# "GA-PSO HYBRID ALGORITHM TO FIND AN OPTIMAL PATH BETWEEN A STARTING AND ENDING POINT IN A GRID ENVIRONMENT"

Project Report Submitted in Partial Fulfilment of the Requirement

for the degree of

**Bachelor of Technology**

**In**

**Computer Science & Engineering**

**Under the supervision of**

**Dr. Anita Yadav**

**Associate professor (HBTU Kanpur)**

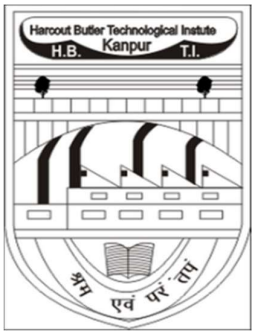

**By:**

**HARSHIT BAJPAI (107/14)**

**NITIN GUPTA (111/14)**

**SHIVAM GOEL (112/14)**

**PUSPENDRA SINGH (101/14)**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**HARCOURT BUTLER TECHNICAL UNIVERSITY, KANPUR**


**(2017-18)**

# CERTIFICATE

This is to certify that this report entitled "**GA-PSO HYBRID ALGORITHM TO FIND AN OPTIMAL PATH BETWEEN A STARTING AND ENDING POINT IN A GRID ENVIRONMENT**" which is submitted by Harshit Bajpai, Nitin Gupta, Shivam Goel and Puspendra Singh in partial fulfilment of the requirement for the award of the degree Bachelor of Technology in Computer Science and Engineering to the Department of Computer Science in Harcourt Butler Technical University, Kanpur, is a record of candidates' own work carried out by them under my supervision. The matter embodied in this thesis is original and has not been submitted for the award of any other degree.

**Date:**

**Dr. Anita Yadav**

**(Associate Professor)**
**Department of computer science and Engineering**
**Harcourt Butler Technical University,Kanpur**

# DECLARATION

We hereby declare that the project work which is being presented in the project entitled "**GA-PSO HYBRID ALGORITHM TO FIND AN OPTIMAL PATH BETWEEN A STARTING AND ENDING POINT IN A GRID ENVIRONMENT**"in partial fulfilment of the requirement for the award of the degree of Bachelor of Technology in Computer Science and Engineering submitted in the department of Computer Science and Engineering, HBTU, Kanpur is an authentic record of our work carried out during the period August 2017 to May 2018 under the supervision of Mrs. Anita Yadav.

The matter embedded in this report has not been submitted by us for the reward of any other degree.

**Date:**

**H.B.T.U.,Kanpur**

**Harshit Bajpai**

**Nitin Gupta**

**Shivam Goel**

**Puspendra Singh**

# ACKNOWLEDGEMENT

Development of this project was a meticulous job and requires lot of commitment. It is a pleasure for us to express our thanks and heartiest gratitude to all those who helps us directly or indirectly during the development of this challenging project. We would like to take this opportunity to thank them all.

While we cheerfully share the credit for the accurate aspect of this project report, the mistakes and omissions we have to claim as our alone. Please bring them to our attention.

We would like to thank **Dr. Anita Yadav (Associate Professor)** for providing us the required useful information and guiding us through this project. We would like to express our heartiest gratitude to **Dr. Narendra Kohli (Professor)** for their highly valuable support require for development of this project. They were very encouraging and helpful throughout this project. We would also like to thank Mr. Indresh Gupta (faculty, CSE dept.) for his valuable support throughout the project duration.

Our sincere gratitude to H.B.T.U, in general, for providing excellent material and labs with all facilities for project development.

# ABSTRACT

Traditional method to solve network routing problems performs a limited exploration of search space and results in inferior solution. Some other traditional method are general purpose optimization that usually involve large set of parameters that need to be fine-tuned, thus PSO(Particle swarm optimization) is a simple algorithm that seems to be effective for optimizing  a wide range of function. We have implemented an paper on FPGA-based (field-programmable gate array) hybrid heuristic GA (genetic algorithm)-PSO (particle swarm optimization) algorithm for mobile robots to find an optimal path between a starting and ending point in a grid environment. GA has been combined with PSO in evolving new solutions by applying crossover and mutation operators on solutions constructed by particles. This hybrid algorithm avoids the premature convergence and time complexity in conventional GA and PSO algorithms. The initial feasible path generated from the hybrid GAPSO planner is then smoothed using the cubic B-spline technique, in order to construct a near-optimal collision-free continuous path. Experimental results are conducted to show the merit of the proposed hybrid GA-PSO path planner for global path planning for mobile robots.

# INDEX

# LIST OF FIGURES

# 1. INTRODUCTION

## 1.1 GRID COMPUTING

Grid computing is the collection of computer resources from multiple locations to reach a common goal. The grid can be thought of as a distributed system with non-interactive workloads that involve a large number of files. Grid computing is distinguished from conventional high-performance computing systems such as cluster computing in that grid computers have each node set to perform a different task/application. Grid computers also tend to be more heterogeneous and geographically dispersed (thus not physically coupled) than cluster computers. Although a single grid can be dedicated to a particular application, commonly a grid is used for a variety of purposes. Grids are often constructed with general-purpose grid middleware software libraries. Grid sizes can be quite large.

Grids are a form of distributed computing whereby a "super virtual computer" is composed of many networked loosely coupled computers acting together to perform large tasks. For certain applications, distributed or grid computing can be seen as a special type of parallel computing that relies on complete computers (with onboard CPUs, storage, power supplies, network interfaces, etc.) connected to a computer network (private or public) by a conventional network interface, such as Ethernet. This is in contrast to the traditional notion of a supercomputer, which has many processors connected by a local high-speed computer bus.

## 1.2 Quality of Service

With the rapid development of Internet, mobile networks and high-performance networking technology, multicast routing has become a very important research issue in the areas of networks and distributed systems. Currently multicast-based applications have pervasive presence and influence in wide area networks. This increased the demand for multicasting algorithms that can efficiently manage network resources and satisfy requirements . The provision of quality-of-service guarantees is of utmost importance as the internet expands many new real-time communication services such as multimedia teleconference, audio/video broadcasting, and remote education. These services usually require the transmission of information from one source node to a large number of destination nodes according to a multicast distribution tree . Their analysis is classified as multiple destinations routing . The main objective of the MDR problem is to construct the optimal multicast tree in the distributed network that determines the best routing for the delivery of a message from the source node to multiple destination nodes while optimizing a certain performance criteria and meeting all QoS requirements. Recently, with the high demand of fast and better quality of services, a number of rigid QoS criteria, such as bandwidth, delay, jitter, and packet loss rate, have been considered.

## 1.3 Grid Computing and multicast Routing

This QoS multicast routing problem has drawn wide spread attention from researchers who have been using different methods to solve the problem using conventional algorithms, such as exhaustive search routing and greedy routing. Typical approaches include applying Dijkstra algorithm to find the shortest path, seeking the minimum network cost using Steiner tree routing algorithm, and finding multicast tree that the paths between source node and the destination nodes are connected and their cost is minimized .

The multicast routing problem is known to be NP-Complete and for a large scale network with high real time response, it is expensive or even infeasible to find the optimal multicast trees. Thus, most previous researchers have focused on developing heuristic algorithms that take polynomial time and produce near optimal results for solving the multicasting routing problem . The advent of evolutionary computation has inspired new resources for optimization problem solving. Many evolutionary algorithms, such as genetic algorithm (GA) , simulated annealing (SA) , and ant colony optimization(ACO),have been proposed for solving the QMR problem. However GA, SA, and ACO have practical limitations in real-time multicast routing. The GA climbing capacity is weak and premature easily. Both the efficiency of the SA algorithm and the quality of the solution depends on procedures that are sensitive to the influence of random annealing sequence. The ACO algorithm has many parameters and cannot guarantee convergence to the global optimal.

Until now, limited papers have discussed the application of particle swarm optimization (PSO) algorithm to solve the QMR problem. Liu et al. and Wang et al. proposed a PSO based algorithm to solve the QMR problem in by means of serial path selection to realize the optimization of a multicast tree. The multicast tree can obtain a feasible solution by exchanging paths in the vector. Experimental results indicate that the proposed algorithm could converge to the optimal or a near-optimal solution with lower computational cost. Another algorithm was described by Sun et al. based on the quantum-behaved PSO(QPSO). It was inspired by quantum mechanics and seemed to be a promising optimization problem solver. The proposed method converts the QoS multicast routing problem into an integer-programming problem and then solves the problem by QPSO. Additionally, combining PSO with other optimization techniques to deal with QoS routing was also proposed in the literature. In Xi-Hong et al. proposed an ACO-PSO algorithm to solve the QMR problem. The solution generated by ACO is regulated by position update strategy of PSO, which extends the search scope efficiently and avoids premature convergence to local optima. The simulation results demonstrate its superiority to other algorithms such as the GA and the ACO. Li et al. described a new evolutionary scheme for the optimization of multicast QoS routing based on the hybrid of GA and PSO, called HGAPSO. In HGA PSO, the upper-half of the best-performing individuals in a population are regarded as elites. Instead of being reproduced exactly in the next generation, these elites are enhanced first. The group constituted by the elites is regarded as a swarm, and each elite corresponds to a particle within it. Wang et al.  proposed a new method for tree-based optimization. The algorithm

optimizes the multicast tree directly, unlike the conventional solutions to finding paths andintegrating them to generate a multicast tree. The algorithm combines PSO with tree based optimization to the solution to control the optimization orientation of the tree shape.

## 1.4 Need of Hybrid

The aim is to develop an efficient heuristic algorithm that can solve the QMR problem. The proposed algorithm utilizes PSO algorithm that has emerged as a new heuristic that can efficiently solve large-scale optimization problems. This study differs from existing literature in the following aspects: First, in this study various QoS measures are considered such as cost, bandwidth, delay and jitter. The proposed model treats these constraints separately, and can be extended to add more constraints. Second, new discrete PSO operators have been presented to modify the original PSO velocity and position update rules to the discrete solution space in the multicast routing problem. Third, a new adjustable PSO-GA hybrid multicast routing algorithm which combines PSO with genetic operators was proposed. The performance of the adjustable hybrid model is optimized by two driving parameters that give preference to either PSO or GA. The proposed hybrid algorithm can overcome the disadvantages of both PSO and genetic algorithm, and can achieve better Quality of Services performance.

We propose a mechanism that exploits the benefits of a channel approach for providing lower delay paths for real time applications, while at the same time utilizes the flexibilities of a hybrid channel approach for providing higher throughputs for non-delay sensitive applications. According to this approach, we design a protocol that can, depending on the type of traffic being routed, control the channel allocation strategy of the nodes.

GA and PSO are both population based algorithms that have proven to be successful in solving very difficult optimization problems . However, both models have strengths and weaknesses. The PSO algorithm is conceptually simple and can be implemented in a few lines of code. PSOs also have memory, whereas in a GA if an individual is not selected the information contained by that individual is lost. In PSO the collaborative group interactions enhance the search for an optimal solution, whereas GAs have trouble finding an exact solution and are best at reaching a global region. However, without a selection operator PSOs may waste resources on a poor individual that is stuck in a poor region of the search space. Comparisons between GA and PSOs have been performed by both Eberhart and Angeline and both studies suggested that a hybrid of the standard GA and PSO models would lead to a very effective search strategy.

The least-cost multicast routing with quality-of-service constraints, such as bandwidth, delay and delay jitter, is a challenging problem in high speed multimedia networks. Computing such a constrained problem is an NP-complete problem. This paper presents GA, PSO and hybrid GA-PSO based algorithms to solve this problem. In this work, routing tables are constructed for source-destination pairs in the given network, such that each path in these routing tables satisfies the bandwidth and delay constraints. Accordingly, a fitness function is proposed, to be used by the proposed algorithms, which depends only on the cost and delay jitter

constraint. This approach reduces the search space used to locate the least cost multicast tree, and ensures that it satisfies the specified constraints. Also, during the evolution process in the three algorithms, any generated individual that does not represent a multicast tree is discarded. The paper presents the results of the experiments that have been conducted to evaluate the performance of the proposed algorithms in solving the QoS multicast routing problem, and the proposed fitness function.

This paper presents three proposed algorithms for solving the QMR problem, based on GA, PSO and hybrid GA-PSO. The hybrid GA-PSO algorithm combines the GA and PSO algorithms as described in . It consists in a strong co-operation of GA and PSO, since it maintains the integration of the two techniques for the entire run. In our work, before applying any of the proposed algorithms, a routing table is constructed for each source-destination pair in the given network, such that each path in the routing table satisfies the bandwidth and delay constraints. Accordingly, a fitness function has been proposed, to be used by the proposed algorithms in evaluating the generated individuals, which depends only on the cost and the delay jitter constraint. Also, during the evolution process in the three algorithms, any generated individual that does not represent a multicast tree is discarded.

# 2. Literature Review

This project is inspired by the work done by S. Bhattacharya, V. Kalivarapu, J. Oliver, R. Zau, E. Winer, Mustapha C.E. Yagoub, Saeid Saryazdi on Improving Performance of Clustering Algorithm over Wireless Networks.

In the paper included herein, there is a review of the various problems that arise in Data Flow in wireless network. Various factors contribute to the performance of a Data Flow in wireless network like the physical obstructions, network range and distance between devices, wireless network interference signal sharing, network usage and load, local environment characteristics and so on. Modifications and adjustments to any of the above mentioned parameters may have different forms of impact on the throughput of the network.

The packages if allowed on only single path may lead to increased latency and congestion on a single route. This can be overcome by the proper implementation of Genetic Algorithm. In addition to Genetic Algorithm proper use of clustering algorithms like Particle Swarm Optimization enhance the efficiency of Data Flow in wireless network. Some methods of implementation are taken into account and their effect on the throughput is theoretically analysed.

# 3. OBJECTIVE

## Project Aim

The main aim of this project is:

- Implement various optimization algorithm in network routing
- Implement PSO algorithm in network routing to improve its efficiency
- Implement GA algorithm in network routing to improve its efficiency
- Improving efficiency of network routing
- Implement hybrid of PSO and GA to improve efficiency of network routing

# 4. METHODOLOGY

## 4.1 Particle swarm optimisation

In PSO algorithm, these are following steps to be followed-

1. Evaluate the fitness of each particle
2. Update individual and global best fitness and positions
3. Update velocity and position of each particle

These steps are repeated until some stopping condition is met.

## 4.2 Genetic Algorithm

In applying Genetic algorithm in network routing, these are steps to be followed-

1. Initialization
2. Fitness assignment
3. Selection
4. Crossover
5. Mutation

This iteration from step 2 to step 5 is continued until stopping condition becomes true. So to formalize a definition of a genetic algorithm, we can say that it is an optimization technique, which tries to find out such values of input so that we get the best output values or results. The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each iteration called a generation. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. A practical variant of the general process of constructing a new population is to allow the best organism(s) from the current generation to carry over to the next, unaltered. This strategy is known as elitist selection and guarantees that the solution quality obtained by the GA will not decrease from one generation to the next

## 4.3 Hybrid of GA and PSO

In applying hybrid of PSO and GA in network routing, these are steps to be followed-

1. Initial population of PSO is assigned by solution of GA.
2. Total no of iterations are equally shared by GA and PSO.
3. First half of iterations are run by GA and solutions are given as initial population of PSO.
4. Remaining half iteration are run by PSO.

# 5. Introduction to Python

## 5.1 Python Overview:

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

## 5.2 History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

## 5.3 Installation of Python 3 on Linux

1. Check to see if Python is already installed:

```
$ python --version
```

**Note**

If your Linux distribution came with Python, you may need to install the Python developer package in order to get the headers and libraries required to compile extensions and install the AWS CLI. Install the developer package (typically named python-dev or python-devel) using your package manager.

2. If Python 2.7 or later is not installed, install Python as follows:

```
# wget https://www.python.org/ftp/python/3.6.3/Python-3.6.3.tar.xz

# tar xJf Python-3.6.3.tar.xz

# cd Python-3.6.3

# ./configure

# make

# make install
```

**Figure 5.1 : Installing Python**

3. Open a command prompt or shell and run the following command to verify that Python installed correctly:

```
$ python3 --version
Python 3.6.2
```

Sample outputs:

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  python3.1-minimal
Suggested packages:
  python3.1-doc python3.1-profiler
The following NEW packages will be installed:
  python3.1 python3.1-minimal
0 upgraded, 2 newly installed, 0 to remove and 13 not upgraded.
Need to get 5,444 kB of archives.
After this operation, 19.9 MB of additional disk space will be used.
Do you want to continue [Y/n]? y
Get:1 http://debian.osuosl.org/debian/ squeeze/main python3.1-minimal amc
Get:2 http://debian.osuosl.org/debian/ squeeze/main python3.1 amd64 3.1.3
Fetched 5,444 kB in 27s (201 kB/s)
Selecting previously deselected package python3.1-minimal.
(Reading database ... 280220 files and directories currently installed.)
Unpacking python3.1-minimal (from .../python3.1-minimal_3.1.3-1_amd64.deb
Selecting previously deselected package python3.1.
Unpacking python3.1 (from .../python3.1_3.1.3-1_amd64.deb) ...
Processing triggers for man-db ...
Processing triggers for menu ...
Processing triggers for desktop-file-utils ...
Processing triggers for gnome-menus ...
Setting up python3.1-minimal (3.1.3-1) ...
Setting up python3.1 (3.1.3-1) ...
Processing triggers for menu ...
```

**Figure 5.2 : Sample Output**

## 5.4 Installing Pip

Pip is a tool for installing and managing Python packages.

Python is a programming language. It is quite popular and has a design philosophy that emphasizes code readability. It is widely considered to be a very easy programming language to learn and master because of its focus on readability. Python is open source, and will run a a multitude of platforms including, but not limited to: Various Linux/UNIX distributions (CentOS, Ubuntu, Fedora, Debian, etc.), Microsoft Windows, and Mac OS X.

Pre-Flight Check

- These instructions are intended specifically for installing Pip, a tool for installing and managing Python packages.
- I'll be working from a Liquid Web Core Managed Ubuntu 14.04 LTS server, and I'll be logged in as root.

Step 1a: Install Pip with apt-get, The Installation

As a matter of best practice we'll update our packages:

```
apt-get update
```

Then let's install python-pip and any required packages:

```
apt-get -y install python-pip
```

Step 1b: Install Pip with Curl and Python, The Installation

We can also use curl and python to download and install Pip.

```
curl "https://bootstrap.pypa.io/get-pip.py" -o "get-pip.py"
```

```
python get-pip.py
```

Step 2: Verify The Installation

View a list of helpful commands:

```
pip --help
```

Check the version of Pip that is installed:

```
pip –V
```

## 5.5 Installing Pygame

Step 1: Change to your home directory.

Step 2: Get Pygame source code.

```
Sudo apt -get install mercurial

Hg clone https://bitbucket.org/pygame/pygame

cd pygame
```

Step 3: Install dependencies.

```
sudo apt -get install python3-dev python3-numpy libsdl-image1.2-dev
\

libsdl-mixer1.2-dev libsdl-ttf2.0-dev libsmpeg-dev libportmidi-dev \

libavformat-dev libswscale-dev libjpeg-dev libfreetype6-dev \
```

Step 4: Build and install Pygame.

```
Python3 setup.py build

Sudo python3 setup.py install
```

```
1  import pygame
2  pygame.init()
3  def main():
4      background = pygame.Surface((640,480))
5      background.fill((0,0,0))
6      for i in range(1,320,3):
7          pygame.draw.circle(background,(0xFF,0x00,0x00),(i,i),i,1)
8          pygame.draw.circle(background,(0x00,0x00,0xFF),(640-i,i),i,1)
9          pygame.draw.circle(background,(0x00,0x00,0xFF),(i,480-i),i,1)
10         pygame.draw.circle(background,(0xFF,0x00,0x00),(640-i,480-i),i,1)
11         pygame.draw.circle(background,(0xFF,0xFF,0xFF),(320,240),i,1)
12     pygame.image.save(background,"circles.bmp")
13     print "look in current directory"
14 if __name__ == "__main__":
15     main()
```

**Figure 5.3 : Sample Implementation of Program with Pygame**

# 6. Particle Swarm Optimization

## 6.1 PSO Overview

In computer science, particle swarm optimization (PSO) is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. It solves a problem by having a population of candidate solutions, here dubbed particles, and moving these particles around in the search-space according to simple mathematical formulae over the particle's position and velocity. Each particle's movement is influenced by its local best known position, but is also guided toward the best known positions in the search-space, which are updated as better positions are found by other particles. This is expected to move the swarm toward the best solutions.

PSO is originally attributed to Kennedy, Eberhart and Shi and was first intended for simulating social behaviour, as a stylized representation of the movement of organisms in a bird flock or fish school. The algorithm was simplified and it was observed to be performing optimization. The book by Kennedy and Eberhart describes many philosophical aspects of PSO and swarm intelligence. An extensive survey of PSO applications is made by Poli. Recently, a comprehensive review on theoretical and experimental works on PSO has been published by Bonyadi and Michalewicz.

PSO is a metaheuristic as it makes few or no assumptions about the problem being optimized and can search very large spaces of candidate solutions. However, metaheuristics such as PSO do not guarantee an optimal solution is ever found. Also, PSO does not use the gradient of the problem being optimized, which means PSO does not require that the optimization problem be differentiable as is required by classic optimization methods such as gradient descent and quasi-newton methods.

## 6.2 Algorithm

A basic variant of the PSO algorithm works by having a population (called a swarm) of candidate solutions (called particles). These particles are moved around in the search-space according to a few simple formulae. The movements of the particles are guided by their own best known position in the search-space as well as the entire swarm's best known position. When improved positions are being discovered these will then come to guide the movements of the swarm. The process is repeated and by doing so it is hoped, but not guaranteed, that a satisfactory solution will eventually be discovered.

Formally, let f: $\mathbb{R}$n $\rightarrow$ $\mathbb{R}$ be the cost function which must be minimized. The function takes a candidate solution as an argument in the form of a vector of real numbers and produces a real number as output which indicates the objective function value of the given candidate solution. The gradient of f is not known. The goal is to find a solution a for which $f(a) \leq f(b)$ for all b in the search-space, which would mean a is the global minimum. Maximization can be performed by considering the function h = -f instead.

Let S be the number of particles in the swarm, each having a position xi ∈ ℝn in the search-space and a velocity vi ∈ ℝn. Let pi be the best known position of particle i and let g be the best known position of the entire swarm.

## 6.3 Pseudocode

```
for each particle i = 1, ..., S do
   Initialize the particle's position with a uniformly distributed random vector: x_i ~ U(b_lo, b_up)
   Initialize the particle's best known position to its initial position: p_i ← x_i
   if f(p_i) < f(g) then
       update the swarm's best known  position: g ← p_i
   Initialize the particle's velocity: v_i ~ U(-|b_up-b_lo|, |b_up-b_lo|)
while a termination criterion is not met do:
   for each particle i = 1, ..., S do
      for each dimension d = 1, ..., n do
         Pick random numbers: r_p, r_g ~ U(0,1)
         Update the particle's velocity: v_i,d ← ω v_i,d + φ_p r_p (p_i,d-x_i,d) + φ_g r_g (g_d-x_i,d)
      Update the particle's position: x_i ← x_i + v_i
      if f(x_i) < f(p_i) then
         Update the particle's best known position: p_i ← x_i
         if f(p_i) < f(g) then
            Update the swarm's best known position: g ← p_i
```

## 6.4 Parameter selection

The choice of PSO parameters can have a large impact on optimization performance. Selecting PSO parameters that yield good performance has therefore been the subject of much research.

The PSO parameters can also be tuned by using another overlaying optimizer, a concept known as meta-optimization or even fine-tuned during the optimization, e.g., by means of fuzzy logic.

Parameters have also been tuned for various optimization scenarios.

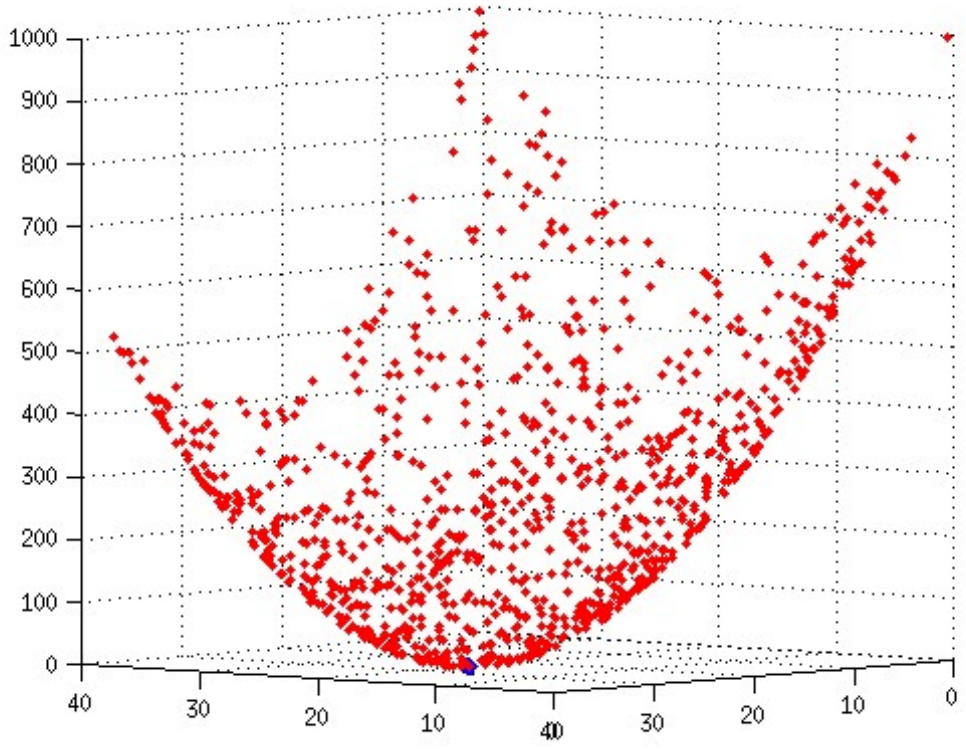



**Figure 6.1 : Performance landscape showing how a simple PSO variant performs in aggregate on several benchmark problems when varying two PSO parameters.**

## 6.5 Strategy

The goal of the algorithm is to have all the particles locate the optima in a multi-dimensional hyper-volume. This is achieved by assigning initially random positions to all particles in the space and small initial random velocities. The algorithm is executed like a simulation, advancing the position of each particle in turn based on its velocity, the best known global position in the problem space and the best position known to a particle. The objective function is sampled after each position update. Over time, through a combination of exploration and exploitation of known good positions in the search space, the particles cluster or converge together around an optima, or several optima.

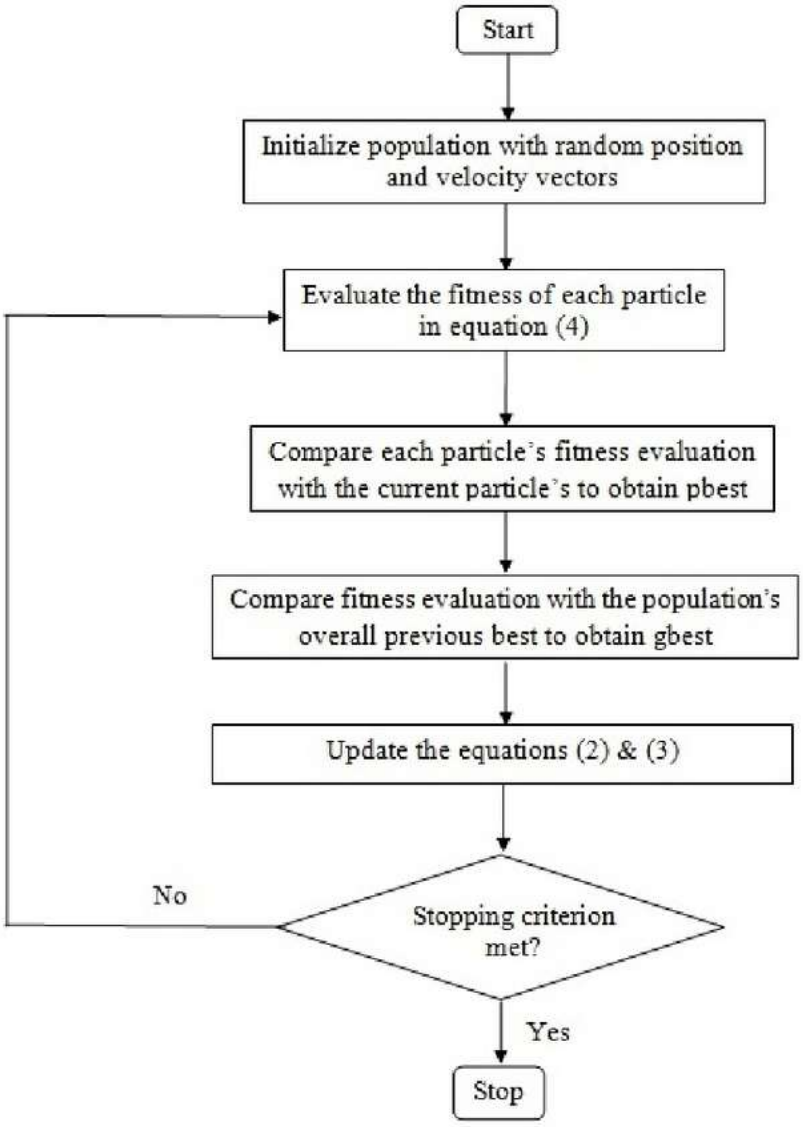



**Figure 6.2 : Flow chart of Particle Swarm Optimization Algorithm**

## 6.6 Applications of PSO and Current Trends

The first practical application of PSO was in the field of neural network training and was reported together with the algorithm itself (Kennedy and Eberhart 1995). Many more areas of application have been explored ever since, including telecommunications, control, data mining, design, combinatorial optimization, power systems, signal processing, and many others. To date, there are hundreds of publications reporting applications of particle swarm optimization algorithms. For a review, see (Poli 2008). Although PSO has been used mainly to solve unconstrained, single-objective optimization problems, PSO algorithms have been developed to solve constrained problems, multi-objective optimization problems, problems with dynamically changing landscapes, and to find multiple solutions.

A number of research directions are currently pursued, including:

- Theoretical aspects
- Matching algorithms (or algorithmic components) to problems
- Application to more and/or different class of problems (e.g., multi-objective)
- Parameter selection
- Comparisons between PSO variants and other algorithms
- New variants

# 7. GENETIC ALGORITHM

## 7.1 Genetic algorithm Overview

The genetic algorithm is a method for solving both constrained and unconstrained optimization problems that is based on natural selection, the process that drives biological evolution. The genetic algorithm repeatedly modifies a population of individual solutions. At each step, the genetic algorithm selects individuals at random from the current population to be parents and uses them to produce the children for the next generation. Over successive generations, the population "evolves" toward an optimal solution. You can apply the genetic algorithm to solve a variety of optimization problems that are not well suited for standard optimization algorithms, including problems in which the objective function is discontinuous, no differentiable, stochastic, or highly nonlinear. The genetic algorithm can address problems of mixed integer programming, where some components are restricted to be integer-valued.

The genetic algorithm uses three main types of rules at each step to create the next generation from the current population:

- Selection rules select the individuals, called parents that contribute to the population at the next generation.
- Crossover rules combine two parents to form children for the next generation.
- Mutation rules apply random changes to individual parents to form children.

The genetic algorithm differs from a classical, derivative-based, optimization algorithm in two main ways, as summarized in the following table.

| **Classical Algorithm** | **Genetic Algorithm** |
|---|---|
| Generates a single point at each iteration. The sequence of points approaches an optimal solution. | Generates a population of points at each iteration. The best point in the population approaches an optimal solution. |
| Selects the next point in the sequence by a deterministic computation. | Selects the next population by computation which uses random number generators. |

Standard genetic algorithms (GA) have revolutionized computing technology representing solid and adaptive models that are based on the idea of natural selection. GA are designed to emulate the processes observed in natural evolution. GA are used as search and random optimization techniques but are structured algorithms that demonstrate their efficiency by operating on stored information to create new offspring with superior performance.

## 7.2 FLOW CHART

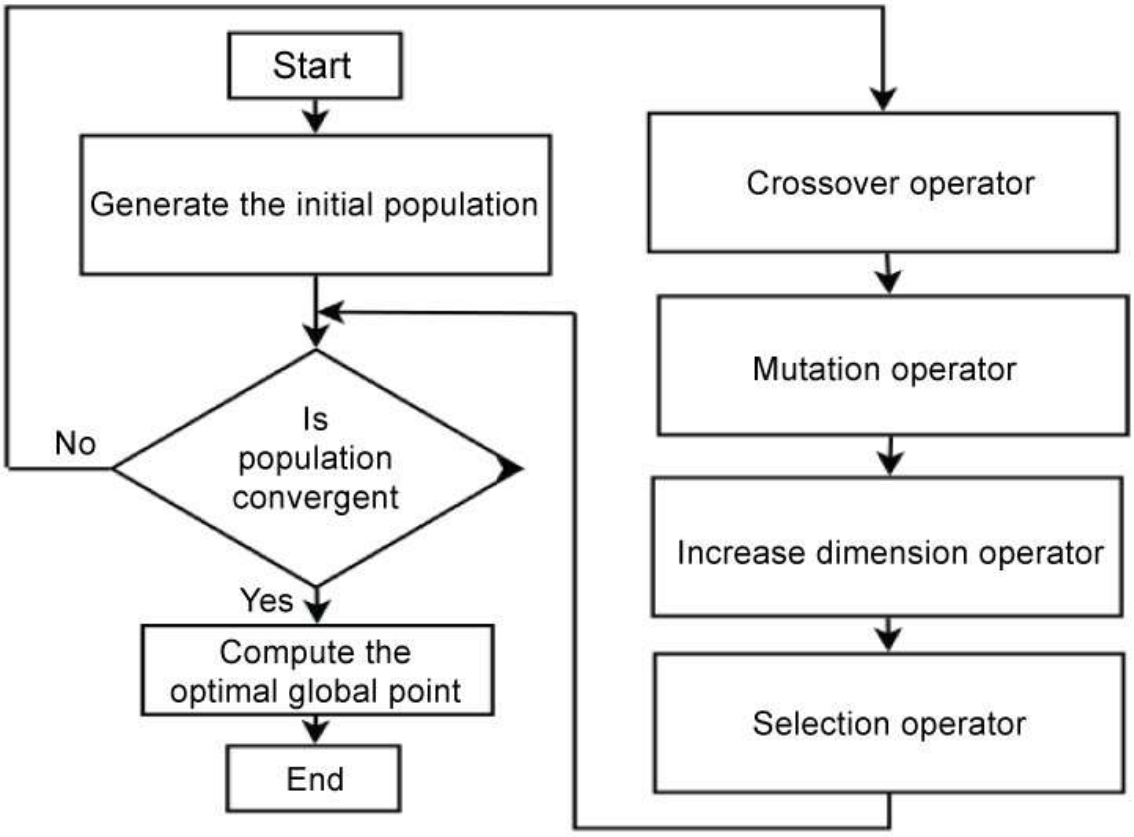



**Figure 7.1: Flow chart of the improved genetic algorithm**

A typical genetic algorithm requires:

- Genetic representation of the solution domain.
- Fitness function to evaluate the solution domain.

A standard representation of each candidate solution is as an array of bits. Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations. Variable length representations may also be used, but crossover implementation is more complex in this case. Tree-like representations are explored in genetic programming and graph-form representations are explored in evolutionary programming; a mix of both linear chromosomes and trees is explored in gene expression programming.

Once the genetic representation and the fitness function are defined, a GA proceeds to initialize a population of solutions and then to improve it through repetitive application of the mutation, crossover, inversion and selection operators.

## 7.3 OPERATORS

### 1. Selection Operator

- Key idea: give preference to better individuals, allowing them to pass on their genes to the next generation.
- The goodness of each individual depends on its fitness.
- Fitness may be determined by an objective function or by a subjective judgement.

### 2. Crossover Operator

- Prime distinguished factor of GA from other optimization techniques.
- Two individuals are chosen from the population using the selection operator.
- A crossover site along the bit strings is randomly chosen.
- The values of the two strings are exchanged up to this point.
- If S1=000000 and s2=111111 and the crossover point is 2 then S1'=110000 and s2'=001111.
- The two new offspring created from this mating are put into the next generation of the population.
- By recombining portions of good individuals, this process is likely to create even better individuals.

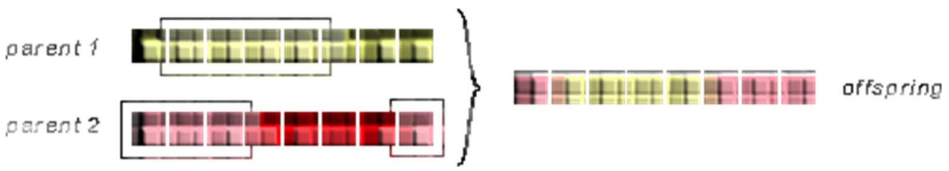



**Figure 7.2: Crossover**

### 3. Mutation Operator

- With some low probability, a portion of the new individuals will have some of their bits flipped.
- Its purpose is to maintain diversity within the population and inhibit premature convergence.
- Mutation alone induces a random walk through the search space
- Mutation and selection (without crossover) create a parallel, noise-tolerant, hill-climbing algorithms

### 4. Effects of Genetic Operators

- Using selection alone will tend to fill the population with copies of the best individual from the population
- Using selection and crossover operators will tend to cause the algorithms to converge on a good but sub-optimal solution
- Using mutation alone induces a random walk through the search space.
- Using selection and mutation creates a parallel, noise-tolerant, hill climbing algorithm

## 7.4 Pseudocode of Genetic Algorithm

1) Randomly initialize population's p

2) Determine fitness of population

3) Until convergence repeat:

   a) Select parents from population

   b) Crossover and generate new population

   c) Perform mutation on new population

   d) Calculate fitness for new population

## 7.5 Applications in Real World

- Engineering Design
- Traffic and Shipment Routing (Travelling Salesman Problem)
- Robotics

# 8. HYBRID OF GA-PSO

## 8.1 Overview

There are few drawbacks of using GA and PSO algorithm alone so we make the hybrid of GA and PSO, It can be done in 2 ways. The first algorithm is called GA-PSO, while the second algorithm is called PSO-GA. In PSO-GA algorithm, PSO generates an initial population for GA, while in GA-PSO algorithm, the initial population is generated by using GA for PSO. The final results showed that the PSOGA hybrid algorithm outperforms the GA-PSO version as well as simple PSO and GA versions.

A Hybrid GA-PSO algorithm combines the strengths of both algorithms to address the workflow scheduling problem. The efficiency of the proposed algorithm is evaluated against other algorithms to prove its effectiveness to find an optimal path between a starting and ending point in a grid environment. Moreover, the Hybrid GA-PSO algorithm may not get trapped in the local optimal solution, because of the use of the GA mutation operator that enhances the accuracy of the solutions.

In summary, the GA-based algorithms provide better results than other algorithms when the number of iterations is large. However, increasing the number of iterations means that the GA algorithm will consume more time to reach the optimal solution. On the other hand, the PSO-based algorithms provide better results than the other algorithms and in less time. However, the results may not be accurate due to the fast convergence of the PSO-based algorithms to the solution, which may cause being stuck in the local optimal solution. Therefore, the proposed Hybrid GA-PSO algorithm is distinguished by the characteristics of the GA and the PSO algorithms. The Hybrid GA-PSO algorithm is expected to work faster with different sizes of workflow applications compared to other algorithms with the same objectives. Moreover, the Hybrid GA-PSO algorithm may not get trapped in the local optimal solution, because of the use of the GA mutation operator that enhances the accuracy of the solutions.

The main steps of the GA-PSO algorithm are shown in Figure below. The GA-PSO algorithm starts with generating a random population and defines a specific number of iterations as a parameter to the algorithm. The population represents several solutions to the workflow tasks problem and each solution is a distribution of the whole workflow tasks over the available VMs. The initialized population is passed through the GA algorithm with the first half of the defined iterations; that is, if the number of the iterations is (), then the GA algorithm will be repeated () times. The reason behind using () iteration is to reduce the complexity of the proposed algorithm, as the performance of the GA algorithm depends mainly on the method used to encode solutions into chromosomes and particles and what the fitness function is measuring, as well as the size of the population, that is, the number of iterations. These parameter values can be adjusted after evaluating the algorithm's performance on a few trial runs. Through experiments, the GA-PSO algorithm's performance was the best, when the defined number of iterations is divided equally between the GA and PSO algorithms. This also agrees with the concept of the divide and conquer that divides one problem into two subproblems to produce a complexity equal to , where  is the divide and conquer time. The solution for such equation depends on and the complexity is if (by master theory).

Moreover, it is also known that both the GA and the PSO require many function evaluations because each needs to evaluate the objective of every population member in the current sample. Therefore, decreasing the population size in a GA or PSO (i.e., decreasing the number of iterations) is a common practice to avoid degrading the GA or PSO performance in terms of the accurate results and reduction rate.

The Hybrid GA-PSO algorithm is initialized to a specific number of iterations. A solution is initiated randomly at the first iteration. After the first iteration, a sequence of new populations are created and recursively enhanced using the previous solutions to form a set of suggested solutions as illustrated in Figure 3. The population in the GA algorithm is called chromosome. The length of the chromosomes is equal to the number of the workflow tasks, and the genes of each chromosome represent the different VMs. The randomly generated chromosomes represent the input to the proposed GA-PSO algorithm. The GA algorithm represents the first part of the proposed GA-PSO algorithm which will be used to generate different solutions to the workflow scheduling problem.

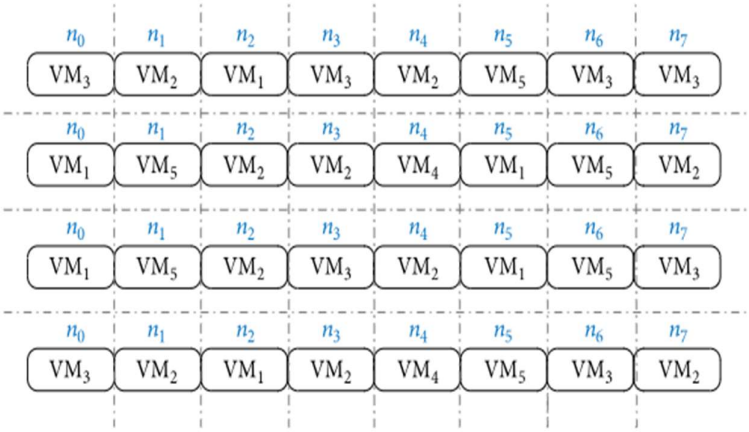



**Figure 8.1 : An example of a randomly initiated population.**

# 9. Implementation of a hybrid of GA-PSO algorithm

## 9.1 Hybrid of GA-PSO algorithm

To circumvent the problems in conventional GA and PSO algorithms, the proposed hybrid GA-PSO algorithm has a standard PSO hybridized with GA operators, including crossover and mutation operations to generate new population. In the following, these two algorithms for global path planning are briefly described and the hybrid GA-PSO approach is then presented.

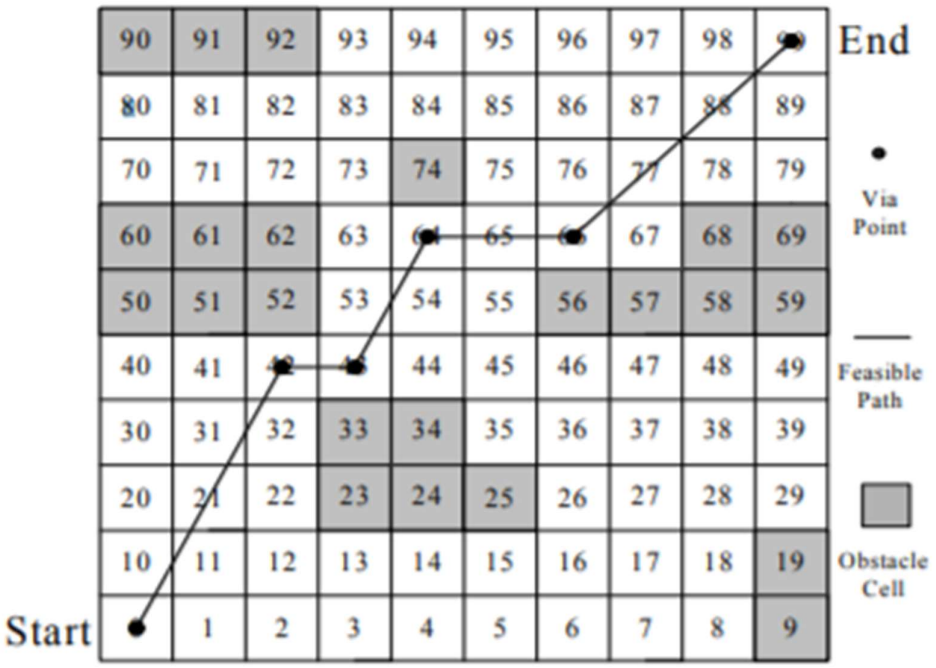



**Figure 9.1 : A grid-based environment with obstacles and the planned collision-free path.**

## 9.2 GA for global path planning

GA is an adaptive and heuristic search method. Its main idea is to construct a fitness according to the objective function to evaluate all chromosomes in a population. The optimal collision-free path is evolved by the genetic operators. As shown in Fig. 1, a collision-free path (chromosome) contains a set of the grid points (genes of a chromosome), including the start point, via-points and the end point. The length of a chromosome was variable, ranging from two to maximum length max M. A path can be either feasible (collision-free) or infeasible by evaluating the fitness function expressed by

$$F = \sum_{i=1}^{M_{max}} (d_i + \alpha_i T)$$

Here $M_{max}$ is the number of line segments of a path, $d_i$ is the distance of the near two nodes forming the line segment, T is a constant and

$$\alpha_i = \begin{cases} 0 & \text{if the ith line segment is feasible} \\ \sum_{K=1}^{N} k & \text{if the line segment intersects obstacle(s)} \end{cases}$$

where N is the number of obstacles that the line segment intersects.

## 9.3 PSO for global path planning

The PSO algorithm is a population based optimization method.

This algorithm can be applied to solve for the optimal problems with the multimodal function $f(x) = f\ (x_1, x_2, x_3, \dots\dots, x_n)$ by using a population of particles. The fitness of each particle is given by the fitness function f(x). Each particle represents a feasible path solution of the global path planning problem.

In PSO algorithm, $x_i$(t) denote the position of particle I in the search space at discrete time steps t. The position of the particle is changed by adding a velocity, $v_i$(t) to the current position, given by

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

with an initial position $x_i(0)$. The velocity vector drives the PSO optimization process and reflects both the experimental knowledge of the particle and socially exchanged information from the particle's neighbourhood. The best position reached by the single particle (gbest) is responsible for the former type of attraction.

On the other hand, the best location found by the rest of the swarm (pbest) is the second factor, which indicates the influence of the swarm on the single particle. Each particle i moves around the search space, and renew its velocity using its past experience and the population's experience given by

$$v_{ij}(t+1) = w v_{ij}(t) + c_1 \varphi_1\ (p_{best} - x_{ij}(t)) + c_2 \varphi_2\ (g_{best} - x_{ij}(t))$$

On the other hand, the best location found by the rest of the swarm

(pbest) is the second factor, which indicates the influence of the swarm on the single particle. Each particle i moves around the search space, and renew its velocity using its past experience and the population's experience given by

$v_{ij}$(t+1) = $wv_{ij}$(t) + $c_1\varphi_1$ ($p_{best}$ - $x_{ij}(t)$) + $c_2\varphi_2$ ($g_{best}$ - $x_{ij}(t)$)

where $v_{ij}$(t) is the velocity of particle i in dimension j at time step t, $x_{ij}(t)$ is the position of particle i in dimension j at time step t, $c_1$ and $c_2$ are positive acceleration constants used to respectively scale the contribution of the cognitive and social components. $\varphi_1$ and $\varphi_2$ are uniform random numbers. These two random values introduce a stochastic element to the PSO algorithm. w is called inertia weight which control the momentum of the particle by weighting the contribution of the previous velocity.

## 9.4 Hybrid of GA-PSO algorithm for global path planning

This subsection aims to employ the GA-PSO hybrid algorithm to design an optimal path planner for mobile robots. Each PSO particle is composed of the point sequence, including starting point, via point and ending point. The optimal feasible path will be evolved by the efficient GA-PSO hybrid algorithm described by the following steps.

**Step 1:**

Initialize the swarm size, neighbourhood size, search space, acceleration coefficients, and number of iterations.

**Step 2:**

Randomly generates particles and initialize the position and velocity of the particles.

**Step 3:**

Calculate the fitness value for all the particles.

**Step 4:**

GA crossover and mutation operations are executed after all particles have constructed a solution.

**Step 5:**

(1) Search of personal best population.

(2) Search of global best population.

**Step 6:**

(1) Update the velocity using (4).

(2) Update the position using (3).

**Step 7:**

Check the stop criterion. If the stop criterion is not matched, go to Step 3 and set t = t+1, otherwise, output the optimal path and its corresponding feasible path and stop the algorithm.

## 9.5 FPGA implementation

This section presents the FPGA implementation of the proposed GA-PSO using hardware/software co-design technique. Fig. 2 depicts the architecture of the FPGA implementation for the proposed GA-PSO. The user IP cores (custom logic) for this GA-PSO operators have been developed by VHDL (VHSIC Hardware Description Language), including random number generator (RNG) module, particle updating module, GA operations, and fitness module. Note that the soft-core processor Nios II is embedded as path smoother to smooth the discontinuous path from the GA-PSO path planner. The software-based path smoother and hardware-based custom logics for the GA-PSO are connected to the system interconnect fabric via Avalon memory-mapped interface in one FPGA chip.

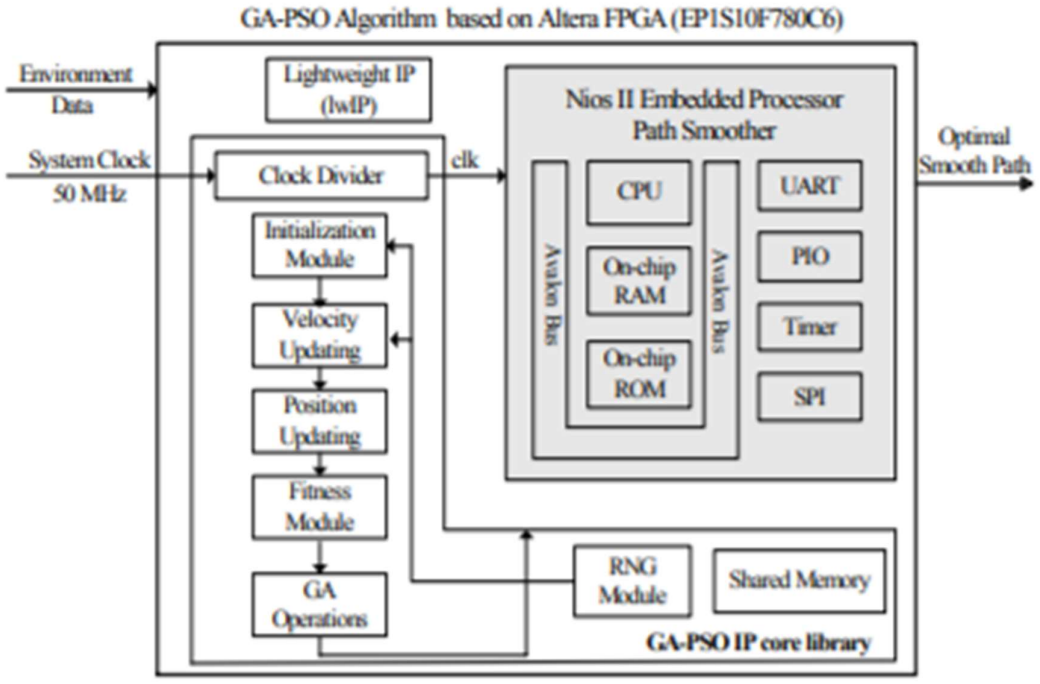



**Figure 9.2 : FPGA implementation of the proposed hybrid GA-PSO algorithm for global path planning.**

In VHDL GA-PSO processor, the particle position and velocity are continuously updated according to the best p and best g information, thereby moving the particles toward the optimal solution. The update process can be implemented by using a simple adder and a multiplier to update the position and velocity of the particles in the proposed GA-PSO algorithm. This high-performance hardware module is implemented by VHDL to efficiently update the particles. The two random numbers, $\varphi_1$ and $\varphi_2$ are generated from the linear feedback shift register (LSFR) RNG module. The initial particle data stored in the swarm

memory is also obtained from the LSFR-based RNG module. As mentioned above, the fitness function can be defined for its corresponding optimal problem. The purpose of this fitness evaluation module is to evaluate the particles after position and velocity update and find out the $p_{best}$ and $g_{best}$ in the GA-PSO algorithm. By taking the advantage of high performance of hardware implementation, the fitness evaluation module has been efficiently implemented by VHDL, thus significantly improving the execution performance. After the evaluation process, the $p_{best}$ and $g_{best}$ for the PSO algorithm will be stored in the swarm memory. The FPGA-based GA-PSO for solving the global path planning problem with the fitness function in (1) is described by the following steps.

**Step 1**:

Use the LSFR-based RNG to randomly generate the initial particles from the start to the goal.

**Step 2**:

In the VHDL-based GA-PSO, execute the procedure of particle updating and also check whether new particles are acceptable. If the new particles do not satisfy the requirement, repeat this procedure until acceptable particles are obtained.

**Step 3**:

Evaluate the fitness values of the particles.

**Step 4**:

Check the stopping conditions. If the stopping criterion is not met, go to Step 2.

**Step 5**:

Smooth the discontinuous path using B-spline.

## 9.6 Python code for hybrid of GA-PSO

```
CROSSOVER_PROB = 0.7

def instantantiate(pop):
    """Takes the number of particles and returns a swarm of particles"""
    print "Initializing swarm..."
    swarm = []
    for i in range(0, pop):
        swarm.append(Particle())
    print "Swarm initialized."
    return swarm

def instantantiate_map(obstacles):
    """ Takes the positions of the obstacles and returns a new map. A
    free position is denoted with a 0, an obstacle is denoted with 1
    """
    new_map = []
    for i in range(0, 100):
        if i in obstacles:
            new_map.append(1)
        else:
            new_map.append(0)
    return new_map


def selection(swarm):
    """Performs a selection of particles for the next generation.
    Particles with higher fitness hava a higher probability of being
    selected
    """
    #The following algorithm of selection found in the following page
    #http://arxiv.org/pdf/1308.4675.pdf
    #each step is marked below

    #Probability of each chromosome to be selected
    fit_r = []
    for particle in swarm:
        fit_r.append(1/particle.fit)

    #Probability over total probability
    fit_r_sum = sum(fit_r)
    selection_probability = []
```

```
    for relative_fit in fit_r:
        selection_probability.append(relative_fit/fit_r_sum)

    #Cumulative probability
    cumulative_probability = []
    the_sum = 0
    for a in selection_probability:
        the_sum += a
        cumulative_probability.append(the_sum)

    #For the new generation, we compare a random number between 0 and 1
    #and we select the particle that has the next greater cumulative
    #probability
    probability = random()
    for i in range(0, len(cumulative_probability)):
        if probability <= cumulative_probability[i]:
            new_kid = swarm[i]
            break
    #Make new copy
    a_new_kid = Particle()
    a_new_kid.v = new_kid.v[:]
    a_new_kid.x = new_kid.x[:]
    a_new_kid.p_best = new_kid.p_best[:]
    return a_new_kid

def find_best_fit(swarm):
    """Returns the particle with the best fit in the swarm
    in order to perform elitism.
    """
    fitt = []
    for particle in swarm:
        fitt.append(particle.fit)
    minimum = min(fitt)
    index_of_min = fitt.index(minimum)
    return swarm[index_of_min]


def remove_duplicates(path):
    """Takes a path and returns it with duplicate nodes removed."""
    final_path = []
    final_path.append(0)
    for i in range(1, len(path) - 1):
        if path[i] != path[i - 1]:
            final_path.append(path[i])
    if final_path[len(final_path) - 1] != 99:
        final_path.append(99)
```

```
    return final_path

def algorithm(pop, generations, OBSTACLES):
    """Runs the main pso - ga algorithm as described on the paper"""
    swarm = instantantiate(pop)
    print "Swarm population: ", pop
    print "Max generations: ", generations
    print "-----------------------------------"
    my_map = instantantiate_map(OBSTACLES)
    Particle.THE_MAP = my_map
    best_history = [] #keeps track of best history to terminate program

    print "Searching..."
    for i in range(0, generations):
        #---------------Step 3-------------------
        #print "--------Start----------"
        for particle in swarm:
            particle.fit = particle.calculate_fit(particle.x)
            #print particle.fit
        #print"------------------------"

        #----------------Step 4------------------
        new_gen = []
        the_best = find_best_fit(swarm)
        elite = Particle()
        elite.v = the_best.v[:]
        elite.x = the_best.x[:]
        elite.p_best = the_best.p_best[:]
        new_gen.append(elite)
        for j in range(1, len(swarm)):
            #Decide for crossover
            dont_crossover = random()
            if dont_crossover < CROSSOVER_PROB:
                parent1 = selection(swarm)
                parent2 = selection(swarm)
                a_new_kid = parent1.crossover(parent2)
            else:
               a_new_kid = selection(swarm)
            a_new_kid.mutate()
            new_gen.append(a_new_kid)
        swarm = new_gen
        #print "------After mutation-------"
        for particle in swarm:
            particle.fit = particle.calculate_fit(particle.x)
            #print particle.fit
        #print "--------------------------"
```

```
#----------------Step 5------------------
#Find p_best of each particle
for particle in swarm:
    if particle.fit < particle.calculate_fit(particle.p_best):
        particle.p_best = particle.x[:]

#Find g_best
fitt = []
for particle in swarm:
    fitt.append(particle.calculate_fit(particle.p_best))
minimum = min(fitt)
if minimum < particle.calculate_fit(Particle.g_best):
    position = fitt.index(minimum)
    Particle.g_best = swarm[position].x[:]

fitt = []
for particle in swarm:
    fitt.append(particle.fit)
#print fit
minimum = min(fitt)
position = fitt.index(minimum)
best_x = swarm[position].x[:]
best_history.append(minimum)
print "Generation number: %d, best fit: %f" % (i, minimum)
#---------Uncoment the following six lines for faster results----------
if abs(best_history[i] - best_history[i - 1]) < 0.00000000001 and i > 0:
   same += 1
else:
    same = 0
if same >= 20:
    Break
#print "-----After step 5-------"
for particle in swarm:
    particle.fit = particle.calculate_fit(particle.x)
    #print particle.fit
#print "------------------------"
#----------------Step 6------------------
#print "--------Update position-------"
#print "position: ", position
for i in range(len(swarm)):
    if i != position:
        swarm[i].update_velocity()
    else:
        Pass
```

```
    for i in range(len(swarm)):
        if i != position:
            swarm[i].update_position()
        else:
            Pass

    for particle in swarm:
        particle.fit = particle.calculate_fit(particle.x)
        #print particle.fit
    #print "------------------------------"
    #raw_input()


#Check if the solution is valid
line_segments = []
for i in range(0, len(best_x) - 1):
    line_segments.append(best_x[i:i+2])
obstacle_factor = []
obstacles = Particle.obstacles_per_segment(line_segments)
valid = True
for nmbr_of_obstacles in obstacles:
    if nmbr_of_obstacles != 0:
        valid = False
        Break
if valid:
    print "Path found, press Enter to view."
    sol = remove_duplicates(best_x)
    raw_input()
    return sol
else:
    print "A valid path could not be found. Press Enter."
    raw_input()
    return False
```

# 10. Code implementation and execution

## 10.1 Instructions

- If not already installed, install the [PyGame Library] (http://www.pygame.org/news.html)
- Place all the source files in the same directory.
- If under Windows OS:

  Open __main__.py with IDLE and run it

  If under UNIX-like system:

  - Open a terminal
  - Move to the proper directory
  - Run the program with python __main__.py

## Steps to run the program on Linux

1. Traverser into the directory where code is saved. Run the file _main_.py by running the command

   python _main_.py

2. Select the dead nodes in the GUI and press space after you are done with selection.
3. To select dead nodes, simply click on the node present once and it will turn black.
4. Enter the swarm populations.
5. Enter max number of generations.
6. Hit enter to run the program.
7. Please wait for the program to complete its iteration.
8. Enter space to view the path found.

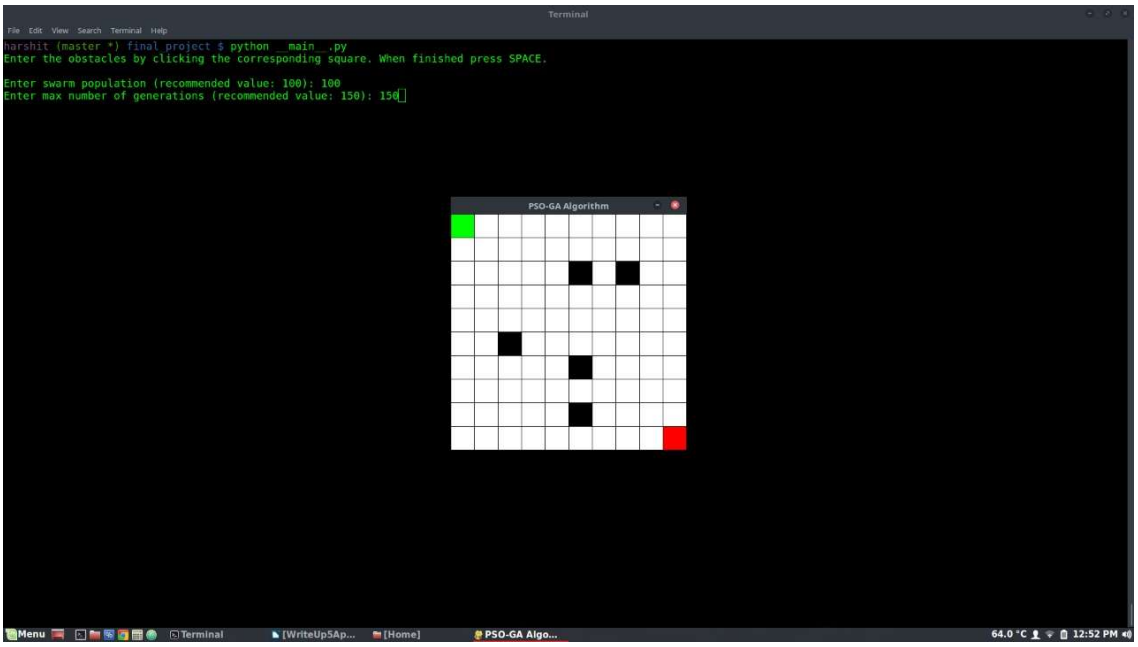



**Figure 10.1 : Runing program by hitting enter after selecting dead nodes**

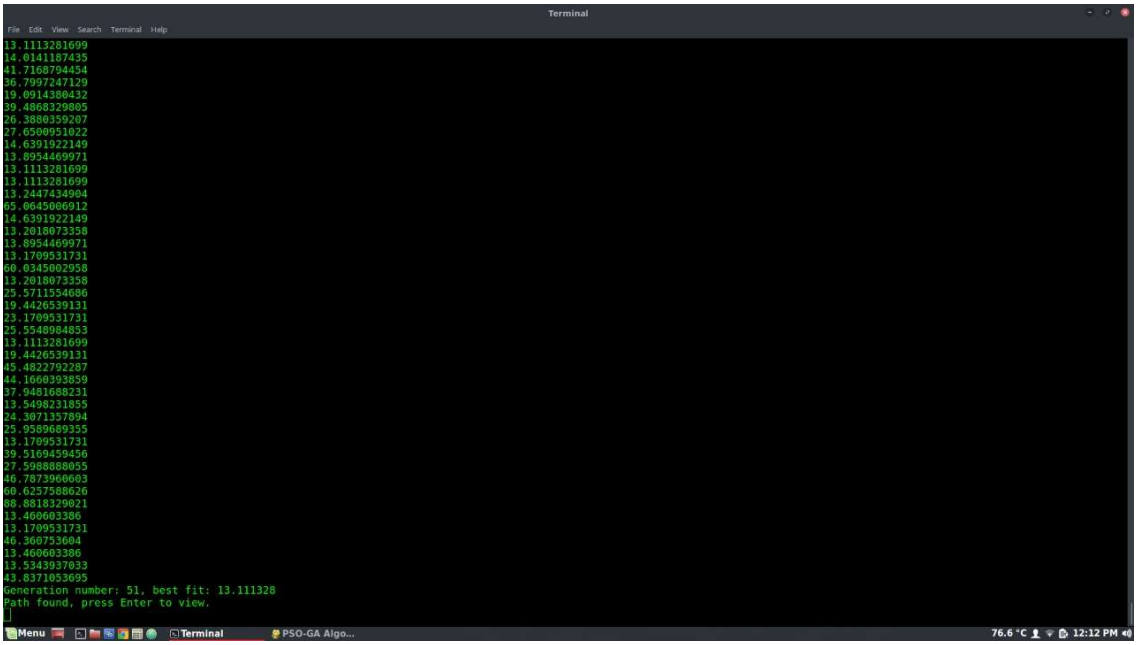



**Figure 10.2 : Please wait for the iterations to complete.**

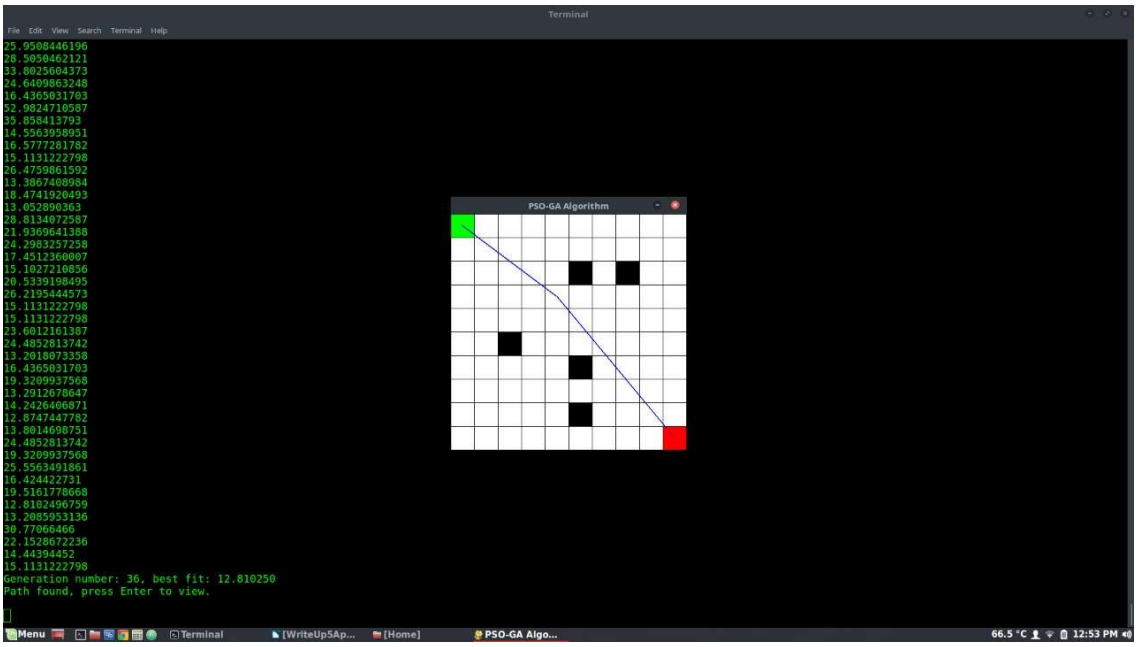



**Figure 10.3 : Final path**

**Code base can be found at this URL**

**https://github.com/stormindia/GA-PSO-hybrid**

# 11. RESULT

## 11.1 Experimental results

The following experiments are conducted to illustrate the feasibility and merit of the proposed hybrid GA-PSO path planner together with the B-spline path smoother in different environments. In addition, the resultant fitness values are presented for illustration of effectiveness of the proposed GA-PSO path planner. These simulations are performed with the parameters: w = 0.8 and $c_1 = c_2 = 1.5$. The crossover probability is 0.7, and the mutation probability is 0.1.

### A. Path planning in a complex environment

The first experiment was conducted to present the path planning and smoothing results for mobile robots in a complex environment. Fig. 3 depicts the discontinuous feasible path from the proposed hybrid GA-PSO algorithm and the smooth collision-free path using the B-spline modelling. Fig. 4 presents the fitness value of the GA-PSO for solving the global path planning problem. In order to exploit the merit of the proposed hybrid GA-PSO, the conventional GA used to solve the same problem is also presented in Fig. 4. As shown in Fig. 4, the hybrid GA-PSO has better convergence behaviour than GA does.

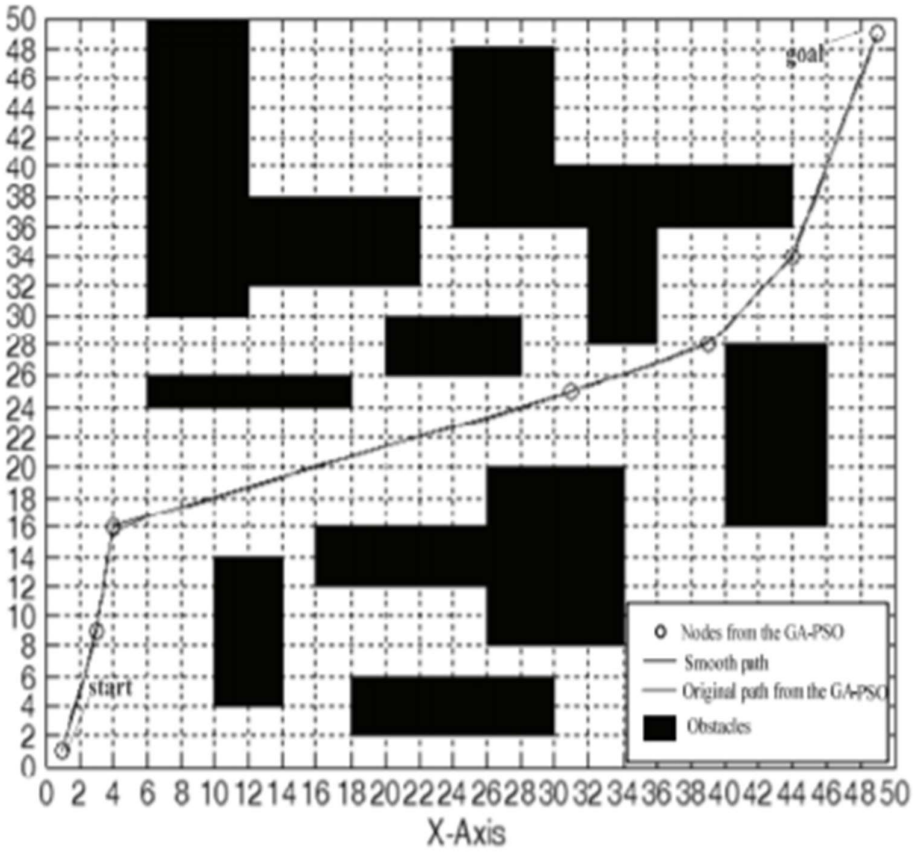



**Figure 11.1 : Experimental results of the proposed GA-PSO and GA path planning in a complex environment**

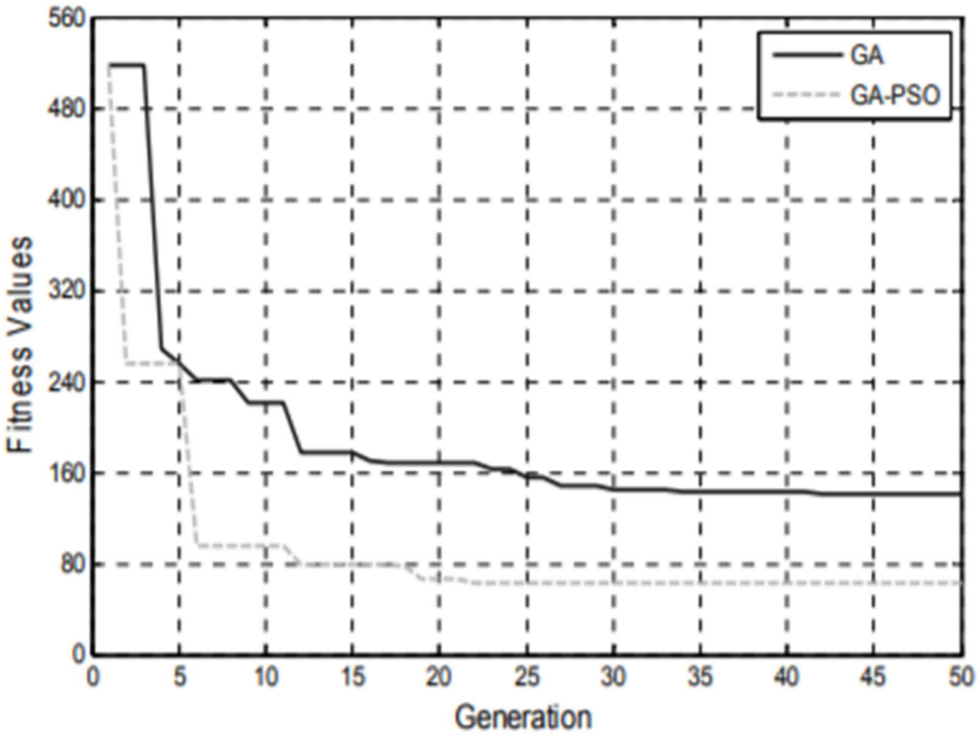



**Figure 11.2 : Fitness values of the GA-PSO**

## B. Path planning in a double U-shape environment

Fig. 5 presents the path planning result for mobile robots in a double U-shape environment. This result has shown that the proposed GA-PSO is capable of evolving optimal collision-free path in this environment. Fig. 6 depicts fitness values for solving the global path planning in a double U-shape environment using the proposed GA-PSO and conventional GA. As shown in Fig. 6, the proposed GAPSO algorithm converges to the optimal collision-free path with better fitness value, namely that the proposed GA-PSO outperforms conventional GA to solve the optimal problem.

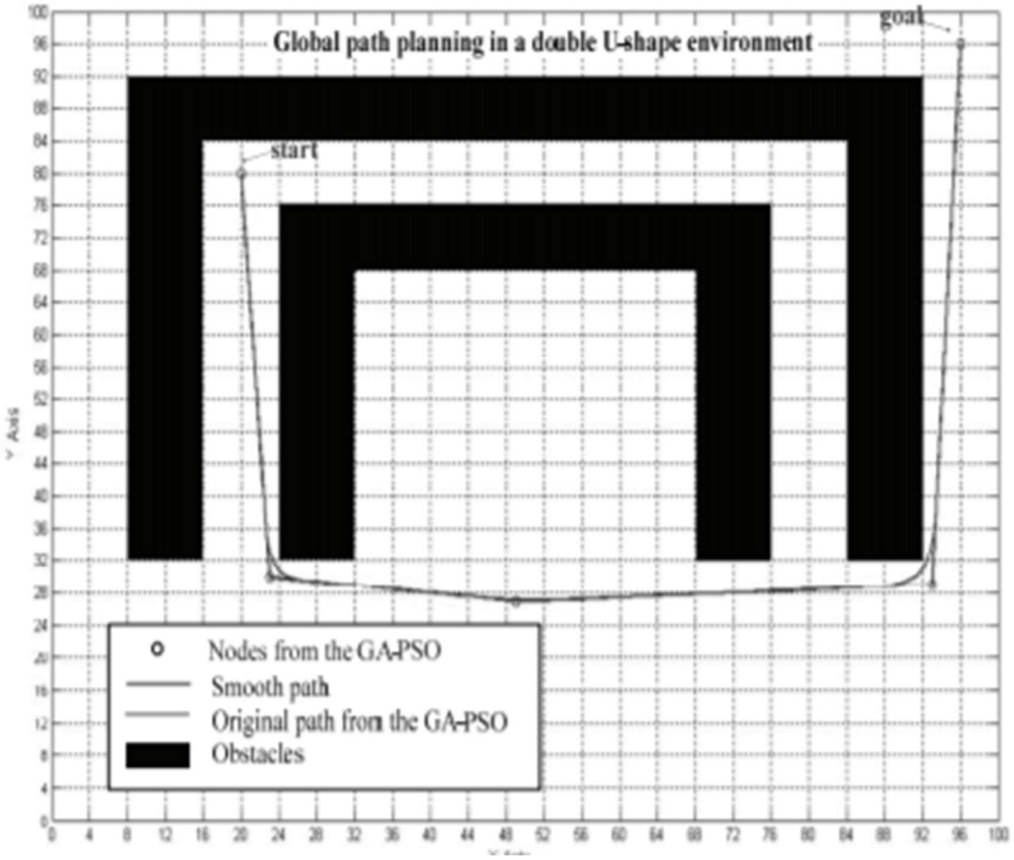



**Figure 11.3 : Experimental results of the proposed GA-PSO and GA in a double U-shape environment**

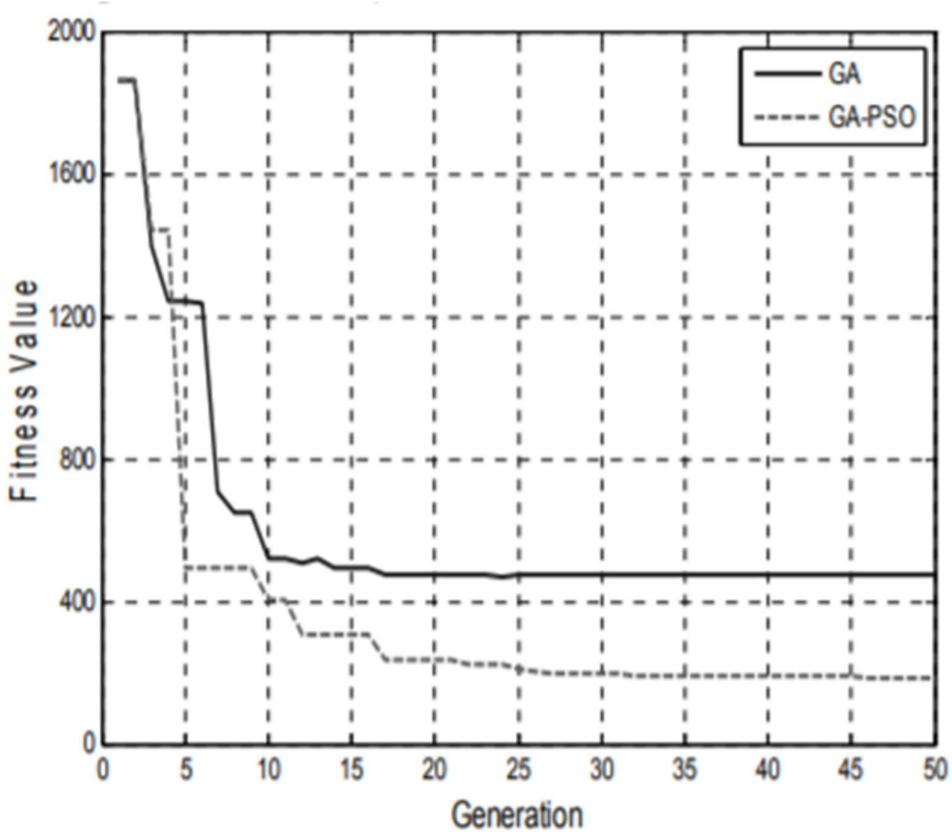



**Figure 11.4 : Fitness values of the GA-PSO and GA in a double U-shape environment**

# 12. Conclusion

This project presents a hybrid metaheuristic GA -PSO algorithm to find an optimal path between a starting and ending point in a grid environment. The proposed GA-PSO has been efficiently resolved for the global path planning problems in a structure environment with obstacles. In order to smooth the planned paths from the proposed GA-PSO planner, the path smoother has been proposed using the B-spline smoothing technique. Through experimental results, the proposed GAPSO has been shown to find the feasible paths in different environments. These experimental results clearly indicate that the proposed GA-PSO outperforms conventional GAs.

# 13. Summary of work

So far we have taken network grids as our primary concern to improve network routing using hybrid of genetic algorithms and particle swarm optimizations and we tried to improve our model as compared to GA.

# 14. Future work

- Allow the user to select the starting and goal position
- Allow the user to switch between other searching algorithms

# 15. REFERENCES

[1] Global Path Planning for Autonomous Robot Navigation Using Hybrid Metaheuristic GA-PSO Algorithm by Hsu-Chih Huang and Ching-Chih Tsai.
[2] PSO-HC: Particle Swarm Optimization Protocol for Hierarchical Clustering in Wireless Sensor Networks by Mustapha C.E. Yagoub, Senior Member, IEEE Riham S. Elhabyan
[3]GSA: A Gravitational Search Algorithm by Esmat Rashedi, Hossein Nezamabadi-pour *,Saeid Saryazdi
[4] https://en.wikipedia.org/wiki/Particle_swarm_optimization
[5] https://en.wikipedia.org/wiki/Genetic_algorithm
[6] https://www.youtube.com/watch?v=OQ3T575sbI8
[7] http://ieeexplore.ieee.org/document/5451746/