Int J Syst Assur Eng Manag
DOI 10.1007/s13198-017-0663-z

ORIGINAL ARTICLE

Three-dimensional path planning for unmanned aerial vehicles using glowworm swarm optimization algorithm

Prashant Pandey[1] · **Anupam Shukla**[1] · **Ritu Tiwari**[1]

Received: 28 February 2017 / Revised: 20 August 2017

Abstract Robot path planning is a task to determine the most viable path between a source and destination while preventing collisions in the underlying environment. This task has always been characterized as a high dimensional optimization problem and is considered NP-Hard. There have been several algorithms proposed which give solutions to path planning problem in deterministic and non-deterministic ways. The problem, however, is open to new algorithms that have potential to obtain better quality solutions with less time complexity. The paper presents a new approach to solving the 3-dimensional path planning problem for a flying vehicle whose task is to generate a viable trajectory for a source point to the destination point keeping a safe distance from the obstacles present in the way. A new algorithm based on discrete glowworm swarm optimization algorithm is applied to the problem. The modified algorithm is then compared with Dijkstra and meta-heuristic algorithms like PSO, IBA and BBO algorithm and their performance is compared to the path optimization problem.

✉ Prashant Pandey
94prashant.pandey@gmail.com

Anupam Shukla
dranupamshukla@gmail.com

Ritu Tiwari
tiwariritu2@gmail.com

[1] Robotics and Intelligent System Design Lab, Indian Institute of Information Technology and Management, Gwalior, India

Keywords Path planning · Glowworm swarm optimization · Meta-heuristics · Swarm intelligence · Robotics

1 Introduction

The modern robots are made adept in performing several complex tasks by themselves which usually required human support. Some of the recent developments like the Mars Spirit Robot and Self-Driving Cars (Levinson et al. 2011) are an excellent example of the today's technological advancements in the fields of mobile robots. The robots of today are extremely intelligent and are capable of finding their own way of completing the task given to them. The most crucial tasks of mobile robots are to move from one place to another safely without damaging themselves or their corresponding environment.

The task of finding a safe path from a source and a destination for a mobile robot is known as robot path planning. Path planning is done by a robot by devising a strategy for their movement of the robot while avoiding the obstacles in the environment and to reach its goal safely. Path planning is required for all types of robots in different terrains like air, water and land. Land robots are usually confined to 2D space whereas air and water robots have the option for a movement in three dimensions. Three-dimensional path planning comprises of generating an optimal path between a source point and a destination point while avoiding obstacles present in the environment where the robot is free to change is position in any direction.

Three-dimensional path planning is an essential part for every unmanned aerial vehicle (Bortoff 2000). The Unmanned Aerial Vehicles are fitted with complex

Published online: 15 September 2017

programmed path planners that manage their behavior by setting an optimal path for their flight trajectory. The path planner is implemented in the robots and along with other features like task assignment, coordination, etc., a vehicle moves autonomously in its environment. The techniques used for generating the path are of different types, involving the generation of the full path before vehicle movement starts or the vehicle generating the path on the go. The former type of path planning is referred as offline path planning while the latter as online path planning.

Unmanned Aerial Vehicles operate in air for completing complex tasks like surveillance, cargo drops, etc. and are deployed in areas where human operations are difficult. UAVs require an aerial path from the source and the destination points and thus fall under the domain of 3D path planning. Three-dimensional path planning can be defined as an NP-Hard problem and the main goal is to generate an optimized path between a source point and a destination point. The path is in the form of a set of co-ordinate points which are in close proximity and forms a connected path as the optimized trajectory. 3D path planning algorithms for mobile robots include the one based on visibility graphs (Valavanis and Vachtsevanos 2014), randomly exploring algorithms (Valavanis and Vachtsevanos 2014), Probabilistic Road Maps (Valavanis and Vachtsevanos 2014), deterministic search algorithms like Dijkstra algorithm, Heuristic search algorithms like A*, D* (Valavanis and Vachtsevanos 2014), and various meta-heuristic algorithms. The path planning problem falls under the domain of optimization problem where the goal is to obtained the most optimized solution in the form of a connected trajectory from the source to the destination. The steps involved in the path planning are usually categorized as follows:

1. The first step is to model the environment, design and identify the obstacle and non-obstacle positions in environment. The environment is usually modelled using a grid map where a point in the grid map denotes the corresponding point in the co-ordinate space.
2. The next step is to devise a set of connecting points from the source point and the destination point without having any collision prone point in the respective path. This is obtained using a solution algorithm that can be deterministic or non-deterministic.
3. The final step is to generate a smooth trajectory out of the path obtained which can be viable for the vehicle to follow during the actual run. B-splines are effectively employed to obtain a smooth trajectory from a path.

There are various path planning solutions in literature that employ a meta-heuristic algorithm for generating the viable path between source and destination in the configuration space. The main specialty of these algorithms is that they generate near-optimal paths in a complex environment while taking considerably less time than normal deterministic algorithms. These algorithms are very robust with large application domain and generate good quality solutions for various types of problems. Several algorithms have been previously applied for the three-dimensional UAV path planning problem. The authors in Li and Sun (2008) applied A* algorithm and proposed a route planning method for UAVs. Foo et al. (2009) and Foo et al. (2006) proposed a 3D path planning algorithm using PSO algorithm. The authors of Fu et al. (2012) used GA to solve 2D path planning problem while Wu et al. (2011) used the GA to solve the multi-constraints 3D UAV path planning problem. A different variant called the hybrid multi-population GA was applied for the UAV path planning problem in Arantes et al. (2016). The authors of Roberge et al. (2013) compared the parallel GA with the PSO algorithm in the real-time 3D path planning problem. Zhang and Duan (2014) proposed Predator–Prey Pigeon Inspired Optimization for 3D UAV path planning and compared them with PIO and PSO algorithm. The algorithm performance has also been checked in the dynamic environment in Zhang and Duan (2017) and produces good results. In Duan and Li (2014) the authors analyzed several bio-inspired algorithms for various concepts of Unmanned aerial vehicles relating to single and multi-UAV path planning (Table 1).

Several latest types of research have been focused on UAV path planning. The authors in Chen et al. (2016) presented a research where they applied modified Central Force Optimization for the problem of UAV path planning where they tested the algorithm for six degrees of freedom quad rotor and compared the results with several other meta-heuristics. Swarm intelligence algorithms have been a widely used choice by researchers for choosing the solution for their optimization algorithm. Several different algorithms like PSO (Foo et al. 2009), ABC (Jadon et al. 2014; Bansal et al. 2014) are widely used for solving optimization problems while new algorithms like Spider Monkey (Bansal et al. 2014), Shuffled Frog Leaping Algorithm (Sharma et al. 2017) are also giving good results for different set of problems. Various different meta-heuristics like Cuckoo search (Xie and Zheng 2016), Grey wolf optimizer (Zhang et al. 2016), biogeography-based optimization (Zhu and Duan 2014), firefly algorithm (Wang et al. 2012), improved bat algorithm (Wang et al. 2016), etc. have been applied to the path planning problem for UAV and are observed to show good results in the 2D and 3D environment. Authors in Zhou and Wang (2016) applied Improved Flower Pollination Algorithm for path planning in Unmanned Undersea Vehicles which can be considered analogous with the UAV path planning.

Table 1 A comparison between 2D path planning and 3D path planning

Features	2D path planning	3D path planning
Configuration space	In 2D path planning involves two dimensions (usually x, y) while the third dimension or real world space is usually considered constant	In 3D path planning, the vehicle is able to change it position in all the three dimensions (usually x, y and z), the vehicle movements are not restricted in any dimension and is free to move
Complexity	The exclusion of the third dimension makes the environment less compex that the 3D case with similar conditions	3D environment in more complex and more computations are required to generate paths while taking into account all the dimensions
Applicability	2D path planning is suitable for motion planning for land vehicles. Aerial and underwater vehicles path planning can also be checked in 2D using constraints on one dimension	The path for underwater and aerial vehicles is more realistic when generated in 3D as these vehicles are designed to be able to move in the complete 3D space in real life
Solution approach	Deterministic and heuristic approaches are applied in the domain	Deterministic solutions are rarely applied because of high complexity. Heuristic approaches are more common

Glowworm swarm optimization algorithm is a meta-heuristic introduced by Krishnanand and Ghose (2009). This algorithm has been applied for various types of optimization problems. GSO algorithm exploits the behavior of glowworms: a class of luminescent insects similar to fireflies. GSO algorithm specializes searching for multiple solutions in the objective space simultaneously and is known to obtain multi-modal solutions efficiently. GSO algorithm is known to show extremely good results for the various class of problems in the previous literature. The GSO algorithm has been tested to solve the shortest path problem in a graph (Zhao et al. 2015) and is observed to show good results. GSO is also applied in path planning problem for Autonomous Land Vehicles (Peng-zhen et al. 2014). The authors in Tang and Zhou (2015) implemented GSO algorithm for 2D UCAV path planning problem and obtained better results than several different meta-heuristic algorithms. This paper proposes a new modified glowworm swarm optimization(GSO) algorithm for solving the 3D path planning problem for Unmanned aerial vehicles. The algorithm is modified for solving the path planning problem more accurately within less time frame by proposing several changes which make the creation of new path fast and more efficient. The paper also takes into consideration several constraints like turning angle and trajectory smoothness while devising a collision free path from the source to the destination.

The further structure of the paper has been divided into different sections as follows: Sect. 2 describes the problem formulation describing the path planning problem environment and related parameters taken. Section 3 explains the GSO algorithm and the modification applied for generating optimal solutions for the problem. Section 4 describes the implementation procedure for the solution to the path planning problem and the trajectory generation. Section 4.3 presents the experimental results with comparison with various other meta-heuristic algorithms like Dijkstra, PSO, IBA and BBO algorithm. The paper is then concluded in Sect. 5.

2 Problem formulation

2.1 Terrain construction

In order to start with the path planning problem, an environment is first needed where the vehicle is required to generate its path. The environment consists of several areas where the movement is restricted known as obstacles and the robot is supposed to avoid these areas during its run. The obstacles are mapped in the form of cuboids of different magnitude. However, in the real environment, the shape and size of obstacles is not a perfect geometrical figure, however since the modeling of the unevenly shaped obstacles is a little difficult, it abstains in the experimentation. However, it is ensured that the algorithm created sufficiently avoids obstacles of uneven dimensions. For that, the irregular obstacles are completely encased during the environment modeling and testing. The main goal of the path planning algorithm is to devise a set of connecting points from the source point 'S' to the destination point 'T'.

The whole environment is designed in the 3D co-ordinate space. Any point in the environment can be represented using their co-ordinate point $P = (x, y, z)$. The source node is mapped using coordinates $S = (X_s, Y_s, Z_s)$ while the destination node using co-ordinates $T = (X_t, Y_t, Z_t)$. The obstacles are mapped in the co-ordinate space in a way that all the points which lie in the obstacle domain are marked as a set of points which are inaccessible for the UAV. The whole of the space is defined as a grid and then given an integral value where a single integer denotes a co-

ordinate of three dimensions. The points which are not susceptible to a collision with an obstacle are safe points while the points lying on the position of the obstacle are defined as unsafe points. For the path planning task, the unsafe points are avoided during the course run of the vehicle and only safe points are accessed. The boundary and obstacles information for the two maps included in the paper have been specified in Tables 2 and 3 respectively.

2.2 Cost function

There are various factors on which the cost of a potential solution depends. In our solution, since the paths are generated randomly, it is possible that the generated path may not reach the destination node, because of getting stuck at an obstacle point and not able to move further. Hence, apart from the path length cost, altitude cost, we add one cost where the path is not complete and there is a distance between the path end and goal point.

Various cost functions with respect to the problem statement are:

1. C_{fuel} = Cost of the flight path due to the distance travelled from the source node to the destination node. The shorter the path the less is this cost and thus less is the time and fuel consumption from the source and the destination.
2. C_{turns} = Cost of the sharp turns in the flight path. Flight path should be smooth and sharp turns should be avoided in order to reduce fuel consumption and this cost maintains the smoothness of the path generated.
3. C_{end} = This cost is added to the total cost if the obtained path does not reach the destination due to getting stopped by a dominant obstacle. This cost is the most important and is needed to be made zero in the best solution and is given the highest priority.

When running the algorithm for path planning using the GSO algorithm, solutions are obtained as a collection of points denoting a UAV's path from a source point 'S' trying to reach the destination point 'T'. The solution is denoted as '$L_{x,\ y,\ z}$' such that:

$$L_{x,y,z} = \{S, P_1, P_2, P_3, P_4 \ldots, P_n/T\} \quad (1)$$

Table 2 3D map boundary information

Map	Start boundary	End boundary
Map 1	(0,−5,0)	(10,20,6)
Map 2	(0,0,0)	(20,5,6)
Map 3	(0,−5,0)	(10,20,6)

Here $P_1, P_2, P_3, P_4 \ldots, P_n$ denotes the intermediate points in the path from the source to the destination.

Now, various cost functions are set according to their respective properties. The cost related to the distance traveled by the UAV during its run is expressed according to the amount of the fuel dissipated during the run. Assuming the speed of the UAV to be nearly constant in its run, we calculate the fuel cost as the total length of the path traversed by the vehicle. The fuel cost can be mathematically explained as follows:

$$C_{fuel} = \sum (L_{x,y,z}). \quad (2)$$

$L_{x,\ y,\ z}$ is the set of the points in continuation which denote a path for an unmanned aerial vehicle.

The other cost is the C_{turns} and can be written as the cost addition where the direction of point change from P_{i-1} to P_i is different from the change of direction from P_i to P_{i+1}. Mathematically, C_{turns} can be calculated by finding the instances in the generated path which have a significant turning angle between the two. Consider an instance with three points in the cartesian space A (x, y, z), B (x′, y′, z′) and C (x″, y″, z″). We obtain AB and BC vector respectively. If the cross product of both the vectors is a non-zero value, then there is turn with respect to the vehicle movement, otherwise, the vehicle is moving on the same trajectory. We calculate the total instances where the cross product is non-zero and count it as the turn cost of the path.

The cost parameter C_{end} denotes the cost addition in case the final point in the solution does not reach the final node. The C_{end} can be defined as the Euclidean distance between the end point and the goal point. In case the end point is same as the goal point the C_{end} is zero, other it is raised to a large amount due to its importance. The quantity C_{end} can be defined as:

$$C_{end} = \sqrt{(x_{end} - x_t)^2 + (y_{end} - y_t)^2 + (z_{end} - z_t)^2} \quad (3)$$

Here (x_{end}, y_{end}, z_{end}) denotes the end points of the path obtained from the solution while (x_t, y_t, z_t) denotes the goal point.

The overall cost function is the summation of all the cost function parameters and is used to determine the best path among the set of paths.

The total cost of the path planning solution can be given as:

$$C_{Total} = k_1 * C_{fuel} + k_{-2} * C_{turns} + k_3 * C_{end} \quad (4)$$

Where k_1, k_2, k_3 are experimental parameters whose value is set according to the problem statement parameters. So if the problem statement requires the path length to be given maximum preference than other parameters then k_1 is set to a large value, similarly, k_2 gives importance to te smoothness of the path and k_3 gives importance to the path

Table 3 3D map obstacle information

Obstacle number	Map 1	Map 2	Map 3
1	(0,2,0)–(10,2,5,1.5)	(3.1,0,2.1)–(3.9,5,6)	(0,2,0)–(10,2,5,1.5)
2	(0,2,4)–(10,2.5,6)	(9.1,0,2.1)–(9.9,5,6)	(0,2,4)–(10,2.5,6)
3	(0,2,1.5)–(3,2.5,4.5)	(15.1,0,2.1)–(15.9,5,6)	(0,2,1.5)–(3,2.5,4.5)
4	(7,2,1.5)–(10,2.5,4.5)	(0.1,0,0)–(0.9,5,3.9)	(7,2,1.5)–(10,2.5,4.5)
5	(3,0,2.4)–(7,0.5,4.5)	(6.1,0,0)–(6.9,5,3.9)	(3,0,2.4)–(7,0.5,4.5)
6	(0,15,0)–(10,20,1)	(12.1,0,0)–(12.9,5,3.9)	(0,15,0)–(10,20,1)
7	(0,15,1)–(10,16,3.5)	(18.1,0,0)–(18.9,5,3.9)	(0,15,1)–(10,16,3.5)
8	(0,18,4.5)–(10,19,6)	NA	(0,18,4.5)–(10,19,6)
9	NA	NA	(0,−2,0)–(10,−1.5,1.5)
10	NA	NA	(0,−2,3)–(10,−1.5,5.5)
11	NA	NA	(0,7,0)–(10,7.5,0.5)
12	NA	NA	(0,7,2)–(10,7.5,5.5)
13	NA	NA	(0,11,0)–(10,11.5,2.5)
14	NA	NA	(0,11,4)–(10,11.5,5.5)

reaching the destination and the solution not leaving the path incomplete.

In the problem statement, the vehicle tends to have information only about its surroundings and can change its position among one of its neighboring cells for the next iteration. Thus, the UAV do not have information on where and when it is going to find an obstacle in the start and thus it has to decide and reassess its course only after it encounters an obstacle during its path generation algorithm.

3 Methodology

3.1 Glowworm swarm optimization

GSO algorithm was first introduced by two Indian authors named as Krishnanand and Ghose (2009). The algorithm is based on the foraging behavior of luminescent insects named as Glowworms. These insects move in disjoint groups giving rise to multiple areas of high fitness giving rise to multi-modal solutions in the problem domain. The algorithms and their variants are observed to have high efficiency in terms of speed, quality of solutions and global search ability while showing low chances of premature or local optima convergence. The GSO algorithm relies on local information exchange for the movements of the glowworms and the movement is governed by a quantity known as luciferin, denoting the intensity of the luminosity. The algorithm is mainly depending on a total of four phases after its initialization.

3.1.1 Glowworm distribution phase

In this phase, the randomly generated glowworms are distributed in the solution space. These glowworms are assigned an equal quantity of initial luciferin l_0 and an initial range of r_0.

3.1.2 Luciferin update phase

The luciferin value of the glowworms is updated according to previous luciferin level and the value of the objective function for the glowworm. The update rule for the luciferin update phase can be stated as follows:

$$l_i(t) = (1-\rho)l_i(t-1) + \gamma J(x_i(t)) \tag{5}$$

Here, $l_i(t)$ refers to the luciferin value of the ith glowworm during the xth iteration, $l_i(t-1)$ denotes the luciferin value of glowworm ith glowworm at time t − 1, γ be the luciferin decay constant and $J(x_i(t))$ is the value of the objective function for the glowworm i.

3.1.3 Movement phase

This phase deals the movement of glowworm according to the neighbors having a higher level of luciferin count than its own. The neighbor in which the direction the glowworm moves is chosen according to a probabilistic mechanism. The probability equation which decides the movement of the glowworm is as follows:

$$p_{ij}(t) = \frac{l_j(t) - l_i(t)}{\sum_k \int_{N_i(t)} (l_k(t) - l_i(t))} \tag{6}$$

Here, j belongs to a set $N_i(t)$ given as,

$$j \int N_i(t), N_i(t) = \{j : d_{ij}(t) < r_d^i(t); l_i(t) < l_j(t) \quad (7)$$

where d denotes the distance between glowworm i and glowworm j during the current iteration and rd denotes the range of the glowworm which gets adjusted after each iteration of the algorithm. The glowworm then moves in direction of its selected neighbor according to the following path-update equation:

$$x_i(t+1) = x_i(t) + st * \left(\frac{x_j(t) - x_i(t)}{x_j(t) - x_i(t)}\right) \quad (8)$$

where $x_i(t)$ is a real number and denotes the location of the glowworm i during the tth iteration, st denotes the step size and $||\cdot||$ denotes the euclidean distance operator.

3.1.4 Neighborhood range updation phase

This phase deals with the updation of the neighborhood range domain for the corresponding glowworms. The neighborhood range update equation is given as:

$$r_d^i(t+1) = \min\{r_s, \max\{0, r_d^i(t) + \beta(n_t - N_i(t))\}\} \quad (9)$$

where $r_d^i(t+1)$ denotes the range of the glowworm in the t + 1th iteration, $r_d^i(t)$ denotes the range of the glowworm in tth iteration,r_s denotes the minimum sensory radius, β denotes a constant and n_t denote the parameter to control the neighborhoods' quantity for the glowworm.

3.2 Path generation for the glowworms

For the problem statement, the solution is in the form a continuous set of points starting from source node to the destination node as given in Eq. 1. In the generation of the solution, for each point, there are potential 6 directions in the 3D domain to the point to be moved. For example, for a point P (x, y, z) the six potential neighbor points in the objective space are:

$$\begin{aligned} &\mathrm{A}\,(\mathrm{x} + \delta, \mathrm{y}, \mathrm{z}),\ \mathrm{B}\,(\mathrm{x} - \delta, \mathrm{y}, \mathrm{z}),\ \mathrm{C}\,(\mathrm{x}, \mathrm{y} + \delta, \mathrm{z}),\\ &\mathrm{D}\,(\mathrm{x}, \mathrm{y} - \delta, \mathrm{z}),\ \mathrm{E}\,(\mathrm{x}, \mathrm{y}, \mathrm{z} + \delta),\ \mathrm{F}\,(\mathrm{x}, \mathrm{y}, \mathrm{z} - \delta) \end{aligned} \quad (10)$$

Here, δ denotes the smallest addition to the co-ordinate point specified to reach the next point in the objective space. Technically, it is the shortest distance defined between two points on the map. This makes the space to form a grid- based structure. The diagonal points can be reached using these points hence only the basic additions are specified.

The path generation is defined as generating a set of connecting points by using the neighbors of a point which do not lie within an obstacle and to generate a path from the source to the destination. Here we define a constant factor of 'N' as to the maximum number of nodes possible in the path matrix. If the path matrix exceeds the value of N, the currently generated path is returned by the solution. The algorithm for generating random paths is as follows:

Algorithm 1: Generating initial path using random selection of neighbors

Step 1: Initialize the path solution matrix P and insert the source node 'S'. Define the current node as the source node 'S'

Step 2: Generate neighbors of the current node. Select the non-collision nodes out the neighbors generated

Step 3: Randomly select a node out of the set of non-collision neighbor and add it to the path solution matrix 'P'. Set the selected node as the current node

Step 4: If the current node is the goal node or the number of the nodes in the path solution matrix exceeds 'N' then move to step 5 with the solution matrix 'P', otherwise move to step 2

Step 5: Remove loops from the solutions in the path set by deleting the nodes present between two repeating nodes in the path set 'P'. This results in the shortening of the path and makes the pass more efficient. Return the path set 'P'

The following algorithm generates the glowworms as a connecting path between two points in the configuration space. The solution depicts a solution and has a cost value which is defined according to the cost functions defined in the previous section. This cost value determines the fitness of the solution. The less the cost of the solution, the better is the solution for the current problem. However, it is seen that due to the randomness of the neighbor taken from neighbor set the solution to tend to get into loops or get into the opposite direction to that required by the solution frequently. This results in inefficiency in the quality of the solutions and takes a number of iterations to obtain a viable solution let alone the optimal solution.

The recommended way is to generate initial paths according to a pseudo-random strategy in which the neighbors are chosen from the neighbor set which is along the viable direction towards the goal point based on a heuristic to the goal point. According to this strategy, instead of choosing the six neighbors of a point on the map, choose the three viable neighbors of the point which decrease the distance from the point to the destination node. In case a viable neighbor is not present due to being blocked by an obstacle, then select the one random neighbor from the non-Viable Neighbor list. However, count the number of non-Viable selections and if the non-viable selections exceed a certain number, it means the path is not improving in its run and it is time to terminate the further path generation. This path can, however, be improved using further strategies in order to obtain the best possible paths.

Therefore, a new collection of nodes known as Viable-neighbors is generated with stores the neighbors reducing the distance of the current node from the goal node.

Viable Neighbors = {CurrentNeighbour || CurrentNeighbour is a viable direction towards the goal}

The new algorithm to generate the solutions is given in Algorithm 2.

Algorithm 2: Generating initial solutions using semi-random selection of neighbors.

Step 1: Initialize the path solution matrix P and insert the source node 'S'. Define the current node as the source node 'S'. Define the viable directions towards the goal point.

Step 2: Generate neighbors of the current node. Select the non-collision nodes out the neighbors generated. Select the neighbors according to the viable directions and add them to the Viable-Neighbors set.

Step 3: Randomly select a node out of the set of non-collision Viable-Neighbors and add it to the path solution matrix 'P'. If the Viable Neighbor set is empty, select a node from the CurrentNeighbour set and add it to the path matrix 'P' and add one to Non-Viable Node count. Set the selected Node as the current Node.

Step 4: If the current Node is the goal node or the count of Non-viable nodes exceeds the threshold, then move to step 5. Otherwise, go to step 2.

Step 5: Remove loops from the solutions in the path set by deleting the nodes present between two repeating nodes in the path set 'P'. This results in the shortening of the path and makes the pass more efficient. Return the path set 'P'.

The algorithm results in generation of pseudo-random paths between a source point and destination point. This algorithm is used for defining initial glowworms as well as during the glowworm update phase of the GSO algorithm (Fig. 1).

3.3 Glowworm swarm optimization algorithm for discrete path optimization

The original GSO algorithm is fit to be applied to optimization functions related to continuous functions. The position update equation updates the glowworm's position according to the current glowworm positions and the step size. However, in the path optimization problem, a glowworm corresponds to a path starting from the source node moving towards the destination node. Hence, the glowworm swarm optimization algorithm is needed to be adjusted to cater the discrete solutions in the objective domain. Therefore, some modifications are introduced to the glowworm swarm optimization algorithm. The most important are the definition of path update equation using a hybrid genetic operator like reproduction and mutation to improve weak solutions as well as introducing chaotic factor explore new solutions from the current solution. The operations can be properly explained in the following section.

The GSO algorithm deals with local information exchange in its variable neighborhood. The algorithm chooses a random glowworm among the one which has greater luciferin than the current glowworm while being present in the range of the glowworm and changes the current solution according to the chosen glowworm. The glowworm in the current problem denotes a set of points denoting a path on the map for the vehicle. The path has to be changed making sure it maintains its properties of reaching the destination while getting improved from the better solution. The technique which is used for the purpose are the introduction of genetic operators and a chaotic factor for the glowworm to reach its destination by exploring new paths.

The algorithm for path update is as follows:

Algorithm 3: GSO algorithm for UAV path planning

1. Define parameters: population size: n, initial luciferin: l_0, initial neighborhood range: r_0 k_1, k_2, k_3, $NVNodes_{max}$, max iterations $iter_{max}$
2. Generate initial glowworm solutions using the steps described in Sect. 3.2. Assign a same quantity of initial luciferin l_0 and a same initial neighborhood range r_0.
3. While iter $<$ $iter_{max}$
   for i = 1 to n
   1. Calculate luciferin value of glowworm i according to (5).
   2. Find the set of neighbors of glowworm using Eq. (7).
   3. Calculate the probability of movement using Eq. (6).
   4. Select a neighbor j among the set using a probabilistic mechanism.
   5. Move the glowworm i towards neighbor j according the procedure described in Sect. 3.3. If no neighbor found use, mutation strategy as described Sect. 3.3 to generate new glowworms.
   6. Update the neighborhood range using Eq. (9).
   end for
   end while
4. Output the best glowworm solution as the most suitable path.

Case 1 For a current glowworm, there are one or more than glowworms within its range having a larger luciferin value. In this case, one of the neighbor glowworms gets selected to update the current glowworm's position. Now the task would be to use the new glowworm path matrix to update the current ones. This would be done by first a common point in both the paths of the chosen glowworms.

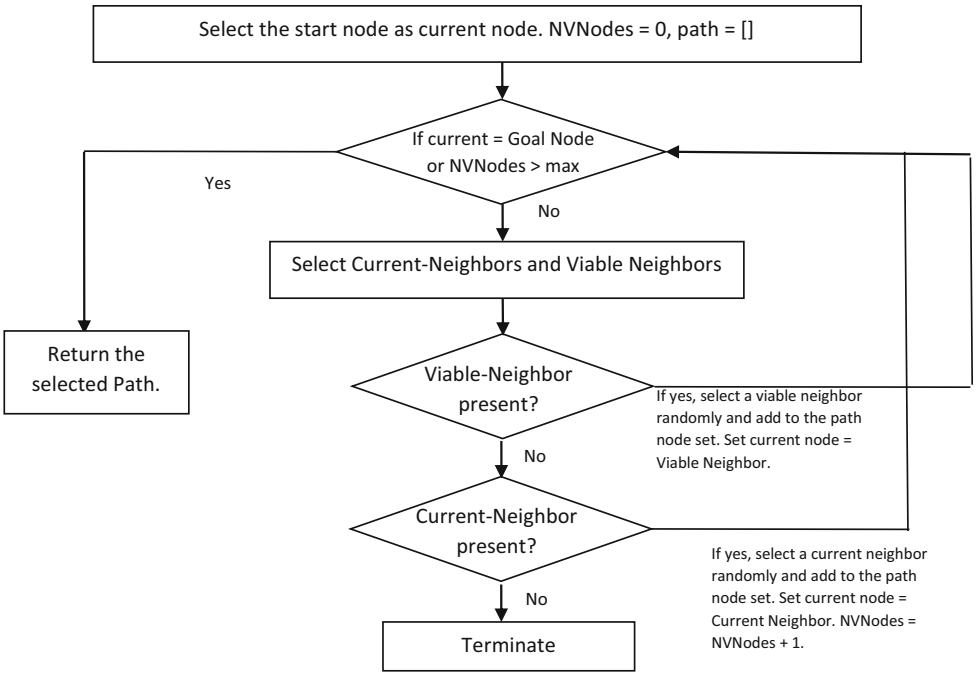

Fig. 1 Initial path generation flowchart for application of GSO algorithm for 3D path planning

If a common point is found This would mean that current glowworm can be improved by choosing its path direction to the other one. This can be done by choosing a random point in the neighbor glowworm after the common point found earlier. Now, copy the path contents of the neighbor glowworm between the common points and new chosen point and replace it with the current glowworm's path positions after the common point. After the random point, define the rest of the path according to the path generating algorithm defined in the previous section.

If a common point is not found If a common point is not found, then the current glowworm path is improved according the chosen neighbor's path completely. This is done by choosing two random points in both the paths on the glowworms, then a new path is tried to be formed between the two chosen points. If a complete path is able to be generated within 'n' number of tries using the path generation algorithm, then both the points are connected, replaced in the current glowworm path and then the rest of the path between the point and the goal point is then copied from neighbor glowworm or generated according the path generation algorithm depending on whether the neighbor path successfully reaches the goal point.

Case 2 In this case, a glowworm does not have neighbor glowworm in its range having a higher luciferin than it, then we need to use the mutation property to find a different solution from the current which may or may not is a better one than the current one. This can be done by choosing a random point from the path set and then mutate it to any other point on the map which is not in the obstacle and then try to generate a path between the chosen point in the glowworm and the new point chosen. If a path is not generated, then choose another point on the map until a connected path is generated between the two points. If the path is generated between the two points, replace the newly generated path in the original glowworm path after the previously chosen point and then generate the path from the end point of the newly generated path to the goal point. This is the new glowworm for the current iteration.

After the path is generated, return the new path as the corresponding glowworm, calculate the cost according to the cost functions and check with the best path's cost. If the glowworm's cost is less than the current-best cost, then replace the best cost with the current glowworm's cost. Run the algorithm for all the glowworms, update the luciferin value and the range of the glowworm. These steps are then repeated until the number of iterations is reached.

After a specified number of iterations, the result is the best path obtained after all the directions. This best path can then be used by the UAV for its run on the map. The path is then plotted on the map and then passed to the trajectory generator in order to generate a viable trajectory using the path submitted by the GSO algorithm. The path generated when plotted on the map is erratic and not a smooth trajectory due to the random striations and movements for the vehicle. Thus it is essential that the path before being employed for the real robot is to be smoothed and all the unnecessary turns are to be removed.

The algorithm for generating paths in the mGSO algorithm is given as follows (Fig. 2):

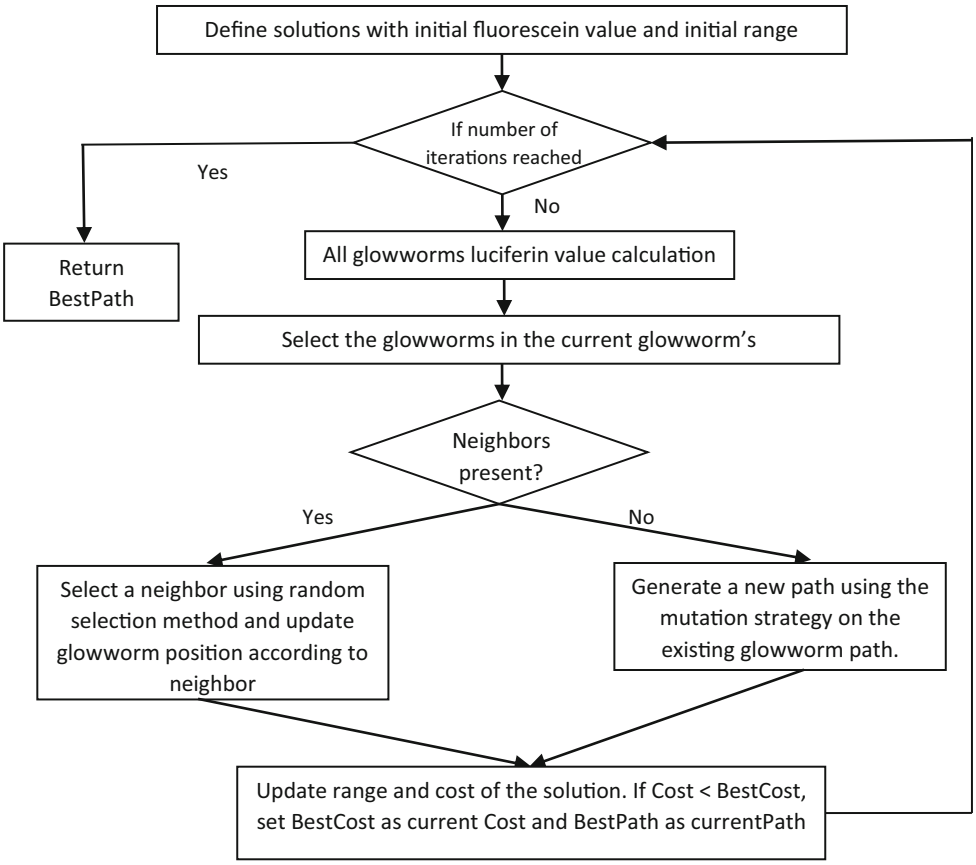

Fig. 2 GSO algorithm flowchart for 3D Path Planning

3.4 Trajectory generation

Trajectory generation is done to make the path smoother and flyable by the UAV. The received path after the GSO algorithm may have sharp turns that are not feasible for the UAV flight and can be further improved. Velocity and acceleration for the vehicle movement are generated as a function of time using polynomial functions. A quintic (5th order) polynomial is used to define the initial and final values and is usually considered smooth which signifies that the first few temporal derivatives are continuous. Thus, we can effectively obtain velocity and acceleration values by calculating the derivatives of the polynomial. A typical quintic polynomial would look like:

$$S(t) = At^5 + Bt^4 + Ct^3 + Dt^2 + Et + F \quad (11)$$

The first step in the trajectory generation would be to extract waypoints in the path. The waypoints would be the important points in the generated path where the vehicle is either having a sharp turn near an obstacle or two points within which an obstacle lies and a line of sight path is not there. After the waypoints are generated, the velocity and acceleration between the points are calculated using the quintic polynomial based trajectory fitting method (Li and Sun 2008) and the equations are stored in variables. The velocity and accelerations for the whole path are then calculated using the same method and then the trajectory is generated using these values and the vehicle can then move through this trajectory.

The steps to generate waypoints from the path obtained are as follows:

Algorithm 4: Waypoint generation algorithm

1. Define a waypoint vector W. Copy the path obtained from Algorithm 3 into W.
2. Eliminate the straight line path points and keep only those points where there is a co-ordinate change in the different co-ordinate axis than the previous one. This will result in the path containing only the potential turn points of the path.
3. For all the turn points, check till which turn point contains a path which does have any collidable object for the vehicle in the area between the points. Those turn points can then fit a simple line and no turn is necessary. Start the process for the next turn point. If there a collidable obstacle between two turn points, generate the smallest path between the one before that point and continue with the normal path for the identified point.
4. The obtained vector W contains the waypoints W, in which a curve can be fitted using the curve fitting methods (Li and Sun 2008).

4 Simulation experiments

In order to test the effectiveness of the modified GSO algorithm, a series of experiments were performed in MATLAB programming language. The Matlab 2009a version was chosen for performing experimentation for the Path planning problem. The PC chosen had a 3.4 GHz of CPU and a 2 GB of RAM.

4.1 2D path planning

The first set of experiments were performed on 2D environment where the obstacles were in the form of circular threat areas. There were three circular threat areas chosen for the UAV to generate the path. The circular threat areas behaved as pseudo obstacle where the UAV instead to prevent collision had to resist entering the area as much as possible. The start and goal co-ordinates for the path planning are chosen as (0,0) and (4,6) respectively. A point size vehicle is chosen for the test to maintain focus on path planning problem. The information for the threatening areas is present in Table 4 and Fig. 3.

The parameters chosen for the GSO algorithm are as follows:

1. Population Size: 100.
2. Number of iterations: 100.
3. Initial Luciferin: 25.
4. Initial Range: 5.0.
5. Luciferin decay: 0.4.
6. Luciferin enhancement: 0.6.
7. Range boundary: 50.2.
8. Range constant = 0.5.

The initial experiments of the GSO algorithm path planning were performed to test the effectiveness of the algorithm implementation and to have a conjecture on the quality of solution for the 3D path planning of UAV. The running tests for the calculated paths for the planning are shown in Fig. 4.

4.2 3D path planning

After testing the algorithm for 2D path planning, the algorithm is then implemented for the path planning in the 3D environment. The experimentation is done in MATLAB 2009a in a PC with CPU of 3.4 Ghz and a RAM of 2 GB. The algorithm is run in the environment described in the previous section. Three different maps are chosen for the algorithm to be tested demonstrated in Fig. 4a–c respectively. The initial parameters taken are defined as follows:

1. Population size: 20–30.
2. Number of iterations: 25–50.
3. Initial Luciferin: 25.
4. Initial Range: 5.0
5. Luciferin decay: 0.4
6. Luciferin enhancement: 0.6
7. Range boundary: 50.2
8. Range constant: 0.5

The algorithm is run on all the maps and then the results are tested and compared with four other algorithms which are Dijkstra (Deterministic), PSO, BBO and IBA algorithm respectively. The generated paths for all the maps is shown on Fig. 4 while the average running time with the number of iterations for the meta-heuristic algorithms and the Dijkstra algorithms are presented in Tables 5, 6 and 7 for Map 1,2 and 3 respectively.

The comparison of running times and best costs of all the three maps with different iteration numbers and population counts has been presented in Fig. 5. It can be noted that the results are improved when the path planning is done using GSO algorithm over and above Dijkstra algorithm. The results are also comparatively better that the other meta-heuristics in most of the cases.

After the generation of the path, the trajectory is generated for the algorithm and has been presented in Fig. 6. The trajectory is a result of the path smoothing and optimization procedure applied on the path obtained after applying the GSO algorithm in order to make it more flyable for the aerial vehicle reducing the number of sharp twists and turns in the path wherever possible.

The algorithm performance is compared with that of Dijkstra, PSO, IBA and BBO algorithm to for in terms of training time taken for generating a viable path from the source to destination. The algorithms were made to generate paths for all the mazes and were made to go from one end to another end. It is observed that for Dijkstra algorithm deterministic results were obtained but the time required for running the algorithm was quite high rather than for other meta-heuristics. It is also noted that as the map size or obstacles increases the Dijkstra algorithm become infeasible for finding the path between source and destination in reasonable time. The comparison for the running times for various algorithms made to run with different start and end points in the map are given in Tables 4 and 5.

Table 4 Obstacle information for 2D map for UAV path planning using GSO

Obstacle number	Center (units)	Radius (units)
1	(1.5,4.5)	1.5
2	(4.0,3.0)	1.0
3	(1.2,1.5)	0.8

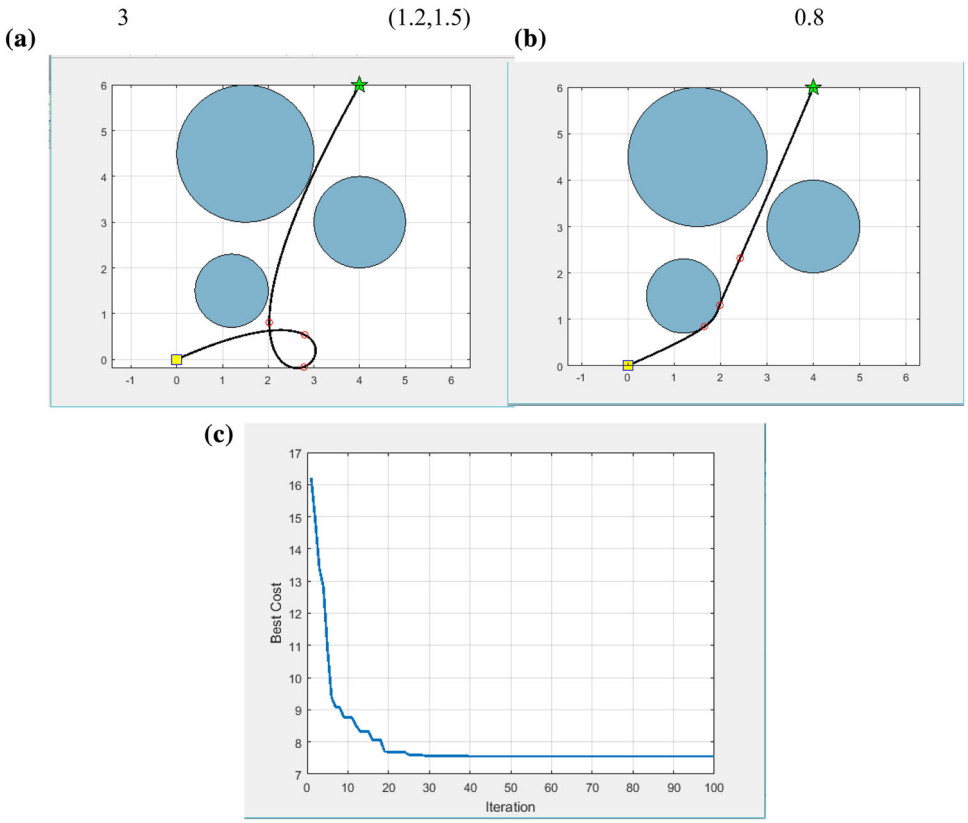

Fig. 3 **a** Initial path generated during application of GSO algorithm for 2D path planning in radar environment. **b** Final obtained path. **c** Best cost vs number of iteration graph

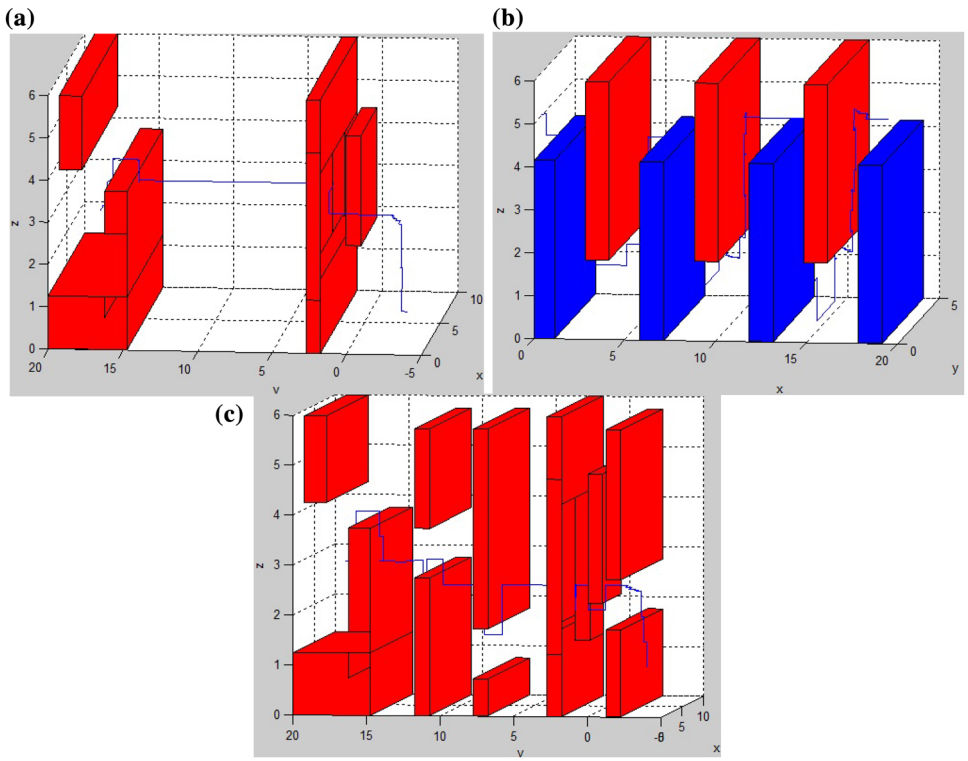

Fig. 4 **a** The path generated for Map 1 using GSO algorithm. **b** The path generated for Map 2 using GSO algorithm. **c** The path generated for Map 3 using GSO algorithm

From Tables 4 and 5, it can be inferred that the modified GSO algorithm performed good results and took less time than Dijkstra and other meta-heuristic algorithms in most of the cases. mGSO algorithm passes the BBO algorithm marginally in the best cost and total time and performs better than the other algorithms in both the parameters.

Table 5 Running time of algorithms for Map 1 for different population count and iterations from start (0, −4, 1) and end (2, 17, 3)

Algorithm	Population size	Iterations	Best cost (points)[a]	Time (units)
Dijkstra	20	25	308.20	127.03
GSO	20	25	314.70	8.71
PSO	20	25	311.00	12.42
IBA	20	25	346.80	11.75
BBO	20	25	317.60	8.70
Dijkstra	25	40	308.20	131.75
GSO	25	40	305.40	15.93
PSO	25	40	322.80	23.45
IBA	25	40	310.60	22.31
BBO	25	40	308.70	15.29

[a]Best cost formula: Cost = Fuel cost + 80*(distance of end point to the goal point of the path) + Turn cost (refer Sect. 2.2)

Table 6 Running time of algorithms for Map 2 for different population count and iterations from start (0, 1, 5) and end (19, 1, 5)

Algorithm	Population Size	Iterations	Best cost (points)[a]	Time (units)
Dijkstra	20	25	256.60	18.56
mGSO	20	25	293.90	10.95
PSO	20	25	357.80	28.37
IBA	20	25	370.90	16.00
BBO	20	25	363.10	19.21
Dijkstra	30	40	256.60	18.48
GSO	30	40	337.90	14.54
PSO	30	40	368.20	36.42
IBA	30	40	364.30	19.67
BBO	30	40	333.10	25.07

[a]Best cost formula: Cost = Fuel cost + 80*(distance of end point to the goal point of the path) + Turn cost (refer Sect. 2.2)

Table 7 Running time of algorithms for Map 3 for different population count and iterations from start (0, −4, 1) and end (2, 17, 3)

Algorithm	Population size	Iterations	Best cost (points)[a]	Time (units)
Dijkstra	20	25	309.30	123.79
mGSO	20	25	324.60	10.56
PSO	20	25	320.40	15.59
IBA	20	25	337.40	13.50
BBO	20	25	325.00	10.09
Dijkstra	25	40	309.30	123.23
GSO	25	40	312.32	18.70
PSO	25	40	368.20	26.67
IBA	25	40	364.30	25.35
BBO	25	40	333.10	18.76

[a]Best cost formula: Cost = Fuel cost + 80*(distance of end point to the goal point of the path) + Turn cost (refer Sect. 2.2)

The Best Cost of various meta-heuristic algorithms after several iterations has been compared in Table 6 and visualized using Fig. 7. It can be seen that modified GSO algorithm explores the solution space efficiently by getting out of the local optima while other algorithms tend to get stuck in local optima where the solution Best Cost cease to decrease after a certain number of iterations (Tables 8, 9, 10; Figs. 8, 9).

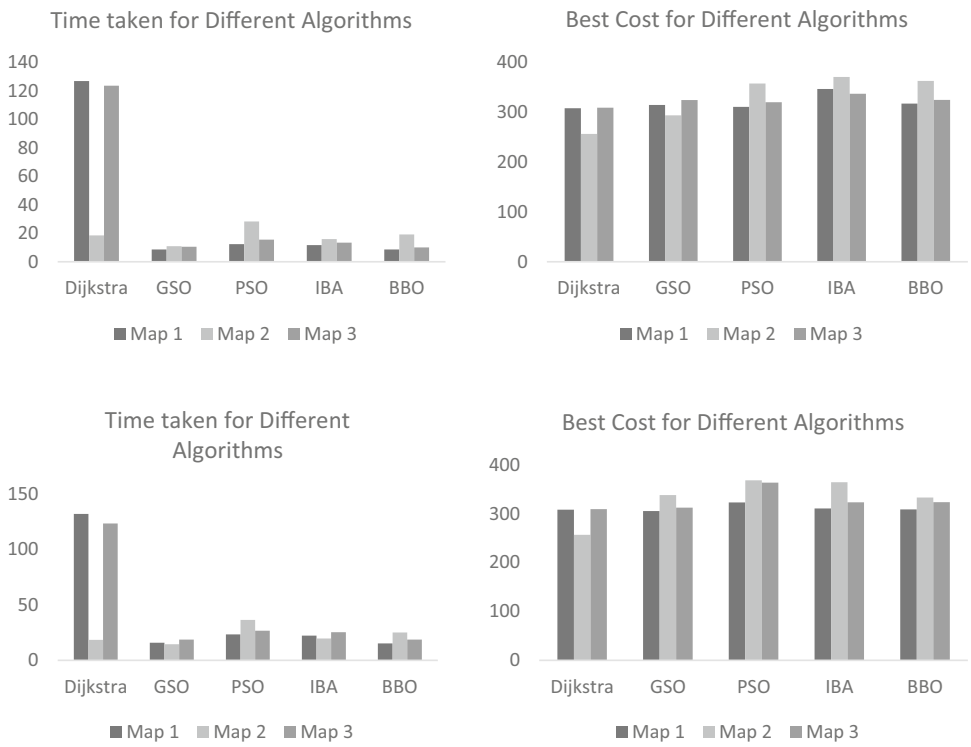

Fig. 5 Analysis of running time and best cost for three maps on different iteration number and population. **a** Running time. **b** Best cost for population size: 20 and iterations: 25. **c** Running time. **d** Best cost for population size: 25 and iterations: 40

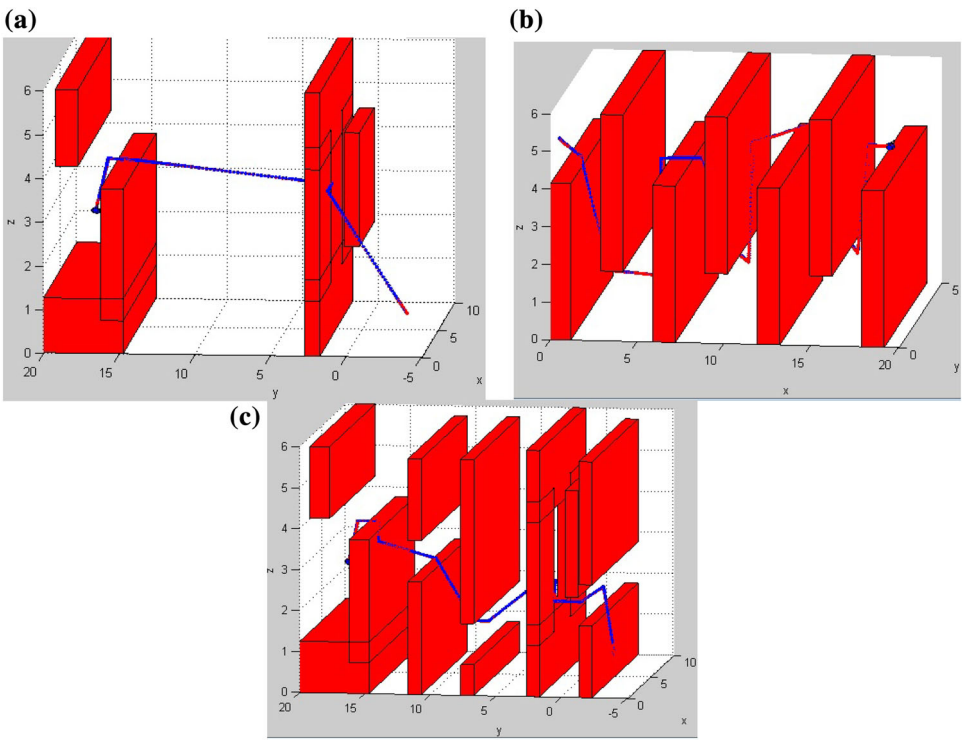

Fig. 6 **a** Trajectory generated for Map 1 using GSO algorithm. **b** Trajectory generated for Map 2. **c** Trajectory generated for Map 3 using GSO algorithm

4.3 Convergence and complexity analysis of GSO algorithm

The algorithm is a combination of GSO algorithm, with heuristic search and genetic operators. The presence of heuristic operators makes the algorithm more convergent in the case on less clutter in the environment. In the case of a map with little or no obstacles, it can be observed that the GSO algorithm generates a near-optimal solution in less number of iterations. The more obstacles are present in the

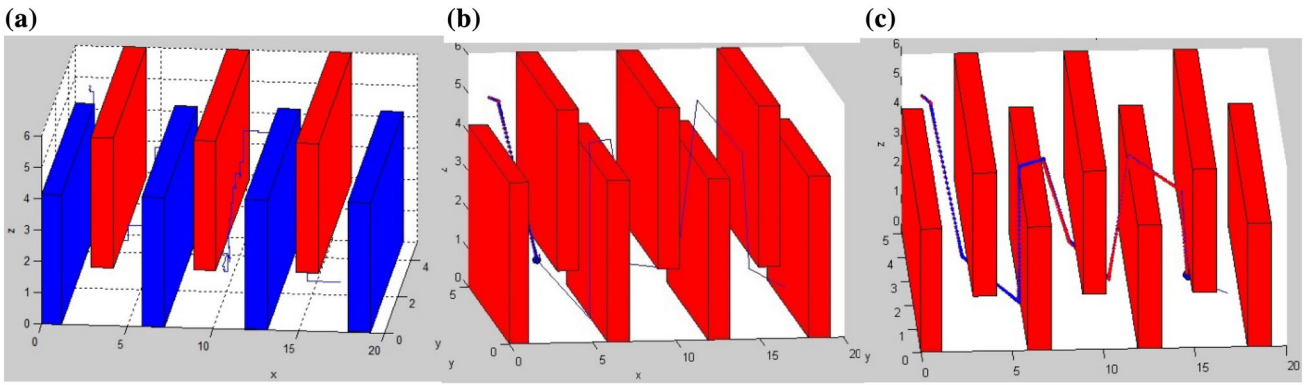

Fig. 7 **a** Path generated. **b** Trajectory movement (start). **c** Trajectory Movement (end) for Map 2 with start: (1, 4.5, 5) and end (17, 1, 1])

Table 8 Best cost per iteration count for various meta-heuristic algorithms applied on Map 1 in start (0, −4, 1) and end (2, 17, 3)

Iteration number	GSO	PSO	BBO	IBA
1	360.90	371.10	355.80	455.80
5	330.40	364.50	337.20	334.50
10	326.40	345.70	337.20	334.50
15	301.60	330.80	307.20	334.50
20	301.60	330.80	307.20	334.50
25	301.60	330.80	307.20	334.20
30	301.60	330.80	307.20	334.20
35	301.60	330.80	307.20	331.20

Table 9 Best cost per iteration count for various meta-heuristic algorithms applied on Map 2 in start (0, 1, 5) and end (19, 1, 5)

Iteration number	GSO	PSO	BBO	IBA
1	991.15	762.68	392.10	1128.7
5	779.57	472.50	392.10	702.59
10	686.42	361.10	392.10	442.60
15	667.66	360.40	362.80	442.60
20	667.66	360.40	362.80	417.10
25	645.25	360.40	362.80	368.40
30	631.24	360.40	362.80	368.40
35	319.90	360.40	362.80	352.80
40	319.90	360.40	362.80	352.80
45	319.90	360.40	362.80	352.80
50	319.90	353.80	362.80	352.80

environment; more iterations are needed for the algorithm to converge. After a certain number of iterations, the probability of improvement of solutions becomes low and then getting new solutions which are better becomes less.

After the application of the algorithm, it is seen that the converged solutions are generated a lot faster for the small and decluttered maps and are generated with more iterations in case of large and cluttered maps. On application of algorithm on the small map i.e. Map 1 the solutions are generated on as low as 2 iterations of the algorithm and become converged after 10 iterations. On the application of a typical map i.e. Map 2 and Map 3 in which there is no direct line of sight path between the source and destination. The path is generated usually after 10 iterations and comes to convergence after 30-40 iterations.

Table 10 Best cost per iteration count for various meta-heuristic algorithms applied on Map 3 in start (0, −4, 1) and end (2, 17, 3)

Iteration number	GSO	PSO	BBO	IBA
1	377.00	411.40	491.40	505.70
5	367.40	354.10	382.60	412.70
10	367.40	324.60	325.00	366.60
15	333.30	324.60	325.00	345.30
20	333.30	324.40	325.00	343.60
25	328.10	324.40	325.00	342.80
30	328.10	324.40	325.00	342.80
35	328.10	324.40	325.00	342.80

From the results obtained in the previous section, it is seen that the GSO algorithm shows good convergence speed among all other meta-heuristic algorithms in most of the cases while it showed exponentially better results from deterministic algorithms like Dijkstra algorithm. The running time of all the algorithms is almost similar when compared to small maps like Map 2 while for larger maps a significant improvement is observed for the GSO algorithm. Thus it can be observed that the GSO algorithm showed polynomial complexity in terms of algorithm convergence to get near-optimal and satisfactory results in comparison with exponential deterministic algorithms. Among the meta-heuristics, the GSO algorithm performs better results and thus it can be seen as a viable optimization algorithm for its application in the real-world UAVs.

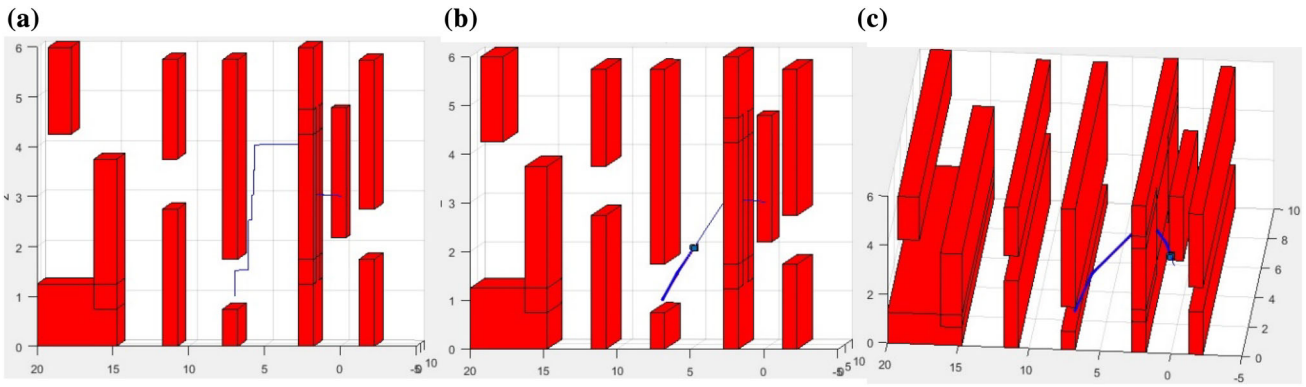

Fig. 8 **a** Path generated. **b** Trajectory movement (start). **c** Trajectory Movement (end) for Map 2 with start: (1, 7, 1) and end (1, 0, 3)

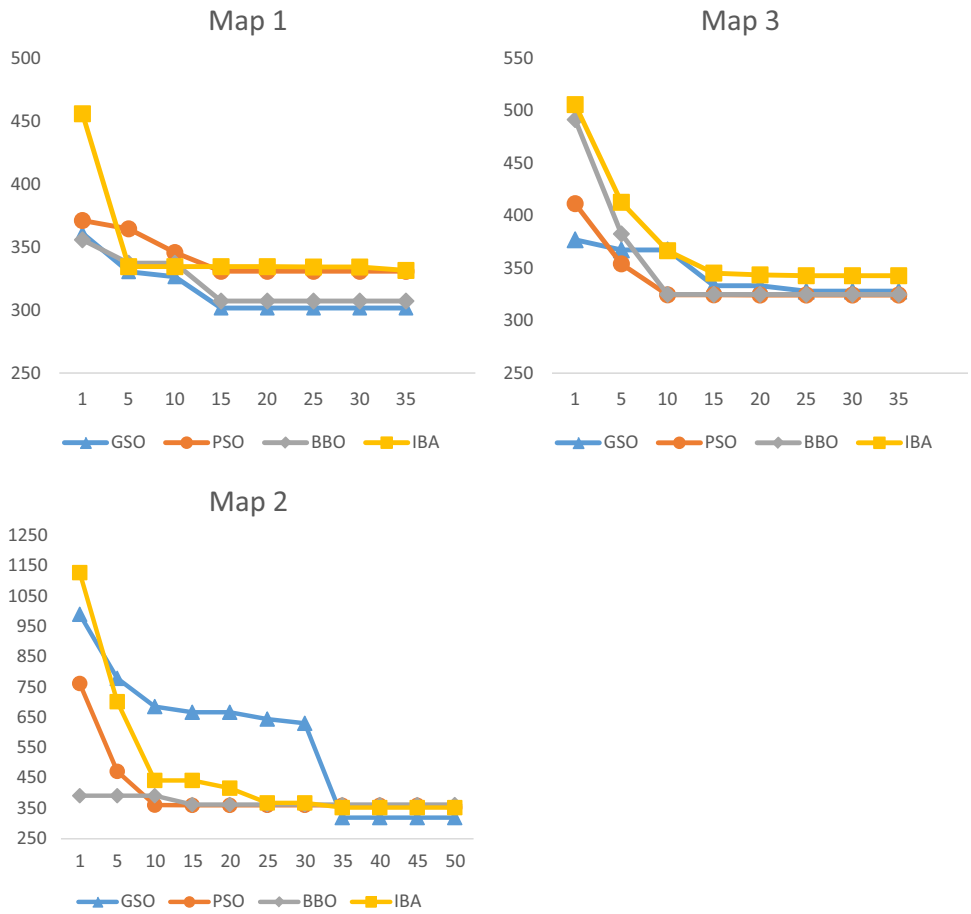

Fig. 9 Comparison chart for best cost over various iterations for different meta-heuristic algorithms

4.4 Effect of parameters on the algorithm performance

4.4.1 Effect of population size

The population size and the number of iterations support each other for generating the most optimal solutions. A considerably high population size becomes unnecessary for the problem and usually slows down the algorithm. In the experiments, a population size of 20–30 glowworms combined with a suitable number of iterations is used for the application of the algorithm. It can be seen that choosing a low population size would make the generation of most viable solutions difficult while a high population size would be redundant in the generation of solutions.

4.4.2 *Effect of the cost parameter constants like path incompletion cost and turn cost constants*

The constants added to the cost functions are according to the importance given to the various components of the cost function. A high importance is usually given to the path incompletion cost to make sure the pseudo-randomly generated path reaches the goal point effectively. Hence a high constant is multiplied by the path incompletion cost, C_{end} than the turn cost. The turn cost is required a low weightage than other costs because in many cases, turns become inevitable. Also, the random nature of path generation creates many turns which are then eliminated during the trajectory generation. The luciferin decay constant governs the importance of the cost function in deciding the luciferin quantity for future generations. Luciferin decay is kept moderate in the experiment so that the solutions' luciferin value gets modified steadily in successive iterations of GSO. Similarly, an average constant for range update (β) is also set for generating new range for the updated glowworms.

4.4.3 *Effect of the map size and number of obstacles in the path*

The number of obstacles in the map highly decides how many iterations could be needed in order to generate a path between source to destination. A large map with no line-of-sight path between the source and the destination would require more number of iterations for the path generation and path convergence. The algorithm in a clean and uncluttered map with line-of-sight path would easily generate an optimal solution using the heuristic properties of the algorithm.

5 Conclusion

The paper presents a modified GSO algorithm for 3D UAV path planning. The algorithm successfully generates an optimized path between the source and destination while keeping the flight constraints to optimal. The introduction to the genetic operators of mutation and crossover to the GSO algorithm improves its exploration and exploitation characteristics and helps the algorithm for better results and faster convergence. As a result, the path cost is less and the obstacles are avoided during its run and thus the algorithm shows promising results after comparison with deterministic algorithms and several meta-heuristic algorithms. The new approach is more explorative and during many iterations, it was observed that the algorithm moves out of the local optima and the solution improves further due to the addition of mutation in the path generation algorithm. The new algorithm shows a good potential solution for the implementation of the path finding for a real-time UAV in the presence of static obstacles in the way.

The future work for this paper would be to add several dynamic functionalities to the 3D map in the form of dynamic obstacles so as to implement path re-planning algorithm which is an extremely important field for the robot path planning domain. The algorithm can also be implemented for 3D multi-UAV path planning and task assignment as a future research work.

References

Arantes MD, Arantes JD, Toledo CF, Williams BCA (2016) Hybrid multi-population genetic algorithm for UAV path planning. In: Proceedings of the 2016 on genetic and evolutionary computation conference, ACM, pp 853–860

Bansal JC et al (2014a) Self-adaptive artificial bee colony. Optimization 63(10):1513–1532

Bansal JC et al (2014b) Spider monkey optimization algorithm for numerical optimization. Memet Comput 6(1):31–47

Bortoff SA (2000) Path planning for UAVs. In: Proceedings of the 2000 American control conference on ACC (IEEE Cat. No. 00CH36334), Chicago, pp 364–368

Chen Y et al (2016) Modified central force optimization (MCFO) algorithm for 3D UAV path planning. Neurocomputing 171:878–888

Corke P (2011) Robotics, vision and control: fundamental algorithms in MATLAB, vol 73. Springer, Berlin

Duan H, Li P (2014) Bio-inspired computation in unmanned aerial vehicles. Springer, Berlin

Foo J, Knutzon J, Oliver J, Winer E (2006) Three-dimensional path planning of unmanned aerial vehicles using particle swarm optimization. In: 11th AIAA/ISSMO multidisciplinary analysis and optimization conference, Portsmouth, Virginia, pp 123–156

Foo JL et al (2009) Path planning of unmanned aerial vehicles using B-splines and particle swarm optimization. J Aerosp Comput Inf Commun 6(4):271–290

Fu SY, Han LW, Tian Y, Yang GS (2012) Path planning for unmanned aerial vehicle based on genetic algorithm. In: 2012 IEEE 11th international conference on cognitive informatics and cognitive computing (ICCI*CC), IEEE, pp 140–144

Jadon SS, Bansal JC, Tiwari R, Sharma H (2014) Artificial bee colony algorithm with global and local neighborhoods. Int J Syst Assur Eng Manag. doi:10.1007/s13198-014-0286-6

Krishnanand KN, Ghose D (2009) Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions. Swarm Intell 3(2):87–124

Levinson J et al (2011) Towards fully autonomous driving: systems and algorithms. In: 2011 IEEE intelligent vehicles symposium (IV), Baden-Baden, pp 163–168

Li J, Sun X (2008) A route planning's method for unmanned aerial vehicles based on improved A-star algorithm [J]. Acta Armament 7:788–792

Peng-zhen DU et al (2014) Global path planning for ALV based on improved glowworm swarm optimization under uncertain environment. Acta Electron Sin 3:031

Roberge V, Tarbouchi M, Labonté G (2013) Comparison of parallel genetic algorithm and particle swarm optimization for real-time UAV path planning. IEEE Trans Ind Inform 9(1):132–141

Sharma P, Sharma N, Sharma H (2017) Locally informed shuffled frog leaping algorithm. In: Proceedings of sixth international conference on soft computing for problem solving, Springer, Singapore

Tang Z, Zhou Y (2015) A glowworm swarm optimization algorithm for uninhabited combat air vehicle path planning. J Intell Syst 24(1):69–83

Valavanis KP, Vachtsevanos GJ (2014) Handbook of unmanned aerial vehicles. Springer, Berlin

Wang G, Guo L, Duan H, Liu L, Wang H et al (2012) A modified firefly algorithm for UCAV path planning. Int J Hybrid Inf Technol 5(3):123–144

Wang GG, Chu HCE, Mirjalili S (2016) Three-dimensional path planning for UCAV using an improved bat algorithm. Aerosp Sci Technol 49:231–238

Wu JP, Peng ZH, Chen J (2011) 3D multi-constraint route planning for UAV low-altitude penetration based on multi-agent genetic algorithm. IFAC Proc Vol 44(1):11821–11826

Xie C, Zheng H (2016) Application of improved Cuckoo search algorithm to path planning unmanned aerial vehicle. Springer, Berlin, pp 722–729

Zhang B, Duan H (2014) Predator-prey pigeon-inspired optimization for UAV three-dimensional path planning. In: International conference in swarm intelligence, Springer, Berlin, pp 96–205

Zhang B, Duan H (2017) Three-dimensional path planning for uninhabited combat aerial vehicle based on predator-prey pigeon-inspired optimization in dynamic environment. IEEE/ACM Trans Comput Biol Bioinform 14(1):97–107

Zhang S et al (2016) Grey wolf optimizer for unmanned combat aerial vehicle path planning. Adv Eng Softw 99:121–136

Zhao X, Chen S, Sheng Y (2015) Research on the problem of the shortest path based on the glowworm swarm optimization algorithm. Optimization 52(4):65–95

Zhou Y, Wang R (2016) An improved flower pollination algorithm for optimal unmanned undersea vehicle path planning problem. IJPRAI 30(4):1659010

Zhu W, Duan H (2014) Chaotic predator–prey biogeography-based optimization approach for ucav path planning. Aerosp Sci Technol 32(1):153–161