Shape Reconstruction using Particle Swarm Optimization

https://github.com/raustin9/ParticleSwarmVis

Jacob Hawkins
University of Tennessee
Dept. of Electrical Eng and Computer Science
Knoxville, Tennessee
jhawki41@vols.utk.edu

Lakelon Bailey
University of Tennessee
Dept. of Electrical Eng and Computer Science
Knoxville, Tennessee
lbaile31@vols.utk.edu

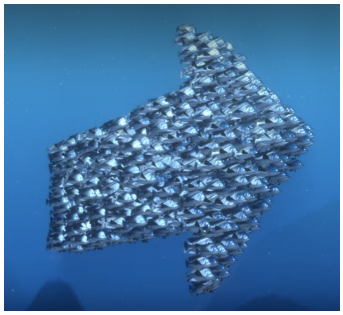
Reagan Austin
University of Tennessee
Dept. of Electrical Eng and Computer Science
Knoxville, Tennessee
rautin9@vols.utk.edu

***Abstract*—Particle Swarm Optimization (PSO) is a biologically-inspired computation technique used to optimize a problem by iteratively trying to improve a candidate solution in relation to an established measure of quality. This technique is based off of swarming patterns found phenomena such as schools of fish or bird murmuration. In this project, we sought to answer the question of whether or not PSO can be effectively used to generate 2-dimensional shapes from particles. This involved implementation of baseline particle movement rules, such as velocity, turning radius, inertia, and collision. Then, PSO behavior was built on top of this baseline in order to simulate natural particle movement and swarming. We created a system in which the particles track a personal best scent and global best scent and applied these values to the standard PSO equation, including a social weight, cognition weight, and an inertia weight. Experimental data was produced by varying each relevant parameter in isolation and then measuring the average distance to the shape across all particles upon convergence. Our results indicated that a higher ratio of the cognition weight the social weight leads to a better overall solution with a lower average distance to shape. This aligns well with the nature of PSO, as a higher social relative social weight would normally cause the particles to pay more attention to one another rather than the shape, leading to a higher average distance to shape. These results answered our research question by establishing that complex 2-dimensional shapes can be created using PSO when a higher emphasis is placed on the cognition weight.**

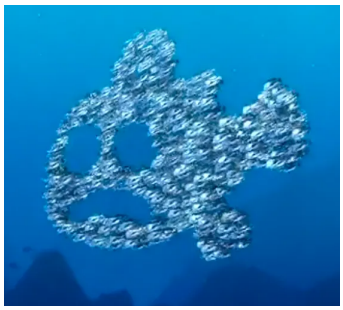
I. Introduction and Motivation

In the movie *Finding Nemo*, there is a scene in which the protagonists encounter a school of fish that communicate with the protagonists by organizing themselves into complex, 2-dimensional shapes such as an arrow, a fish, and a boat (re 1). While there are obvious cartoon elements of this scene, our team was interested to test its validity by formulating it into a Particle Swarm Optimization (PSO) problem. Our goal was to create a 2-dimensional particle simulation where particles move around and target locations based on PSO-based agent communication.

Our interest in simulating this particular scene arised from general fascination with swarming patterns found in Biology. As a group, we have always been fascinated with behaviors such as bird murmuration and migration patters, insect swarm-



(a) Arrow Formation



(b) Fish Formation

(c) Boat Formation

Fig. 1: Fish Formations in *Finding Nemo*

ing, schools of fish, and more. Because of this, using PSO to test the validity of a fictional scene is a fascinating project that allows us to both explore our interest in the subject as well as develop a PSO solution that could be used for a variety of applications.

II. Related Work

Since the original Particle Swarm Optimization algorithm was introduced in 1995, there have been a number of variations, both substantial and minimal, used to perform research and put to use. This is natural and through this, the performance of Particle Swarm Optimizations have improved. However, this also introduces confusion and complexity into the field, and [1] introduces a standard that future implementations can be based around and compared against.

A common issue present in Particle Swarm Optimizations is the drop in performance as the number of iterations that have to occur to search the entirety of the space increase dramatically as the size of the space and number of particles increases. [2] reasons to solve this effectively by partitioning the space and introducing some concurrency. More importantly, they use "cooperative learning" using a shared "blackboard" that the particles can all see and learn from, so hints can be shared amongst all particles allowing them to converge more easily.

SwarmViz [3] is a visualization tool that allows the user to see the swarm's progress. This allows researchers to be able to monitor the swarm to make observations about their parameter choices, but it also serves as a learning tool for students to see the impact of parameter choice to help teach different optimization techniques. SwarmViz includes multiple forms of visualization, and it is a good tool for graphing and charting the outputs. This is similar to the purpose of this paper, as this paper intends to create a form of visualization for specific particle swarms.

[4] uses a particle swarm for shape-fitting in the field of Computer Vision. Shape-fitting is a very important problem in Computer Vision for tasks like object recognition and compression. They use a particle swarm to construct a fitting to these images and evaluated different parameter choices to find the best fit. They find some success, and it is an effective method of creating vertices to form a shape. As the purpose of their method is to read shapes rather than construct them, their particle swarm does not converge in a manner that visualizes well to shapes, but it rather converges in a manner that is better suited for image-recognition.

As particle swarms are suited to finding optimal regions of a space, it is a great application for finding multiple solutions within a single problem. This is explored in [5] where the authors apply a Particle Swarm Optimization to find multiple solutions within a multi-objective problem. For this research, they limit the scope of the solution's dimensions to only 2, so their findings are not too conclusive to the effectiveness of higher dimensional problem spaces, but they do find Particle Swarm Optimization to be effective for the cases they tested on.

[6] uses a Particle Swarm Optimization to use as an algorithm for stock and investment portfolio optimizations. Their paper measures the effectiveness of a particle swarm against genetic algorithms, Tabu Search, and stimulated annealing which are other heuristics for portfolio selection. They find that none of the 4 heuristics outperform any of the others consistently or significantly, but they do find that Particle Swarm Optimization outperforms the others measurably when it comes to low-risk investments.

The typical conventional wisdom of a Particle Swarm Optimization is to converge on optimal locations within the search space by keeping track of good regions for a particle, their neighbors', or the swarm globally, and using that knowledge to further find better and better regions. In [7] they propose an algorithm that counters that idea by keeping track of the local and global worst locations that have been found, and using that information to avoid poor regions in order to find better ones. The researchers found that this form of PSO can outperform the traditional model in some cases, but the traditional model outperforms the new model in other cases. They propose this new model (NPSO) as an alternative new variation of a PSO rather than a better one.

Particle Swarm Optimizations often have the issue of improper convergences on locally good regions and getting stuck there, and this behavior can be seen in this paper as well. The researchers of [9] explore how genetic algorithms in combination with the behavioral update function can cause particles to break out of undesirable convergences on local optima by mutating over time. They test this technique in combination of multiple functions, and they find that the mutation hinders the performance when using the spherical function, but actually does significantly improve the performance for harder functions for all dimensions of 10, 20, and 30.

Chaos is a technique used in certain forms of optimization, most notably Chaotic Neural Networks, and it is also used in the Chaotic Optimization Algorithm (COA) that is based on the chaotic evolution of variables. Chaos is defined as a characteristic of non-linear systems, and it is behavior that is highly sensitive to initial conditions, yet it is still deterministic within the system. The researchers of [10] find that introducing Chaos into the searching behavior of particles in a PSO, the searching quality and efficiency are greatly improved, and at the same level of searching quality of other implementations, the efficiency and speed of the chaotic PSO is much better and quicker than the others.

Many implementations of PSO constrain particle behavior to be homogeneous, or universally similar across all particles. [8] introduces an implementation that uses heterogeneous behavior for particles that can be selected from a table of potential behavior rules. They implement two versions: a static model where the behaviors are heterogeneous yet unchanging over the run of the model, and a dynamic one where the behaviors do update during runtime. Both implementations initialize behaviors randomly for each particle, and in the dynamic model when a new behavior is selected, it is done randomly from the pool. This pool includes 5 different behaviors, but

this is extensible. They find that the dynamic model drastically outperforms the static model and the homogeneous model, and this is with a small behavior pool and random selection function, and they intend to explore further improvements to this model.

III. Method/Approach/Experimental Setup

A. Environment Setup

For this experiment, we developed our environment using a simple Go server as a foundation. This server serves the static HTML, CSS, and JavaScript files we used to build our simulation. Along with this, the server is set up as an API to receive experimental data from the JavaScript simulation to append to a CSV file. This simple environment allows the simulation to be easily rendered in any browser while also being compatible with the generation of simple experimental data. We did not utilize any additional libraries on top of these languages. We used the JavaScript Canvas API to draw the particle simulation directly to the screen and used simple DOM manipulation for any forms or styling necessary.

B. Configuration

In order to allow for easy manipulation of parameters, we developed the simulation to pull all of its relevant parameters from a configuration object with attributes for each parameter. The default values for this configuration were stored in a separate JavaScript file. Because these values are loaded into a mutable object, it was easy to manipulate different configuration parameters by simply updating the configuration object.

C. Generating Shapes

In order to avoid needing to determine the closest point on a shape, we generated all target shapes using the same Particle class that is used for the main particles. However, the shape particles were given a velocity of 0 and were non-physical, meaning the main particles could not collide with them and simply passed through. This shape generation method allowed us to track the closest point on the shape within the regular nested loop that checks the proximity between each particle and the other particles. Each particle was given a boolean variable to indicate whether or not it was a shape particle so that it could be considered appropriately in the main particle behavior.

D. Particle Behavior

The particles move across the screen in the simulation simply by adding the value of their velocity vector to their position in each frame. However, the velocity vector was manipulated based on the basic movement rules and PSO rules in each frame before being applied to the position.

1) Basic Movement: Before implementing the particle swarming behavior, we implemented a simple simulation where particles fly around at constant speeds and directions, bouncing off one another and the viewport bounds. For each particle, we assigned a random velocity vector at initialization. To handle collisions between particles, we used the 2D vector rotation equation below to rotate each particle's velocity by the angle between their velocity vectors:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

To handle collision with the viewport bounds, we simply multiplied the corresponding axis velocity by -1 to reverse the direction. For example, if a particle contacted the top part of the viewport, we multiplied its y-axis velocity by -1 to reverse its vertical direction but maintain its horizontal direction.

2) PSO Movement: To implement PSO into the particle behavior, we used the following PSO equations to update each particle's velocity and position in each frame:

$$v_{id}^{(t+1)} = w \cdot v_{id}^{(t)} + c_1 \cdot r_1 \cdot (p_{id} - x_{id}^{(t)}) + c_2 \cdot r_2 \cdot (p_{gd} - x_{id}^{(t)})$$
$$x_{id}^{(t+1)} = x_{id}^{(t)} + v_{id}^{(t+1)}$$

where:

- $v_{id}^{(t)}$ and $v_{id}^{(t+1)}$ are the velocity of particle i in dimension d at iteration t and $t+1$, respectively.
- $x_{id}^{(t)}$ and $x_{id}^{(t+1)}$ are the position of particle i in dimension d at iteration t and $t+1$, respectively.
- w is the inertia weight, which controls the influence of the previous velocity of the particle on its current velocity.
- c_1 and c_2 are the cognitive and social scaling coefficients, respectively.
- r_1 and r_2 are random numbers typically drawn from a uniform distribution between 0 and 1, used to maintain the stochastic nature of the algorithm.
- p_{id} is the best known position of particle i in dimension d (personal best).
- p_{gd} is the best known position among all particles in the swarm in dimension d (global best).

The personal best and global best positions were tracked based on nearby particles and shape particles. Each particle's personal best position was calculated in each frame by finding the position of the closest shape particle within the particle's shape detection range. If no shape particles were within range or the closest shape particle is farther away than the particles current personal best, the particle's personal best was not updated. Otherwise, the new personal best was set on the particle. Each time a new personal best position was set on a particle, a new personal best score is set as well, which is the value of the distance between that particle and the shape.

The global best position was calculated for each particle by finding the lowest personal best score of all other particles within the particle's social range. If this score was lower than the particle's existing global best score, the particle's global best position would be set to this positon and its global best score updated as well.

E. Experimental Setup

1) Test Cases: For each weight type (inertia, social, and cognition), we established a default value and a range of possible values from the below table:

TABLE I: Parameter Values

Weight	Default Value	Value Range (start, stop, step)
Inertia	0.5	0.1, 1, 0.1
Social	1	0.1, 4, 0.1
Cognition	1	0.1, 4, 0.1

Then, for each weight, we varied the weight across its value range while keeping the other weights at their default. For each combination of these weights, we added 10 test cases. Adding 10 of each combination allows us to compensate for the randomness associated with the PSO equation and still observe consistent trends. In total, this method produced 900 test cases for our experiment.

2) Experiment Mode: In order to enable experimentation, we created a mode of our simulation called **Experiment Mode**. This mode will run the simulation in a "headless" environment, meaning it does not render any of the graphics and instead simply runs the JavaScript algorithm in the background. This mode runs the simulation on all of the generated test cases until they converge, sending the result of each experiment to the Go server to be added to a CSV file. Particles were considered "converged" when at least 99% of the particles had remained within the same 30 pixel radius for their previous 10 positions. These benchmarks were established based on qualitative analysis to reduce convergence timeouts and produce reliable data points quickly.

F. Additional Parameters

Along with the standard parameters used for PSO, we considered several other parameters to experiment with their effect on the particle behavior.

- **Scent Range:** The scent range parameter represents the radius in which particles are able to detect shape particles. We defaulted this parameter to 100 pixels.
- **Social Range:** The scent range parameter represents the radius in which regular particles are able to detect other regular particles. We defaulted this parameter to 300 pixels.
- **Social Scent Increase Factor:** This parameter was a scalar value that we multiplied by social scents when particles communicated their social scents to one another. This was meant to dilute the effect of a social scent across a chain of particle interactions. However, we found that this factor did not lead to an improved simulation and was redundant in light of the social weight parameter, so we decided to default it to 1 to remove its impact.
- **Number of particles:** We tried many different quantities of particles and used qualitative analysis to determine the best particle number to use for experimentation. We defaulted this value to 1,500 particles.
- **Collision Energy Loss Factor:** We experimented with the possibility of implementing energy loss on collision. However, this led to many particles getting stuck, so we decided to default this parameter to 1 to prevent any energy loss.
- **Max Turning Radius:** In order to produce more natural movement, we experimented with implementing a max turning radius. However, this led to much longer convergence times and less reliable results, so we decided to negate this parameter.
- **Max Particle Speed:** We defaulted the particle speed to a maximum of 5 pixels per second. This allowed for a smooth visualization while keeping the particles from becoming so fast that the simulation is indiscernable.

IV. Results

Upon completion of the experiments, a few key results were found. Firstly, the social and cognition parameters have a much more significant impact on the quality of the resulting image than any other hyperparameter. As seen in figure 2, the social and cognition parameters have an almost inverse relationship to the average distance from the target shape. As the social value increases, the average distance also increases. On the other hand, as the cognition value increases, the average distance decreases.

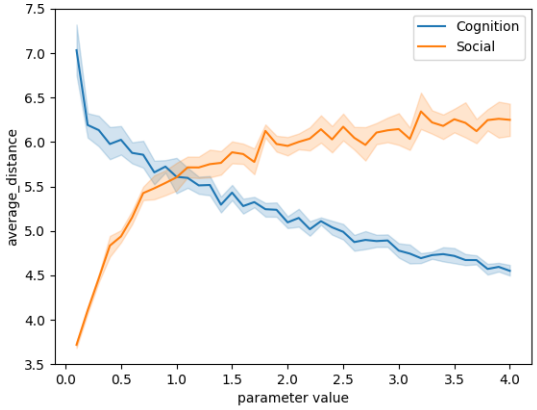
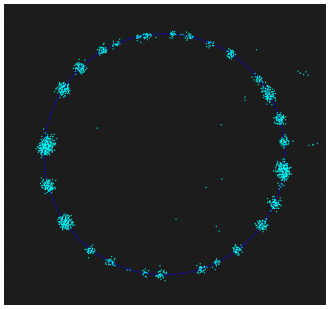
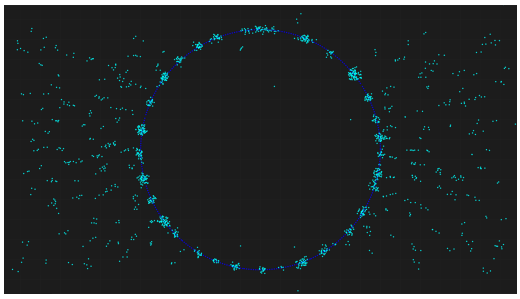


Fig. 2: Social and Cognition Relationship

This can clearly be seen in the visualization of the particle swarm. With higher social values, the particles are more incentivized to be close to each other causing the particles to cluster around the target shape (re 3a). Because the particles can not be on top of one another and will bounce off each other, this causes the average distance to be further from the shape. Another interesting result of having a high social parameter is the particles often form "lanes" to get to the target shape (re 3b). This can be further dramatized if the scent value is decreased. Doing so forces the agents to rely on one another to find the shape rather than simply smelling the shape themselves. Particles follow one another to the shape and create the clusters.



(a) Clustering



(b) Particle Lanes

Fig. 3: Effects of High Social Values

Secondly, changing parameters such as inertia and scent have little effect on the final shape convergence. After testing with a variety of inertia parameter values, it can be seen in figure 4 there is very little significant change in average distance between values. There is a slight upward trend as the inertia increases, but it is not enough to be able to draw conclusions from. Since the inertia parameter balances the algorithm's exploration and exploitation performance, we believe that this does not effect our model as, generally, each particle can always smell the target shape or another agent that can smell the shape. This results in the global best converging quickly without the need for much exploration.

In addition to the inertia parameter, the scent value also does not change how the final shape will converge. The biggest effect we saw from the scent parameter was the number of steps to converge. This makes sense as the scent parameter controls how far away the agent can smell the target shape. The lower this range is, the more the agent will have to rely on other particles around it to find the shape and follow them. The most significant result in testing the scent value came when both the scent and the social parameters were set to low values. This caused particles on the edges furthest from the target shape to wander aimlessly as they could not detect the goal. Otherwise, each particle could either smell the shape themselves or smell another particle that could smell the shape and would converge.

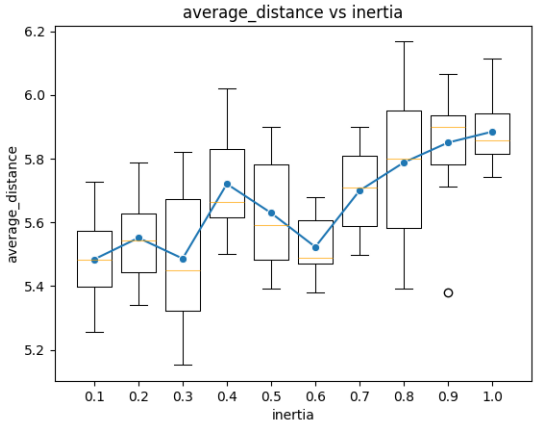


Fig. 4: Change in Inertia

Finally, we found that in order to create the most evenly filled shape with the lowest average distance from the targets, the social parameter has to be basically nonexistent and the scent parameter only has to be high enough for all the particles to smell the shape. When this is the case, the particles' only goal is to get to the closest part of the target shape and results in an evenly distributed shape. When the particles have a social parameter, they are attracted to one another causing the clustering as discussed previously (re 3a).

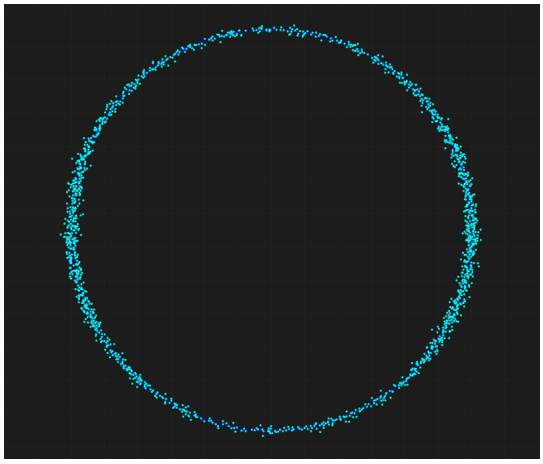


Fig. 5: Successfully Converged Circle

V. Conclusion and Future Work

After analyzing all of our results, we can conclude that we were able to successfully create shapes based purely on agent communication and incentivization. In creating a system that allows particles to communicated with one another, they were able to find and converge to a target shape. Upon testing with different parameters, we found that, to meet our goal, finding the correct balance of social and cognition values was crucial in determining the quality of the shape. We also found that scent range effects the time to convergence and inertia has very little effect in our simulation.

While we were able to achieve our goal, we have found that particle swarm optimization (PSO) is not the appropriate tool to create shapes in this manner. As mentioned in section IV, to achieve the distribution and even spreading of particles to create a well-defined shape, the social parameter has to be practically ignored. This takes a huge part of what makes PSO, PSO. Agent communication is a basic and fundamental part of how the algorithm works and deviating away from that begins to become a completely different concept. In addition to this, particle swarm optimization is typically used to find a single or very few solutions. In this application, we gave it hundreds of targets to optimize too. While this solution could be useful when trying to converge to local optimums where there are multiple solutions, PSO is not made to have this many targets.

Rather than using a particle swarm optimization, it might have been more beneficial and more true to the algorithm to use something like an ant colony behavior where the shape is food. In this algorithm, agents work more independently while still communicating where the goal is. A concept like this also would spread out easier as their social component is more based upon tracking the location of where the food source would be. Using such a technique may stay truer to the core of the concept we are using and would not have to mitigate a major part of the algorithm.

A. Future Work

In the future, we want to work to add parameters to help incentivize particles spread out without taking away the social parameter. This could be some sort of spacing parameter that penalizes particles if there are too many in a specific zone. We also could implement a ”food source” mechanic, as mentioned earlier, where particles are incentivized to reach food but each source can only support so many particles. Both of these solutions would help to spread that particles around the circle while still having a social influence.

Another thing we could work on in the future is adding further customizability options the user interface (UI). Currently, we have options to change the shape, size of the shape, and the number of particles. The back-end code already supports a lot of custom parameters through a configuration file. In the future, adding a UI component that allows the user to access these would help improve the user experience.

Finally, in the future we would like to further flush out our back-end application programming interface (API) to allow others to utilize the visualizer we have created with different particle swarm implementations. Our code already technically supports this, but cleaning up the way it connects into the front-end could prove beneficial for others when trying to visualize their particle swarm.

REFERENCES

[1] D. Bratton and J. Kennedy, ”Defining a Standard for Particle Swarm Optimization,” 2007 IEEE Swarm Intelligence Symposium, Honolulu, HI, USA, 2007, pp. 120-127, doi: 10.1109/SIS.2007.368035. keywords: Particle swarm optimization;Biological system modeling;Testing;Birds;Mathematical model;Measurement standards;Standards development;Algorithm design and analysis;Educational institutions;Statistics,

[2] F. van den Bergh and A. P. Engelbrecht, ”A Cooperative approach to particle swarm optimization,” in IEEE Transactions on Evolutionary Computation, vol. 8, no. 3, pp. 225-239, June 2004, doi: 10.1109/TEVC.2004.826069. keywords: Particle swarm optimization;Stochastic processes;Genetic algorithms;Africa;Computer science;Information technology;Neural networks;Topology;Partitioning algorithms;Space technology,

[3] G. Jornod, E. Di Mario, I. Navarro and A. Martinoli, ”SwarmViz: An open-source visualization tool for Particle Swarm Optimization,” 2015 IEEE Congress on Evolutionary Computation (CEC), Sendai, Japan, 2015, pp. 179-186, doi: 10.1109/CEC.2015.7256890. keywords: Optimization;Visualization,

[4] Panagiotakis, Costas. (2023). PSO based Unconstrained Polygonal Fitting of 2D Shapes. 10.20944/preprints202312.0933.v1.

[5] Parsopoulos, K.E., & Vrahatis, M.N. (2002). Particle swarm optimization method in multi objective problems. In: Proceedings of the 2002 ACM symposium on applied computing, pp 603–607.

[6] Tunchan Cura, Particle swarm optimization approach to portfolio optimization, Nonlinear Analysis: Real World Applications, Volume 10, Issue 4, 2009, Pages 2396-2406, ISSN 1468-1218, https://doi.org/10.1016/j.nonrwa.2008.04.023 (https://www.sciencedirect.com/science/article/pii/S1468121808001259)

[7] Chunming Yang and D. Simon, ”A new particle swarm optimization technique,” 18th International Conference on Systems Engineering (ICSEng’05), Las Vegas, NV, USA, 2005, pp. 164-169, doi: 10.1109/ICSENG.2005.9. keywords: Particle swarm optimization;Organisms;Birds;Marine animals;Educational institutions;Testing;Books;Equations;Systems engineering and theory;Social factors,

[8] Barend J. Leonard, Andries P. Engelbrecht and Andrich B. van Wyk Conference: 2011 IEEE Symposium on Swarm Intelligence, Year: 2011, Page 1 DOI: 10.1109/SIS.2011.5952564

[9] A. Stacey, M. Jancic and I. Grundy, ”Particle swarm optimization with mutation,” The 2003 Congress on Evolutionary Computation, 2003. CEC ’03., Canberra, ACT, Australia, 2003, pp. 1425-1430 Vol.2, doi: 10.1109/CEC.2003.1299838. keywords: Particle swarm optimization;Genetic mutations;Mathematics;Statistics;Convergence;Testing;Birds;Genetic algorithms;Stochastic processes,

[10] Bo Liu, Ling Wang, Yi-Hui Jin, Fang Tang, De-Xian Huang, Improved particle swarm optimization combined with chaos, Chaos, Solitons & Fractals, Volume 25, Issue 5, 2005, Pages 1261-1271, ISSN 0960-0779, https://doi.org/10.1016/j.chaos.2004.11.095. (https://www.sciencedirect.com/science/article/pii/S0960077905000330)