Optimisation of Boids Swarm Model Based on Genetic Algorithm and Particle Swarm Optimisation Algorithm (Comparative Study)

Saleh Alaliyat
Faculty of Engineering and Natural Sciences
Aalesund University College
N-6025, Aalesund, Norway
Email: saal@hials.no

Harald Yndestad
Faculty of Engineering and Natural Sciences
Aalesund University College
N-6025, Aalesund, Norway
Email: hy@hials.no

Filippo Sanfilippo
Department of Maritime Technology and Operations
Aalesund University College
N-6025, Aalesund, Norway
Email: fisa@hials.no

KEYWORDS
Flocking behaviour, genetic algorithms, and particle swarm optimisation.

ABSTRACT

In this paper, we present two optimisation methods for a generic boids swarm model which is derived from the original Reynolds' boids model to simulate the aggregate moving of a fish school. The aggregate motion is the result of the interaction of the relatively simple behaviours of the individual simulated boids[1]. The aggregate moving vector is a linear combination of every simple behaviour rule vector. The moving vector coefficients should be identified and optimised to have a realistic flocking moving behaviour. We proposed two methods to optimise these coefficients, by using genetic algorithm (GA) and particle swarm optimisation algorithm (PSO). Both GA and PSO are population based heuristic search techniques which can be used to solve the optimisation problems. The experimental results show that optimisation of boids model by using PSO is faster and gives better convergence than using GA.

INTRODUCTION

Many animals in the nature move in groups: fish swim in schools, birds fly in flocks, sheep move in herds, insects move in swarm and ants distribute to find a food source and then all ants follow path to the food. Simulating the aggregate motion is an important issue in the areas of artificial life and computer animation and in a lot of their applications such as games and movies. Reynolds (1987) proposed a first computer model of group animal motion such as fish schools and bird flocks. He called his model as "boids model", where boids refer to the generic flocking simulated creatures. The aggregate moving of the simulated boids is the result of the interaction of the relatively simple behaviours of the individual simulated boid. An interesting look at boids can be taken from the perspective of artificial life where is the holistic emergent phenomena is the result of interactions of independent entities (Anthony 2002). The boids model has three simple rules applied to the boids. The rules are: each boid move to avoid crowding with its neighbours, match and coordinate its movements with its neighbours, and move to gather with the others. Other rules such as avoiding obstacles and goal seeking have been included in steering behaviour model (Reynolds, 1999) and in many simulations based on boids model later on. For instance, Delgado (2007) extended the basic boids model to include the effects of fear. Olfaction was used to transmit emotion between animals, through pheromones modelled as particles in a free expansion gas. Hartman and Benes (2006) added a complementary force to the alignment (steer towards the average heading of neighbours) that they call the change of leadership. This steer defines the chance of the boid to become a leader and try to escape.

After 1987, the boids model is often used in computer graphics to provide realistic life-look representations of the aggregate motion of groups. For instance, in the 1998 Valve Video Game Company has used boids model in Half-Life video game for the flying bird-like creatures (Valvesoftware.com 2014). The boids model represented an enormous step forward compared to the traditional techniques used in computer animation for motion pictures. The first animation created with the boids model was in the computer animation shot film called Stanley and Stella in: Breaking the Ice in 1987 (Ice' and Malone, 2014). After that, the boids model was used in a feature film introduction of Batman Returns in 1992 (Returns and Burton et al., 2014). Then the boids model has been used in many games and films and in many other interesting applications.

In the boids swarm model, each rule is represented by a vector. The vector by its two components (magnitude and direction) is adaptive to the environment. The boid moving vector is a linear combination of every behaviour rule vector. The setting of the moving vector coefficients becomes more difficult by increasing more behaviour rules. These coefficients should be determined and optimised to have a realistic moving behaviour. In this context, we use two optimisation algorithms to optimise the boids model by finding the best coefficients values and combination in order to minimise the objective function. Firstly, we propose a GA to optimise the coefficients in a generic boids model. Secondly we substitute the GA by PSO to optimise the coefficients in the same generic boids model and use PSO to find food sources. Then we do a comparison of these two models by focusing on the advantages and disadvantages of each algorithm.

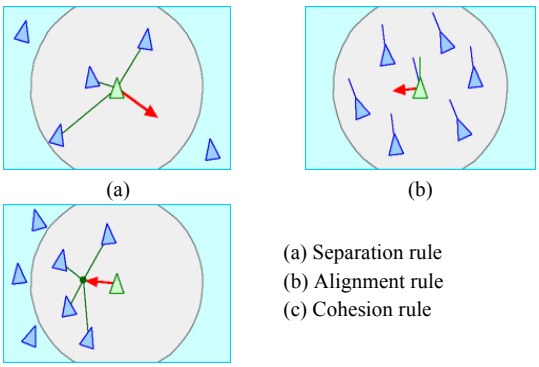
[1] Boids are bird-like objects that were developed in the 1980s to model flocking behaviour.

THE BOIDS MODEL

In 1986, Reynolds has developed the boids model. His published paper about the boids model (Reynolds 1987) was cited so many times and extended in so many different ways. Many of the extensions present additional rules to the boids, some describe constrained solution, some tend to easy solutions usable in computer games, some extend the previous work in spite of computational complexity, etc.

Reynolds (1987) describes the flock behaviour[2] as a result of the motion and the interaction of boids. Each boid has three simple rules of steering behaviours that describe how an individual boid move based on the positions and velocities of its flock mates (social reaction).

- Separation (*figure 1(a)*): each boid keep a distance from other boids nearby to avoid collision and prevent crowding.
- Alignment (*figure 1(b)*): each boid match the direction and the speed of its neighbours. This rule causes boids to follow each other.
- Cohesion (*figure 1(c)*): each boid tends to move to the average position of its neighbours.



(a) (b) (c)

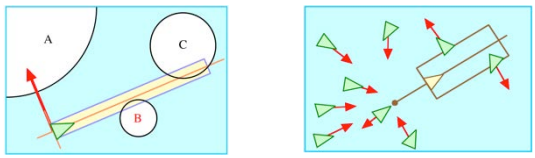
(a) Separation rule
(b) Alignment rule
(c) Cohesion rule

Figure 1: The boids social rules (Reynolds 1987).

Reynolds (1999) has extended the boids model to include more individual-based rules of the steering behaviours, to have more advanced individuals which are capable to finish specific task or adapt to complex environments. Some of these behaviours are:

- Obstacle avoidance (*figure 2(a)*): The obstacle avoidance behaviour allows the boids move in cluttered environment by dodging around obstacles.
- Leader following (*figure 2(b)*): this behaviour causes one or more boids to follow another moving boid selected as a leader.

[2] We mean by *Flock behaviour* in this paper as behaviour of flock, school, herd and swarm.



(a) (b)

Figure 2: Steering behaviours rules (a) Obstacle avoidance (b) leader following(Reynolds 1999).

Based on Reynolds model, we have implemented a generic boids swarm model in Unity3D (Unity3d.com 2014), the program is written in MonoDevelop Unity-C# (Docs.unity3d.com 2014). The aim is to develop a generic model that can be used in simulating the aggregate motion for flocks of birds, schools of fish or herds of animals. The model has five rules:

- Cohesion: steer to move toward the average position of local flock mates (as in the original Reynolds' model). By applying cohesion rule keeps the boids together. This rule acts as the complement of the separation. If only cohesion rule is applied, all the boids in the flock will merge into one single position. Cohesion (Coh_i) of the boid (b_i) is calculated in two steps. First, the center ($\overrightarrow{Fc_i}$) of the flock (f) that has this boid is calculated as in *equation 1*. Then the tendency of the boid to navigate toward the center of density of the flock is calculated as the cohesion displacement vector as in *equation 2*.

$$\overrightarrow{Fc_i} = \sum_{\forall b_j \in f} \frac{\overrightarrow{p_j}}{N} \quad (1)$$

Where, p_j is the position of boid j and N is the total number of boids in f

$$\overrightarrow{Coh_{i=}} \overrightarrow{Fc_i} - \overrightarrow{p_i} \quad (2)$$

- Alignment: steer to match the heading and the speed of its neighbours. This rule tries to make the boids mimic each other's course and speed. Boids tend to align with the velocity of their flock mates. The alignment (Ali_i) is calculated in two steps. First, the average velocity vector of the flock mates ($\overrightarrow{Fv_i}$) is calculated by *equation 3*. Then Ali_i is calculated as the displacement vector in *equation 4*.

$$\overrightarrow{Fv_i} = \sum_{\forall b_j \in f} \frac{\overrightarrow{v_j}}{N} \quad (3)$$

$$\overrightarrow{Ali_{i=}} \overrightarrow{Fv_i} - \overrightarrow{v_i} \quad (4)$$

Where $\overrightarrow{v_i}$ is the velocity vector of boid i

If this rule was not used, the boids would bounce around a lot and not form the beautiful flocking behaviour that can be seen in the nature.

- Separation: steer to avoid collection and overcrowding with other flock mates. There are many ways to implement this rule. An efficient solution to calculate the separation (Sep_i) is by applying *equation 5*. Vectors defined by the

position of the boid b_i and each visible boid b_j are summed, then separation steer ($\overrightarrow{Sep_i}$) is calculated as the negative sum of these vectors.

$$\overrightarrow{Sep_i} = -\sum_{\forall b_j \epsilon f}(\vec{p_i} - \vec{p_j}) \quad (5)$$

If only the separation rule is applied, the flock will dissipate.

- Leader following: steer to follow another moving boid selected as a leader (p_l). The leader following (Led_i) is calculated by *equation 6*.

$$\overrightarrow{Led_i} = L * (\vec{p_i} - \vec{p_l}) \quad (6)$$

Where L is a leader strength factor. (Note: the moving vector (velocity) has limits, minimum and maximum).

- Random movement: this rule is added to have more realistic flock behaviour. This rule is depending on the random number generator inside the game engine (*Unity3D*). The random movement ($Rand_i$) is calculated as in *equation 7*.

$$\overrightarrow{Rand_i} = -ffactor * \vec{r} \quad (7)$$

Where r is a unit sphere random vector and $ffactor$ is a flock random strength factor.

Then the moving vector (V_i) is for boid (b_i) is calculated by combining all the steering behaviour vectors as in *equation 8*.

$$\vec{V_i} = w_1 \overrightarrow{Coh_i} + w_2 \overrightarrow{Ali_i} + w_3 \overrightarrow{Sep_i} + w_4 \overrightarrow{Led_i} + w_5 \overrightarrow{Rand_i} \quad (8)$$

Where w_i are the coefficients describing influences of each steering rule and used to balance the five rules.

We have used the Unity3D to implement the boids model to get the benefits of using a game engine. The first benefit is the amazing visualisation that we get in Unity3D. So we skip wasting time to model and program the boid's shape and its geometry. In Unity3D, it is easy to build a boid such as a bird, a fish or a sheep and attach some life-look animation to it, or import the 3D model of the boid from other programs and attach a built in animation to it or program the animation from the scratch. In this model we exploit the collision detection component in Unity3D game engine to avoid the obstacles. Each obstacle has a physic's collision component that doesn't let other objects to move through the collision bounds (obstacle's space). In another words, the obstacles will be excluded from the boids flocking space. We will show the results in the experimental results section.

GENETIC ALGORITHMS FOR OPTIMISATION OF BOIDS MODEL

In this section, we will give an overview of GA in general and some examples of its applications. Then we present our proposed model (GA for optimisation of boids model).

Genetic Algorithm:

Genetic algorithm (GA) is an optimisation and search technique based on the principles of genetic and natural selection (Haupt, 2003). A GA allows a population composed of many individuals to evolve under specified selection rules to a state that maximise the fitness (i.e., minimizes the cost function). Genetic algorithms (GAs) were invented by John Holland in 1960s and were developed by him and his students in 1960s and 1970s. Holland (1975) presented the GA as an abstraction of biological evolution and gave a theoretical framework for adaptation under GA.

GA belong to the larger class of evolutionary algorithms, which generate solutions to optimization problems using techniques inspired by natural evolution such as selection (reproduction), crossover (recombination) and mutation. The evolution process starts from a population of individuals generated randomly within the search space and continues for generations. In each generation, fitness of every individual is evaluated, and multiple individuals are randomly selected from the current population based on their fitness and modified by recombination and mutation operation to form a new population. Then this new population will be used for the next generation of the evolution. In general, the search process ends when either a maximum number of generations have been produced or a fitness level has been reached for the population. The flowchart of GA is shown in (*figure 3)*.

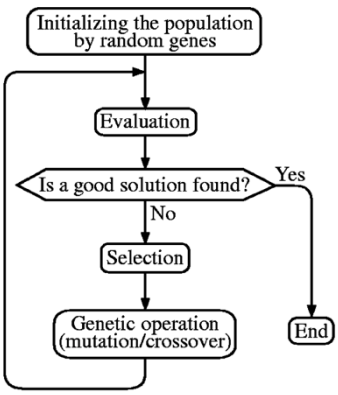


Figure 3: Flowchart of GA

In a GA, it's necessary to be able to evaluate how good a potential solution is relative to other potential solutions. The fitness function is responsible for performing this evaluation and returning a fitness value (positive integer number) that reflects how optimal the solution is. The fitness function is associated with the objective function of the problem. The fitness value of the individual is used to determine the probability with which the individual is selected into the new population. A common metaphor for the selection process is a roulette wheel selection (Fogel, 2000).

Traditional methods of search and optimization are too slow in finding a solution in a very complex search space. GA is a robust search method requiring little information to search effectively in a large or poorly understood search space. In particular a genetic search

progress through a population of points in contrast to the single point of focus of most search algorithms. Moreover, it is useful in the very tricky area of nonlinear problems. GAs have been used to solve optimisation problems in different fields such as automotive design, engineering design, robotics, optimised routing, games, etc. Chen (2006) has applied GAs to optimise the behaviour of a school of fish.

GA for optimisation of moving vector in the boids model:

The moving vector (V_i) in *equation 8* for each boid (b_i) is a combination of all the five steering behaviour vectors. And the movements are balanced by the w_i weight coefficients, so these coefficients should be optimized to have realistic and life-look behaviour. We have removed the random steering behaviour in the moving vector to exclude the random movements for the boids. The new moving vector is:

$$\vec{V_i} = w_1 \overrightarrow{Coh_i} + w_2 \overrightarrow{Ali_i} + w_3 \overrightarrow{Sep_i} + w_4 \overrightarrow{Led_i} \quad (9)$$

We have used GA to optimise these coefficients and getting the benefit from using GA for parameters optimisation and finding a global optimum solution. The goal is to find the optimal solutions in terms of the variables (coefficients). Thus we should define mathematically what is the optimal solution. We begin the GA by defining the chromosome. The chromosome is an array of the coefficients values that will be optimised. In this case the chromosome has four variables and is written as a four-element row vector.

$$chromosome = [w_1, w_2, w_3, w_4] \quad (10)$$

Then we should formulate the cost function that gives a cost for each chromosome.

$$cost = f(chromosome) = f(w_1, w_2, w_3, w_4) \quad (11)$$

In our case, the optimal solution is to have life-look flock behaviour. Measuring the flock behaviour can be very complicated process, expensive computationally and then time consuming. Since the purpose of this paper is to build a generic boids model and optimise it by different optimisation methods, we suggest a simple cost function. In the flowing, we explain the proposed cost function which is divided into five parts.

- Related to the alignment rule: The divergence between the direction of the boid and the average direction of the flock should be minimised. The divergence is the angle θ between the boid velocity vector $\vec{v_i}$ and the average flock velocity vector.

$$cost_1 = \theta \quad (12)$$

- Related to the leader following rule: The divergence between the direction and the distance of the boid and the direction and the distance of the leader should be minimised.

$$cost_2 = d \quad (13)$$

Where d is the distance between p_l and p_i

$$cost_3 = \alpha \quad (14)$$

Where α is the angle between the boid velocity vector $\vec{v_i}$ and the leader velocity vector.

- Related to the separation and cohesion rules: The boids distribution should be optimised to avoid crowding or losing contact and having nice flocking. To do this; we calculate the distance between the boid and the flock center first d_c. Then we check all the boids, if they are nearby (<*keepd*) or far enough (>*keepd*).

If $(|(\vec{p_i} - \vec{p_j})| \leq keepd)$,

$$cost_4 = \sum_{\forall b_j \epsilon f} d_c * \frac{(|(\vec{p_i} - \vec{p_j})| - keepd)^2}{keepd^2} \quad (15)$$

But for the far boids,

If $(|(\vec{p_i} - \vec{p_j})| > keepd)$

$$cost_5 = \sum_{\forall b_j \epsilon f} d_c * \frac{(|(\vec{p_i} - \vec{p_j})| - keepd)^2}{(d_c - keepd)^2} \quad (16)$$

Then the cost is:

$$cost = cost_1 + cost_2 + cost_3 + cost_4 + cost_5 \quad (17)$$

We have used the continuous GA as explained in (Haupt, 2003). We used the parameters in (*Table 1*). We will analyse the results in experiment results section.

Number of optimisation variables	4
Upper limit on optimisation variables	1
Lower limit on optimisation variables	0
Maximum iteration	100
Minimum cost	0
Population size	20
Mutation rate	0.2
Selection rate	0.5

Table 1: GA parameters

PARTICLE SWARM OPTIMISATION OF BOIDS MODEL

In this section, we will give an overview of PSO algorithm in general and some examples of its applications. Then we present our proposed model (PSO for optimisation of boids model).

Particle Swarm Optimisation:

PSO is a computational method that optimises a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. Kennedy and Eberhart introduced PSO in 1995 (Kennedy, 1995). PSO was originally used to solve non-linear continuous optimization problems, but more recently it has been used in many practical, real-life application problems. For example, PSO has been successfully applied to track dynamic systems (Eberhart, 2001) and evolve weights and structure of neural networks (Zhang, 2000). PSO draws inspiration from the sociological behaviour associated with bird flocking. It is a natural observation that birds can fly in large groups with no collision for extended long distances, making use of their effort to maintain an optimum distance between themselves and their neighbours.

Cui (2009) has developed a hypried PSO and boids model (Boids-PSO), where cohesion rule and alignment

rule are both employed to improve the PSO algorithm for boids simulation and to overcome the weekness of biological background of PSO. But in our case we use PSO as an optimisation technique to optimise the coefficients in the moving vector.

The PSO methodology operates by placing a group of individual particles into a continues search space, wherein each particle is initialised with a random position and a random initial velocity in the search space. The position and velocity are updated synchronously in each iteration of the algorithm. Each particle adjust its velocity according to its own flight experience and the other's experience in the swarm in such a way that it accelerates towards positions that have high fitness values in previous iterations. In other words, each particle keeps track of its coordinates in the solution space that are associated with the best solution that has achieved so far by its self. This value is called personal best (*pbest*), Another best value that is tracked by the PSO is the best value obtained so far by any particle in the neighbourhood of that particle. This value is called (*best*). So the basic concept of PSO lies in accelerating each particle toward its *pbest* and the *gbest* locations, with a random weighted acceleration at each time step as shown in (*figure 4*).

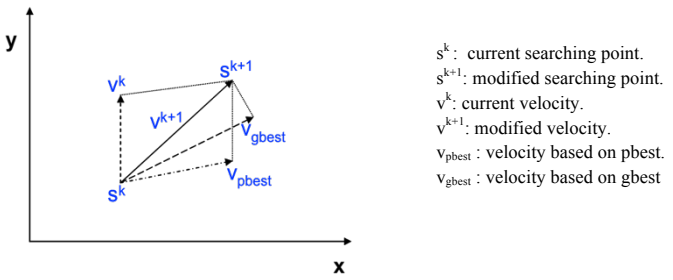


Figure 4: Concept of particle position modification by PSO

The modification of the particle's position can be mathematically modelled according to *equation 18*.

$$\vec{v}(k+1) = \vec{v}(k) + c_1\overrightarrow{R_1}\left(\overrightarrow{pbest} - \vec{s_i}(k)\right) + c_2\overrightarrow{R_2}\left(\overrightarrow{gbest} - \vec{s_i}(k)\right) \quad (18)$$

Where,

$\vec{v}(k)$ is the velocity of a particle at iteration k.

$\overrightarrow{R_1}$ and $\overrightarrow{R_2}$ are random numbers in the range of [0,1] with the same size of the swarm population.

c_1 and c_2 are learning factors which will be fixed through whole the process.

In order to improve the local search precision, Eberhart (2001) added the inertia weight w to *equation 18* to be as following equation.

$$\vec{v}(k+1) = w\vec{v}(k) + c_1\overrightarrow{R_1}\left(\overrightarrow{pbest} - \vec{s_i}(k)\right) + c_2\overrightarrow{R_2}\left(\overrightarrow{gbest} - \vec{s_i}(k)\right) \quad (19)$$

The inertia weight controls the momentum of the particle by weighing the contribution of the previous velocity.

Chatterjee (2006) suggested a dynamic change of inertia weight in his work.

Clerc (1999) indicates that the use of a constriction factor K may also be necessary to ensure convergence of the particle swarm algorithm, defined as when all particles have stopped moving. Then the velocity is calculated by the equation:

$$\vec{v}(k+1) = K[\vec{v}(k) + c_1\overrightarrow{R_1}\left(\overrightarrow{pbest} - \vec{s_i}(k)\right) + c_2\overrightarrow{R_2}\left(\overrightarrow{gbest} - \vec{s_i}(k)\right)] \quad (20)$$

$$K = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|} \quad (21)$$

Where $\varphi = c_1 + c_2$ and $\varphi > 4$.

Then the new position for the particles is the addition of the position at k iteration and the distance that the particles will fly with the new velocity $\vec{v}(k+1)$. The position is updated by *equation 22*.

$$\vec{s_i}(k+1) = \vec{s_i}(k) + \vec{v}(k+1) \quad (22)$$

The flow chart of a general PSO algorithm is shown in (*figure 5*).

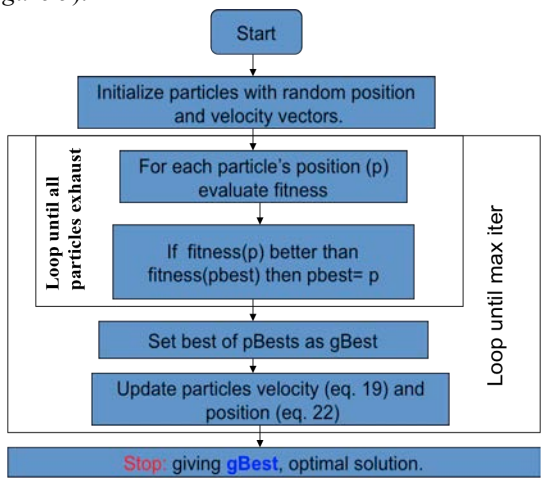


Figure 5: flow chart of general PSO algorithm

Path Planning using PSO:

One of the main applications of PSO is the path planning. PSO has been applied massively for path planning intensely in robots (Chen X 2006 and Nasrollahy 2009).

In the boids model, PSO can be applied to the flock leader, to plan and smooth his path. For this purpose; we have used PSO algorithm to plan the path to the target (i.e. food source). We have used the Euclidian distance between the particle and the target as a fitness function in the PSO.

In PSO for path planning, the inertia weight w is calculated according to the next equation.

$$w = w_{start} - \frac{w_{start} - w_{end}}{K}k \quad (23)$$

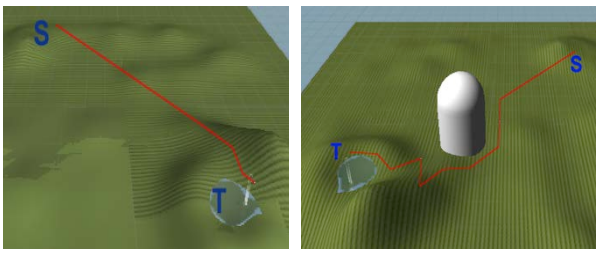
Where, K is the iteration maximum number and k is the current iteration. By linearly decreasing the inertia weight from a relatively large value to a small value, the PSO tends to have more global search ability at the beginning of the run while having more local search ability near the end of the run.

As in robot's applications, PSO gives advantages to the path planning particularly in the dynamics environment containing stationary and moving obstacles.

We have used the parameters in (*Table 2*) for the PSO. In case of facing obstacles; the leader is looking to his target, if there is an obstacle whose obscures the target and the distance to this obstacle is less than a threshold, the leader will change his direction randomly to his right or his left by 45-degrees angle.

Swarm size	20
Dimension of the problem	2
Maximum iteration	100
c1 (cognitive parameter)	2
c2 (social parameter)	2
C (constriction factor)	1
Inertia start	0.9
Inertia end	0.4
Upper limit on optimisation variables	100
Lower limit on optimisation variables	-100

Table 2: PSO parameters for path planning



(a) (b)

Figure 6: PSO for path planning without obstacles (a) and with an obstacle (b).

PSO for optimisation of moving vector in boids model:

As in GA for optimisation of the moving vector in the boids model, we have applied PSO to find the optimum coefficients for the moving vector (V_i) in *equation 9* for each boid (b_i). We have used the same cost function as in equations from 12 to 17. We have used the PSO as explained in (Kennedy, 1995).

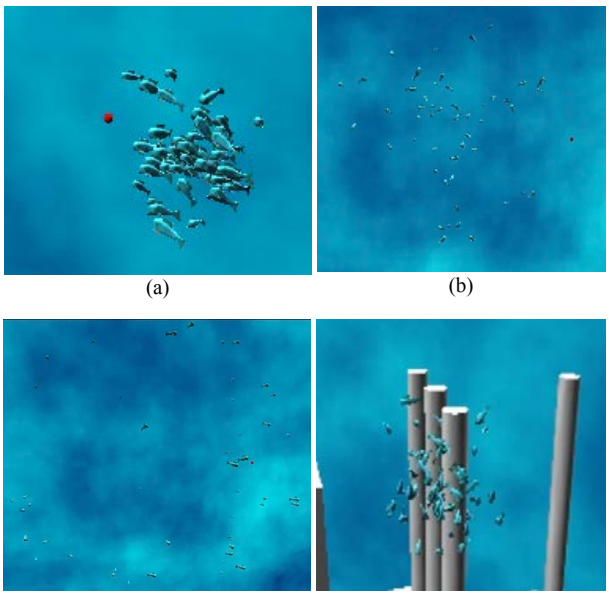
Population size	20
Dimension of the problem	4
Maximum iteration	100
c1 (cognitive parameter)	2
c2 (social parameter)	2
C (constriction factor)	1
Inertia start	0.9
Inertia end	0.4
Upper limit on optimisation variables	1
Lower limit on optimisation variables	0

Table 3: PSO parameters

We have used the parameters in (*Table 3*) and we have used *equation 23* to calculate the Inertia. We will analyse the results in next section.

THE EXPERIMENT RESULTS

For testing the boids model without/with the GA and PSO, we have made a fish school in Unity3D. we have used a ready fish boid from unity website to have a nice fish shape with some animations such as moving the fish tail. The fish school has 50 fish (*figure 7a*). Firstly we have implemented the boids model as in *equation 8* with excluding the random steering, and then we added the random steering and the leader factor. *Figure 7(b,c,d)* shows the simulation from different efforts. It was observed that the model need time to have a nice flocking shape with/without random movement. And the random movement was important to avoid obstacles since we didn't have a separate steering behaviour for avoiding the obstacles. The simulation of a fish school depends on the boids model as in *equation 8* with equal weight coefficients gives a good flocking shape but the model was slow to get the nice flocking shape.



(a) (b)

(c) (d)

Figure 7: (a) Fish school system (b) the boids simulation without random movement after few frames from the start (c) the boids simulation with random movement (d) the boids simulation with random movement and adding leader factor and some obstacles.

Using the GA for optimisation the moving vector by finding the optimal coefficients, made the school of fish getting the nice flock shape sooner, but its very computational costly. The frame rate went down from more than 30 frames per second to almost 3 frames per second (this depends on the parameters of GA). And it is noticeable that after passing the start time and having the flock shape, there is no noticeable difference between the original boids model and the boids model

with GA. *Figure 8(a)* shows a screen shot from the simulation with GA.

(a) (b)

Figure 8: The boids simulation with (a) GA, and (b) PSO

PSO was faster than GA, and gave more noticeable results as shown in (*figure 8(b)*). It is observed that the boids get the flock shape faster and even the flock behaviour is look nicer than before.

We depend on our observation of the simulation results to do the comparison between the different models, because each simulation/run is different from other simulations/runs and it depends on the starting positions and the random numbers. We have selected the same population size and the number of iterations for both GA and PSO algorithms. The other parameters in GA, were selected by running the GA on many standard optimisation problems and from the literature (Haupt, 2003). The other parameters in PSO algorithm, were selected from the literature and the path planning algorithm. These parameters are selected to have a good convergence. The cost function or the objective fuction in general should be connected to the type of simulation. In our experiment, the objective function is to have nice fish school behaviour.

CONCLUSIONS AND FUTURE WORK

The boids model is often used in computer graphics to provide realistic life-look representations of the aggregate motion. For instance, many animations require natural-looking behaviour from a large number of characters (boids). The aggregate motion of the group is the result of the interaction of the individuals, so let each individual generate its own motion, this is easier and produces natural and unscripted motion. The individual's moving is calculated by combining all the steering behaviour vectors. To have a natural behaviour in different environments, the boid's movements should be optimised and adapted. GA and PSO algorithm are used to optimise a generic boids model by optimising the coefficients of the moving vector to minimise the cost function.

The challenge is to write rules to define natural behaviours. In the boids model, we have defined the cost function which is divided into five parts. These parts are related to the steering behaviours in the model. Thus the cost function reflects how the fish school should look in the nature. Our cost function is not computationally costly and measures simply the boids behaviour. The cost function (objective function) should be connected to type-of-problem we want to solve, and reflects how we want the flock/swarm to behave. The setting of moving vector coefficients is determined by the cost function. We use GA and PSO to find the best coefficients values (waights of the behaviours rules) to minimise the cost function which reflects the wanted behaviour.

GA and PSO have many similarities and both of them use population-based approaches. GA is known as a good algorithm to find the global optimal solution where is PSO could stuck in the local optimum. But PSO has advantage over GA concerning the time. In the boids model, where we have adaptive boids in a dynamic environment, we are interested in a nice flocking behaviour (convergence) as in nature, and in time consuming. From the experiments for both GA and PSO, we observed that PSO is much faster than GA, and gives a faster convergence, and because the PSO is computationally less costly than GA, we could notice the convergence more in PSO than GA.

The challenge is to balance between the convergence (nice flocking behaviour and the adaptively) and the time consuming. Having more advanced cost function probably will give better results, but it will be very expansive and lead to a very slow model. Applying the optimisation algorithm not continuously such as applying the optimisation algorithms for only some parts of the simulation such as at beginning of the simulation until the boids get a nice flock shape which is wanted, or when the boids facing obstacles or enemies, will accelerate the model.

REFERENCES

Anthony L. (2002), "Artificial life", Macmillan Press Ltd., Basingstoke, UK.

Chatterjee A., Siarry P. (2006), "Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimisation", Computer & Operations Research 33, 859-871.

Chen, Y.-W.; Kobayashi, K.; Huang, X. & Nakao, Z. (2006), "Genetic Algorithms for Optimization of Boids Model", *in* Bogdan Gabrys; Robert J. Howlett & Lakhmi C. Jain, ed., 'KES (2)' , Springer, , pp. 55-62 .

Chen X.; Yangmin L. (2006), "Smooth Path Planning of a Mobile Robot Using Stochastic Particle Swarm Optimization", *Mechatronics and Automation, Proceedings of the 2006 IEEE International Conference on* , vol., no., pp.1722,1727, 25-28.

Clerc M. (1999), "The swarm and the queen: towards a deterministic and adaptive particle swarm optimization", in Proceedings of the Congress of Evolutionary Computation, Washington, DC, pp. 1951-1957.

Cui Z., Shi Z. (2009), "Boid particle swarm optimisation", *International Journal of Innovative Computing and Applications* 2 (2): 77–85.

Delgado M. C., Ibanez J., Bee S., *et al.* (2007), "On the use of Virtual Animals with Artificial Fear in Virtual Environments", *New Generation Computing* 25 (2): 145–169.

Docs.unity3d.com (2014), *Unity - Getting started with Mono Develop*. [ONLINE] Available at:

http://docs.unity3d.com/Documentation/Manual/HOWTO-MonoDevelop.html. [Accessed 31 January 2014].
Eberhart R., Shi Y. (2001), "Tracking and optimizing dynamic systems with particle swarms", Proc. Congress on Evolutionary Computation 2001, Seoul, Korea
Fogel D. (2000), "Evolutionary Computation: Towards a New Philosophy of Machine Intelligence", IEEE Press, New York.
Hartman C., Benes B. (2006), "Autonomous boids", *Computer Animation and Virtual Worlds* 17 (3-4): 199–206.
Haupt R., Haupt S.(2003), "Practical Genetic Algorithms", 2nd Ed, Wiley, 2003.
Holland J. (1975), "Adaptation in Natural and Artificial Systems", Ann Arbor: University of Michigan Press.
Ice', S. and Malone, L. (2014), "*Stanley and Stella in 'Breaking the Ice' (1987)",* [online] Available at: http://www.imdb.com/title/tt0302371/ [Accessed: 31 Jan 2014].
Kennedy J., Eberhart R. (1995), "Particle Swarm Optimisation", *Proceedings of IEEE International Conference on Neural Networks* IV. pp. 1942–1948.
Nasrollahy, A.Z.; Javadi, H. (2009), "Using Particle Swarm Optimization for Robot Path Planning in Dynamic Environments with Moving Obstacles and Target", *Computer Modeling and Simulation, 2009. EMS '09. Third UKSim European Symposium on* , vol., no., pp.60,65, 25-27.
Returns, B., Burton, T., Kane, B., Waters, D., Keaton, M., Devito, D. and Pfeiffer, M. (2014), "Batman Returns (1992)" *IMDb*, [online] Available at: http://www.imdb.com/title/tt0103776/ [Accessed: 31 Jan 2014].
Reynolds C. (1987), " Flocks, Herds, and Schools: A Distributed Behavioural Model", Computer Graphics, 21:4,1987, 25-34.
Reynolds C. (1999), " Steering behaviour for autonomous characters", http://www.red3d.com/cwr/steer/, first version from 1999.
Unity3d.com (2014), *Unity - Game Engine*. [ONLINE] Available at: http://www.unity3d.com. [Accessed 31 January 2014].
Valvesoftware.com (2014), *Valve*. [ONLINE] Available at: http://www.valvesoftware.com/. [Accessed 31 January 2014].
Zhang C., Shao H., Li Y. (2000), "Particle Swarm Optimisation for Evolving Artificial Neural Network", In the 2000 IEEE International Conference on Systems, Man, and Cybernetics, vol.4, pp.2487-2490.

AUTHOR BIOGRAPHIES

SALEH ALALIYAT was born in Jenin, Palestine. He is currently working as a PhD candidate at Aalesund University College, Norway. He received his Master's degree in Media Technology from Gjøvik University College in Norway.

HARALD YNDESTAD was born in Aalesund, Norway. He has studied cybernetics at the University in Trondheim, obtained a dr.philos degree in 2004 and he is now professor at Aalesund University College. His research interests are complex systems, swarm intelligence and ecosystem dynamics.

FILIPPO SANFILIPPO is a PhD candidate in Engineering Cybernetics at the Norwegian University of Science and Technology, and a research assistant at the Department of Maritime Technology and Operations, Aalesund University College, Norway. He obtained his Master's Degree in Computer Engineering at University of Siena, Italy.