two bodies is defined as:

$$F_{12} = G\frac{m_1 m_2}{|r_1 - r_2|^3}(r_1 - r_2)$$

where r_1 and r_2 are two vectors and the notation $|v|$ indicates the magnitude of a vector v.

Consequently, given n bodies, it is possible to define the acceleration perceived in a point r_0 as:

$$a_0 = \sum_{i=1}^{n} G\frac{m_i}{|r_i - r_0|^3}(r_i - r_0)$$

Since the purpose is to bring the total force to zero, this means bringing a_0 to zero and for this reason it is possible to neglect G, that is constant. The quantity that should be minimized, therefore is:

$$\sum_{i=1}^{n} \frac{m_i}{|r_i - r_0|^3}(r_i - r_0)$$

I have formulated this problem as a maximization problem (the sign is the only part that changes with respect to a minimization problem), therefore the fitness graph starts with negative values and tends to zero.

3.2 Simulation for a 4-body system

I have used the following parameters for a 4-bodies system and the bodies have been placed randomly:

- Number of agents: 20
- Number of iterations: 150
- Initial weight (w0): 0.9
- Final weight (wf): 0.3
- Local search factor: 1
- Global search factor: 1
- Number of agents in a cluster: 20
- Number of celestial bodies: 4

In this case the best position has coordinates $(-5.91, -0.47, -8.40)m$ and the total gravitational acceleration in that position is $6.50 \cdot 10^{-26} ms^{-2}$. This acceleration is equal to $6.62 \cdot 10^{-25}\%$ of the average gravitational acceleration on the Earth.

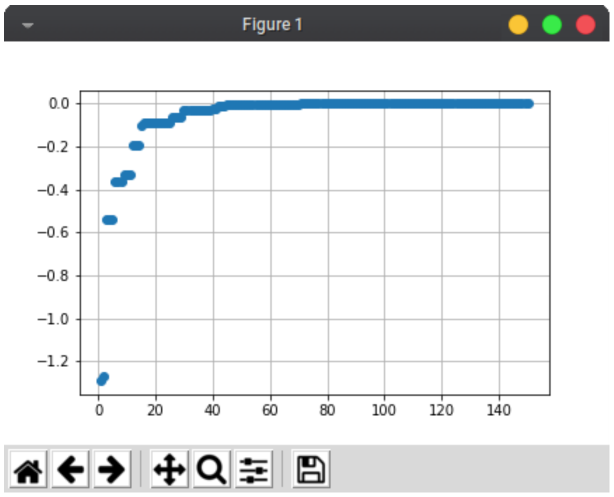

Figure 19: The solution improves and it tends to a total resulting force of zero

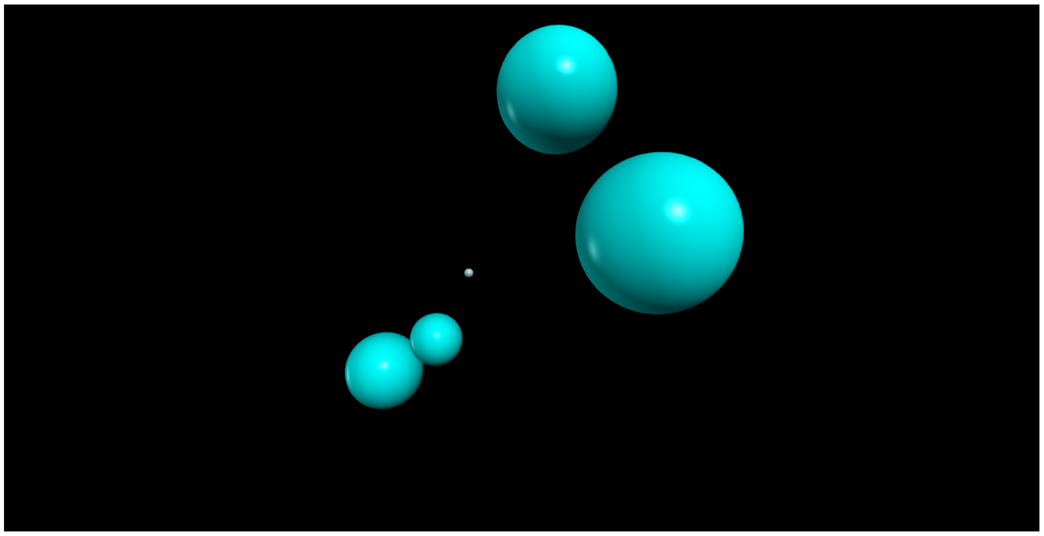

Figure 20: The blue bodies are celestial bodies, while the white sphere is the point with a resulting gravitational force of zero