STATE UNIVERSITY OF NOVI PAZAR

DEPARTMENT OF TECHNICAL SCIENCES

Irfan S. Fetahović

IMPROVEMENT OF ROBOT PATH PLANNING BY BRAIN STORM ALGORITHM

Doctoral Dissertation

Novi Pazar, 2018

Dissertation Data

Doctoral Dissertation Title: Improvement of robot path planning by brain storm algorithm

Abstract

Path planning can be defined as finding optimal route between start and target point, where optimality can be defined in numerous ways. It can be applied to path planning of different objects in various environments under different conditions. Here, the robot is most widely understood as any moving object, but path planning can also refer to motion planning of different artificial hands that are used in the industry. During path planning process many demands and constraints must be taken into consideration. The path along which the robot moves must be the shortest possible so the fuel consumption remains at minimum, and the time needed to reach the target point is minimized. Moreover, the collisions between robot and obstacles or other robots must be avoided, and in the case of UAV vehicles, the exposure of these vehicles to threats, such as radars, missiles or hostile aircrafts, also must be minimized. In addition, path planning for UAVs has even more constraints which must be taken into consideration such as: turning angle, climbing/diving angle, flight altitude etc. Path planning can be understood as a multi-objective constrained optimization problem which includes different objectives, as well as numerous demands and constraints. Unfortunately, these objectives and constraints are often mutually exclusive, thus compromises must be made during path planning process. Path planning problem is a hard optimization problem so there is no successful deterministic method to solve it in reasonable time. Although different methods and techniques were proposed to solving path planning problem, the application of swarm intelligence algorithms to robot path planning is not sufficiently researched. Brain storm optimization algorithm is swarm intelligence algorithm inspired by the human idea generation process during problem solving, i.e. brainstorming process. It was designed by Yuhui Shi in 2011. and since it was successfully applied to solving numerous optimization problems.

In this dissertation, we analysed methods of solving robot path planning problem using nature-inspired algorithms, especially swarm intelligence algorithms. Moreover,

we adjusted and applied brain storm optimization algorithm to robot path planning.

In the first part of our research we applied the original brain storm algorithm for UCAV path planning. Fuel consumption and safety were considered as performance criteria. The proposed method was tested in the environment from the literature, with circular danger zones and different threat degrees. Our proposed method was compared with ten other, nature-inspired metaheuristics from the literature. Based on the simulation results, it can be concluded that our proposed approach is robust, exhibits better performance in almost all cases and has potential for further improvements. It had better performance for smaller problem dimensions, while for larger problem dimensions more iterations were needed. However, the results were further improved compared to other algorithms.

In the second part of the research we applied the original BSO for robot path planning in uncertain environment with static obstacles. We proposed probabilistic model for determining danger degree for sources with unknown certain positions. Two contradicted criteria, path length and safety, are handled by introducing controlling parameter into the objective function. The proposed method deals with infeasible solution by adding penalty to the objective function. Penalty combined with increased exploration was capable to ensure that feasible solutions are always generated. Comparison with concurrent algorithm proves that our proposed method is, even though simpler, more efficient and robust since it obtained best solutions in all cases.

Finally, we considered mobile robot robot path planning problem in two dimensional grid based space. Brain storm optimization combined with the local search method for finding the shortest path in the graph was used for searching the optimal path in environments with static obstacles. Only path length was used as objective. Initial feasible solution for the BSO were generated by local search deterministic procedure and the BSO was used to further optimize the path. The proposed method was tested in five different scenarios and proved to be able to find the optimal feasible solution.

A future work can include: hybridization or modification of the brain storm optimization algorithm in order to improve the convergence speed and to adjust it for larger dimensional problems; analysis of path planning problem in 3D environment; and designing algorithms for self-adaptive and collaborative path planning. Future

research can introduce numerous improvements. Since paths are far from random collections of points, with many inherent relations and dependencies, it may be possible to exploit that and guide path formation in order to make more efficient algorithms. In further research, instead of grid based environment model, real search space can be used and the initial points can be obtained by some guidance instead of using randomly deployed points in the search space.

Keywords: robot path planning, brain storm algorithm, swarm intelligence, nature-inspired metaheuristics
Scientific field: Computer science
Scientific discipline: Artificial intelligence
UDC number:

* * * * * * * * *

First, I would like to thank my mentors, prof. dr Milan Tuba and prof. dr Edin Dolićanin for help and guidance during PhD research. I also thank prof. dr Boško Nikolić for comments and useful suggestions. Special thanks go to my family, father Salih, mother Magbula, sister Azra, and brother Ferid, for their constant and unconditional support that gave me motivation and strength to persist on the path of science and knowledge.

Contents

1 OPTIMIZATION

Most resources we use in everyday life (e.g. time, money, etc.) are limited. In many activities we perform, we are making an effort to optimize those limited resources in order to use them in the best way possible. By optimization here we mean finding the best solution for the given context, which consists of: feasible solution set, defined objectives, and constraints [1]. Thus, optimization as an activity is all around us; it is being used in the economy, computer networks, energy industry, transportation, but also in everyday life, for planning a field trip or a vacation. Optimization is crucial in all fields and activities which require decision-making because it is directly correlated with the act of choosing between several possible alternatives. The choosing of one option (or more) from several alternatives i.e. solutions to the problem, is determined by our desire to make the best possible decision, where the measure of decision quality is defined by the objective we are trying to reach, or if speaking in mathematical terms, objective function.

Most optimization problems can, from a mathematical point of view, be described in the following way [2]:

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}} \quad f_i(\mathbf{x}), \qquad (i = 1, 2, \ldots, M), \tag{1.1}$$

$$\text{subject to} \qquad h_j(\mathbf{x}) = 0, \qquad (j = 1, 2, \ldots, J), \tag{1.2}$$

$$g_k(\mathbf{x}) \leq 0, \qquad (k = 1, 2, \ldots, K), \tag{1.3}$$

where $f_i(\mathbf{x})$, $h_j(\mathbf{x})$, and $g_k(\mathbf{x})$ are functions of design vector

$$\mathbf{x} = (x_1, x_2, ..., x_n)^T \tag{1.4}$$

Vector $\mathbf{x}$ in (1.4) is actually the solution to the optimization problem, and its components x_i are called decision variables or design variables. Function $f_i(\mathbf{x})$ is called the objective function, fitness function, criterion function or performance criteria. The goal of optimization is finding that solution which has the minimum (as a result of a minimization process) or the maximum (as a result of a maximization process)

value of the objective function, as well as the value of criterion functions in that case. Equation (1.1) is given for a case of a minimization problem. The space which is defined by values of the decision variables is called search space or design space $\mathbb{R}^n$, and the space which is defined by values of the objective functions is called solution space. Functions $h_j(\mathbf{x})$ and $g_k(\mathbf{x})$ represent constraints, while functions $g_k(\mathbf{x})$ can also be defined as $g_k(\mathbf{x}) \geq 0$.

1.1 Classification of optimization problems

Optimization problems can be classified based on various criteria. Depending on the number of objectives, optimization can be single-objective, if $M = 1$, or multi-objective, if $M > 1$. Most real-world problems are multi-objective, and they are known to be more complex and harder to solve. If the parameters for equations (1.2) and (1.3) are $J = 0$ and $K = 0$, then we have an unconstrained optimization problem, otherwise it's a constrained optimization problem, where if $J = 0$ and $K \geq 1$ it is called an inequality-constrained problem, and if $K = 0$ and $J \geq 1$ it is called an equality-constrained problem. A part of the search space in which all constraints are met is called feasible space. In the case of a constrained optimization problem, the search for the optimal solution would only make sense within this space.

Classification can also be done on the basis of function forms of the objective functions as well as the functions which define constraints. If all of the functions $h_j(\mathbf{x})$ and $g_k(\mathbf{x})$ are linear, then it's called a linearly constrained problem. Additionally, if the objective functions $f_i(\mathbf{x})$ are also linear, then it becomes a linear programming problem. When discussing classification based on forms of the objective functions and constraints, we mustn't fail to mention the following types of optimization: quadratic programming (objective functions are quadratic functions and constraints are linear), nonlinear programming (objective functions and constraints are nonlinear), geometric programming (objective functions and constraints are posynomials, i.e. polynomials with positive coefficients), convex optimization (objective functions are convex), non-smooth optimization (objective functions and/or constraints are not differentiable). Local optimization is the one in which the search is being performed near the optimal point, while global optimization takes into account the entire domain of the objective function [3].

Classification can also be performed based on variable type. If the variables are continuous, the optimization is referred to as a continual optimization, and if the variables are discrete (e.g. binary or integer) the optimization is referred to as a discrete optimization. Two well-known branches of discrete optimization are integer programming and combinatorial optimization. Integer programming refers to the problems in which all of the variables are integers. If instead of integer or binary variables, discrete structures such as graphs, matroids, etc. are being used, then we have combinatorial optimization. These two branches are interrelated because a lot of combinatorial problems can be modeled as integer programming [4]. In other words, discrete or combinatorial optimization is referring to a search problem with the intent of finding the optimal solution within a finite or countably infinite set of potential solutions. The solutions to these problems can be combinatorial structures such as sequences, combinations, subsets, subgraphs, etc. In order to define the concept of a local minimum in such problems, we first need to define an appropriate metric to express the distance between the solutions, that is, to define the concept of neighborhood in the solution space [5]. It is important to emphasize that these metrics are dependent on the problem that is being solved.

In deterministic optimization, parameters of the optimization problem (variables or functions) are known in advance and the procedure follows a repeatable and a mathematically rigorous procedure in an attempt to avoid any randomness. In this type of optimization, starting from the same point will always lead to the same solution. When the system parameters are random (due to random noise, etc.) or the problem-solving methods involve making random decisions during the search, then such optimization is called stochastic optimization. Unlike unimodal problems which have one solution, multimodal optimization problems can be defined as problems with more global or local optimums, i.e. optimal solutions. For these types of problems it is of interest to obtain the largest amount of solutions possible, for several reasons; on one hand, when the knowledge of a problem is incomplete, the given solution doesn't have to be the best one, because one cannot claim that a better solution can't be found in the search space that has not completely been searched; on the other hand, even if we are certain that the given solution is the best one, there could also be other solutions that are just as good, or even solutions that are slightly worse but would be

given preference for a number of reasons (easier interpretation, easier implementation, etc.), thus making these solutions globally better [6].

In multi-criteria (multi-objective) optimization it's often the case that there is no one single solution that's best according to all criteria, meaning that one solution can be optimal for one criteria function, but not for another. For that reason, a concept for measuring the optimality of the solution, called Pareto optimality, was introduced. Feasible solution $x^* \in X$ is Pareto optimal (efficient, dominant, non-dominated) if there is no other solution $x \in X$ which meets both of the following conditions:

$$f_k(x) \leq f_k(x^*), \quad \text{for} \quad k = 1, 2, \ldots, p \tag{1.5}$$

$$f_i(x) < f_i(x^*), \quad \text{for some} \quad i \in \{1, 2, \ldots, k\} \tag{1.6}$$

where p represents the number of criteria, i.e. objectives, and X is the set of feasible solutions. The solution x^* has weak efficiency if there isn't an x for which the following is true:

$$f_k(x) < f_k(x^*), \quad \text{for each} \quad k = 1, 2, \ldots, p \tag{1.7}$$

A non-dominated solution set is a set which includes all of the solutions which are not dominated by any element of the solution set. A Pareto-optimal set is a non-dominated set of the entire feasible decision space, and the limit defined by the points of the Pareto-optimal set is called the Pareto-optimal front. Methods of solving multicriteria optimization problems can be divided into scalarization and Pareto methods. The first group includes the weighted sum approach, compromise programming, multiattribute utility analysis, goal programming, lexicographic approach and fuzzy logic. The most well-known Pareto methods are exploration and Pareto filtering, weighted sum approach (with weight scanning), adaptive weighted sum approach, normal boundary intersection, and multi-objective nature-inspired metaheuristics.

An algorithm can be defined as a tool or a procedure for solving a well-specified computational problem. An algorithm takes a set of values as input and then processes them through a series, or a sequence, of computational steps in order to produce a result, i.e. output. Considering that the desired relationship between input and output is specified by the statement of the problem, the goal of the algorithm is to define and describe the computational procedures for achieving the desired relationship [7].

Time and space are the two most valuable resources which the algorithm utilizes. The time complexity of an algorithm is defined by the number of steps (in the worst-case scenario) that are needed in order to solve a problem whose size is n. While determining the time complexity of an algorithm, we are not discovering the exact number of steps needed to solve a problem, but rather an asymptotic bound, and thus we use Big-O notation.

Algorithms whose complexity is lesser or equal to polynomial are considered good enough or suitable for application. Their complexity is labeled as $O(n^k)$, where k is a constant and n is the size of the input. Unfortunately, there is a subset of problems that do not belong to the aforementioned category, but it is difficult to determine which problems from that subset can be solved in polynomial time and which ones cannot. Let us consider decision problems, i.e. problems whose output is a single binary value: YES or NO. Two basic classes of decision problems can be defined:

- P (polynomial) - decision problems that can be solved in polynomial time;
- NP (nondeterministic polynomial) - problems with the following characteristic: if the answer is YES, then we can verify this answer in polynomial time. In other words, if there is a solution to the problem, we can determine if that solution is accurate within polynomial time.

Although most consider P and NP classes of problems to be different, this question has not yet been answered in a form of a precise proof. If the existence of a polynomial algorithm for a certain problem X implies the existence of a polynomial algorithm for every NP problem, then it is said that X belongs to the class of NP-hard problems. In other words, NP-hard problems are at least just as hard as NP problems. NP-complete problems are problems for which the following applies: if a polynomial algorithm for one NP-complete problem exists, then it implies that a polynomial algorithm for every NP-complete problem exists.

1.2 Methods for solving optimization problems

Conventional ways of solving optimization problems can be divided into following categories: calculus-based methods, random-based search methods, or enumerative search techniques. On the other hand, optimization methods can also be classified

depending on whether derivatives of the objective function are used or not, and so we differentiate between derivative and non-derivative methods. Derivative methods, which include the Newton method, the Gauss-Newton method, etc., are based on mathematical analysis, i.e. gradient search. These classical methods have proven to be good for solving a wide range of optimization problems when the number of decision variables isn't very large. However, real-world problems are usually discrete and have a large number of decision variables, meaning that the search space is extremely vast [8]. For solving these types of problems the classical methods are unable to provide an acceptable level of performance and thus we resort to other, more modern methods, which include heuristics, or metaheuristics.

Practical problems we encounter in the optimization process differ in complexity, thus making it very hard to find one method or algorithm that would be suitable for solving all types of problems. No-Free-Lunch theorems (NFL) prove that a universal algorithm for all problems doesn't even exist. NFL theorems hold for both deterministic and stochastic optimizations, where the set of parameters is continuous, discrete or mixed, and the set of values of the objective function is finite. The main claim of these theorems is the following: if one algorithm X exhibits better performance compared to another algorithm Y in searching for a minimal or maximal value of the given objective function, then algorithm Y will exhibit better performance in a case of another objective function. For a given problem, the choice of a suitable algorithm for solving the problem depends on the user experience as well as the properties of the algorithm itself, such as computational cost, availability of the software, computing time, etc. If, for a given algorithm, we want to determine the types of problems it would be successful at solving, we need to test the behavior of the algorithm in solving several problems of different characteristics, and then compare the performance of the given algorithm to the performance of other algorithms.

1.3 Test functions

Generally speaking, the performance evaluation of an optimization algorithm can be done using known real-world problems or using artificial problems. Real-world problems can be taken from different fundamental science fields, such as mathematics, physics, chemistry, or from applied fields such as engineering, medicine, etc. Usu-

ally, the mathematical model for such problems is complex, because it is described using complicated mathematical expressions and it is difficult to implement. On the other hand, artificial problems are test or benchmark functions which can have different properties, e.g. a single global minimum, multiple global or local minima, flat surfaces, etc. [9]. These problems can relatively easily be modified and adapted in order to test the behavior of optimization algorithms in various scenarios. Up until now many test functions were developed and proposed, although a standard list of benchmark functions for every algorithm does not exist. When we want to evaluate the performance of an algorithm, we actually want to determine problems for which that algorithm is more efficient in comparison to other algorithms. For that reason, the function set which we use in the process mustn't be too specialized and the functions mustn't be similar in their properties. Therefore, the collection of benchmark functions has to include different functions, which will be used to test the efficiency of the algorithm for different problems, such as unimodal, multimodal, multidimensional, etc. This type of approach is the only one that will allow for a clear and a precise determination of the set of problems that the algorithm is suitable for.

In practice, performance evaluation of algorithms is often done with a set of functions established and used in scientific conferences and congresses that deal with the field of optimization, such as the IEEE CEC (Congress on Evolutionary Computation). In literature, these sets of functions are labeled by names which include the abbreviated name of the congress and the year in which it was held (e.g. CEC 2015).

1.4 Heuristics and metaheuristics

In many real-world situations we are faced with extremely complex problems. Application of optimization methods and techniques is necessary in those situations, although using these methods does not guaranty that the optimal solution will even be found. To make the challenge greater, these problems often belong to the NP-hard class of problems - ones for whom effective algorithms cannot be created. A great number of these problems have to be solved by trial and error, using different optimization techniques [10]. In literature, one can find a great deal of NP-hard optimization problems that belong to different fields, such as graph theory, network design, mathematical programming, sequencing and scheduling, logic, games, and puzzles, etc. [11]. In order to

find a good-enough or acceptable solution to these problems, metaheuristics are often used. Generally speaking, there are two types of stochastic algorithms - heuristics and metaheuristics. Literature does not provide a precise definition of these two terms, so they cannot even be clearly distinguished from one another and thus are often used as synonyms. The term "heuristic" comes from the Greek language and it means "the art of discovering new strategies for problem-solving". Usually, by heuristics we mean a discovery by trial and error. Some of the most well-known heuristic methods are hill climbing, breach and bound method, $A*$, etc. Through further development of heuristic algorithms, metaheuristics were born. The prefix "meta-" also comes from the Greek language and it means "methodology of a higher level". Metaheuristic search methods can be defined as general higher-level methods that can be used as guiding strategies in designing heuristics for solving specific optimization problems [12]. In other words, unlike heuristics which are made and adapted to fit a specific problem, metaheuristic algorithms and methods are general strategies for solving a wider range of problems. The solution to a complex optimization problem acquired by the application of metaheuristics is considered "good enough" and is acquired within a reasonable timeframe. Unlike traditional methods, metaheuristic algorithms make no guarantees that the optimal solution will be found. Also, unlike approximation methods of optimization, metaheuristics provide no information on how close the given solution was to the optimal solution.

In all metaheuristics, a compromise between randomization and local search is made. In the last couple of years, there is even a tendency to classify all stochastic algorithms which are based upon randomization and local search as metaheuristics. The simplest way to implement stochasticity or randomization during a search is a random walk. It is done by randomly going from one place in the search space to another in order to find the optimal solution. It is also important to point out that the majority of metaheuristics are inspired by nature, meaning that they are based off of the principles derived from physics, biology or ethology, i.e. animal behavior.

Generally speaking, metaheuristics can be divided into the following categories: single-solution based and population-based. Single-solution based methods are also called trajectory methods, because during their execution the algorithm creates a trajectory in the search space, moving from one - initial solution, to other solutions that

could be located anywhere in the search space. The most well-known algorithms in this category are simulated annealing (SA), tabu search, GRASP method, variable neighborhood search (VNS), as well as different forms of local searches [13]. Population-based metaheuristics use several agents, i.e. a population of individuals, where the individuals of a population share information amongst each other, thus searching for the optimal solution collectively. In this case, multiple trajectories in the search space are generated. Evolutionary computing (EC) and swarm intelligence (SI) are the two main groups of methods which belong to population-based metaheuristics. Evolutionary computing is based on Darwin's process of evolution, where mutation and recombination are applied to individuals, and those individuals who exhibit the best qualities (as defined by the objective function) are selected for the next generation. Swarm intelligence is a collection of metaheuristic optimization methods inspired by the collective behavior of different animal species, which, through collaboration and information sharing, exhibit the intelligence needed for complex problem-solving.

Considering that all algorithms, including metaheuristics, have deficiencies and disadvantages, combining two or more of complementary algorithms in order to create new hybrid algorithms is often something that is resorted to. These algorithms combine the advantages of each individual primary algorithm while minimizing the effect of their disadvantages. The result of hybridization is often enhancement of performance compared to individual primary algorithms, in regards to computing time or accuracy. Hybrid algorithms can be collaborative and integrative. Collaborative hybrids consist of two or more algorithms running either in sequential or parallel. Collaborative work of the algorithms can be implemented in several ways: 1) multi-stage - the work is divided between algorithms, e.g. one algorithm is in charge of global optimization, while another algorithm performs local search; 2) sequential - algorithms run alternatively, e.g. each algorithm perform a certain number of iterations before proceeding to another algorithm, 3) parallel - all algorithms work in parallel, using the same population. In the case of integrative algorithms, one algorithm is regarded as the master algorithm, while the other is regarded as a subordinate algorithm. An example of such implementation is the incorporation of certain operators from one algorithm to another - primary algorithm.

1.5 Nature-inspired metaheuristics

In the last couple of decades, a new class of optimization algorithms, inspired by the processes that occur in nature, has gained in significance as well as popularity. Nature has two tremendous and efficient mechanisms: selection and mutation. Through selection, nature rewards those individuals who adapt better to their habitat, i.e. the stronger individuals who solve existential problems more efficiently. Mutation is a mechanism based on random processes which allow for new, different individuals to be born. These are the two mechanisms on which nature-inspired optimization and respective algorithms are based upon. Namely, the selection mechanism is the core principle of this type of optimization, while mutation is the mechanism on which stochastic search is based. There are four primary characteristics of nature-inspired metaheuristics [15]:

- they model a phenomenon which exists in nature;
- they are non-deterministic;
- they often present implicitly a parallel structure (multiple agents);
- they have an ability to adapt.

An ideal optimization algorithm would be the one which produces better solutions in each cycle, compared to the previous cycles. Also, it would be preferred if the best possible solution is reached after a minimal number of iterations. However, such algorithm has not yet been developed and probably isn't even possible. Stochastic algorithms are characterized by the fact that the new solutions aren't necessarily better than the previous solutions. Although at first this may seem illogical, during the execution of these algorithms it is necessary to sometimes, in a random way (using randomization), choose solutions which are not the best ones possible but will help the search process and prevent it from trapping in a local optimum. This stochastic mechanism, called exploration or diversification, is at the core of modern metaheuristic algorithms. This mechanism works on a global scale and brings diversity to the search process by enabling a more efficient search of distant regions in the search space, i.e. making the search global, and generating solutions which are distant and differ from the existing solution. In this manner, exploration reduces the probability of the

algorithm trapping in a local minimum. However, the downside of this process is slow convergence, because too much computing resources are being used to conduct a search through regions that may be very distant from the desired global optimum [16].

Another important mechanism of metaheuristics is exploitation or intensification. It refers to a local search, meaning, searching near the existing solutions, in order to exploit information so that new and better solutions could be generated [17]. Information that helps in this process is also local, for example, a gradient. The exploitation process accelerates the convergence of the algorithm, however, too much exploitation can lead to the algorithm trapping in a local optimum. Too much exploration and too little exploitation will increase the probability of reaching a global optimum, but the price of such an approach is slow convergence and large resource consumption. On the other hand, too much exploitation and too little exploration will increase the speed of the algorithm convergence but will decrease the probability of generating a solution that is also the best one in global terms. Taking into account the properties of the two aforementioned processes, their effects and consequences, it is obvious that an efficient algorithm must provide balance, that is - a compromise between the two [18].

Nature-inspired metaheuristics have the attributes of diversity and adaptation. Adaptation means that better performance can be achieved by changing the appropriate parameters such as population size or other, algorithm specific, parameters. As for diversity, one can say that diversity is a direct outcome of randomization and that it can be implemented in different ways, depending on the specific algorithm used, e.g. mutation in genetic algorithms. Of course, these two attributes are directly correlated because, for instance, a greater diversity can be achieved by applying adaptation to the algorithm through parameter modification [19].

Below are described several most known metaheuristics, those being: 1) simulated annealing (SA) algorithm, as a typical representative of the trajectory methods or individual metaheuristic population methods; 2) genetic algorithm (GA) and differential evolution (DE) - as representatives of the population methods. Swarm intelligence algorithms will be discussed in the next chapter.

1.5.1 Simulated annealing

Simulated annealing algorithm (SA) was proposed in the early 80s of the twentieth century by Kirkpatrick and others (1983), and then a couple of years later it was independently proposed by Cerny (1985). The inspiration behind this algorithm came from the process of annealing solid materials and the analogy between this process and the process of solving combinatorial optimization problems [20].

We know from solid-state physics that the annealing process consists of two stages. In the first stage, the solid material is heated until it reaches the maximum value and turns liquid, where the particles of the material are distributed in an arbitrary and random fashion. After this stage comes the slow, gradual cooling of the material, where it transitions to the solid state of a crystal structure with the lowest energy. In the cooling process, the system reaches a thermodynamic equilibrium at every temperature level, which is, in terms of energy, defined by the Boltzmann distribution. If the cooling process is slow enough, at a temperature close to zero degrees, the system will enter into a state of minimal energy. However, if the process is developing quickly, meaning that the system isn't allowed to reach thermodynamic equilibrium at every temperature level, what happens is that the material remains in a metastable amorphous state.

For simulating the evolution of a material to a state of thermodynamic equilibrium for a given temperature level, the Metropolis algorithm was proposed, who was named after a researcher who developed it in 1953. The algorithm is based on a Monte Carlo approach, which generates a sequence of states of the material in the following way. Suppose that the material is in some state i which is determined by its position, i.e. by the distribution of particles and system energy E_i. Changing the state of the system and transitioning into state j, with energy being E_j, is achieved by applying a small, randomly generated perturbation, for example, relocation of particles. If the difference between energies of the new and current state is $\Delta E = E_j - E_i \leq 0$, new state j is accepted as the current state, otherwise the new state is accepted with the probability of $\exp(\frac{-\Delta E}{k_B T})$, where k_B is the Boltzmann constant and T is the temperature level. This rule is also called the Metropolis criterion. Therefore, the state of thermodynamic equilibrium is reached through an iterative process, by generating a large number of transitions from one state to another, following the described procedure.

A combinatorial optimization problem can be viewed as an annealing process if the following two analogies are introduced: 1) solutions of the optimization problem are equivalent to the states of a physical system: 2) value of the objective function is equivalent to the energy of a state. In order to implement this algorithm, a control parameter c is needed, which controls temperature changes in the original Metropolis algorithm. In this way, the simulated annealing method can be observed as a search method, implemented as an iterative sequence of Metropolis procedures, which are run successively, so that in each succeeding execution value of the control parameter is less than the value of that parameter in the current execution. Algorithm 1.1 contains the pseudocode of the simulated annealing metaheuristic [21].

Algorithm 1.1: Simulated annealing

```
1  begin
2      Initialization (i_start, c_0, L_0);
3      k = 0;
4      i = i_start;
5      repeat
6          for i = 1 to L_k do
7              Generate (j from S_i);
8              if f_i ≤ f_j then
9                  i = j
10             else
11                 if exp (f_i − f_j)/c_k > random[0, 1) then
12                     i = j
13                 end
14             end
15         end
16         k = k + 1;
17         CalculateLength(L_k);
18         CalculateCtrlParam(c_k)
19     until stopping criteria;
20 end
```

The function Initialization() generates an initial solution as output and performs initialization of the control parameter c and parameter L, which represents the number of perturbations in the Metropolis algorithm. The function Generate() is used for generating new states (solutions) from the neighboring existing states, and functions

CalculateLength() and CalculateCtrlParam() compute new values for parameters c and L. The convergence speed of the algorithm depends on parameters c and L, and for specific implementations of the algorithm parameter c is modified using the following formula: $c_{k+1} = \alpha * c_k, (k = 0, 1, 2,)$, where the constant α takes values from the range 0.8-0.99. SA algorithm has found application in a wide range of single-objective and multi-objective combinatorial optimization problems. Additionally, a large number of hybrid algorithms have been developed, where one of the components of the algorithm is the SA algorithm [22].

1.5.2 Genetic algorithms

Processes of evolution and natural selection have inspired the development of genetic algorithms (GA). These algorithms belong to population-based, evolutionary meta-heuristics. Genetic algorithms have gained in popularity thanks to two comparative advantages they have over other algorithms: 1) the ability to solve complex optimization problems of different types and different objective functions; 2) parallelism, which is possible due to the fact that multiple genes are convenient for parallel optimization. Basic components of a GA algorithm are the so-called genetic operators: inheritance, mutation, crossover, and selection. At the beginning of the algorithm, the population of individuals is initialized, which is then further developed by applying a genetic operator. Individuals, i.e. solutions, are represented as a binary or decimal string called chromosome, while the genes are individual bits or a string of neighboring bits, which are used to encode a specific element of the solution [23].

New individuals are added with crossover and mutation operators. By applying the crossover operator to the parent individuals, new individuals are added to the population, where the probability of crossover is huge, between 0.7 and 1.0. The operation is performed by having the algorithm randomly chose a location within a chromosome and exchange the corresponding parent bit sequences prior to and following this location, in order to create two individuals. The concept of survival of the fittest and the best, which is present in the natural selection mechanism, is carried out by the selection operator in GA algorithms. In the simplest version of GA, algorithm selection is done by a proportionate scheme, which means that for an individual with a fitness value of f, f/f_{sr} offspring is allocated [24]. A solution

with the fitness value greater than f/f_{sr} is granted more than one offspring, and a solution with the fitness value below f/f_{sr} is granted less than one offspring. Taking into consideration that f/f_{sr} is the expected number of offspring, randomization is introduced in the final step of the decision-making in order to reduce bias towards certain individuals. The simplest way to implement a proportionate scheme is through roulette wheel selection. In this mechanism, each individual is granted a sector of the roulette wheel, which is proportionate to the value f/f_{sr}. A certain individual is allocated an offspring if a randomly generated value within the range of 0 to 2π belongs to the sector which was assigned to the individual. Aside from this mechanism, often used is also the tournament selection. Additionally, the elitist strategy is also used, which allows for the individuals with the best fitness function values to be copied to the next generation [25]. The mutation operation is performed by inverting some of the bits in the chromosome string, and has a smaller probability, usually between 0.001 and 0.05. The solutions are being evaluated based on the objective function of the given optimization problem, and the selection procedure chooses only the best solutions for the next generation [26]. Pseudocode of the basic GA algorithm is shown in Algorithm 1.2.

Algorithm 1.2: Genetic algorithm

```
1  begin
2      Define fitness function F;
3      InitializePopulation();
4      Initialize crossover probabilities p_c and mutation probabilities p_m;
5      EvaluatePopulation();
6      while stopping criteria not met do
7          Perform selection of individuals for the next generation;
8          Perform crossover with probabilities p_c;
9          Perform mutation with probabilities p_m;
10         EvaluatePopulation();
11     end
12 end
```

As with all other metaheuristics, selecting control parameters is an important topic in GA algorithms. If the crossover probability is too big, premature convergence and trapping in a local optimum can occur. On the other hand, if this probability is very small, crossover happens less frequently, which prevents evolution, making

the algorithm less efficient. Mutation is the operator which, essentially, performs exploration and provides solution diversity. A small mutation probability can prevent the generation of solutions that differ from the existing ones, and big probability values can result in the algorithm wandering too much in the search space and thus slowing down convergence, even in the cases where the solution is relatively close. The population size mustn't be too small as to not jeopardize the evolution and diversity and also to prevent premature convergence. However, too big of a population will result in the increase of computational cost. The best value for population size is between 40 and 200, depending on the optimization problem.

1.5.3 Differential evolution

Differential evolution (DE) is a parallel direct search method which belongs to the group of evolutionary algorithms. In this algorithm, population within any generation G is represented with a N D-dimensional vectors: $x_i, i = 1, 2, \ldots N$. In the initialization phase, the vector components are assigned random values from the uniform distribution, while the number of vectors N doesn't change during the execution of the algorithm. In case the preliminary solution is known, the initial population can be generated by adding random values from a normal distribution to the nominal solution $x_{nom,0}$ [27]. As it is the case with all evolutionary algorithms, DE has three main operators: mutation, recombination (crossover) and selection. The mutation operation is executed for each vector x_i by randomly choosing three different vectors x_t, x_r and x_s, and creating a mutated vector x_m:

$$x_m = x_t + F * (x_r - x_s) \tag{1.8}$$

where $F > 0$ is the weighting or scaling factor used to amplify differential variation of two vectors. Authors of the algorithm have proposed that the value of F stays within the range $[0, 2]$, while practice has shown that the most effective values rarely go beyond 1. Besides using equation (1.8), the mutation operation can also be implemented in other ways [28]. A new individual is generated after the crossover operation in the

following manner:

$$x_{new,j} = \begin{cases} x_{i,j}, & \text{if} \quad rand(0,1) < CR \\ x_{m,j}, & \text{otherwise} \end{cases} \tag{1.9}$$

where $rand(0,1)$ is a random variable from a uniform distribution, and CR is a crossover constant which takes values from range [0,1] and its value is user-defined. For each component j from the vector of a new individual x_{new}, it is determined, in a probabilistic way, if the value will be taken from the initial vector x_i or mutated vector x_m, which essentially is the way in which crossover is performed between them. Pseudocode of the basic DE algorithm is given in Algorithm 1.3 [29].

Algorithm 1.3: Differential evolution

```
1  begin
2      Pseudo randomly generate N individuals of initial population;
3      while stopping criteria do
4          for i = 1 to N do
5              Calculate f(x_i)
6          end
7          for i = 1 to N do
8              Select three individual vectors x_r, x_s, i x_t;
9              Calculate x_m using Eq. (1.8);
10             x_new = x_m;
11             for j = 1 to D do
12                 Generate rand(0,1);
13                 if rand(0,1)<CR then
14                     x_{new,j} = x_{i,j}
15                 end
16             end
17             if f(x_new) ≤ f(x_i) then
18                 Save index for replacement x_i = x_new
19             end
20         end
21         Perform replacement;
22     end
23 end
```

Taking into account that, in the initial iterations of DE algorithm solutions are widely distributed in the search space, step size $F(x_r - x_s)$ has greater values, thus

making the exploration process dominating. In the later stages of the algorithm, the solutions are concentrated in certain regions of the search space by gradually decreasing step size. In this way the search becomes localized, making the exploitation process dominant. Although this type of algorithm behavior is desirable, it carries within itself a certain limitation and a potential flaw. Namely, if the algorithm fails to produce new individuals which are better than current ones, step size will not decrease and stagnation will occur, which will prevent the algorithm from converging to suboptimal solutions.

It is obvious that algorithm performance depends greatly on the choice of control parameters, which include population size N, weighting factor F and the crossover constant CR. It was shown that selecting the values for parameters F and CR is extremely hard, and over the years researchers have proposed a huge number of solutions for adjusting their values, so that the efficiency of a DE algorithm is the best it can be [30].

1.6 Application of optimization

As already stated at the beginning of this chapter, optimization is applied in many fields and disciplines. Here we will give an example of one application, which was the subject matter of our scientific research over the years. Namely, the subject of optimization can also be redundancy in nanoelectronics (electronic components used in nanotechnology). The constant reduction in the dimensions of these components, together with the increase in environment's electromagnetic contamination and omnipresent secondary cosmic radiation, significantly reduces the reliability of these components. This especially comes to the fore in the production of nanocomputers. A solution for this type of problem is introducing redundancy to computer memory and other computer systems. However, this solution also brings about the increase in energy consumption, so the level of redundancy needs to be optimized while maximizing the reliability. In [31], the problem of optimizing the redundancy procedure of MOS memory structures in nanocomputers working in an environment with radiation was considered.

Based on the experiment results it can be concluded that a single MOS structure which is exposed to the effects of background radiation would have a 99% chance of

containing the wrong information. Based on that fact, it would be impossible to use nanocomputers for real-world applications. Thus, the idea of creating nanocomputers with individual integrated components was abandoned, and creating nanostructures with redundant components (both parallel and serial) was embraced. Of course, even though nanocomponents are very cheap, the number of ones that are redundant has to be kept at a minimum in order to reduce consumption of energy needed for computing and cooling of such a computer.

In a case of a parallel connection of discrete, independent MOS memories that are manufactured as nanocomponents, the enlargement factor is one-dimensional and equal to the number of memories used. Observing the mentioned example with the enlargement factor being n=10000 and the probability of an individual nanocomponent MOS memory containing the wrong information due to background radiation being 0.005, the probability of the wrong information being displayed in the entire, redundant memory circuit is $0.05 * 10000^{-1}$, or, if we assume that 100 individual nanocomponent MOS structures have the same probability of storing the wrong content, the information obtained from the redundant system is 0,000005 statistically reliable. In standard conditions, this is a satisfactory level of statistical reliability.

2 SWARM INTELLIGENCE

2.1 Introduction

Swarm intelligence is a nature-inspired paradigm for solving optimization problems. The idea for its implementation came from observing biological systems, primarily, social insects such as ants, bees, fireflies, as well as other animals, for example, a flock of birds or a school of fish. Besides this, the inspiration for such algorithms also came from other natural phenomena, such as firework explosion, or social phenomena, such as the behavior of people when generating ideas and problem-solving. The main characteristics of these algorithms are self-organization, coevolution, distribution and decentralized approach.

Many biological organisms, when in a group, behave in such way where they make decisions based on local information they gather from their external surroundings, as well as through interaction with other organisms from the group. These interactions can eventually lead to the phenomenon of collective or social intelligence. It is assumed that this happens because the biological changes in organisms are the result of their adaptation to changes in their environment and in the group they belong to, thus the occurrence of intelligence in developed biological species is a direct or an indirect consequence of complex interactions. Research has shown that groups of organisms belonging to the same species have the ability to solve complex tasks using collective intelligence, i.e. swarm intelligence [32]. Based on the swarm intelligence phenomenon, scientists have developed a large number of methods and algorithms for solving complex problems, which include optimizations based on insect behavior or other animal behavior. Application of these algorithms makes sense because the iterative process of the algorithm resembles the self-organizing evolution of the system [33]. Considering that randomness is an inherent property of these algorithms, it increases the probability of the solution not get trapped in a local minimum, meaning that the algorithm will converge faster to a global optimum solution. Self-organization in swarms is characterized by four properties [34]:

1. Positive feedback: it is used to promote and encourage the creation of suitable structures. Pheromone trails, which the ants leave behind, are a good example of positive feedback;

2. Negative feedback: it counterbalances positive feedback and helps stabilize the population. This mechanism is needed in order to avoid saturation, e.g. in the number of food foragers;

3. Fluctuation: the randomness in movement, task switching, etc. is a vital component to preserving the creativity because it enables for new solutions to be found;

4. Multiple interactions: each individual uses, sends and receives information from other swarm (flock) members, and in that way the information is spread throughout the swarm.

In essence, a swarm intelligence system is based upon very simple rules. Individual agents, e.g. ants or bees, make decisions based on local information, where centralized control or an entity that controls the way agents should behave is nonexistent. Agent behavior is local, random to a certain extent, but the interactions between agents lead to self-organization and a global "intelligent" behavior which is not familiar to individual agents. The conditions which are necessary in order to establish self-organization within a system are feedback, stigmergy (when individuals communicate indirectly through modification of their local environment), multiple interactions, memory and conditions in the environment [34].

In swarm intelligence algorithms each individual represents a solution in the search space. These algorithms have to encompass two types of abilities: capability learning and capacity developing [35]. Capacity developing refers to the process of moving the algorithm search towards areas which have a greater searching potential, while capability learning focuses on the search itself, by starting from the current solution, for single-point optimization algorithms, or starting from the current population, for population-based swarm intelligence algorithms. Swarm intelligence algorithms which have both capabilities are called developmental swarm intelligence algorithms (DSI).

Ant colony optimization (1992), particle swarm optimization (1995), bacterial foraging (2002), artificial bee colony algorithm (2005), monkey search (2007), firefly algorithm (2008), cuckoo search (2009), glowworm algorithm (2009), bat algorithm (2010), fireworks algorithm (2010), brain storm algorithm (2011), krill herd algorithm (2012), gray wolf optimization (2014), lion optimization algorithm (2016) are just a

few of the most known algorithm in the class of swarm intelligence algorithms. In this chapter, a brief description of the most significant swarm intelligence algorithms will be provided, and the brain storm algorithm will be described in detail in the next chapter.

2.2 Ant colony optimization algorithm

Ant algorithms have been developed as an approach to solving multi-agent combinatorial optimization problems such as the travelling salesman problem or quadratic assignment problem. The inspiration for developing these algorithms came from observing ant colony behavior, more precisely, how ants find the shortest path from their nest to the food source [36]. Analogous to biological ants, ant colony optimization algorithm is based on indirect communication of a colony comprised of simple agents, which are called artificial ants, through artificial pheromone trails. These trails are used as distributed numeric information which the ants use in order to construct a solution to a problem in a probabilistic fashion, and are adapted and updated by the ants during the algorithm execution, in order to better represent their search experience [37]. ACO algorithms use the ant population to collectively solve a problem. Information that is collected by the ants in the search process is being kept in the pheromone trails, which are assigned to respective arcs, and these trails represent the long-term memory of the given search process. An ant colony has the following characteristics [38]:

- Ants search for a solution which has the lowest cost;
- An ant k has a memory M_k, which he uses to save information about the path he has covered until the current moment. Memory is used in order to generate feasible solutions, for the evaluation of found solutions, and also for backward path restoration;
- Ant k in a state s can move towards any node j in his feasible neighborhood;
- An ant can be assigned a start state and one or more termination conditions;
- Ants start from the start state and move towards feasible neighbor states, thus building the solution in an incremental manner. The procedure stops if for at

least one ant at least one termination condition is satisfied;

- Ant k located in node i can move towards chosen node j from his feasible neighborhood, where the movement is defined by the probability decision rule;
- The probability decision rule is a function of the following: (1) values which are stored in a local node data structure, which is called ant-routing table, and is obtained by functional composition of pheromone trails, that are locally available to the node and heuristic values, (2) ant's private memory which preserves the movement history of the ant, and (3) defined problem constraints;
- When moving from node i to the neighboring node j, the ant can update the pheromone trail on that link. This is called step-by-step pheromone update;
- When the solution is created, the ant can retrace the same path backward and update pheromone trails on the links. This is called delayed pheromone update;
- After this, the ant dies and frees the allocated resources.

The central component of the ACO algorithm is parameterized probabilistic model which is called the pheromone model. This model consists of model parameter vectors called pheromone trail parameters, which are usually connected to the solution components and have values called pheromone values. The pheromone model is used for probabilistic generation of solutions from a finite set of solution components. During the execution of the algorithm, ACO updates pheromone values using previously generated solutions. The goal of the update is to focus the search on regions which contain quality solutions [39]. The general structure of ACO optimization algorithm is given in Algorithm 2.1.

After the parameter and pheromone trail initialization, the main part of the algorithm consists of three steps, which are repeated in each iteration of the algorithm, until one of the termination conditions is reached. Firstly, a solution is constructed based on the information obtained from pheromones and, eventually, available heuristic information. When the ants complete the construction of the solutions, optionally, they can be improved in the daemon action step, which is a local search. In the end, pheromone values are updated in order to reflect the search experience [40]. The solution construction is performed incrementally by having each ant start off with an

Algorithm 2.1: ACO algorithm

```
1 begin
2     Initialization;
3     while stopping criteria not met do
4         ConstructingSolutions;
5         DaemonActions (optional);
6         PheromoneUpdate;
7     end
8 end
```

empty solution (s_p is an empty set), and then, during the execution of the algorithm, build his partial solution and expand it by adding one feasible component (c_i^j) from the component set $N(s_p)$. The given set is implicitly determined by the process of solution construction which the ant implements. The decision on which component from $N(s_p)$ will be selected is made in each step of the solution construction, and is done in a probabilistic way, by applying the so-called transitional probabilities, while staying in-line with the pheromone model. There are different ways to define probability distributions which are used in that case, but most ACO algorithms use the ant system rule, which is defined by the following equation [41]:

$$p(c_i^j \mid s_p) = \frac{\tau_{i,j}^{\alpha}[\eta(c_i^j)]^{\beta}}{\sum_{c_i^l \in N(s_p)} \tau_{i,l}^{\alpha}[\eta(c_i^l)]^{\beta}} \tag{2.1}$$

where: $\eta()$ is a function which assigns each feasible component of the solution c_i^j a heuristic value (heuristic information); α and β determine the influence of pheromone trails and heuristic information. Different versions of the ACO algorithm differ by the way in which they update pheromone values they use. The goal of pheromone updates is to make those components which belong to good solutions the ones which will be preferred and selected by other ants, in the following iterations. The pheromone update mechanism is done in two parts. First, the pheromone evaporation operation is carried out, which represents a uniform decrease of pheromone values which have been set by the ants in the previous iteration, all with the goal of avoiding rapid convergence of the algorithm in the suboptimal regions. This operation applies a useful form of forgetting, in a way that favors new regions in the search space. The

second operation is the pheromone deposit, which increases pheromone levels of those solution components which belong to the set of good solutions, S_{upd}. Pheromone update is usually done according to the following equation [42]:

$$\tau_{ij} = (1 - \rho) + \sum_{s \in S_{upd} | c_i^j \in s} g(s) \tag{2.2}$$

where ρ is the evaporation rate, and $g(s)$ is the evolution function.

Daemon actions are centralized procedures which cannot be carried out by the ants themselves. An example of this action is the application of additional pheromone deposit to the solution components which have proven to be the best ones. Also, constructed candidate solutions can additionally be improved by applying the local search algorithm as a daemon action.

2.3 Particle swarm optimization

Particle swarm optimization algorithm (PSO) is a stochastic method of optimization developed by Eberhart and Kennedy in 1995. It is based on simulation of animal groups, such as bird or fish, which apply the principle of cooperation when searching for food, in a way that each member of the group continually changes its way of searching in accordance to the information it receives from the surrounding. In order to better understand the creation and development of the particle swarm optimization algorithm, we will get acquainted with a simple Boid model [43], which was introduced in order to simulate the behavior of birds and has been used as a basis for particle swarm optimization algorithms. In this model each bird is represented by a point in the Cartesian coordinate system, which is assigned an initial position and velocity. Considering that the simulation is being carried out in accordance to the rule which states that the individual gravitates towards having the same velocity as the closest neighbors, very soon all the points will have the same velocity, which is a simplified model that does not reflect any realistic scenario. In order to prevent this, meaning to make the simulation better reflect a real-world situation, a random value is added to velocity values in each iteration.

In the Heppner corn model [44] it is assumed that, in the beginning of the simulation, food position, as well as bird position and velocity are randomly determined.

Coordinates of the cornfield are (x_0, y_0), while the coordinates of bird position and bird velocity is (x, y) and (v_x, v_y), respectively. Performance of the current bird position and velocity is determined based on the distance between the birds and the food; the further the distance the better the performance, and vice versa. If we assume that birds have the ability to remember the best state and adopt the following notation: *pbest* - best position, a - constant for adjusting velocity, *rand* - random number in the interval $[0, 1]$, then the following applies:

$$v_x = \begin{cases} v_x - rand * a, & \text{if} \quad x > pbestx \\ v_x + rand * a, & \text{otherwise} \end{cases} \tag{2.3}$$

$$v_y = \begin{cases} v_y - rand * a, & \text{if} \quad y > pbesty \\ v_y + rand * a, & \text{otherwise} \end{cases} \tag{2.4}$$

If we also assume that birds within a swarm can communicate and that they remember the best swarm position (*gbest*), then the following also applies:

$$v_x = \begin{cases} v_x - rand * b, & \text{if} \quad x > gbestx \\ v_x + rand * b, & \text{otherwise} \end{cases} \tag{2.5}$$

$$v_y = \begin{cases} v_y - rand * b, & \text{if} \quad y > gbesty \\ v_y + rand * b, & \text{otherwise} \end{cases} \tag{2.6}$$

Starting from previously mentioned models, Kennedy and Eberhart have made a new optimization algorithm which they called particle swarm optimization algorithm because in their solution individual birds are observed as particles without mass or volume, where of interest are only their velocity and position. The rule by which particle velocity and position is updated in this algorithm can be described by the following expressions [45]:

$$v_x = v_x + 2 * rand * (pbestx - x) + 2 * rand * (gbestx - x) \tag{2.7}$$

$$x = x + v_x \tag{2.8}$$

In the particle swarm optimization algorithm, each particle represents a potential

solution to the optimization problem in D-dimensional search space. Considering that each particle remembers its best position, as well as the best position of the flock, in every iteration, based on the given information, particles can calculate a new velocity, thus determining a new position. PSO algorithm is performed in a way that particles adjust their velocity in every iteration, which makes them move in the direction of the prior best position (*pbest*) and globally the best position in the flock (*gbest*), where [46]:

$$\forall i \in \{1, 2, \ldots, N\} \; pbest(t) = arg \; min[f(P_i(k))], k = 1, 2, \ldots, t \tag{2.9}$$

$$gbest(t) = arg \; min[f(P_i(k))], i = 1, 2, \ldots, N, k = 1, 2, \ldots, t \tag{2.10}$$

where: i is the particle index, N is the total number of particles, t is the current iteration, f is the fitness function, and P is the particle position. Generalized formulas for updating particle velocity and position are [46]:

$$V_i(t+1) = \omega V_i(t) + c_1 r_1 (pbest(i,t) - P_i(t)) + c_2 r_2 (gbest(t) - P_i(t)) \tag{2.11}$$

$$P_i(t+1) = P_i(t) + V_i(t+1) \tag{2.12}$$

where V is the particle velocity, ω is the weight factor of the iteration, which is used for balancing global exploration and local exploitation, r_1 and r_2 are random variables from a uniform distribution, from the range $[0, 1]$, and c_1 and c_2 are positive constants which are called acceleration coefficients. Pseudocode of a standard PSO algorithm is given in Algorithm 2.2 [47].

In an effort to improve the original PSO algorithm a great deal of research has been done and a great number of different versions have been proposed. Modifications of the PSO algorithm can be done through modifying several aspects of the algorithm: algorithm structure, change of parameters or change of topology. Modification of the algorithm structure can be implemented in several ways, from which the most known are: introducing subpopulations, changing strategies for particle velocity or position updates, applying techniques for preserving population diversity or by combining with other nature-inspired algorithms in order to create hybrid algorithms. Likewise, structure changes are also necessary when solving multi-objective and multimodal problems, as well as problems which include discrete or binary variables.

Algorithm 2.2: Standard PSO algorithm

```
1  begin
2      For each particle i = 1, 2, ..., N:
3      Initialize particle positions with uniform distribution from the interval
         [LB, UB], (0) ~ U(LB, UB), where LB and UB represent lower and
         upper limit of search space;
4      Initialize pbest as pbest(i, 0) = P_i(0);
5      Initialize gbest on minimal value of the swarm: gbest(0) = arg minf[P_i(0)];
6      Initialize velocities: V_i ~ (-|UB - LB|, |UB - LB|);
7      while stopping criteria not met do
8          For each particle i = 1, 2, ..., N:
9          Randomly select values r_1, r_2, that belong to uniform distribution,r_1,
             r_2, ~ U(0, 1);
10         Modify particles' velocities according to formula (2.7);
11         Modify particles' position according to formula (2.8);
12         if f[P_i(t)] < f[pbest(i,t)] then
13             Update best position of a particle i: pbest(i,t) = P_i(t);
14             if f[P_i(t)] < f[gbest(t)] then
15                 Update best position in a swarm: gbest(t) = P_i(t)
16             end
17         end
18         t ← (t + 1);
19     end
20     Output value gbest(t) that contains the best solution;
21 end
```

2.4 Artificial bee colony algorithm

A food-seeking bee model has the following three components: food sources, employed foragers, and unemployed foragers. Goals of a bee colony are cooperative work in finding good food sources and abandoning food sources which lack quality. Food sources are estimated based on several characteristics, such as distance from the beehive, location, nectar richness and ease of extracting food. Employed foragers are those who are currently employed at extracting nectar, and then share information about the food source with other bees. Unlike employed foragers, the goal of unemployed foragers, who can be ether scouts or onlookers, is to constantly search for food or wait in the beehive and share information they receive from employed foragers [48]. Information about food sources is carried on by forager bees to other bees in the hive using a

particular dance. In the beginning, the role of scout bees is to find a location where the food source is located and pass on that information to the employed bees and onlookers, which then perform exploitation of the given source. When one food source is completely exhausted, employed bees become scouts and the process of finding a new food source begins all over again. Inspired by this process, the artificial bee colony optimization algorithm (ABC) was developed, where the position of a single food source represents one solution to the problem, and the quality of food (nectar richness) corresponds to the fitness function. The general structure of the ABC algorithm is given below [49].

Algorithm 2.3: General structure of ABC algorithm

```
1 begin
2     Initialization phase;
3     repeat
4         Employed bees phase;
5         Onlooker bees phase;
6         Scout bees phase;
7         Memorize the best solution until current moment;
8     until maximal number of iterations reached or maximal CPU time
        reached;
9 end
```

In the initialization phase, the ABC algorithm at first generates a uniformly distributed population which consists of SN solutions. Each solution x_i $(i = 1, 2, \ldots, SN)$ is a D-dimensional vector, where D is the number of variables in the given optimization problem. Also, x_i represents i-th food source, which is generated in the following way [50]:

$$x_i^j = x_{min}^j + rand(0,1)(x_{max}^j - x_{min}^j), \forall j = 1, 2, \ldots, D \tag{2.13}$$

where x_{max}^j and x_{min}^j represent the boundaries of solution x_i in dimension j.

In the employed bee phase, a bee modifies the current solution based on its own individual experience and fitness value of the new food source (solution). If the fitness value (amount of nectar) of the new source is greater than the one of the old source, the bee will update her position and discard the old position. In other words, the greedy selection mechanism is being applied as a selection operation between the old

and the new, candidate solution. The update equation for j-th dimension is:

$$v_{ij} = x_{ij} + \phi(x_{ij} - x_{kj}) \tag{2.14}$$

where $\phi(x_{ij} - x_{kj})$ is step size; $k \in \{1, 2, \ldots, SN\}$ and $j \in \{1, 2, \ldots, D\}$ are two randomly chosen indexes, with the condition that i has to be different from k; ϕ_{ij} is the random number from the interval $[-1, 1]$.

The onlooker bee phase begins when the employed bees pass on the information about fitness values of the new and updated solutions, i.e. food sources, and their positions, to the onlooker bees which are in the beehive. Onlooker bees have the task of analyzing received information and choosing a solution based on the probability, which is a function of the fitness value of the solution. There are several ways for determine this probability, and one of the possible ways is illustrated by the following formula (fit_i is the fitness values of i-th solution) [50]:

$$p_i = \frac{fit_i}{\sum_{i=1}^{SN} fit_i} \tag{2.15}$$

The onlooker bee compares the fitness values of the selected solution and prior solution, which is stored in the memory, and eventual positions modifications are based on that. If the fitness value of a new solution is higher than the previous, new position is recorded and stored in the memory, while the old one is discarded. If the position of the food source does not change over several iterations, the scout bee phase is initiated because it is assumed that the food source is exhausted and abandoned. Control parameter which determines the number of iterations needed to activate the scout phase is called limit for abandonment, or limit. In that case, the employed bee, which initially found the abandoned food source, becomes a scout bee, and it replaces the abandoned food source with a new source x_i according to the equation (2.13), meaning that the abandoned food source is replaced by a randomly chosen source from the search space [51]. With that said, one can conclude that the exploration phase of the ABC algorithm is determined by the scout phase, and exploitation is done in employed bee and onlooker phases. Pseudocode of the ABC algorithm is provided in Algorithm 2.4 [52].

Algorithm 2.4: Pseudo code of ABC algorithm

```
1  begin
2      Initialize population of solutions, x_i, i = 1, 2, ..., SN;
3      Evaluate population;
4      cycle = 1;
5      repeat
6          Generate new solutions v_i in employed bees phase using Eq. 2.14 and
           evaluate them;
7          Apply greedy selection process u the employed bees phase;
8          Calculate probabilities p_i for solutions x_i according to Eq. 2.15;
9          Generate new solutions v_i in onlooker bees phase, starting from x_i that
           are selected according to p_i, and evaluate them;
10         Apply greedy selection process in the onlooker bees phase;
11         In scout bees phase determine if there are abandoned food sources, and
           if so, replace them with randomly selected solutions x_i, based on Eq.
           (2.13);
12         Memorize the best solution until now;
13         cycle = cycle + 1;
14     until cycle = Maximal cycle number(MCN);
15 end
```

ABC algorithm has several control parameters which need to be finely tuned in order to achieve better performance, and those are: number of food sources, limit, ϕ_{ij} (weighting factor applied to the difference between the current food source and the randomly generated food source), and probability p_i, which is used for selection in onlooker bee phase. Based on the analysis given in [53] we can conclude that the bee colony size should be between 50 and 10 bees. Also, the value of the limit parameter, which determines the beginning of the scout bee phase and consequently effects balance between exploration and exploitation, needs to be equal to value $\frac{veli\text{Д}Hina\ kolonije}{2} * D$. When it comes to parameter ϕ_{ij}, which influences step size, consequently influencing the diversity of the search process of the ABC algorithm, it is recommended for it to be a random value chosen from the uniform distribution whose interval is $[-1, 1]$. The effect of control limit and step size (i.e. scaling factor) parameters was also observed in [54]. Results of the research have shown that 1 is the most suitable value for the step size parameter, and 200 is the best value for the limit parameter, both in cases of unimodal or multimodal functions. Of course, these

values depend on the dimension of the problem which is being solved.

2.5 Firefly algorithm

There are several thousand firefly species in the world and most of them, through a process called bioluminescence, have the ability to create short light of certain intensity and rhythm, where the characteristics of the light depend on the type of firefly. Two main purposes of the light that the fireflies emit is to attract a potential mating partner or potential prey. Also, a shining light can be used to warn and deflect other species that could harm them. The light that fireflies emit and the attraction between them inspired Xin-Shi Yang [55] to develop an optimization algorithm called the firefly algorithm (FA). In this algorithm the emitted light is formulated in a convenient way and tied to the objective function, which needs to be optimized. FA uses three idealized rules:

- Fireflies are sexless, which means they will be attracted to each other regardless of their sex;
- Attraction is directly proportionate to the intensity of the light, which is decreasing with the increase in distance, so in the case of two fireflies, a firefly whose light is weaker will move towards a firefly whose light is brighter;
- The intensity of a firefly's light is determined by the characteristics of the objective function.

The main steps of FA algorithm are given in a form of a pseudocode in Algorithm 2.5 [56].

In the simplest case, the light intensity I of a firefly in a certain location x can be defined as $I(x) \propto f(x)$. However, the attraction β is relative, because it's being evaluated by other fireflies or an observer. So, the attraction is changing with distance r_{ij} between firefly i and firefly j. It is known that the light intensity decreases by increasing distance from the light source, because a part of the light is being absorbed by the transfer medium, thus, attraction can be observed as a function of absorption level. If we adopt that light intensity decreases with the square of the distance from the source, and if the medium absorption coefficient is labelled γ, we end up with a formula used for calculating light intensity [57]:

Algorithm 2.5: Firefly algorithm

```
1  begin
2      Objective function f(x), x = (x_1, x_2, ..., x_d)^T;
3      Generate initial population of fireflies, x_i, i = 1, 2, ..., n;
4      Light intensity I_i of firefly x_i is determined with f(x_i);
5      Define light absorption coefficient, γ;
6      while t < MaxGeneration do
7          for i = 1 to n do
8              for i = 1 to n do
9                  if I_i < I_j then
10                     Move firefly i toward firefly j
11                 end
12                 Vary the attraction with distance using exp[−γr];
13                 Evaluate new solutions and update light intensity;
14             end
15         end
16         Rank fireflies i find current global best solution g;
17     end
18     Post processing and visualization;
19 end
```

$$I = I_0 e^{-\gamma r} \tag{2.16}$$

If we adopt that the attraction of a firefly is directly proportionate to the light intensity other fireflies see, then the attraction of a firefly can be expressed by the following formula:

$$\beta = \beta_0 e^{-\gamma r^2} \tag{2.17}$$

where β_0 is the attraction at point $r = 0$. In the algorithm implementation, the attraction function can be any monotone decreasing function, like the following, generalized form:

$$\beta(r) = \beta_0 e^{-\gamma r^m}, (m \geq 1) \tag{2.18}$$

Distance between any two fireflies i and j, which are at positions x_i and x_j, re-

spectively, is given by the Cartesian distance:

$$r_{ij} = \|x_i - x_j\| = \sqrt{\sum_{k=1}^{d} (x_{i,k} - x_{j,k})^2} \tag{2.19}$$

where $x_{i,k}$ is the k-th component of the spatial coordinate x_i for the i-th firefly. In a case of a $2D$ system, we have:

$$r_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{2.20}$$

Movement of the firefly i, which is attracted by a more appealing (brighter) firefly j, is determined by the following equation [58]:

$$x_i = x_i + \beta_0 e^{-\gamma r_{ij}^2}(x_j - x_i) + \alpha\epsilon_i \tag{2.21}$$

The second term in the equation (2.21) is used to determine the effect of the attraction, and the third addend is a randomization, where α is the randomization parameter, and ϵ_i is a vector of random numbers which can belong to Gauss or uniform distribution. In the simplest form, the random number vector can be replaced by the expression $rand\,(-1/2)$, where $rand$ is a random number generator, which has a uniform distribution in the range $[0, 1]$. Parameter α is used to control randomization, thus also controlling the diversity of solutions. It can be adjusted so that its values change throughout algorithm execution. One of the ways in which this parameter can be defined is by the following expression:

$$\alpha = \alpha_0 \delta^t \tag{2.22}$$

where α_0 is the initial scaling factor of randomization, and δ is the cooling factor. In algorithm implementations, values for δ are usually taken from the range 0.95-0.97 [59]. If we label the average scale of the problem of interest with L, it is good to adjust α_0 to be $0.01L$. Parameter γ is used to define attraction variations, and its value is very important in determining the convergence speed of the FA algorithm. It has been shown that the value of this parameter should also be adjusted in respect to L, in a way that $\gamma = 1/\sqrt{L}$. The value of this parameter is usually between 0.1 and 10.

Also, population size n is usually taken to be between 15 and 100 [60].

By analyzing equation (2.21) it can be shown that the FA algorithm can be reduced to other optimization algorithms, namely that differential evolution algorithms, accelerated PSO algorithm, simulated annealing and harmony search are all special cases of the FA algorithm. For example, if γ is very large, then the second term in equation (2.21) becomes small, reducing the algorithm to simulated annealing. On the other hand, if the parameter γ has a small value (which approaches zero), then it turns into differential evolution algorithm. Similar can also be shown for other mentioned algorithms.

2.6 Cuckoo search

Cuckoos are birds which exhibit parasite behavior during reproduction. Most species of cuckoos exhibit this by laying eggs in the nests of other birds. In doing so, they sometimes remove the host's eggs in order to increase the likelihood of their own eggs being hatched or so that food in the nest is only available to their young. If the host bird discovers cuckoo's eggs, it can discard them or leave the nest [61]. Optimization algorithm which is based on the behavior of cuckoo birds is called cuckoo search (CS).

In order to successfully implement the algorithm, three simple rules are introduced [62]: 1) each cuckoo can only lay one egg at the given moment, in a randomly chosen nest; 2) the best nests, which have the best quality of the eggs (solutions) are passed on to the next generation; 3) the number of available nests is fixed, and the probability of the host discovering the intruder's egg is p_a, where it leaves the nest and goes to build another one, or discards the intruder. Pseudocode of the CS algorithm is shown in Algorithm 2.6 [63].

Generating a new solution for the cuckoo i is done by applying Levy flight, in accordance to the equation (2.23):

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \oplus Levy(\lambda) \tag{2.23}$$

where $\alpha > 0$ is the step size, which depends on the size of the problem that is of interest, and usually is equal to 1. Equation (2.23) is a stochastic equation for a random walk, which, generally speaking, is a Markov chain whose next status/location

Algorithm 2.6: Cuckoo search

```
1  begin
2      Objective function f(x), x = (x_1, x_2, ..., x_d)^T;
3      Genetate initial population of nests, x_i, i = 1, 2, ..., n;
4      while t < MaxGeneration or stopping condition do
5          Randomly, using Levy flight, select cuckoo (i);
6          Evaluate its quality/fitness F_i;
7          Randomly select a nest j (from n possible nests);
8          if (F_i > F_j) then
9              Replace j with new solution;
10         end
11         Abandon part (p_a) of the worst nests (i generate new ones on new
            locations, using Levy flight);
12         Keep the best solutions (nest with the best quality);
13         Rank solutions and find the best solution;
14     end
15     Post processing and visualization;
16 end
```

is determined by the present location (first addend) and the transitional probability (second addend). Levy flight is a way of implementing random walk, where the random step length belongs to Levy distribution: $Levy \sim u = t^{-\lambda}, (1 < \lambda \leq 3)$

2.7 Bat algorithm

Micro-bats are usually insectivores who use echolocation in order to detect prey or avoid threat. They emit very strong and short sound signals of a certain frequency and listen to the echo which comes from the objects in their surroundings. The sound signals are usually a couple of milliseconds long, with a constant frequency being within the range 25-100 kHz. A bat emits 10 to 20 signals a second and can emit up to 200 signals per second when chasing a prey. The loudness of the emitted sound signal goes up to 110 dB, with the frequency being higher if the bat is chasing a prey, and lower upon returning from a hunt [64]. Bats use time delay between signal emission and echo detection, the time difference between the two ears, and variation in echo loudness in order to construct a three-dimensional visualization of the environment. Bat optimization algorithm (BA) is developed by imitating the behavior of bats, more

precisely, their echolocation capabilities. In BA, idealized echolocation properties are used, which can be expressed through the following three rules [65]: 1) All bats use echolocation to detect distance, and also in some "magical way" they can tell food/prey and obstacles apart in their environment; 2) Bats, when searching for food, have the flight speed v_i, in position x_i, emitting sound of a fixed frequency f_{min}, with a changing wavelength λ and loudness A^0. They can automatically adjust, i.e. update the wavelength (or frequency) of the emitted sound signals or impulses, as well as adjust the speed of signal emission $r \in [0, 1]$, depending on how close the target is; 3) it is assumed that the loudness of sound impulses can vary amongst a large (positive) value A^0 and a minimal value A_{min}. Pseudocode of the BA algorithm is shown in Algorithm 2.7 [66].

Algorithm 2.7: Bat algorithm

```
1  begin
2      Objective function f(x), x = (x_1, x_2, ..., x_d)^T;
3      Generate initial population of bats, x_i, i = 1, 2, ..., n, and velocities v_i;
4      Define sound frequency f_i on the position x_i;
5      Initialize pulse emitting speed r_i, and loudness A_i;
6      while t < Maximal number of iterations do
7          Generate new solutions by adjusting frequency and updating velocities
             and locations/solutions (Eqs. (2.24)-(2.26));
8          if rand < r_i then
9              Select solution among the best solutions;
10             Generate local solution in the neighbourhood of selected best
                 solution;
11         end
12         Generate new solution by random flight procedure;
13         if (rand < A_i & f(x_i) < f(x*)) then
14             Accept new solution;
15             Increase r_i and decrease A_i;
16         end
17         Rank bats and find current best solution x*;
18     end
19     Post processing and visualization;
20 end
```

Rules for adjusting frequency, speed and location/solution are given by equations

(2.24), (2.25) and (2.26), respectively.

$$f_i = f_{min} + (f_{max} - f_{min}) * \beta \tag{2.24}$$

$$v_i^t = v_i^{t-1} + (x_i^t - x^*) * f_i \tag{2.25}$$

$$x_i^t = x_i^{t-1} + v_i^t \tag{2.26}$$

3 BRAIN STORM OPTIMIZATION ALGORITHM

Brain storm optimization algorithm (BSO) as a swarm intelligence algorithm which was proposed by Yuhui Shi in the year 2011. It's a new method of optimization based on the collective behavior of humans when solving a problem, i.e. the brainstorming process [67]. Brainstorming process is interpreted and defined as an algorithm which consists of several steps. These steps simplify the entire process, but they hold enough elements in order to implement the optimization algorithm which has both exploration and exploitation capabilities. All swarm intelligence algorithms have simple agents which, through several generations, use information sharing to improve globally best solutions. Exploitation is referring to a local search in the regions which look promising, while exploration is a random search used in order to prevent the algorithm from getting trapped in a local optimum.

This algorithm has two main operators: divergent and convergent operator, and a good enough suboptimal solution is generated by a recursive use of divergence and convergence in the search space. Divergence refers to the exploration of new search spaces, while convergence represents exploitation of the existing regions which can contain good solutions. Considering that the brain storm algorithm belongs to the group of developmental swarm optimization algorithms, it has two main abilities: capability learning and capacity developing. Divergent operator corresponds to capability learning, while the convergent operator corresponds to capacity developing. One of the traits of the BSO algorithm is solution clustering, where new solutions are generated by the mutation of clusters or existing solutions. As said previously, the BSO algorithm is based on the brainstorming process which is unfolded when a group of people collectively solve a problem. Typical steps for this process, as well as Osborn rules for idea generation, are shown in Tables 3.1 and 3.2, respectively [68].

Rule 2 states that any idea which comes to mind during the brainstorming process as in interesting one and shouldn't be discarded, but instead should be shared with others, while it any form of judgment is forbidden, as well as labelling ideas as meaningless and worthless (Rule 1). It is necessary to generate as much ideas as possible (Rule 4), starting from the current ideas and their mutual combinations (Rule 3), and then repeat the procedure until a satisfying solution is reached. Starting from the previously described brainstorming process and the rules that should be followed,

Table 3.1 Steps in a Brainstorming Process

Step 1. Get together a brainstorming group of people with as diverse background as possible;
Step 2. Generate many ideas according to the rules in Table 3.2;
Step 3. Have several, say 3 or 5, clients act as the owners of the problem to pick up several, say one from each owner, ideas as better ideas for solving the problem;
Step 4. Use the ideas picked up in the Step 3 with higher probability than other ideas as clues, and generate more ideas according to the rules in Table 3.2;
Step 5. Have the owners to pick up several better ideas generated as did in Step 3;
Step 6. Randomly pick an object and use the functions and appearance of the object as clues, generate more ideas according to the rules in Table 3.2;
Step 7. Have the owners to pick up several better ideas;
Step 8. Hopefully a good enough solution can be obtained by considering and/or merging the ideas generated.

Table 3.2 Osborn's Original Rules for Idea Generation in a Brainstorming Process

Rule 1. Suspend Judgment;
Rule 2. Anything Goes;
Rule 3. Cross fertilize and Piggybacking;
Rule 4. Go for Quantity.

BSO algorithm was created, whose pseudocode is shown in Algorithm 3.1 [68].

In the brain storm algorithm ideas are d-dimensional vectors and represent simple agents. Brainstorming process begins by generating n initial random solutions, after which the ideas are divided into m clusters. For clustering, a k-means clustering algorithm is usually used, but other methods can be used as well. In each cluster the best idea (i.e. the idea with the best fitness function value) is selected as the cluster center. In the iterative process new solutions are created by combining the existing ideas (solutions). With the probability p_{6b} one solution $X^d_{selected}$ is chosen, which will be changed to generate new solution. After comparing the old and the new solution, the better solution (i.e. the solution with the better fitness function value) is kept.

Algorithm 3.1: Brain storm algorithm

```
1  begin
2      Initialization;
3      Randomly generate n potential solutions (individuals);
4      repeat
5          Cluster n solutions into m clusters;
6          Rank solutions in each cluster and set the best one as cluster center;
7          Randomly generate a value r between 0 and 1;
8          if r < p_5a then
9              Randomly select a cluster center;
10             Randomly generate an individual to replace the selected cluster;
11         end
12         repeat
13             Generate new solutions;
14             Randomly generate a value r between 0 and 1;
15             if r < p_6b then
16                 Randomly select a cluster with probability p_6bi;
17                 Randomly generate a value r_1 between 0 and 1;
18                 if r_1 < p_6bii then
19                     Select the cluster center and add random values to it to generate
                       new individual;
20                 else
21                     Randomly select a solution from the chosen cluster and add
                       random value to the solution to generate new one;
22                 end
23             else
24                 Randomly select two clusters to generate new individual;
25                 Generate random value r_2 between 0 and 1;
26                 if r_2 < p_6c then
27                     Two cluster centers are combined and then added with random
                       values to generate new individual;
28                 else
29                     Two individuals from each selected cluster are randomly selected
                       to be combined and added with random values to generate new
                       individual;
30                 end
31             end
32             The newly generated solution is compared with the same solution index
               and the better one is kept;
33         until n new solution is generated;
34     until Maximal iteration number is reached;
35 end
```

Generating new ideas (individuals) is done in 6 steps, according to the following formula:

$$X_{new}^{d} = X_{selected}^{d} + \xi * n(\mu, \sigma) \tag{3.1}$$

where: X_{new}^{d} is d-th dimension of a newly created individual; $X_{selected}^{d}$ is d-th dimension of the individual chosen for generating of the new individual; $n(\mu, \sigma)$ as a function of the Gauss distribution; and ξ is the weight coefficient which is used to determine the contribution of the Gauss random value. This coefficient is also called step size and its value can be calculated using the following expression:

$$\xi = logsig(\frac{0.5 * maxIteration - currentIteration}{k}) * rand() \tag{3.2}$$

where $logsig()$ is log-sigmoid transfer function; $maxIteration$ is the maximal number of iterations; $currentIteration$ is the number of the current iteration; k is the value which alters the value of the gradient for the function $logsig()$; $rand()$ the random variable from range (0,1). Additionally, with the probability of $1 - p_{6b}$ two solutions (individuals) will be selected, which will be combined in order to generate a new solution as the mean value of the selected solutions. The exploration process is controlled by another algorithm parameter, p_{5a}, and is implemented by replacing the cluster center with a new random solution. Beside mentioned parameters, BSO algorithm has several other parameters which have to be defined. Parameter p_{6bi} refers to the probability of the cluster center, used for selecting a solution (individual) which will be used for generating a new solution, being selected. The probability p_{6bi} is proportionate to the number of ideas in each cluster. Parameters p_{6bii} and p_{6c} are the probabilities of using the cluster center or the random solutions from the selected clusters. Parameter p_{6bii} is the probability of updating the cluster center or random solution from the cluster. Finally, probability p_{6c} is used to determine if the combination process will involve two cluster centers or two random ideas from two clusters.

3.1 Modifications of BSO algorithm

The original BSO algorithm, described in the previous section, has two main components: convergent and divergent operator. The convergent operator is used to emulate the operation of choosing better ideas by the problem owner so that they can be used

to create new ideas in the next round of the process. Therefore, this operator maps one population to a smaller set of individuals. The divergent operator is in charge of the process of generating new ideas based on better ideas, which are chosen in the previous iteration. From the aspect of algorithm execution, it is possible to differentiate between three different operators: grouping operator, replacing operator, and creating operator. The grouping operator groups all ideas generated in one generation into different groups (clusters), and the creating operator generates new ideas by adding a random value to a single idea from a group or a combination of ideas from multiple groups. Implementation of this algorithm uses a k-means clustering method as its grouping operator, while the Gaussian random noise is used in the creating operator as the random value that is added during new idea generation. Shortly after the introduction of the original BSO algorithm, suggestions and variations which seek to improve its performance appeared. Different strategies and forms of operator modifications have been proposed, i.e. modifications of the original BSO algorithm parameters, which resulted in faster convergence speed and smaller time complexity of the proposed algorithms. Below, an overview of the most important variations and modifications of the original BSO algorithm is given.

Authors in [69] introduce two new modifications of the original BSO algorithm in order to enhance performance. The creating operation is modified by implementing a dynamic step size parameter control strategy. The grouping operation is modified by proposing a random grouping strategy instead of a k-means clustering method. Considering that BSO is a stochastic algorithm, this strategy increases the chance of finding good-enough solutions in the heuristic mode of search. The random grouping strategy can be represented by a procedure given in Algorithm 3.2.

Pseudocode of the RGBSO algorithm is given in Algorithm 3.3.

Zhan et al. [70] suggest new methods and strategies for the grouping and creating operators. The suggested modified brain storm algorithm uses the simple grouping method (SGM) as a grouping operator and the idea difference strategy (IDS) as a creating operator. SGM method is implemented through a procedure described in Algorithm 3.4.

The inspiration behind the IDS approach comes from the fact that, at the beginning of the brainstorming process, people generate different ideas, with the idea diversity

Algorithm 3.2: Random grouping strategy

begin
Randomly divide N ideas in m groups, based on group size ($N = s * m$), $G = \{G_1, G_2, \ldots, G_m\}$;
For each group G_i, compare fitness values of all individuals in each group;
Select central idea for each group as one which has minimal fitness value;
Equation to set step size is:

$$\xi = rand() * \exp(1 - \frac{maxIteration}{maxIteration - currentIteration + 1}) \qquad (3.3)$$

where $rand()$ is random value between 0 and 1;
end

Algorithm 3.3: BSO algorithm with random grouping - RGBSO

```
begin
    Randomly initialize N ideas and determine their fitness values;
    Initialize cluster centers (m < N);
    while stopping criteria not met do
        Execute random grouping procedure;
        for i = 1 to N do
            if rand() < p_one then
                if rand() < p_one_center then
                    Select center of the group as X_selected
                else
                    Randomly select idea from the group as X_selected
                end
            else
                if rand() < p_two_center then
                    Combine centers from two groups to form X_selected
                else
                    Combine two random ideas from two groups to form X_selected
                end
            end
            Create X_new using X_selected with formula (3.1) i (3.3);
            Accept X_new if f(X_new) is better then f(X_i)
        end
    end
end
```

Algorithm 3.4: SGM method

begin

Randomly select M different ideas from current generation, as seeds of M groups. These seeds are denoted as S_j,$(1 \leq j \leq M)$;

For each idea X_i,$(1 \leq i \leq M)$ from current generation, calculate its distance to each group using formula:

$$d_g = \|X_i, S_j\| = \sqrt{\sum_{d=1}^{D} \frac{(x_i - s_j)^2}{D}}$$

;

Compare all distances and assign X_i to the closest group;

Go to step 2. Repeat procedure until all ideas are assigned to some group;

end

decreasing as the process reaches its end. So, during the process of creating new ideas based on the existing ones, the diversity of current ideas has to be taken into account. If we use Y_i to label a new idea, and X_a and X_b to label two randomly chosen ideas which represent idea diversity, then the process of creating an idea using the IDS method can be described in the following way:

$$y_{id} = \begin{cases} random(L_d, H_d), & if \quad random(0,1) < p_r \\ x_{id} + random(0,1)_d * (x_{ad} - x_{bd}), & otherwise \end{cases} \tag{3.4}$$

The enhancement of BSO algorithm performance by applying the reinitialization mechanism, as well as modification of the step equation, i.e. the equation which defines the weight coefficient for determining the contribution of the random value, is suggested in [71]. Each idea is assigned a counter which increments each time an idea is not improved. If the counter reaches a certain value, the idea is reinitialized in the search space. The process of reinitialization (Algorithm 3.5) can be performed in two ways: 1) random reinitialization of an idea in the search space; 2) by combining three randomly selected ideas in order to generate a new idea by using the differential evolution update equation.

The second modification consists of modifying the equation which is used to calculate parameter ξ. Introduction of the multiplier α is proposed, which is proportionate

Algorithm 3.5: Reinitialization procedure

```
begin
    for i = 1 to NumberofIdeas do
        if counters(i) ≥ threshold then
            if rand < 0.5 then
                Randomly select three different ideas j, k, l
                  idea(i) = idea(j) + F * (idea(k) − idea(l))
            else
                idea(i) = LB + rand(1, D) * (UB − LB)
            end
        end
    end
    return new ideas;
end
```

to the search space size, so the new equation has the following form:

$$\xi = rand * e^{\frac{1-MaxIteration}{MaxIteration-CurrentIteration+1}} * \alpha \tag{3.5}$$

Application of the globally-best concept in combination with per-variable updates and fitness-based grouping in order to enhance the BSO algorithm is given in [72]. The suggested algorithm (globally-best brain storm optimization algorithm, GBSO) also contains a reinitialization mechanism which is activated by the current state of the population. The concept of globally-best information is borrowed from PSO, and here it is applied to the updating equation in the following way:

$$nidea(i) = nidea(i) + rand(1, DimSize) * C * (GlobalBest - nidea(i)) \tag{3.6}$$

where C is given as:

$$C = C_{min} + \frac{CurrentIteration}{MaxIteration} * (C_{max} - C_{min}) \tag{3.7}$$

Unlike the original BSO algorithm, where new ideas are generated by simultaneously updating all variables in one step, in this algorithm variables are updated one by one. This means that the first variable of the problem can be updated by using a center of

one, randomly chosen cluster, updating the next one based on a combination of ideas from two randomly chosen clusters, etc. Also, while the original BSO algorithm uses one or at most two existing ideas in order to generate a new idea, in this algorithm combination of more than two ideas is allowed, which enables a stronger cooperation between the individuals.

Changes to step size updating and generating new individuals, in order to enhance the performance of the original BSO algorithm, is given in [73]. The proposed algorithm calls for an adaptive step size, which is updated in accordance with a dynamic range of individuals in each single dimension. In the modified algorithm, step size, which defines the effect of Gaussian noise, can be described in the following manner:

$$\xi_i^{center} = k_l * (x_{nmax,i} - x_{nmin,i}) \tag{3.8}$$

$$\xi_i^{individual} = k_2 * (x_{nmax,i} - x_{nmin,i}) \tag{3.9}$$

where ξ_i^{center} and $\xi_i^{individual}$ are step size values in the i-th dimension and are used in different steps of the algorithm, k_1 and k_2 are coefficients, and $x_{nmax,i}$ and $x_{nmin,i}$ are the minimum and maximum value in the entire population in i-th dimension, respectively. Based on equations (3.8) and (3.9) it is clear that the step size is adaptive and that it adapts to the dynamic range of the population of individuals.

Also, it is suggested that individuals be generated in batch mode, and then selection for the next generation to be performed. In this manner, the algorithm crates more individuals in order to fully utilize every referent point instead of constantly creating the same number of individuals, which corresponds to the population size. Parts of the algorithm that have been changed in the original BSO algorithm belong to processes of generating individuals and their selection, and are presented in algorithm 3.6.

Xue et al. [74] suggest an algorithm for multi-objective optimization based on the brainstorming process (MOBSO), which consists of six parts, where three parts are specific to BSO algorithms: clustering strategy, generating process, and global archive updating. Apart from standard operations which are applied in traditional multi-objective optimization algorithms, in the proposed algorithm the clustering strategy is performed in the objective space, by applying k-means clustering algorithm in order to group the population in k clusters based on each objective. Pseudocode of the

Algorithm 3.6: BSO algorithm modification

begin
  Generating individuals
  Randomly select cluster center. Probabilities for the selection of any cluster center is in accordance with the cluster size. Based on the selecter center, generate new individual using Eq. (3.9). Repeat this step n times (where n is population size), thus generating n individuals;
  Randomly selecter two cluster centers, and connect them to create new center. Based on new center generate new individual using Eq. (3.8). Repat this step n times, thus generating n individuals;
  Randomly select an individual as a referent point. Add Gauss noise to it, using Eq. (3.9) in order to create new individual. Repeat this step n times, thus generating n individuals;
  Selection
  Evaluate new generating individuals, total of $3n$;
  Randomly sort total of $4n$ individuals in n groups with equal number of individuals per group (4). In each group select an individual which has the best fitness value and copy it in the next generation.
end

clustering strategy is given in Algorithm 3.7.

Algorithm 3.7: Clustering strategy

begin
  Initialize $Elite\_set = \emptyset$, $Normal\_set = \emptyset$;
  Evaluate population and update $Archive\_set$ according to Pareto domination;
  Initialize cluster centers based on fitness value of an objective M;
  For each objective f_m: cluster the population into k clusters. Select the cluster that has the best fitness value and denote it as $Elite\_cluster_m$;
  For each individual: if the individual is any from $Elite\_cluster_m$, add it to $Elite\_set$; otherwise, add the individual to $Normal\_set$.
end

After clustering, new ideas are generated in accordance to the selection process, dispersion step, as well as the operators of selection and mutation. The mutation operator generates new solutions based on the existing ones, while the selection operator is used to decide whether the newly generated solutions will be sent to the next generation. The process of generating a new individual is given in Algorithm 3.8.

Algorithm 3.8: New individual generation process

```
1  begin
2      if rand() < P1 then
3          if rand() < P2 then
4              if rand() < P3 then
5                  randomly choose an individual from the set Elite_set and
                     denote it as X_selected
6              else
7                  randomly choose X_selected from the set Normal_set
8              end
9          else
10             randomly select X_selected from the set Archive_set
11         end
12     else
13         dispersion step: randomly generate individual X_selected
14     end
15 end
```

The mutation operation can be implemented as a Gaussian mutation or a Cauchy mutation. In classical evolutionary algorithms, the former is usually used, and it's given in equations (3.1) and (3.2), while it has been shown that Cauchy mutation is an effective search operator for a large number of multi-modal optimization problems. This mutation can be described by the following equation:

$$X_{new}^{d} = X_{selected}^{d} + \xi * C(\mu, \sigma) \tag{3.10}$$

where $C(\mu, \sigma)$ is the Cauchy random function with the mean μ and variance σ. The selection operator is based on Pareto dominance and can be described in the following way: if X_{new} dominates $X_{selected}$, then X_{new} survives; if $X_{selected}$ dominates X_{new}, then $X_{selected}$ survives; if neither of them dominates each other, then the new individual is randomly chosen between them. Results have shown that both versions of the algorithm (Gaussian and Cauchy) have great performance.

In the original BSO algorithm and its many modifications, the k-means clustering method is used as a clustering operator. The problem with this approach is that it calls for the number of clusters to be specified before the execution of the algorithm, although the exact number of clusters that are needed can't be known ahead of time.

Also, even though the number of clusters changes over time during the execution of the algorithm, k-means method uses a fixed number of clusters during the entire iterative process. Authors in [75] suggest that instead of using k-means clustering method, another method - the affinity propagation clustering method, is used because this method doesn't need the number of clusters to be determined or approximated before the algorithm starts. The affinity propagation clustering method is a new clustering method based on the message exchange technique. In short, if a set of points $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ and a function s, which calculates the similarity between two points, is given, then it hold that $s(i,j) > s(i,k)$ if α_j is more similar to α_i than α_k. The algorithm is executed by having two steps of message exchanging being performed in order to update two matrices - a responsibility matrix and an availability matrix, in order to find exemplars, i.e. members of the input set which are cluster representatives. Value $r(i,k)$ from the responsibility matrix determines how suitable is α_k as an exemplar for α_i. Value $a(i,k)$ from the availability matrix quantifies how suitable is for α_i to choose α_k as his exemplar. A creating operator with no arguments is also introduced, and it is based on undefined, distributed information from several clusters, by applying ideas from the cloud drops algorithm. More precisely, several clusters are selected, which are then quantified as three numeric characteristics: expected value (Ex), entropy (En), and hyperentropy (He). New solutions are generated based on these numeric characteristics.

Authors in [76] suggest an improved BSO algorithm, which uses the affinity propagation clustering strategy and an improved creating operator, which uses structure information of one or more clusters. In order to make the information extraction process easier, the fitness values of candidate solutions are mapped in a uniform confidence interval. Then, structure information for each cluster C is being extracted and labelled as vector $C\{u, v, w\}$, where u is the kernel of the cluster, and v and w are the candidate solution coverage and dispersion, respectively. Algorithm 3.9 contains the pseudocode of the procedure for the structure information extraction of each cluster.

In the case of more clusters, structure information can be obtained by applying certain formulas. Creating new individuals is, at the end, based on using different structure information. Pseudocode for creating new individuals is given in Algorithm 3.10.

Algorithm 3.9: Algorithm for extracting structural information

begin
  Select a cluster with K individuals in M dimensions;
  Calculate fitness values for each individual: $f(x), i = 1, 2, \ldots, K$;
  Map fitness values into trust interval $[0, 1]$;
  Determine the best individual that represents kernel in the selected cluster $u = [u_1, u_2, \ldots, u_M]$;
  Calculate coverage as: $v = [v_1, v_2, \ldots, v_M] = \frac{1}{L-1} \sum_{i=1}^{L} (x_i - u)^2$;
  For each par which consists of individual and its fitness value, $(x_i, f(x_i))$ calculate $o_i = \sqrt{\frac{-(x_i-u)^2}{2 \ln f(x_i)}}$;
  Calculate average value of o_i as o_{sr}, $o_{sr} = \frac{1}{L} \sum_{i=1}^{L} o_i$;
  Calculate dispersion of the selected cluster in the following way: $w = [w_1, w_2, \ldots, w_M] = \frac{1}{L-1} \sum_{i=1}^{L} (o_i - o_{sr})^2$;
end

Algorithm 3.10: Algorithm form creating new individuals

begin
  Select cluster with K individual in M dimensions;
  Create new G individuals based on structural information $C(u, v, w)$;
  for $j = 1$ **to** G **do**
    Generate vector p with normal distribution using kernel v and coverage w, as $p = NormRand(v, w)$;
    Generate new individuals x_i with normal distribution, using kernel u and coverage p, as $x_i = NormRand(u, p)$;
  end
end

Zhu and Shi [77] have noticed that the k-means clustering algorithm, which is used in the original BSO algorithm, is suitable for programming and computing. However, it also has certain disadvantages which can effect algorithm efficiency. Namely, during the process of updating the cluster center, certain outliers can have a negative effect, because in that process each individual calculates its mean value. Also, if the data set is big, k-means algorithm might require a very long computing time. In order to

solve these problems, the authors suggest a new clustering algorithm which is based on using medians instead of means. The main difference between these two algorithms is in the process of updating the cluster center; the new algorithm uses a median for the cluster center instead of the mean value of individuals. The median coordinate in multidimensional space is the median for each single dimension.

Shi [78] suggests a modification of the convergent operator so that it is implemented in l-dimensional objective space instead of solution space, making the computation time dependent on the population size, but not the dimension of the problem. The author proceeds from a standpoint that, even though the clustering method has proven well for implementing a convergent operator, it is not necessary for its implementation, and it can be replaced by any method which will allow selection of better ideas from the population. He suggests a convergent operator in a one-dimensional objective space for application in BSO algorithm, which solves single-objective optimization problems, in the same way that it has been applied in the case of multi-objective optimization problems in [74]. Procedure for the BSO algorithm in the objective space is shown in Algorithm 3.11.

Algorithm 3.11: BSO procedure in objective space

1 Population initialization;
2 **while** *not end* **do**
3 Evaluate individuals;
4 Take the first $perc_e$ percent of individuals as elite, and the rest as normal;
5 Disrupt randomly chosen individual;
6 Updating individuals;
7 **end**
8 Output individuals;

Steps 3 and 6 are identical to the ones in the original BSO algorithm. Step 4 of the proposed algorithm replaces the clustering operation, and step 5 modifies the disrupting (replacing) cluster operation in the original BSO. Step 4 performs ranking of the individuals based on their fitness value. Unlike the original BSO algorithm, where individuals are grouped in m clusters, here the $perc_e$ % best individuals are placed in the elite category, while others are placed in the normal category. When it comes to updating (generating) new individuals, the same formulas from the original BSO are applied, but firstly, it is decided whether new individuals will be generated based on

the elite or the normal individuals, and also will one or two selected individuals be used (although, generally speaking, every new individual can also be created based on more than two selected individuals). Pseudocode for this operation is given in Algorithm 3.12 (p_e - the probability that elite instead of normal individuals will be used for generating new individuals; p_{one} - the probability that one instead of two individuals will be used for generating new individuals).

Algorithm 3.12: Pseudo code for generating new individuals

```
1  if rand < p_e then
2      if rand < p_one then
3          generate new individual based on randomly selected elite individual
4      else
5          generate new individual based on two randomly selected elite
             individual
6      end
7  else
8      if rand < p_one then
9          generate new individual based on randomly selected normal individual
10     else
11         generate new individual based on two randomly selected normal
             individual
12     end
13 end
```

Step 5, which refers to the disrupting operation of the randomly chosen variable, is done so that a value of only one, randomly selected dimension of the given individual, is disrupted, where the given value is modified by a randomly generated value. This is done in order to decrease the randomness created by the disruption operation, which, in the original BSO, is done by replacing the selected individual with a randomly generated individual. For compensation purposes, step 5 is performed in each iteration, instead of each iteration with a certain probability, as it is the case with the original BSO.

Most swarm intelligence algorithms have the disadvantage of premature convergence. This, among other reasons, happens because the solutions group together in small regions relatively quickly, which means that the population diversity decreases rapidly during the search, thus making further divergence harder. This can result in having exploration and exploitation fall out of balance, thus causing for the algorithm

to get trapped in a local optimum. A suggestion to overcome this flaw is to apply two solution reinitialization strategies which should diversify the population [79]. The general idea is to apply partial solution reinitialization after several iterations, i.e. for a part of the solution to reinitialize its position and speed in the entire search space, which increases the probability that the solutions will leave the local optimum. Two strategies are being analyzed: 2) for half of the solutions to be reinitialized after several iterations; 2) for the number of reinitialized solutions to decrease over the course of the search process, so that more than half of the solutions are reinitialized at the beginning of the search, and the number of reinitialized solutions decreases linearly with every reinitialization. Algorithm 3.13 shows the pseudocode of the modified BSO algorithm which implements the reinitialization process.

Algorithm 3.13: Modified BSO algorithm that applies reinitialization procedure

```
1  begin
2      Initialization: Randomly generate n potential solutions (individuals) and
         evaluate them;
3      while good enough solution not found or maximal number of iteration not
         reached do
4          Clustering: Cluster n individuals in m clusters using clustering
             algorithm;
5          Generating new individual: randomly select one or two clusters for
             generating new individual;
6          Selection: New generated individual is compared with current one
             which has the same individual index, better one is kept and record as
             new individual;
7          Reinitialization: perform partial reinitialization of some solutions after
             certain number of iterations;
8          Evaluate n individuals;
9      end
10 end
```

The issue of premature convergence, which can cause the solutions to get trapped in a local minimum, can be solved in the original BSO algorithm by applying the chaotic operation [80], thanks to the chaos properties such as ergodicity, intrinsic stochastic property, and sensitivity to initial conditions. The original BSO algorithm needs to be altered so that, after updating the individuals, one randomly chosen dimension of

one randomly chosen cluster is updated in chaos mode, while the entire search space is considered as the range of chaotic movement. Specific dynamic mode of simple chaotic map is give by the following equation [81]:

$$x_{+1} = rx(1 - x) \tag{3.11}$$

where x is a value from the range $(0, 1)$, r is a parameter which controls the behavior of the chaotic map. If $r = 4$, x becomes completely chaotic in the range $(0, 1)$. Considering that the initial value of the chaotic process falls within the range $(0, 1)$, the current position is transformed to a value from this range, which is obtained by applying equation (3.12):

$$fposi = \frac{(cposi - lbound)}{(ubound - lbound)} \tag{3.12}$$

where $fposi$ is the position in a fractional form, $cposi$ is the current position in the search space, $lbound$ and $ubound$ are lower and upper bounds in the search space, respectively. After the chaos operation is applied, fractional form is transformed to a position in the search space by the following formula:

$$newposi = fposi * (ubound - lbound) + lbound \tag{3.13}$$

The new position obtained by the formula (3.13) is then evaluated, and if its fitness function is better than the fitness function of the previous position, a replacement is made.

3.2 Hybrid algorithms

A hybrid algorithm for solving continuous optimization problems, based on BSO and simulated annealing algorithm, is given in [82]. The proposed algorithm integrates the simulated annealing process into the brain storm optimization algorithm. The part which is being integrated is in charge of creating new individuals in the later stages of the evolutionary process, by changing the creating operator of the BSO algorithm. More precisely, the new algorithm integrates the SA process into BSO in a serial hybrid mode, which means that the SA process is being executed after each renewal of the

population, in order to make the new, updated population stable. This is a way of decreasing instability, which is introduced due to the random nature of the creating operator of the BSO algorithm.

In [83], a hybrid self-adaptive algorithm for efficient search in a multi-dimensional domain is designed, which is a combination of learning principles from the BSO algorithm and teaching-learning based optimization algorithm (TLBO). TLBO is based on interactions between students and teachers in a class, and also between students themselves. The student with the largest knowledge in the given moment is considered to be like a teacher. Osborn rule number 3 (Table 3.2), as a crucial mechanism in the brainstorming process, is applied in the TLBO algorithm in a way that combines ideas generated in the learning and teaching phases into new ideas. This hybrid algorithm consists of four phases: initialization, teaching phase, learning phase and brainstorming phase. Initialization is performed using a matrix, whose size is MxN, where M is the population size, and N is the dimension of the problem being solved. In the teaching phase, the mean of the class is determined by determining the mean value of each dimension, after which they are combined using an appropriate formula. In the current iteration, the teacher is the best solution of the given population. In the case of a multi-objective problem, the best individual is found by applying a suitable non-dominating sorting method. The equation which describes the mutation process in this phase contains a learning factor, which can be made adaptive by introducing a random value into the formula which is used to calculate it, which makes the algorithm a self-evolving one. In the learning phase, the students go through a process of improvement and enhancement by using differential mutation, where the time gradient which exists between the students is being used for implementing the mutation process. In the brainstorming phase, obtained populations from the previous phases are combined, and then the mutation operator, implemented through nonlinear functions, is being applied, as in the original BSO.

The hybrid algorithm, which integrates mutation and crossover procedures of the DE algorithm into inter-cluster and intra-cluster creating operators of the BSO algorithm, is given in [84]. The reason for applying a DE strategy is that it's mostly based on information about distance and direction, its advantage being that it doesn't have a bias towards a certain direction. The mutation operation of the hybrid algorithm

is implemented in accordance to the mutation strategy of the classical DE algorithm, meaning that the strategy of differential evolution is added to normal ideas in order to create new, different ideas. DE operators of the hybrid algorithm are intra-cluster differential evolution operator and inter-cluster differential evolution operator. Intra-cluster operator can be described by the equation (3.14), which shows that a new idea is generated using the difference between two randomly selected ideas from one cluster, and from the cluster center.

$$X_{new} = X_{center} + F * (X_{r1} - X_{r2}) \tag{3.14}$$

where F is the mutation scaling factor, which affects the differential variation between two ideas, and indexes $r1$ and $r2$ are mutually exclusive integers randomly chosen in the given selected cluster. After this, the crossover operation of the DE algorithm is being applied in order to generate new solutions. The inter-cluster operator can be described using the equation (3.15), where it is shown that a new idea is generated using the difference between two randomly selected ideas from two different clusters and a globally best ideas (for all clusters).

$$X_{new} = GlobalIdea + F * (X_{r1} - X_{r2}) \tag{3.15}$$

where $GlobalIdea$ is the best idea for all clusters, and X_{r1} and X_{r2} are normal ideas selected from two different clusters. After this, the crossover operation of the DE algorithm is applied in order to generate new solutions. Also, for the purpose of a more effective convergence speed control, the proposed algorithm introduces a new way of computing the step size:

$$\xi = rand * e^{1-\frac{maxIteration}{maxIteration-currentIteration+1}} \tag{3.16}$$

3.3 Multiobjective and multimodal BSO algorithms

The original BSO algorithm is meant for solving single-objective optimization problems. However, modifications and variations that can be used for solving multi-objective and multimodal optimization problems have been developed. As it is known, multi-objective optimization problems don't have a single solution, but rather a set of

candidate solutions, so that no one solution is better than another solution from that solution set, considering the defined objectives. This set is called the Pareto-optimal set, and the associated objective vectors form a surface in the objective space, which is called the Pareto front. One of the main challenges of multi-objective optimization problems is that it's very hard to obtain a sufficient number of non-dominated solutions which correspond to the knee region of the Pareto front, which represents the maximum compromise between the objectives. Multi-objective BSO algorithm, based on estimation in the knee region and clustering in the objective region [85], can be applied in order to obtain a knee point of the Pareto-optimal front. Unlike the original BSO algorithm, in this case, the clustering strategy is applied directly in the objective space instead of the solution space, which speeds up the process of finding Pareto-dominant regions in the next iteration. Clustering is done by using the k-means method, by firstly generating the initial distribution of k cluster centers in the objective space. Then, each individual is assigned to a cluster whose center is the closest to it, in accordance with its fitness value. At the end, cluster centers are also updated so that the cluster center becomes the center for the mass of all particles which belong to that cluster. Clusters without non-dominated solutions are mapped in the decision variable space and create a new population $Q1$. Mutation operation is applied if clusters without non-dominated solution exist. In that case, mutation is applied to the new generation in order to generate solutions instead of individuals whose position deviates from the non-dominated solutions. Mutation is implemented as a mutation operation from the differential evolution, and it can be described with the following equation:

$$Z_i = P_i + F * (P_m + P_n) \tag{3.17}$$

where Z_i is the trial vector, P_i is the target vector, F is the mutation parameter, P_m and P_n are parameter vectors, randomly selected between non-dominated solutions which are all different. The probability of mutation changes in accordance to the number of iterations in order to improve the convergence of the algorithm. Selection operation, i.e. the process of deciding whether the newly generated solutions will be selected for the next generation, is based on Pareto dominance. Considering that the position of the knee point changes after several iterations, it is necessary to perform

a knee region estimation. The knee region estimation algorithm selects several best solutions from the μ matrix in order to create candidate solutions. If the Euclidian distance between two potential knee points is less than ϵ, these two solutions are located in a single knee point, which is estimated using the mean value method; otherwise, two potential knee points are two different knee points. Also, a local mutation parameter is also applied in order to enhance the ability of the local search in the decision space, which is mapped by the knee region, and to avoid getting trapped in a local minimum of the knee region. For this purpose, Cauchy mutation is used, which can be described using the following formulas:

$$x_i(t+1) = x_i(t) + \xi * C(\mu, \sigma) \tag{3.18}$$

$$g(x) = \frac{a}{\pi * (x^2 + a^2)} \tag{3.19}$$

where $C(\mu, \sigma)$ is the Cauchy random function with mean μ and variance σ, and $g(x)$ is the density function of the Cauchy distribution.

Modified multi-objective brain storm algorithm which uses clustering strategy in the objective space, DBSSCAN clustering algorithm and DE mutation strategy is given in [86]. The clustering strategy in the objective space works in such a way that it suggests potential Pareto-dominant areas in the next iteration, and in this solution, a density-based algorithm for finding clusters in large spatial databases with noise is used [87]. The proposed algorithm uses the fact that the main reason we can identify a cluster (of points) is due to the fact that a cluster has greater point density than the region outside of the cluster, e.g. the region with noise. The key idea is that, for each cluster point, the surrounding area of that point, which is limited by a radius, has to contain at least a minimal number of points, meaning that the density of the surrounding area has to cross a certain threshold. The process of generating a new individual is done by two operators: mutation and selection. Mutation is based on the differential mutation strategy, because Gaussian and Cauchy mutation operators have shown bad performance in the form of slow convergence when applied to solving multi-objective and multimodal optimization problems. When a new idea has to be generated based on the current ideas, the difference between current ideas has to be

taken into account. If we label the newly generated idea as X_{new}, the current idea as $X_{selected}$, with $X_a = (x_a^1, x_a^2, \ldots, x_a^d)$ and $X_b = (x_b^1, x_b^2, \ldots, x_b^d)$ being two different random ideas selected to represent idea difference, then the process of generating a new idea by applying differential mutation can be described by the following equation:

$$x_{new}^d = x_{selected}^d + rand(0,1)_d * (x_a^d - x_b^d) \tag{3.20}$$

The selection operator is based on Pareto dominance. Pareto set is updated with new non-dominated solutions. In this step, each new non-dominated solution acquired in the current iteration is compared to other members of the Pareto set. If the Pareto set size exceeds the maximum value, cut-off is performed with respect to diversity.

Unlike single-objective optimization problems, multimodal optimization requires for several local and global minima to be obtained simultaneously. In [88] a self-adaptive brain storm optimization algorithm (SBSO) is being proposed for solving multimodal problems. This algorithm uses max-fitness grouping as a clustering method, which can be presented with pseudocode given in Algorithm 3.14.

Algorithm 3.14: Max-fitness grouping as a clustering method

```
1 begin
2     In the search space randomly initialize population P which has N ideas;
3     Find idea which has the best fitness value and denote it as a seed X;
4     Combine M − 1 ideas from population P which are the closest to the idea
        X, in order to form subpopulation;
5     Eliminate these M ideas from P;
6     Repeat steps 2-4 until the population P is divided into P/M
        subpopulation
7 end
```

In order to balance exploration and exploitation processes, self-adaptive control parameter is being applied. By applying this parameter, the convergence of the algorithm towards different optima in different clusters is rapid. The mutation operator works in a way that new idea $u_{i,j}$ is generated by applying the binomial crossover operation to the idea acquired in the last iteration $x_{i,j}$ and the idea after the mutation $v_{i,j}$:

$$u_{i,j} = \begin{cases} v_{i,j}, & \text{if} \quad rand_j(0,1) \leq C_r \quad \text{or} \quad j = j_{rand} \\ x_{i,j}, & \text{otherwise} \end{cases} \tag{3.21}$$

where $i = 1, 2, \ldots, N$, $j = 1, 2, \ldots, n_d$, j_{rand} is a random integer value from $\{1, 2, ..., n_d\}$, and $rand_j(0,1)$ is a random value from the interval (0,1). In each generation, the crossover value Cr_i of each idea is independently generated in accordance to the normal distribution with the mean value Cr_m and standard deviation being 0.1.

$$Cr_i = rand(Cr_m, 0.1) \tag{3.22}$$

$$Cr_m = mean(S_{cr}) \tag{3.23}$$

where S_{cr} is a set of all successful crossover values from the previous generation. At the beginning, Cr_m is 0.5, and S_{cr} is an empty set. The selection operation is done through comparison (of the fitness values) u_i and the closest individual in the population x_s.

3.4 Theoretical analysis

As already stated, in the original BSO algorithm one can identify three main operators: grouping operator, replacing operator, and creating operator. In these operators three control parameters are present, i.e. three probabilities, those being: (1) p_{5a} which regulates the replacing operator, meaning that it controls whether or not the cluster center will be replaced by some randomly generated idea (disruption of the cluster center); (2) p_{6b} which regulates the creating operator, meaning that it controls if new ideas will be created using one or two clusters; and (3) p_{6bii} and p_{6c} which control whether a cluster center or a randomly selected idea will be used for creating new ideas.

How these parameters affect the performance of the BSO and MBSO algorithms are examined in [89]. The simulation results show that p_{5a} value has very little impact on the overall performance of the algorithm. New research on the replacing operator modification possibilities is needed, and it is noticed that it would be possible to create BSO variants without using the replacing operator, in a way that will not degrade the performance of the algorithm. When it comes to parameter p_{6b}, it is shown that its impact on MBSO is much greater than its impact on BSO, where a large value for p_{6b}

is a good fit for MBSO, and a small value for p_{6b} is a better choice for BSO. When the value p_{6b} is smaller, BSO has a greater chance of generating new ideas based on two clusters, which gives the algorithm an opportunity to use more information from the population, thus improving performance. On the other hand, the values of parameters p_{6bii} and p_{6c} have a greater impact on MBSO than BSO. In the case of the BSO algorithm, larger values of parameter p_{6bii} give better results when applied to unimodal functions; the opposite is true when applied to multimodal functions. The reason may be that the probability of BSO using a cluster center for creating new ideas is greater when p_{6bii} is greater. This can increase the speed of convergence and enhance the performance in the case of application to unimodal benchmark functions. When the value of p_{6bii} is small, BSO uses information from the population to create new ideas, making it a good choice for multimodal functions. However, if $p_{6bii} = 0$, that can have adverse effects on the performance of the BSO algorithm. As for MBSO, it has been shown that neither small or great values of p_{6bii} have good results. In this case, the best results in terms of providing a balance between exploration and exploitation are obtained if p_{6bii} has a mean value of approximately 0.4.

The issue of premature convergence is typical of all swarm intelligence algorithms. It happens when the solutions are grouped in clusters (i.e. they converge), and then their divergence does not occur. Clustering mechanism used in BSO algorithm has no adverse effects, but rather, it is used to guide individuals to the better solution regions. Analysis of clustering strategies and other properties of the BSO algorithm is presented in [90]. Through analysis using three unimodal and three multimodal functions, it has been determined that BSO algorithm has great convergence speed at the beginning of the search, which goes to prove that good-enough regions can be located within a few clustering iterations. However, it is necessary to improve the prevention of premature convergence, in order to improve the algorithm's ability to escape from local minimum. Generally speaking, the algorithm has better performance when solving unimodal (simple) problems, while in solving complex functions there is a lot of oscillatory movement involved.

A large number of different optimization problems exist: single-objective, multi-objective, limited, combinatorial, multimodal, and also algorithms meant for the static and dynamic environment. It's known that there isn't a single algorithm for solving

different optimization problems. It is very difficult to find an algorithm which is the best fit for a certain type of problem if we have no prior knowledge of the problem and its environment. The question then becomes: can an algorithm, and in which way, develop its learning capacity in order to better solve the problems which are not known it the moment of the execution of the said algorithm. Therefore, an ideal optimization algorithm should have the ability to self-adapt in order to develop a learning and problem-solving capacity in its specific environment, that is to develop the potential and learning capacity that suits the problem and its environment, which will allow the algorithm to learn better and efficiently solve the problem. The necessity of embedding developmental learning in the swarm intelligence algorithms and an analysis of several swarm intelligence algorithms from a developmental learning point of view are presented in [35]. A framework for developmental swarm intelligence algorithm is developed, which should help in better understanding of the existing developmental swarm intelligence algorithms and developmental evolutionary algorithms, and the implementation of new algorithms of this kind.

3.5 Application of BSO algorithm

BSO algorithm has proven to be successful in solving real-world problems from different fields, such as electromagnetics and energetics, communication, wireless networks, aeronautics, finance, etc. In these applications, special modifications of the original BSO algorithm are used in order to solve a specific problem in the given field. In electrical systems, there is a problem of minimizing the cost of production of the required amount of electric energy (the economic dispatch problem, ED). This is an optimization problem that is especially visible in the case of systems which use wind energy because it is hard to predict. In [91] a BSO algorithm is used for solving the ED problem in a system which contains both thermal and wind power plants. The hybrid algorithm, which is based on the brainstorming process and teaching-learning based optimization (TLBO), is proposed for solving the ED problem in [92]. Finding the optimal location and adjusting flexible AC systems for energy transfer is a complex multi-objective, multimodal optimization problem with constraints, where an attempt to solve this problem by using BSO algorithm is given in [93]. Brain storm optimization is used to solve the ED problem in [94], [95] and [96] as well.

Job scheduling is an important question in systems where execution time needs to be minimized. Optimization of cost and minimization of execution time are objectives which are in mutual conflict because faster resources are usually the more expensive ones. A multi-objective algorithm based on the BSO algorithm, which solves the optimization problem of job scheduling in a grid environment, is presented in [97].

Performances of a wireless sensor network in many ways depends on the network node deployment strategy. However, finding the optimal node deployment strategy in a wireless sensor network, which would achieve multiple objectives such as reduction of cost, robustness in terms of node failure, reduction in computing time and guaranteeing a high level of coverage while preserving connection is a very hard optimization problem. One of the potential solutions to this problem, by using a modified BSO algorithm, is presented in [76].

Prediction of stock market indices is a very important and necessary tool for both investors and the government. However, due to great variability, high noise level and non-linearity of the stock market indices, prediction of this sort is a demanding task and a complicated optimization problem. In [98], an attempt of solving this problem by applying a hybrid approach is presented, with an approach which combines both the BSO algorithm and the gray neural networks model (GNN), where the gray neural network parameter initialization is done by the help of the BSO. It is shown that the proposed algorithm has the ability to overcome the deficiencies of the traditional GNN model with randomly initialized parameters by solving the problem of local optimum and low prediction accuracy.

Optimal reconfiguration of the formation created by multiple satellites in the geostationary orbit is an optimization problem with constraints such as minimal fuel consumption, final configuration and avoiding collision. For solving this problem, Sun and others [99] suggest a modified BSO algorithm based on a closed loop (closed-loop BSO). Three versions of this algorithm have been developed, all of which replace the creating operator from the original BSO with the closed loop strategy. This approach improves performance by using feedback from the search process.

BSO algorithm has found its application even in aeronautics, for the design of automatic systems for plane landing on an aircraft carrier. This system is very critical because it has to enable landing in extremely severe conditions such as poor visibility,

strong wind, and unfavorable sea conditions. A new method for optimizing control parameters in the automatic system for landing on an aircraft carrier for F/A-18A, based on a simplified BSO algorithm, is developed in [100].

Lastly, let us also mention the application of BSO algorithm in electromagnetics: for solving the Loney's solenoid problem [101] and for brushless DC motors [102].

4 ROBOT PATH PLANNING

4.1 Basic concepts

Rapid development of technology creates new and improved possibilities in many different aspects of life. Nowadays, one of promising technologies is the robotic technology. Robotics is a multidisciplinary scientific field that includes computer science, electronics engineering, mechanics and mechanical engineering etc. since it deals with the design of robots, their construction, development of systems for their control, data processing etc.

Robot is a programmable machine that mimics actions or looks like an intelligent being, most often a man, and it needs to have the following properties: 1) sensing and perception) in order to obtain the results from its environment; 2) ability to perform different tasks: mobility or performance of physical actions such as shifting or object manipulation etc. 3) reprogramming; 4) autonomous functioning and/or interaction with human beings. In 1979, Robot Institute of America defined robot as reprogrammable, multifunctional manipulator, designed to move material, parts, tools or specialized devices in order to perform various tasks. Robots are capable of performing tasks without the help of a man or with minimum human activity, usually performing tasks which are dangerous, stressful, hard, or boring for humans. Robots can be classified into several types such as robot manipulator, mobile robot manipulator, legged robot, wheeled mobile robot, underwater robot or autonomous underwater vehicle, areal robot, unmanned aerial vehicle, humanoid robot. Basic components of robots are base, microcontroller, user interface, sensors, actuators, energy conversion unit and manipulators. Manipulators are created by connecting solid parts (links) using joints, which allows relative mobility, i.e. actuation of neighbouring parts. Actuation of joins is done electromechanically, i.e. with the help of electric engines, which enables a robot to perform a certain physical task [103]. The base of a robot can be fixed, like with robot manipulators used in industry, or mobile, designed as a platform with movable parts like legs or wheels. The development of science and technology made possible the presence of robots in everyday life, with their increasing number in industrial and service sectors. The most frequent applications of robots in industry include material handling and transfer, part connection and separation, machine and

component assemblage, welding, painting, inspection etc.

Unmanned aerial vehicles (UAVs) are remote controllable and self-controllable. They became known under the name "robotic aircrafts" and their use is widely spread for both military and civil purposes. They have numerous advantages such as low cost, good ability for manoeuvring and high rate of survival. UAVs are more convenient for monitoring, reconnaissance and observation in hostile and dangerous environments in comparison to the manned aerial vehicles. In addition to the listed qualities, UAVs can contain various devices and equipment such as cameras, sensors, weapons etc. Initially, these vehicles were used for military applications and were known as unmanned combat aerial vehicles, (UCAV). Nowadays however, due to their qualities and characteristics, UAVs are used in agriculture, industry, monitoring and observation and for many other scientific and commercial purposes. Numerous factors contributed to the development of UAV use such as 1) technological development which provided the construction of mighty sensors, microprocessors and other systems for the realisation of efficiency and autonomy that go beyond human possibilities; 2) effectiveness of use of these vehicles for military purposes contributed to the increased investment in development of UAV technologies; 3) UAVs can work in the environments and under the conditions impossible for manned vehicles, such as too high or low altitudes. The technologies that enable UAVs to be operable autonomously without human presence are as follows: navigation sensors and microprocessors, sophisticated communication systems and ground off-board command, communication and control systems (C3). Regarding C3 infrastructure, the following issues can be identified: human-machine interface, voice control, multi-aircraft C3, target identification etc. According to their characteristics, UAVs can be classified into the four following groups [104]:

- Fixed-wing UAVs, which are also known as unmanned airplanes. When they launch they require running or catapult launching;
- Rotary-wing UAVs (rotorcraft UAVs) launch and land vertically, which allows great possibilities for manoeuvring. Various configurations of these vehicles are possible: with main and side propellers (like in helicopters), coaxial propellers, tandem propellers, multiple propellers, etc;
- Balloon or airship-shaped UAVs, which are usually large, endurable and move

with low velocities;

- Flapping-wing UAVs whose configuration is inspired with insect wings.

Besides the listed, other configurations are also known, e.g. vertical launching like a helicopter, but airplane-like flight.

The vehicle is an object which can move and be viewed as a robot that does not contain manipulators. The vehicle is represented by using position and orientation vectors and geometrical model. Position vector defines the position of a vehicle in two-dimensional and three-dimensional space. World space is a physical space where a vehicle moves, and it is usually three-dimensional Euclidean space, although sometimes, for simplification of the problem, movement in two-dimensional space can be observed. Configuration of a vehicle is a set of values that uniquely define a vehicle; it usually contains six values: three components of position vector, and three components of orientation vector. If a robot contains manipulators, the configuration is more complex, since the degree of freedom of each manipulator adds a new parameter to the configuration. Configuration space (C-space) is a set of all possible configurations which a vehicle may have. In the analysis of motion of a robot, it is often necessary to include the concept of state, which consists of configuration of the robot and parameters related to the change of configuration, e.g. in the case of aerial vehicle, the state consists of 12 parameters: three coordinates of positions, three coordinates of velocity, three orientation angles and three angles of orientation change.

State space is a set of all possible states. The number of parameters necessary for configuration or state representation is called number of degrees of freedom. The world space i.e. configuration or state space can be divided into two spaces: free space, which is a set of points that a vehicle may cover during the motion, and the obstacle space, which is a set of points where the vehicle must not be found, because it will thus be in collision with other vehicles. The path is a curved line drawn by a vehicle during motion. It need not be smooth, it does not take into account time as a parameter, and it can include segments so that each segment can be trajectory. Trajectory is a path that includes time as a parameter and can be mathematically described as a polynomial function of time $X(t)$, so that velocity and acceleration can be computed by determining the derivative of the mentioned function. It has to take into account kinodynamic constrains such as the constraints of velocity, acceleration,

rotation, change of direction and so on.

Motion planning can be defined as path or trajectory planning, and it results in a path or trajectory from the initial to target state or configuration [105], Path planning is aimed at finding a continuous curved line in configuration space which begins from initial position and ends in final position. Trajectory planning usually follows the path planning; it takes the algorithm of path planning and determines how a robot can move along the given path. Algorithm of path planning is complete if it is successful in generating the path if the path exists, i.e. it notifies that the path cannot be generated if it does not exist really. Sound planner always guarantees that the robot will reach the desired destination, necessarily avoiding collision with obstacles. This is one of the most significant characteristics of a planner, especially for UAVs, since the consequences of collision can result in permanent damages and catastrophic outcomes.

The problems in path planning can be classified by several criteria. The problem is static if complete knowledge of environment is known. In the case the knowledge is not complete, or changes during the motion of a robot, the problem is dynamic. If the obstacles do not move, the problem is time-invariant, otherwise time-variant problem is discussed. Differentially constrained problem is where equations of vehicle movement are constrains, so that the path must be a trajectory of dynamic system. The best known types of motion planning problems are point robot, point robot with differential constrains, Jogger problem, bug problem, weighed region problem, Mover problem, general vehicle with differential constrains, time variable environments and multi-mover problem.

4.2 Path planning problem

Let mobile robot with k degree of freedom be given, which can move in two-dimensional or three-dimensional space. The space is with obstacles known to the robot. If we assume that initial and final desired position of the robot are known, the problem of robot's motion is 1) determination whether continuous movement of the robot from the initial to the desired position is possible;2) the path planning if the answer to 1) is positive. Basic issues of interest, i.e. the steps in formulation of the problem of motion planning are the following: computation of configuration, representation of

the object, approaches to motion planning, search methods and local optimization of motion [106].

Path planning of mobile robots has been a very active field of scientific research since nineteen sixties. This filed drew greater scientific attention after publishing of a paper [107]. There, the algorithm is proposed for the case of motion of polyhedron object among obstacles of the same shape. The algorithm works in an iterative way, beginning by creation of path in the form of a straight line between the initial and end points. If such path contains the locations of collision, a new path is proposed based on the information on collision, and the given procedure is repeated until the path without collisions is created.

Depending on the fact whether complete knowledge of environment where the robot moves is known to the robot, path planning can be classified into two types: global path planning or deliberate approach, and local planning or reactive approach. Global path planning (GPP) is a procedure by which the path without obstacles is determined in the case when the robot has complete information of its environment. GPP can be done offline. If the information on the environment is only partially known or completely unknown, then local path planning (LPP) or online path planning is carried out. In the case when the terrain is insufficiently known or undefined, GPP procedure is not sufficiently robust, hence LPP can be successfully used for finding optimum path.

4.2.1 Path constraints

When planning the path or trajectory, numerous demands and constrains must be taken into account. Robot's motion should take the shortest path possible so that fuel consumption is minimum, and the path from the initial to end position is passed in shortest time. Besides, another demand is that a robot should not collide with obstacles or other robots; especially it is necessary to minimize the exposure of UAVs to threats. UAV must follow the path with minimum probability to be exposed to threats of enemy radars, rockets or other aerial vehicles. A constraint may be related to kinodynamic properties of robots or constraints of the environment. Additional constraints that have to be considered during path planning of UAVs are: constraints of turning angle α, constrains of climbing/diving angle β, height of flight h and angle

of approaching the target position. Constraint of turning angle requires the path where the turning of vehicle is smaller or equal to certain threshold value in order to avoid damages of its structure or its collision with other vehicles. Constraint of climbing/diving angle has basically the same sense like the previous constraint except that it refers to the direction of altitude. It can be positive or negative, depending on the course of movements (it is negative in diving). It is required that abrupt changes should be avoided, so that the risk of collisions is minimized. A constraint of minimum flight height is necessary so as to minimize the probability of detection by the enemy, since low flight provides masking effect of the terrain. On the other hand, very low height has the risk of vehicle crash and permanent damage, which must be avoided. In some applications such as attack operations, the optimum direction of motion is determined in advance, so the robot has to follow that direction, i.e. that specific angle of approaching the target position.

In addition, the characteristics and constraints of environment such as distances between obstacles, probability of appearance of dynamic obstacles, configuration of terrain, forbidden zones of flight, flight map range, etc. have also to be considered. The environment where the robot moves may have various characteristics related to obstacles and threats which the robot faces. Obstacles may be static or dynamic, and the sources of threat can be permanently present and certain or uncertain, whereby uncertainty means that it is impossible or too expensive to define their exact position in advance, such as e.g. fire in rescue missions, or enemy and mines of a battlefield.

Discontinuities of a curve that represents the path bring numerous shortcomings such as instability of control system, overshooting, unpleasant feeling of passengers, while in some applications they may even cause mechanical damage or failure. The path in the shape of continuous curved line proved to improve stability and control of industrial vehicles [108]. Kinodynamic constraints of a robot demand the construction of continuous smooth curved path, whereby these demands are more important for UAVs, since their kinodynamic constraints are considerably stricter. Namely, the aerial vehicles cannot move along the discontinuous paths, because abrupt turnings can be harmful for the structure of a vehicle. The methods and strategies for path smoothing are very often a part of path planner. Sometimes it is incorporated in optimization algorithm, and sometimes it is realised after generation of optimum paths in the form

of straight lines.

Smoothness can be represented as the sum of reflection angles by any three neighbouring nodes on the path. Direct computation of smoothness is usually time demanding, therefore the procedures for indirect computing are often applied. One of possible procedures is given in [109], and it uses to parameters S_c and S_p, in order to compute path smoothness. The parameter S_c is a relation of deflection angles that are less than given expected value and total number of deflection angles, while S_p is a relation of number of path segments that are larger than a number of segments in the path with the smallest number of segments and total number of path segments. Smoothness can then be computed as following:

$$S = \alpha * S_c + \beta * S_p \tag{4.1}$$

$$S_c = 1 - \frac{DA_l}{N_f - 1} \tag{4.2}$$

$$S_p = 1 - \frac{S_{min}}{N_f} \tag{4.3}$$

where N_f is a total number of path segments, DA_l is a number of deflection angles higher than the expected value; S_{min} is a number of segments in a path with the smallest number of segments; α and β are weight coefficients.

For robots with limited turning angle Dubin's paths were often used to achieve smoothness. Dubin's research [110] has shown that the shortest path which can belong to one of six possible path types can be constructed for robots with known initial and final configuration in two-dimensional environments without obstacles. This path must consist of not more than three parts in the shape of straight lines or arcs. Such solutions based on this one were proposed for achievement of UAV smoothness in 2D and 3D surroundings [111], [112].

To obtain smooth path the algorithms based on spline curves have been frequently used lately, among which the most famous are Bezier curves, B-Spline curves and NURBS (Non Uniform Rational B-Splines) curves [113]. Bezier curves were used in CAD applications and were developed for the needs of car industry. Bezier curve $c(u)$ with degree n is defined by the equation (4.4), where u is normalized parameter of the curve, and $B_{n,i}(u)$ is Bezier blending function for i-th control point P_i. These

functions are expressed by the equation (4.5).

$$c(u) = \sum_{i=0}^{n} B_{n,i}(u) P_i \tag{4.4}$$

$$B_{n,i}(u) = \frac{n!}{i!\,(n-i)!} u^i (1-u)^{n-i} \tag{4.5}$$

Blending functions do not have local effect on generated curve, which may be a limiting factor in the situations when the modifications of the path are demanded due to detected obstacles. By using Bezier curves, the path can be represented by smaller number of parameters in comparison to the case when complete geometrical description of path is used, which considerably improves the performances of the whole algorithm. Therefore, Bezier curves with different numbers of control points are used to achieve the path smoothness. The order of Bezier curve depends on the number of control points and the construction of Bezier curves demands the defining of all coordinates (horizontal and vertical) of the corresponding control points in advance, because otherwise the use of high degree curves is necessary for generating the path, which is computably inefficient in the case of long paths.

B-Spline curves are mostly used for generating the path of industrial robot manipulators, and their use for mobile robots is still limited. B-Spline curve of degree p, $c(u)$, is defined by using n control points P_i and node vector $\hat{u}$ which includes m non decreasing real numbers, whereby number of nodes m is equal $n + p + 1$. If u is a normalized parameter of the curve length, and $N_{i,p}(u)$ is i-th basic function of B-Spline curve of degree p, then it can be expressed by the following equation:

$$c(u) = \sum_{i=0}^{n} N_{i,p}(u) P_i \tag{4.6}$$

The number of base functions is equal to the number of control points, and for their computing, the recursive algorithm is used. Unlike Bezier curves, the order of B-Spline curves does not depend on the number of control points, and base functions have local effect on the generated curve, which means that the modification of a curve is possible when necessary, e.g. for detection of an obstacle.

NURBS are weight modification of Non-Uniform-B-splines, and their use in robotics

has not been developed yet. They are applied where accuracy and computing efficiency are necessary, then for generation of tool path, blood vessel modelling, finite element analysis etc. NURBS curves are given in the equation (4.7):

$$c(u) = \frac{\sum_{i=1}^{n} \omega_i N_{n,i}(u) P_i}{\sum_{i=1}^{n} \omega_i P_i} \tag{4.7}$$

Weight coefficients ω_i are assigned to control points P_i which enables curve to move according to defined control points. The introduction of these coefficients provides better flexibility of generating the curves by changing control points, nodes and weights. With respect to base functions and orders, the characteristics of NURBS curves are similar to B-Spline curves.

One of ways of obtaining path smoothness is applying the k-trajectories dynamic strategy [111], which can be explained as following. Let a path segment be defined by three control points ω_{i-1}, ω_i, ω_{i+1}, and $\overrightarrow{q_i}$ is a unit vector in the direction from ω_{i-1} to ω_i, and $\overrightarrow{q_{i+1}}$ is a unit vector in the direction from ω_i to ω_{i+1}. Also, let φ be angle between $\overrightarrow{q_i}$ and $\overrightarrow{q_{i+1}}$,while $\hat{\mathrm{C}}$ is an arc with radius R and centre C_i that lies on the bisector of angle created by the three control points, where R and C_i are given by the following equations:

$$R = 0.5min\left\{\|\omega_i - \omega_{i-1}\|, \|\omega_{i+1} - \omega_i\|\right\} \tan\frac{\varphi}{2} \tag{4.8}$$

$$C_i = \omega_i + \left(\frac{R}{\tan\frac{\varphi}{2}}\right) \frac{\overrightarrow{q_{i+1}} - \overrightarrow{q_i}}{\|\overrightarrow{q_{i+1}} - \overrightarrow{q_i}\|} \tag{4.9}$$

The arc cuts the lines ω_{i-1},ω_i and ω_i,ω_{i+1}, so now the original segment of the path that consists of three control points is transformed into a segment defined by two lines A'A and B'B and arc $\hat{\mathrm{C}}$, as shown in the Figure 4.1. Waypoints are represented by circles, while control points are represented by squares.

This method has a few advantages: it is easily integrated into algorithms for path planning that generates a path in the form of connected straight lines; the method does not require new computing demands, because smooth trajectories are generated in real time, during the motion of a vehicle along the trajectory; it minimizes the time

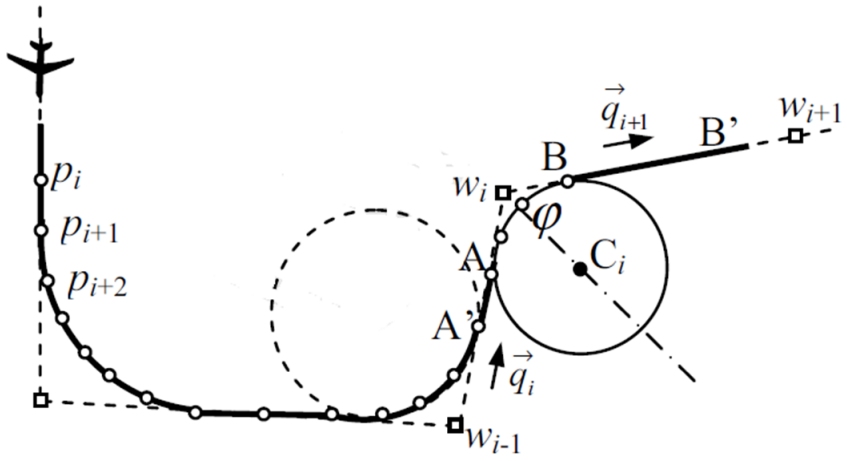


Figure 4.1 Path smoothness strategy

of vehicle deviation from originally generated path in the shape of straight lines.

4.3 Methods of solving path planning problem

Mobile robot path planning is an active subject of scientific research, of great practical importance hence many methods were proposed for its solving. Generally viewed, in mobile robotics two types of methods for path planning are known: traditional (classic) and heuristic. When solving problems of path planning most traditional methods take into account only minimization of path length. Traditional methods require longer computing time and their shortcoming is related to trapping in local minimum [114]. When the scientific community concluded that the path planning problem was NP-complete optimization problem, the attention was directed towards development and application of heuristic methods for its solution.

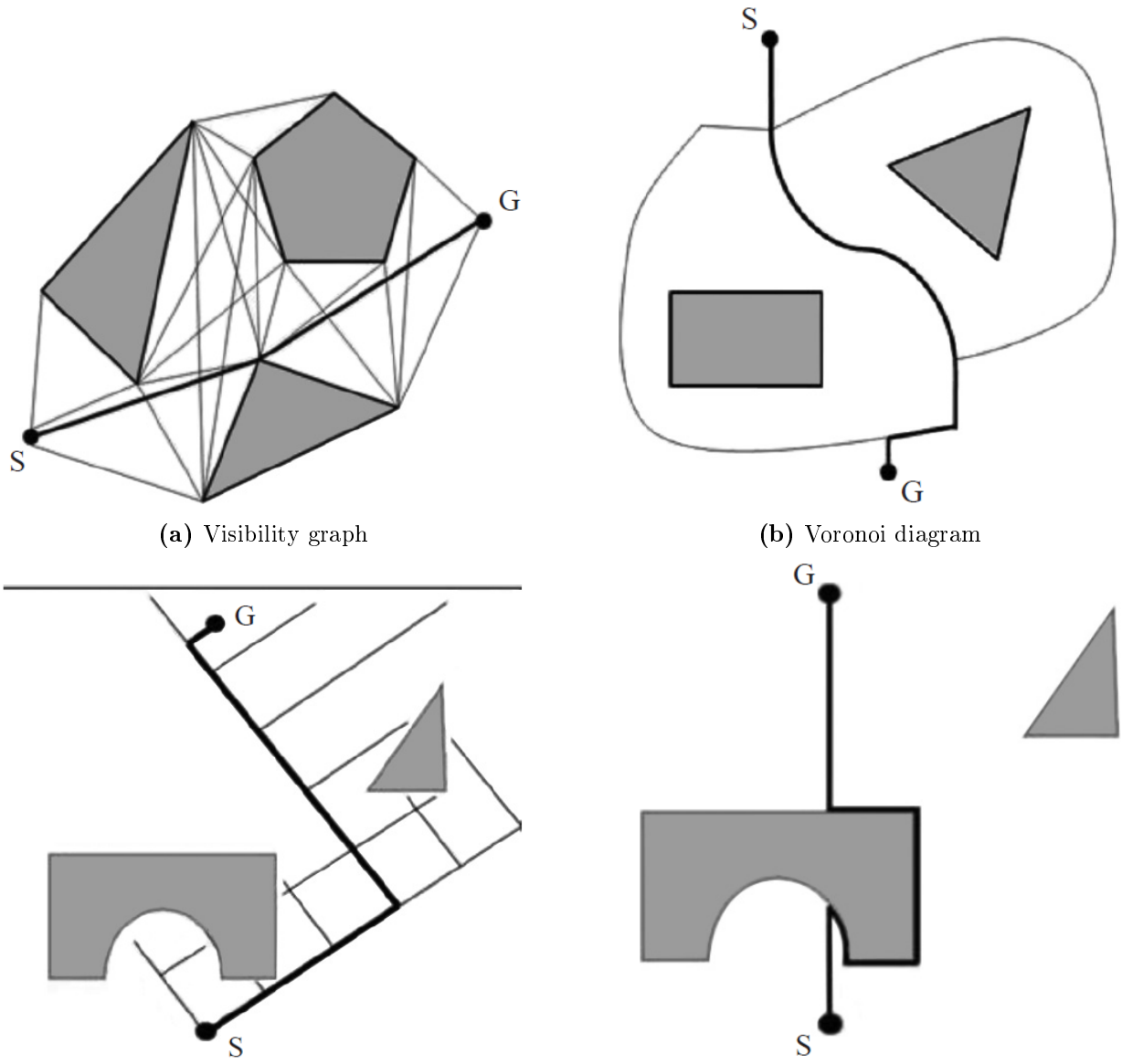
Methods and approaches for solving path planning problem can be divided in a few different groups. The first approach is called skeleton, also known as roadmap or highway approach. In this approach, configuration space is mapped as a network of one-dimensional lines and the problem of motion planning is thus transformed in the problem of searching the appropriate graph. Motion planning is done in three steps: 1) Robot moves from starting configuration towards the point on a roadmap; 2) robot moves from target configuration towards the point on the roadmap; 3) two points are connected using lines on roadmap. Roadmap has to present all topologically different

feasible paths in the configuration space, otherwise the algorithm for motion planning is not complete. The best known roadmap methods are Voronoi diagram, visibility graph, freeway method, silhouette method and subgoal networks (Fig. 4.2).

Visibility graph provides exact solution for the problems of point robot, and in that case its complexity is quadratic. However, its shortcoming is that it is applicable only in two-dimensional environments, because in more complex configuration space its solution belongs to NP-hard class. This approach is based on the fact that the shortest part touches polygonal obstacles on their nodes, thus creating roadmap of lines that connect each node with other nodes that are visible from its position. By applying the visibility graph, optimum path is often close to obstacles, which is risky in the problems where the position of obstacles is not exact.

Bearing in mind that it is very difficult to control and prevent collisions in the case of path with minimum distance from obstacles, many roadmap approaches based on skeleton are suggested, among which Voronoi approach is the best known. Voronoi diagram is a kind of graph that is used as a general solution for the problem of proximity in 2D environment, which can be descried as following: if the set S of n points in a plane is given, it is necessary for every point s in the set S to determine the region that consists of all points in the plane closer to the point s than any other point s in S. Voronoi approach produces a skeleton which has maximum distance from obstacles and finds the path of minimum distance that follows the skeleton. The algorithm works in two-dimensional space and its complexity is $O(NlogN)$. Hierarchical Voronoi graph is an attempt to generalize this approach and apply it to multiple dimensions. Freeway method also makes a skeleton which is distant from obstacles, so that free space is filled with cylinders.

Cell decomposition the second approach and is classified as exact cell decomposition and approximate cell decomposition. In exact cell decomposition, free configuration space is decomposed in the set of convex polygons, which are interconnected by a graph. The optimum path is searched by using the methods of graph search, whereby Dijkstra algorithm is the most often used. The best known methods of exact cell decomposition are trapezoid decomposition, decomposition based on critical curve, cylindrical algebraic decomposition and the method of connected balls in free space. Trapezoid (vertical) decomposition divides free space into trapezoid regions



(a) Visibility graph

(b) Voronoi diagram

(c) Subgoal networks

(d) Silhouette method

Figure 4.2 Roadmap methods [115]

which do not share obstacles. Then, roadmap is created by connecting midpoints of adjacent trapezoids; the search algorithm is then applied to the obtained graph. This approach is suitable for application in the problem of point vehicles and its complexity is $O(NlogN)$. For rigid vehicles with ability to rotate, the approach based on critical curve is applied. This decomposition is carried out by division of free space into critical and non-critical regions, so that the boundaries of these regions are part by part a

polynomial curve. The regions are connected into a graph, which is then searched by using familiar methods. The complexity of approach is $O(N^2 logN)$. The methods of exact cell decomposition also include cylindrical algebraic decomposition which is the extension of decomposition based on critical curve in case of three-dimensional problems, and method of connected balls in free space. The methods of approximate cell decomposition include rectangular decomposition and $2m$ tree decomposition. Rectangular decomposition divides free space into rectangular regions so that each can be completely filled (black) partly filled (grey) or completely empty (white). Square of cubic grids are the most frequently used approaches, while A* or D* methods are usually used for search. $2m$ tree decomposition has been increasingly used lately because it reduces the number of necessary points for presentation of obstacles in comparison to other methods of decomposition.

The third approach is called the potential field approach, where potential function is determined based on information about obstacles, and the path is generated by application of optimization of steepest gradient descent method on that function. The concept of potential field was introduced in 1986; it defines a potential function as a differentiable real function whose value can be considered as energy, thus its gradient is a force. The gradient of the given potential function is a vector which shows direction that maximizes potential energy. In the potential field approach, a mobile robot is observed as a point in space which is under the influence of artificial potential field U, whereby local variations of the field give information on the structure of free space. Potential function can be defined as a sum of attraction potential, which pulls robot towards the final configuration and repulsive potential which repels the robot from obstacles. The potential field approach is a method which is still used a lot for solving the problem of robot motion planning thanks to small computing complexity, especially in the case when the degree of freedom is great [106]. The potential field approach include following methods: potential field with gradient descent, search guided by potential field, harmonic potential functions, continual Dijkstra, wave front expansion, wave front expansion with skeleton. Potential field with gradient descent method (virtual force field) is original and the oldest method in the group. Target position is assigned a decay function with minimum negative value, and each obstacle is assigned special decay function with maximum positive value. The value of these

functions is then summed up, thus total potential field is obtained. Since the previous method can be trapped in local minimum, search gguided by potential field can be used instead. Continual Dijkstra method is used for two-dimensional problems, and it works by dividing the space into visibility polygons, while wave front expansion is a version of the previous method that can be used for multi-dimensional problems, and is very similar to complete grid search by using dynamic programming.

The fourth group could include all other heuristic or hybrid methods such as fuzzy logic, neural networks, random trees, probabilistic roadmaps and nature-inspired metaheuristics.

The insight into literature reveals that the greatest number of research is dedicated to the analysis of path planning problems in two-dimensional space. However, the environments where the robots move, such as forests, urban or underwater environment are usually non-structured and abundant with unpredictable factors, hence three-dimensional (3D) algorithms are necessary. Current 3D algorithms for robot path planning can be classified into the following categories: sampling based algorithms, node based optimal algorithms, mathematical model based algorithms, bio-inspired algorithms and multifusion based algorithms [116].

Sampling based algorithms require the information on the whole environment, i.e. it is necessary to create a mathematical model for space description. Usually, environment is firstly sampled in the node-like, cell or other parts, and then the search is carried out in order to obtain feasible paths. These algorithms can be active, which include rapidly exploring random trees (RRT), dynamic domain RRT (DDRRT), RRT-Star and artificial potential field;and passive, among which the most significant are 3D Voronoi, RRG, probabilistic random maps (PRM) and modifications such as kPRM and sPRM etc. Node based optimal algorithms can be divided in three groups: the first includes Dijkstra algorithm, A*, Theta* and LPA* are in the second group while D*, D*-Lite and similar are in the third group.

Mathematical model based algorithms are linear algorithms and optimal control. These methods first use various equalities and inequalities to model kinodynamic constraints that serve as constraints of objective function, and then the optimum solution is searched. Flatness based model, mixed integer linear programming (MILP), binary linear programming (BLP) belong to this group of algorithms. Multifusion

algorithms are basically a combination of a few different methods in order to remove their shortcomings and achieve better performances.

Most traditional methods are based on constructing the graph by which geometric structure of environment is represented. By using the given graph a graph searching technique is applied in order to find the best path with the initial and end point. Geometric structure of a graph depends on the approach used for problem solution. The most often used methods for search of thus generated graph are A* search, Voronoi diagram search method, mathematical programming, bi-level programming and D-lite algorithm. Most of these algorithms use Eppstein k-best algorithm for finding optimal path. Its shortcoming is that it does not take into account UAV motion constraints, hence it cannot be used in real situations. The most of the listed traditional methods have general disadvantage which is manifested in trapping the solution in local optimum [117].

4.4 Path planning as optimization problem

Path planning can be understood as a multi-objective optimization problem which includes numerous requirements and constrains. Unfortunately, these objectives and requirements are often mutually contradictory, hence certain compromises are necessary in problem solving. The problem of path planning is a hard optimization problem, thus no deterministic methods for its solving in reasonable time are known. While solving path planning problem by application of nature-inspired optimization algorithms it is necessary to take the following steps: to adopt suitable mathematical model of environment, create an adequate objective function and select the optimization algorithm. There are several mathematical models for presentation of environments; the best known are vector model, where the obstacles are shown as polygons, grid or occupancy cell, and graph model where the best known are MAKLINK graph, Voronoi diagram and visibility graph.

MAKLINK graph serves for modelling free space where mobile robot is moving. This approach works under the following assumptions: 1) a robot moves in limited 2D environment; 2) environment has a polygonal shape and contains polygonal obstacles; 3) obstacle boundaries are extended for the value equal to half length or width of robot to which minimum distance among relevant sensors is added. Thus the robot can be

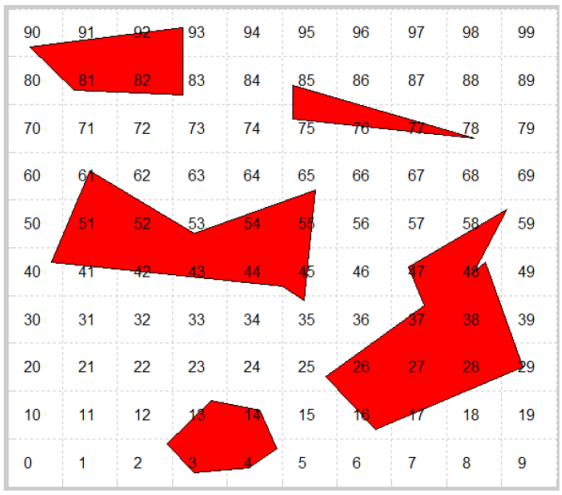


Figure 4.3 Grid model of environment with marked polygonal obstacles [119]

considered a point in further algorithm execution. In MAKLING graph model, each line of a polygon that represents an obstacle is surrounded by a few free MAKLINK lines which can be defined as following [118]: 1) endpoints of a line are two nodes of various obstacles, or one node is on the obstacle while the other is on the boundary of environment; 2) free MAKLINK line cannot cut any obstacle.

Grid model can be realized in two ways: as X-Y coordinate plane, or as an orderly numbered grid, whereby the latter approach is dominant in the literature. In orderly numbered grid, empty cells denote the space where the robot can move freely, while grey cells denote the space with obstacles, whereby the size of grey part is defined by the size of obstacle to which the safe distance is added depending on the size of robot. In this way, the potential path consists of segments that connect the grid cells with different numbers and can be expressed as a series of numbers of grid cells occupied by the robot during its movement along the path.

After the constructing of suitable mathematical model of environment where the robot moves, it is necessary to create mathematical model of the path in accordance with the selected method for solving the path planning problem. The very core of path planning problem includes the establishing and defining efficient model for representation of objective function i.e. cost function of the path, because the value of the function is the criterion for path evaluation; the lower the cost, the better the path is, and vice versa. Objective function has to be defined in such a way to take

into account objectives, requirements and constrains.

When multi-objective optimization problem is solved, then more solutions can be found that optimize one objective at the cost of another. Such solutions are called Pareto optimal set or front, and any solution in this set can be taken as valid. Also, there are such problems where the advantage of one solution in relation to the other is given according to non-numerical qualitative information or heuristics. In the case of path planning problem, it may refer to fewer changes of motion, motion through fewer number of grid nodes, more turnings to the right than to the left etc. Bearing in mind numerous requirements and constrains, it is very difficult to create a mathematical model for the problem of path planning which would be solved by optimization methods. If the optimization problem is such that all constraints cannot be satisfied, then it is called constraint satisfaction problem (CSP). It may happen that the number of constraints in path planning problem is too large to successfully solve them by algorithm in reasonable time. Then, the solutions that are not in the form of Pareto optimal set, but have CPS form are accepted or preferred.

5 ROBOT PATH PLANNING BY NATURE-INSPIRED METAHEURISTICS

In recent years large number of nature-inspired algorithms is proposed for the mobile robot path planning problem. A review of the research aimed at solving the path planning problem of robots, UAVs and UCAV aircraft using these algorithms, with a special emphasis on the algorithms of the intelligence swarms, is given in this chapter. In [120], a simulated annealing algorithm was applied to the problem of mobile robot path planning in a two-dimensional space, taking into account three representations of the path: linear, Bezier curve and spline interpolated curve. The aim is to find the optimal path without collision, and the criteria function is defined based on the length of the path. The simulated annealing algorithm combined with the traditional artificial potential field method in order to alleviate its shortcoming which refers to trapping at the local minimum is used in [121], [122].

5.1 Evolutionary algorithms

A real-time path planning system based on an evolutionary algorithm is proposed in [123]. The planner has the ability to work in an environment where the changes are unpredictable, and it takes into account different constraints of the problem, such as the minimum length of the path, flight altitude, the maximum angle of rotation and the fixed approach vector, i.e. a certain approach angle to the target position, and the planner can also be used to work in the system with one and more vehicles. In [124], a path planning algorithm for multiple UAV vehicles in real-world conditions is proposed. The proposed solution is based on evolutionary algorithms using the approach of multiple coordinated agent co-evolution. The obtained paths are calculated based on the properties of the real UAV, terrain characteristics, radars and missiles, and are structured by different priority levels depending on the mission of the aircraft. The planner works in offline and online mode in order to deal with unpredictable risks during the flight and, if necessary, to recalculate parts of the path. The evolutionary approach is also used in [125] for finding the path in a complex environment, as well as in [126].

5.1.1 Genetic algorithm

Pehlivanoglu [127] proposed a multirequency vibrational genetic algorithm (mVGA) as a solution for the problem of UAV path planning. The author implemented a new mutation strategy, and in the initial phases of the algorithm the clustering methods and the Voronoi diagram are applied. The mutation operator is applied twice after the crossover operator. The first application is done in order to ensure global random diversity in the population, while the other aims at local diversity in the neighbourhood of an elite individual. The initial population is presented as the sum of the random population and the rest of the population. The random population consists of individuals who are generated in a random way, in accordance with the constraints of applied model of the terrain. The rest of the population is generated using Voronoi diagram. The path modelling is done using Bezier curves, because they allow the calculation of smooth dynamic paths. The objective function is designed to achieve three requirements: path length minimization, path smothness, and maintaining the safe distance form ground. It is implemented as a liner combination of these three requirements. The proposed algorithm is compared with three other competing GA based algorithms, in two different 3D environments: the sinusoidal and urban model. The results confirmed the superiority of the proposed solution in terms of the required calculation time.

[128] an improved version of the genetic algorithm for mobile robot path planning is described. The modifications refer to crossover, mutation and deletion operations of the basic GA algorithm, as well as the integration of a control algorithm based on fuzzy logic that self-adaptively adjusts the probabilities of crossover and mutation operations. For mathematical modeling of the problem, an orderly numbered grid is used. One of the shortcomings of the GA algorithm in its application path finding the optimal path of mobile robots is the mutation operator, which results in the generation of unfeasible paths. In that sense, the authors [129] designed a new GA algorithm with a modified mutation operator for mobile robot path planning in a dynamic environment with obstacles. The environment model is a orderly numbered grid. The aim of the algorithm is path minimization, and the criteria function is designed to use the penalty method for unfeasible paths, i.e. paths that contain obstacles. The penalty is added to the total path length, and its value is greater than the longest generated

path. A computer-efficient algorithm for robot path planning, where the criteria are path length and safety is given in [130]. The environment is represented by a grid, and the wavefront method is used to create numerical potential fields, for both target points and obstacles. AL-Taharwa et al. [131] proposed the implementation of the GA algorithm for mobile robot path planning in a static environment that is modelled with a grid, analyzing two cases of movement: with or without obstacles.

The application of a genetic algorithm with an elitist non-dominated sorting to solving a multi-objective vehicle path planning is shown in [132]. Four different schemas for path representation are proposed, and the environment is represented in the form of a grid. The objectives are defined in a way to optimize the path length and safety, and path smoothness is also given as the secondary objective. In addition, several important issues related to the problem of path optimization, such as dealing with constraints, identification of an efficient scheme for path representation, differences between single-objective and multi-objective path optimization, as well as evaluation of the proposed algorithm on very large grids where the obstacles are densely deployed are analysed. Cheng et al. [133] proposed an immune genetic algorithm (IGA) with an elitist approach for UAV path planning. The modified algorithm uses the immune operator and the concentration mechanism, which reduces the inherent shortcomings of the original GA algorithm related to premature and slow convergence. Nikolos et al. [134] described an algorithm that represents a combination of multiple genetic algorithms for offline/online UAV path planning in 3D, using B-Spline curves whose control point coordinates are artificial genes of evolutionary algorithm.

Three-dimensional UAV path planner implemented using the multi-objective non-dominated sorting genetic algorithm II (NSGA II) is given in [135]. The environment is mathematically represented in the form of meshed 3D surface. The algorithm has two objectives: minimization of the path length and maximal safety margin, and the generated path is represented by B-Spline curve, so that the B-Spline control points are decision variables of the genetic algorithm. Solutions for the robot and autonomosu vehicle path planning problem that are based on the GA algorithm and its modifications are also given in [136], [137], [138], [139], [140].

5.1.2 Differential evolution

Zhang and Duan [141] described an improved version of the DE algorithm for solving the UAV path planning problem in 3D environment. The authors designed their solution considering two objectives: a short path and a low flight altitude. In addition, many constraints existing in real flight conditions, such as: maximum angle of rotation, maximum climbing/diving slope, terrain configuration, forbidden flight zone, threat zone, range of flight map are considered. Mathematical representation of the environment and the path is the same as in our research in 6.1, with the difference of a added third height coordinate. The proposed algorithm uses dynamic strategy for path smoothness, and the search space is limited in order to make the process more efficient. The objective function is represented by the sum of two components, J_L and J_H, which relate to the path length and flight altitude, sequentially. The path length is calculated as the sum of the segments, where each segment is calculated as the Euclidean distance between two points. It is desirable that UAV aircraft fly to lower altitudes, since it this way it better avoids radars, due to masking of the terrain. The flight altitude costs can be calculated using the formula $J_H = \int_{P_{UAV}} H_p dl$, where $H_p = 0$, if z_k is less than 0, and otherwise it has the value z_p (the value of the altitude in point p). The optimization problem is considered as constrained optimization problem, where constraints which refer to maximal rotation angle, climbing/diving slope, and terrain constraints are represented with inequalities (inequality constraints), while constraints referring to forbidden flight zones, threat zones, and flying outside the map range are represented with equalities (equality constraints). For more efficient solving of the path planning problem, the authors performed the transformation of the constraned optimization problem to the unconstrained problem using α level comparison technique. With this technique, the α satisfaction level is introduced for constraints, and it defined how well the potential solutions meet the constraints, e.g. if the level of satisfaction is less than 1, the solution is not feasible. Thus, the selection operator of the original DE algorithm is modified using the α level comparison and differential evolution with a comparison level (DELC) is created. Additionally, the authors suggested a modified strategy for updating α values using the sigmoid function. The proposed DELC algorithm is compared using numerical experiments with six existing optimization algorithms with constraints and five methods based on penalty functions

(which traditionally serve to deal with constraints). The results showned that DELC algorithm exhibited good performance in terms of quality, robustness, and possibility to satisfy constraints. The implementation of 3D trajectory planner for UAV aircraft using an improved version of DE algorithm is described in [142]. In order to eliminate the drawbacks of the original DE algorithm regarding premature and slow convergence problem, the authors proposed the usage of a chaotic search in the mutation operator. The criteria function (cost function) is calculated as the sum of costs related to length and height of the path, and threats along the path.

The aim of the paper [143] is to examine the possibility of using the DE algorithm for offline planner design in a static sea environment, which has has the ability to generate 2D paths for coordinated navigation of multiple UAV aircrafts. Starting from the assumption that each UAV moves from a different initial location, the algorithm should generate a path with a certain desirable distribution of velocities along the paths, so that the aircraft arrive at the destination while avoiding collisions with each other and with obstacles, as well as meeting the path and coordination objectives and constraints. Cooperative path planning problem of multiple robots is also analysed in [144]. The proposed solution uses parallel differential evolution algorithms, and the authors compared two possible approaches to problem solving: centralized and distributed. The results showed that the distributed approach was better than centralized, and that it was competitive with PSO implementation of the planner.

5.2 Particle swarm optimization

In [145] a preliminary study of multi-swarm sharing scenario for particle swarm optimization and its application in solving the UAV path planning problem is given. In order to implement the proposed algorithm, which works with multiple swarms, the authors included two new processes: swarm crossover and swarm manager. The swarm crossover is implemented following the crossover concept from genetic algorithms. First, two parents are selected from the set of R globally best positions (R is the number of swarms), and then two crossover points are randomly chosen from selected parents; their information behind chosen point change, and thus new offsprings produced which expand or contract search dimension. The offsprings are then evaluated and their fitness values are compared in order to choose a better one that will

serve to generate a new swarm. In order to remove unwanted particles and swarms, and to prevent unreasonable growth of population, swarm manager is introduced, which in each generation deletes particles with the worst fitness value. This approach has a positive influence on the calculation time of the algorithm. The proposed method proved to be good in finding a suitable and feasible path in the 3D flight model.

Modified membrane inspired particle swarm optimization (mPSO) is proposed in [109] for solving mobile robot path planning problem in two dimensional environment with dynamical obstacle. The path planning problem is treated as multi-objective optimization problem where three goals are considered: distance, safety, and smoothness. In order to obtain a compromise between mutually opposing goals and improve the convergence of the algorithm, point repair algorithm and smoothness approach are introduced. The safety degree is the a sum of the deviation degrees C_i $(i = 1, 2, \ldots, N)$ between any segment of the path and the closest obstacle, and it is defined as:

$$SD = \sum_{i=1}^{n-1} C_i = \begin{cases} 0, d \geq \alpha \\ \sum_{i=1}^{n-1} e^{\lambda - d_i}, d < \alpha \end{cases} \tag{5.1}$$

mPSO algorithm uses the dynamic structure of the membrane, where OLMS and D-OLMS are altered in order to adjust population of particles, that represents a potential robot path and specify different rules such as membrane division, transformation and communication rules, and membrane decomposition. PSO particle dimension is dynamically changed during algorithm execution. Point repair algorithm is used to convert unfeasible paths into feasible. The smoothness algorithm removes unnecessary nodes, thus reducing particle dimension. Moreover, motion direction adjustment technique is used to accelerate the convergence of the algorithm. The proposed algorithm is compated with PSO and GA in different environments with three grid models and five obstacle types, and efficiency of mPSO algorithm is proved.

The aim of the research in [146] is to analyse intelligent vehicle path planning problem in a dynamic environment. The proposed method is based on PSO algorithm and behavior dynamics method. The behavior dynamics method is responsible for generating competitive behavior problem, while PSO approach is used to improve behaviour coordination. Dynamic model is constructed based on behaviour variables

and behaviour pattern that the intelligent vehicle follows. Behavioru variables are heading angle and velocity of intelligent vehicle. The objective of general vehicle behaviour consists of two elements: the behavior of reaching target position and the behaviour of avoiding obstacles. For each of these two elements, appropriate models are constructed, separately for each of behaviour variables. After defining different dynamic models, the proposed method requires that coordination of these behaviour models is performed by fusing together several variants of behaviour, i.e. to make a fusion of variables using corresponding weight coefficients, and then to implement vehicle path planning. PSO optimization algorithm is here used to optimize mentioned weight coefficients, whereby the fitness function is defined to reflect the relationship that exists between the vehicle and the environment, i.e. obstacles. The appropriate type of behaviour should be a priority depending on the movement dynamics: if the obstacle is close to the vehicle, behaviour of avoiding obstacles should take priority, and vice versa, if the vehicle is far from the obstacle, the behaviour of reaching the target position is a dominant one. The simulation results showed that the proposed method was efficient in terms of reliability and real time performances, proving that PSO algorithm is a good solution for behaviour coordination problems.

Three-dimensional algorithm for UAV path planning, based on adaptive sensitivity decision operator combined with PSO algorithm is given in [147]. Mathematical modelling in the proposed method uses cardinal spline functions, while a cylindrical model is used to define deterministic threats such as radars, artillery, and missiles. The mathematical representation of the path is done in a way that is often used in the literature. Start and target points are marked and connected to form a straight line. The straight line is the divided using vertical lines perpendicular to the line into $M+1$ segments, and the intersection points are called waypoints. Threats are represented with circles, so the sensitivity to the threat is given by a probability which is inversely proportional to the distance from the center of the threat; when the distance from the path segment to the centre of the threat is greater, the probability that threat poses a risk to the aircraft is smaller. the less likely the threat will be the danger to the aircraft. Path planning objectives are avoiding all thereat areas with minimum costs, finding the shortest path i lowest possible altitude. The mathematical representation of the problem requires the transformation of the coordinates in order to accelerate

the algorithm. The straight line connecting starting and target point becomes a new horizontal axis, and the new vertical axis is obtained by rotating the old one in the counter clockwise direction for the angle θ between the new horizontal axis and the old one. The geometrical relationship between the waypoints in the new and old coordinate system is given by the following equation:

$$\begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_O - x_S \\ y_O - y_S \\ z_O - z_S \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ z_S \end{bmatrix} \tag{5.2}$$

where index r denotes new, transformed coordinate system, O is start point and S is target point. The objective function is defined as:

$$J_{obj} = J_{hit} + J_{in} + J_{out} + J_{fall} + J_{dis} + J_{alti} + J_{ToD} + J_{lim} \tag{5.3}$$

where: J_{hit} is the total number of waypoint points within the terrain; J_{in} - the UAV threat exposure degree in a given point; J_{out} - number of points outside the specified flight area; J_{fall} - total effect of threats; J_{dis} - path length; J_{alti} - accumulated difference between UAV altitude and terrain in a given point; J_{ToD} - average distance between waypoints and destination; J_{lim} - component that refers to search space constraints. The difference between proposed algorithm and the standard PSO is the presence of an adaptive sensitivity decision area, which is formed to improve the performances of the algorithm. This area makes it possible to determine the potential locations of particles with high probability and to remove other candidate solutions. In order to avoid premature convergence, the search space is kept within certain boundaries, and a paricles' relative orientation from the current location enables search improvements. By applying the straight line rate index (SLR), which is defined as a ratio between candidate path length and length of the straight line connecting start and target position, and the paired T-Test, the authors showed that the proposed method was better compared to standard PSO and genetic algorithm.

Zhang et al. [148] proposed multi-objective robot path planning algorithm in uncertain environment based on the PSO optimization algorithm. An uncertain environment implies the existence of threats whose position cannot be determined with

precision. The mathematical formulation of the path is the same as in our research, path length and threat degree are performance criteria, and the path has to be without collisions. The path length is calculated in the standard way, as the sum of Euclidean distances between the points on the path, i.e. as the sum of the segments of the path. When the threat degree is considered, the authors first analyse the case where threats, i.e. danger sources have static positions, when the threat influence is represented as a linear fuzzy function; and then the case when the positions of the danger sources are uncertain is analysed. In this way, the path planning problem with uncertain threat sources is reduced to constrained two-objective optimization problem with uncertain coefficients. The authors proposed modifications to the original PSO algorithm in the part related to particle update, which is based on random sampling and uniform mutation. It is known that in multi-objective optimization problems with constraints, the solution evaluation is not done only by the fitness function value, but also by the degree of constraints violation. In this paper, for the evaluation of particles, the authors proposed improved version of constrained dominance relationship, which is based on imprecise dominance relationship. Constrained-violated degrees are introduced which are calculated on the basis of number of paticle's collisions with obstacles, i.e. with danger sources. Additional modification is introduction of infeasible solutions archive, that is used to store unfeasible non-dominated solutions which can serve as a bridge towards exploring isolated feasible solutions.

Phase angle-encoded quantum behaved PSO (θ-QPSO) is designed and implemented on solving UAV path planning problem in three-dimensional environment [149]. Instead of position and velocity vectors, θ-PSO uses vectors of phase angle and its increment, and in θ-QPSO only phase angle vectors are used, which reduces calculation costs and consumption of memory resources. The path cost function takes into account the path length, thereats, turning angle, climbing/diving angle and height i.e. altitude.

UAV reconnaissance path planning algorithm, based on PSO algorithm, is presented in [150]. The objective function is calculated as a ratio of reconnaissance costs and reconnaissance effects, whereby the reconnaissance cost is calculated as a sum of threat costs and fuel consumption. Reconnaissance effects are related to target value, hence the effective reconnaissance path represents an amount of information the UAV

collects about the target.

Authors in [151] used MAKLINK graph for mathematical modelling of the problem, then they applied Dijkstra algorithm to find the shortest distance in the formed graph, wherby the modified PSO algorithm is used for the optimization of the path. A path planner based on improved PSO, in which heuristic threat mechanism is integrated, is proposed in [152]. Heuristic information are used in the particle velocity update formula, thus guiding the movement of particles and improve the performances of the PSO algorithm. For reducing particle dimensions, a minimal risk surface is used, and the algorithm applies online approach to path planning, in order to deal with unpredictable threats. In order to reduce the computing complexity of the path planning process, authors suggest [153] an improved stochastic PSO algorithm that has a high capacity of exploration, and they apply it on mobile robot path planning in environment with static obstacles. Unlike the other methods, the proposed algorithm provides the generation of smooth paths.

Two novel robot path planning algorithms are proposed in [154]. The path objectives in the first algorithm are shortest path length, and is implemented as a hybrid of PSO algorithm and the probabilistic roadmap method (PRM), so the PSO is used as a global, and PRM as a local planner. In the second algorithm, the smoothness of the path appears as a objective, and the proposed solution is a combination of a new or negative PSO (NPSO) and PRM method. The NPSO works in such a way that direction of the search is determined on the basis of positions of the worst particles, and not on the basis of the best particles' positions. PSO and PRM are combined in the following way: the best positions are added as auxiliary nodes for the random nodes generated by PRM. The results showed that the NPSO solution is better compared to the first algorithm. Gong et al. developed an algorithm for robot path planning in environment that contains danger sources, based on multi-objective PSO optimization [155]. A free space is represented by a map consisting of a series of horizontal and vertical lines, whereby the objective function takes into account path length and danger degree. Multi-objective PSO algorithm used to generate optimal path contained self-adaptive mutation operator based on the degree of the path blocked by an obstacle. An additional modification is the existence of an archive that keeps unfeasible solutions, in addition to the already existing archive of feasible solutions, which is

aimed at improving the algorithm in the part related to exploration.

In [156], a global robot path planning algorithm based on a binary PSO algorithm is presented. The obstacles are represented by polygons, and the polygon nodes are numbered from 1 to n. The length of the PSO particle is n, and the value of each variable can be 0 or 1, depending on whether the node is on the path or not. The algorithm uses a mutation operator that prevents premature convergence. Other papers where PSO algorithm and its modifications are applied in the path planning are [157], [158], [159], [160].

5.3 Ant colony optimization algorithm

Tan et al. [161] proposed path planning method that uses advanced Dijkstra algorithm and ACO optimization. The first step is environment modelling using MAKLINK graph, then improved Dijkstra algorithm is applied for finding suboptimal collision free path, and finally ACO optimization is performed. At the beginning of algorithm execution path points are found as midpoints on MAKLINK lines, and the aim of the algorithm is to further optimize its position on MAKLINK lines. Possible location can be defined in the following way: let $P_0, P_1, P_2, \ldots, P_{d+1}$ be path point obtained by using Dijkstra algorithm, where P_0 is start, and P_{d+1} target position. Then the possible location on the MAKLINK line can be defined as:

$$L = \sum_{i=1}^{d} length(P_i(h_i), P_{i+1}(h_{i+1})) \tag{5.4}$$

The aim of the optimization is to find set of parameters h_i so the function L has minimal value.

Application of ACO algorithm in autonomous underwater vehicle path planning is given in [162]. Visibility graph is used for modelling, and it is based on grid model of the environment. The algorithm is improved with a few new rules for pheromone update, and two new parameters are introduced for path smoothness: cut off operator and insertion-point operator. Underwater vehicle path planning optimization by improved ACO algorithm is given in [163]. The space is modelled using cuboid with appropriate dimensions, which is further divided with horizontal and vertical planes into 3D space grid. The algorithm uses pheromone exclusion approach. In this algo-

rithm pheromones contain attractive and repulsive part. An ant is attracted by its own pheromones, and repelled by pheromones of other ants. ACO algorithm is also proposed for finding optimal path for underwater vehicles in [164] and [165].

Improved ACO algorithm for path planning of UAV that fly on low altitudes is proposed in [166]. The environment is represented by grid map, and criteria function consists of three components: the component which is related to path length, i.e. cost of deviation from straight line that connects start and final position; penalty, if the path is close to the threat zones; and the component related to minimizing the flight altitude. The algorithm has two modifications that refer to adaptive selection of the next node during ant movement, and adaptive pheromone update. Pheromone evaporation parameter has initial value of 0.1 and it is changed adaptively during algorithm execution. Pseudo code of the algorithm is given in Algorithm 5.1.

Algorithm 5.1: Improved ACO algorithm form UAV path planning

1 Form the original pheromone matrix T;
2 M ants are set in start position;
3 Each ant chooses the next node in grid map, based on diversion rule, finally reaching destination node, and thus forming a path;
4 Calculate cost function of all generated paths in (3) and save the optimal solution;
5 Update pheromone of each ant, based on cost function, in accordance with the update pheromone rule;
6 Evaluate optimal solution and decide if update of pheromone evaporation gene is needed;
7 Examine if the stopping condition for the algorithm is met, and repeat the steps (1) - (6) if not;

Coordinated trajectory replanning of multiple UAVs in dynamic and uncertain environment by using Max-Min adaptive ACO algorithm is proposed in [167]. In this algorithm, ants belonging to each subpopulation, when deciding about collisions, use data not only form its own subpopulation, but also from other subpopulations. UAV coordination include: simultaneous arrival at the destination, which requires determining estimated time of arrival of the group (ETA) and collision avoidance. The environment is divided and represented as a two-dimensional grid. Threat costs are calculated exactly like in our research in 6.1. The algorithm takes into account constraints such as: turning angle minimization, time coordination constraint (all

UAV should arrive at the destination simultaneously) and collision avoidance. Path replanning consists of three parts: coordination decider, trajectory planner and path smoother. Limiting the influence of pheromone trails is introduced to avoid search stagnation by setting explicit boundaries (min and max) for pheromone trail values, which are valid for all pheromones.

Chen et al. [168] proposed modified ACO algorithm for UCAV path planning in 3D environment. First, the solution for the case of static environment is given, and then path planner for UCAV in dynamic 3D environment, when recalculation of the path is needed due to unpredictable threats, is proposed. Criteria function takes into account fuel consumption costs, threat costs and altitude costs, and also several constraints are defined: 1) turning angle constraint, in horizontal and vertical plane; 2) maximal and minimal flight altitude (low altitude brings risk of collision with obstacles, and high altitude increases probability of detecting aircraft by enemy radar); and 3) the fastest flight range (the vehicle should not be without fuel). Several modifications of the original ACO are proposed. Weight function is introduced in the equation that defines the transition probability, in order to increase selection speed and selection probability when the intensity of the pheromone on the path is high. Moreover, when the number of ants on a route is higher than one third of total number of the ants, arithmetic operator is introduced that replaces pheromone evaporation parameter, in order to improve the probability of local best solutions. When path replanning is concerned, authors integrated two well-known approaches: the first one, which performs replanning of all segments of the path, from current node to target node; the second one, which only performs path replanning in the threat area. The shortcoming of the first approach is long calculation time, while shortcoming of the second approach is the fact that new path is only local best. The authors propose that after detection of a new threat selection of the appropriate window which includes detected threat area is performed. In addition, path planning must be done in the window area, before the aircraft arrives at the replanned start position.

In [169] method for mobile robot path planning that uses Simple Ant Colony Optimization Meta-Heuristic (SACO-MH) is proposed. SACO-MH is modified so the decision making depends on the distance d between source node and destination nodes, and ants have a memory m where visited nodes are saved, which helps in solving

stagnation problem. New algorithm, SACO-MHdm uses fuzzy criteria function for evaluation of the best paths, and fuzzy inference system is modified using Simple Tuning Algorithm. The proposed solution can be used as a path planner in static and dynamic environment, for both virtual environments and real time conditions. Mobile robot path planning using intensified ant colony optimization algorithm is described in [170].

5.4 Firefly algorithm

Adaptation of firefly algorithm and its application in solving UCAV path planning problem was proposed in [171]. Firstly, the mathematical problem was adopted to transform the path planning problem into D-dimensional optimization problem. Using simple mathematical procedure the coordinate system was transformed, so the horizontal axis of the new coordinate system was the straight line which connects starting and target point on the path. New horizontal axis X was divided into D equal segments, and optimization of vertical coordinates Y was performed for every node in order to obtain group of points which represent vertical components of D points. Finally, the path was obtained by connecting these before mentioned points. Safety and fuel consumption were considered as performance indicators, therefore, the path should be chosen so the degree of threat exposure during the flight is minimized, and the fuel consumption is the smallest possible. Total performance indicator is given by the following equation, where J_t is threat related indicator, J_f is fuel consumption indicator, while k represents coefficient for creating balance between demands to minimize threat and fuel consumption, and its value is in the interval [0,1], so the greater value means that the priority is given to the safety:

$$minJ = kJ_t + (1-k)J_f \tag{5.5}$$

The authors modified the original firefly algorithm to increase the convergence speed. The first modification was to add Levy flight with step size α. This was done to enforce exploitation in the phases when the individuals, i.e. fireflies are close to the solution. The second modification was to add information exchange between the best, i.e. the brightest (top) fireflies. There are two new parameters in the modified

Algorithm 5.2: Modified firefly algorithm for UCAV path planning problem

```
1  begin
2      Initialization. Set generation counter G = 1. Randomly initialize
         population P of n fireflies, where each firefly represents the potential
         solution of the problem; define light absorption coefficient γ, set
         parameter α which controls step size, and set initial attraction β_0 for
         γ = 0;
3      Generate rotated coordinate system. Transform the original
         coordinate system into the new rotated coordinate system; convert the
         information about threats on the battle field in the rotated coordinate
         system, and divide new axis X into D equal segments. Each potential
         solution P = {p_1, p_2, ..., p_D} is an array of real numbers representing
         coordinates of D;
4      Evaluate threat cost/light intensity J for each firefly in P using equation
         (5.5);
5      while (stopping criteria not met) or (G < MaxGeneration) do
6          Sort firefly population P from best to worst, according to the threat
             cost/light intensity J of each firefly;
7          Information exchange between top fireflies;
8          for i = 1 to n do
9              for j = 1 to n do
10                 if J_j < J_i then
11                     Move firefly i towards firefly j;
12                 end
13                 Vary the attraction with distance r using the formula:
                     exp[−γr^2];
14                 Evaluate new solutions and update threat cost/light intensity;
15             end
16         end
17         Evaluate threat cost/light intensity J for each firefly in P;
18         Sort firefly population P from best to worst, according to the threat
             cost/light intensity J of each firefly;
19         G = G + 1;
20     end
21     Inverse transformation of rotated coordinate system into the original one
         and generate the output;
22 end
```

algorithm: light absorption coefficient and ratio of top fireflies. It was shown that values 1.0 i 0.25 were the most suitable for these parameters. The modified algorithm

proposed by the authors to solve UCAV path planning problem is given in Algorithm 5.2. Firefly light intensity i is represented by objective function given in equation (5.5), so that intensity of the light emitted by the firefly is inversely proportional to the threat cost. The simulations proved that modified firefly algorithm is superior or competitive with the original firefly algorithm, and also it exhibits better performances compared to other analysed population based optimization methods, namely: ACO, biogeography-based optimization (BBO), DE, ES, GA, probabilistic-based incremental learning (PBIL), PSO i stud genetic algorithm (SGA).

Modified firefly algorithm is used for 3D path planning of autonomous underwater vehicles [172]. The algorithm was modified by designing random movement step which was set to be equal to the distance between two fireflies. Moreover, autonomous strategy was designed to avoid wrong flight of fireflies, where new position of the firefly is estimated before it moves. In other words, the firefly will change its position only if that is useful in terms of problem solving context. Two novel operators were also introduced: exclusion operator, which is used for better collision avoidance; and retraction operator, used to increase convergence speed and path smoothness. Criteria function has two objectives: minimization of path length and safety. The proposed algorithm was compared with PSO, GA and basic FA algorithm, and 6 standard test functions were used: sphere, rosenbrock, rastrigin, grienwank, ackley and zakharo.

5.5 Cuckoo search

New method for mobile robot path planning in unknown or partially known two-dimensional environment with static obstacles is described in [173]. The algorithm is based on Levy flight and cuckoo search metaheuristic algorithm. Robot should move towards destination while avoiding obstacles on the path. When the robot is near the obstacle (when its sensors detect the obstacle) cuckoo search algorithm is activated to find the best nest position of the robot (according to defined objective function) thus avoiding the obstacle. Mobile robot path planning depends on the two types of behaviour: avoiding obstacles and reaching target position. When avoiding obstacles, robot tries to make the distance between nest position and obstacle maximally safe. Behaviour which refers to reaching target position is realized in such a way that the best nest position is always on the minimal distance from the target position.

Objective function which satisfies both of these behaviours is given by the following equation:

$$F = C_1 \frac{1}{\min\limits_{OB_j \in OB_d} \|Dist_{N-OB_d}\|} + C_2 \|Dist_{N-G}\| \tag{5.6}$$

where OB_j are obstacles from the environment within sensor range of the robot, $\|Dist_{N-G}\|$ is the distance between the nest and target position, $\|Dist_{N-OB_d}\|$ is the distance between the nest and the obstacle, while C_1 and C_2 are control parameters. If a value of the parameter C_1 is big, the robot will be far from the obstacle, while if the distance is small, it will be close, thus increasing the probability of collision with obstacles. Parameter C_2 influences path length; greater value means shorter path length. The correct setting of these parameters influences convergence and performances of the algorithm, and values used in simulations are chosen by using trial and error approach, like in other nature-inspired metaheuristics. Pseudo code of the proposed algorithm is given in Algorithm 5.3.

Algorithm 5.3: Mobile robot path planning by cuckoo search

1. Initialize start and target position;
2. Moving towards the target until reaching the obstacle;
3. When the obstacle is on the path, apply cuckoo search algorithm;
4. Generate initial nest population, where each nest represents one solution of the proposed optimization problem;
5. Calculate current best nest and find global best nest;
6. Discard the worst nest and make new nest using Levy flight;
7. Continue moving of the robot towards the best nest position;
8. Repeat steps 2 - 7 until the robot avoids all obstacles or target destination is reached;

The efficiency of the algorithm is verified using simulations and real time experiment. The experiment is implemented using obstacles with different sizes and shapes, and Khepera-III mobile robot.

5.6 Artificial bee colony

Application of the modified artificial bee colony algorithm (ABC) in optimization of path planning in two-dimensional environment is given in [174]. The modification is implemented by applying balanced evolutionary strategy, which facilitates usage of

convergence status in each iteration of the algorithm, in order to manipulate the accuracy of exploration/exploitation, and to achieve the balance between local exploitation and global exploration. New algorithm, named BE-ABC, is different from the original ABC in using the parameter $trial(i)$ to set the accuracy of exploration/exploitation, and has a new strategy for generating bee scouts. In addition, convergence factor is added into the equations which define crossover and mutation operations, in order to set up the accuracy of exploitation, while the number of elements that participate in the afore mentioned operations is adaptive. Mathematical model of the problem is identical to the one used in our research in 6.1, and the organization of the simulations is very similar. Results of the simulations showed improvement compared to basic ABC algorithm, and two of its modifications, I-ABC i IF-ABC.

ABC algorithm where the improvement is implemented by the application of chaos theory is applied on UCAV path planning problem in [175]. The problem modelling is done exactly like in our research in 6.1, and the threats are represented with circles, so that the threat vulnerability is represented by the probability which is inversely proportional to the distance to the centre of threat area. Criteria function considers threat cost and fuel consumption costs, and it is defined like in our research in 6.1. The chaos is instability of deterministic systems in definite phase space, which is often present in non-linear systems, so that the small variance of some variable can induce great change of other system parameters. The applied modified ABS algorithm tends to exploit ergodicity and irregularity of chaotic variable with the aim of finding optimal parameters and pulling out from the local optimum trap. After search phase of each bee, the chaotic search is applied in the vicinity of the current best solution to chose better solution for future generation.

5.7 Other nature-inspired metaheuristics

For solving UCAV path planning problem in 3D dynamic environment, which includes moving threat zones, in [176] is proposed predator-prey pigeon-inspired optimization (PPPIO). Pigeon inspired optimization (PIO) belongs to swarm intelligence algorithms, and is based on movement of pigeons. Domestic pigeons find way to their home by using three different tools: magnetic field, sun and certain landmarks. During the algorithm execution it is possible to use two operator models: map and compass

model, which is based on magnetic field and sun; and landmark model, which based on area landmarks. Convergence problem, which is inherent to this algorithm, is here solved by introducing the concept of predator-prey. Predators are introduced into the population with the aim to remove the worst pigeons (individuals), and to force other individuals to move from these solutions. Cost function is designed to include path characteristics, constraints and demands to be satisfied. U these paper it is represented as a sum of the costs of path length, altitude, threat area, required power, collision and smoothness. Mathematical model of the path is constructed beginning from the start and target point, which are connected, and then the projection of the line ST on XY plane is made. Afterwards, the projection ST is divided into $D+1$ equal segments, and the vertical plane ST is defined on the each point of the segment. The point are taken in each of these plane and they are connected to form the flight path. In this way, the path planning problem is transformed into the problem of finding optimal values of the coordinates which will proved the best fitness function values, i.e. into the optimization problem. The path generated by this algorithm consists of line segments, which is not good solution for UCAV, therefore k-trajectory strategy is applied for path smoothness. Simulation results showed that PPPIO was more efficient compared to basic PIO, PSO and DE in solving UCAV path planning problem in 3D environment.

Gravitational search algorithm is used for mobile robot optimal path planning [177], and for UCAV path planning in [178]. For UCAV trajectory planning, Duan et al. [179] proposed intelligent water drops optimization algorithm, and in [180] improved artificial fish swarm algorithm (IAFSA) is introduced, and its application in solving mobile robot path planning problem is proposed. The experiments for performance evaluation of the proposed method were implemented using robot operating system on Pioneer 3-DX mobile robot.

5.8 Hybrid algorithms

Solution for the problem of UAV path planning on the sea, implemented by hybrid algorithm, which is based on differential evolution and particle swarm optimization with quantum behaviour (QPSO) is given in [181]. The proposed algorithm consists of two phases: in the first phase QPSO is applied for population update, and in the second

phase the modified DE algorithm is used. In canonical DE donor vector is generated based on three individuals from the current population, while the generation of donor vector in the modified DE is done by adding weighted difference of two or more randomly selected personal best positions to the global best position obtained by applying QPSO in the first phase. Also, like in the canonical DE, modified crossover operation is done after mutation operation. Before path construction, the authors propose simple method for terrain pretraitment, which is realized in two phases. Contours of the island are find in the first phase, which entails finding border points of the object. Adjustment of the points obtained in the first phase, in order to match an ellipse, is done in the second phase. Authors adopted the assumption that UAV flight altitude is constant, thus reducing the problem to 2D search. In this way, computing time and necessary memory is significantly decreased. The path is divided by n points, and is defined by an array of n elements, where each element represents the coordinates of given point. Thus, each particle which represents solution of the problem (potential path) has a dimension $2n$. Authors used B-Spline curve to construct a path. Since the presumption of constant altitude is adopted, only costs related to path length, threat exposure, and turning angle constraint are considered. Comparative analysis of performances between the proposed algorithm and several nature-inspired methods, such as genetic algorithm, differential evolution, standard PSO, hybrid PSO with differential evolution, and QPSO, is performed. The experimental results showed superiority of the proposed method in terms of robustness, quality of solutions, and convergence.

Hybrid algorithm based on non-linear time-varying PSO (NTVPSO) and modified, self-adaptive DE algorithm, whose mutation operator is based on ranking (ranking based self-adaptive DE, RBSADE) is designed for mobile robot path planning in 2D environment [182]. Algorithm works in such a way that after each iteration, the modified DE algorithm is applied to make evolution of particles' personal best positions more efficient, that is, to bring out particles from stagnation. Self-adaptive mechanism of the proposed method serves for setting up three control parameters: weight inertia, cognitive and social acceleration. Mathematical modelling of the working environment and the path is done exactly like we did in our research in 6.1. In this paper obstacles are represented as polygons, while optimization objectives are to minimize the costs related to path safety and its length. Path safety is defined as a

sum of minimal distances between each segment of the path and closest obstacle. The observed problem is constrained optimization problem, because the aim is to find the path without collisions with obstacles. After the constraints violation degree, which is equal to the ratio of number of constraints violations and total number of obstacles, is calculated, the rule for selection of the best path among multiple possible paths is applied. The rule is applied in the following way: (1) if the two paths the same constraints violation degree, the path with better fitness value is selected; (2) if the two paths have different constraints violation degree, the path with smaller value of this degree is selected. Simulation results showed that the proposed solution is more efficient compared to several popular nature-inspired methods: JADE, time-varying PSO, gravitational search, and modified DE algorithm.

Mo and Xu proposed hybrid algorithm for solving robot path planning problem in static environment. The method is based on combination of biogeography-based optimization (BBO) and standard PSO algorithm [183]. In BBO algorithm, islands are used to represent the solutions, where island features, in terms of their habitability, are represented by the set of variables which are called suitability index variables (SIV). Potential solutions are evaluated using habitat suitability index (HSI), where the greater value of this index denotes the island more suitable for habitability, i.e. represent better solution of the optimization problem. Better solutions suffer smaller modifications compared to worse solutions, but they also share their properties with worse solutions. BBO algorithm has migration operator which operates in the following way: when the island is selected for modification, its immigration degree is observed to determine in a probabilistic manner which variables from SIV should be modified; based on emigration degree of other islands, it is probabilistically determined which island should migrate randomly selected SIV value to the observed solution, i.e. to the island which is being modified. After the migration, mutation process is executed, which provides population diversity. The approximate Voronoi boundary network (AVBN) is used for path modelling, and the generalized Voronoi diagrams in raster based space are used. The environment where the robot moves is realized as a grid with dimensions 400x400, and each element of the grid is one small area from the real environment. Presence of an obstacle in a grid element is denoted with integer values, where value 0 denotes space without obstacles. The role of PSO

in this hybrid algorithm is to update particle positions, thus increasing population diversity in BBO algorithm. In basic BBO, if no island is selected for immigration, then also islands selected for emigration will not be modified, which has a negative influence on population diversity. In the modified algorithm, if this is the case, PSO and its particle update strategy is applied in order to modify island which are not selected for immigration.

Novel approach for determining optimal path in an environment which contains multiple robots, and which is based on combining improved particle swarm optimization (IPSO) and differentially perturbed velocity algorithm (DV), is described in [184]. The aim of the algorithm is to minimize path length, that is, to minimize the time to reach the destination, for all observed robots. The robots are not allowed to collide with each other (dynamic obstacles), nor to have collisions with the obstacles in the environment (static obstacles). In addition, optimal paths should be such that they minimize fuel consumption, which is considered as a number of turns a robot must do to reach a destination. IPSO algorithm is designed by modifying basic PSO with the application of adaptive weight adjustment and acceleration coefficients, which have positive effect on algorithm convergence speed. Moreover, IPSO-DV uses differential operator from DE algorithm in the equation for speed adjustment of IPSO algorithm. It is applied on the corresponding vector positions of neighbouring particles, which are not their best positions. Total objective function is given by the following equation:

$$F = \lambda_1 F_1 + \lambda_2 F_2 + \lambda_3 F_3 + \lambda_4 F_4 \tag{5.7}$$

where: F_1 - part of the objective function related to shortest path; F_2 - repulsive function, defined as a relative distance between a robot and an obstacle; F_3 - prediction of dynamic positions of the obstacles in the environment; F_4 - part of the objective function related to path smoothness, that is defined as an angle between hypothetical lines connecting target point and two neighbouring robot best positions in each iteration. Coefficients λ_i, which represent contribution of each part of the objective function, are adjusted during simulation. The effectiveness of the proposed solution is tested using simulations in the environment that contains 12 soft bots with circular shape, and seven obstacles with different shapes. Also, real experiment is performed with Khepera-II miniature robot, with 7cm diameter. The environment consists of

two robots and eight static obstacles. During simulation procedure, the proposed algorithm is compared with DE i IPSO, while in the experimental part of the testing, the algorithm is compared with PSO, gravitational search, IPSO and DE. The results showed that performances of the proposed solution were significantly better than competitors.

Wang et al. designed path planing algorithm for unmanned combat aerial vehicles using bat algorithm with mutation (BAM) [185]. Mutation operator from DE algorithm is applied to contribute to increasing the speed of original BA algorithm while preserving its robustness. Reasons for the application of hybrid operator are the improvement of bad solutions and improvement of exploration capabilities of the new algorithm. DE mutation operator is applied in such a way to improve the original bat algorithm by generating new solution for each bat, with the probability $1 - r$, using random walk. Mathematical model of the problem used in this paper is very famous, often used in the literature, and identical to the model that we used in our research in 6.1. The authors used two performance indicators: safety indicator and fuel consumption indicator. The calculation of these indicators is done exactly like in our reseach. Pseudocode of proposed algorithm is given in Algorithm 5.4. The algorithm showed superiority compared to several well known metaheuristics: ACO, BBO, DE, ES, GA, PBIL, PSO, and SGA, and the comparison was performed for different maximal number of generations and different problem dimensions.

Hybrid algorithm named genetic annealing, which represents the combination of genetic algorithm and simulated annealing is proposed in [186] for multiple robot path planning in dynamic environment. Hybrid approach solves the problem of premature convergence of the genetic algorithm, and converges faster than genetic algorithm and simulated annealing.

Three-dimensional UCAV path planner, implemented using hybrid metaheuristic algorithm which uses ACO and DE, where DE is used for pheromone trail optimization during pheromone update process of improved ACO algorithm, is described in [187]. The algorithm has two objectives: fuel consumption minimization and maximum safety. After optimal path generation, k-trajectory method is applied to achieve smoothness. Hybrid algorithm for mobile robot path planning in static environment, which consists of ACO and improved GA is described in [188]. The proposed al-

Algorithm 5.4: BAM algorithm for UCAV path plannning

begin
  Initialization;
  Generating rotated coordinate system. Transformation of the original coordinates; conversion of battle field information into the rotated coordinate system, and divide X axis into D equal parts. Each feasible solution is an array of D coordinate, which are real numbers.;
  Evaluate threat cost for each bat from the population P;
  while *(stopping criteria not met) or* $(t < MaxGeneration)$ **do**
    Sort population of bats P, from the best to the worst, by order of threat cost;
    for $i = 1$ **to** NP **do**
      Randomly select uniform $r_1 \neq r_2 \neq r_3 \neq i$;
      $r_4 = \lceil NP * rand \rceil$;
      $v_i^t = v_i^{t-1} + (v_i^t - x_*) * Q$;
      $x_i^t = x_i^{t-1} + v_i^t$;
      if $rand > r$ **then**
        $x_u^t = x_* + \alpha\epsilon^t$
      else
        $x_u^t = x_{r1}^t + F(x_{r2}^t - x_{r3}^t)$
      end
      Evaluate fitness function for the offspring x_u^t, x_i^t, x_{r4}^t;
      Select the offspring x_k^t with the best fitness among the offsprings x_u^t, x_i^t, x_{r4}^t;
      if $rand < A$ **then**
        $x_{r4}^t = x_k^t$
      end
    end
    Evaluate the threat cost for each bat in P;
    Sort the population of bats P from best to worst by order of threat cost;
    $t = t + 1$;
  end
  Inversely transform the coordinates in final optimal path into the original coordinate, and output;
end

gorithm first applies ACO algorithm to find suboptimal collision free paths, which serve as a initial population for GA algorithm. Unlike the original GA algorithm, the proposed improvement contains deletion operation which is based on domain heuristic

knowledge, so it is suitable for optimal path planning for mobile robots. The proposed algorithm is firstly applied on single objective optimization problem, where only path length is considered, and then also multi-objective path planning problem is defined and solved, with the following objectives: path length, safety, and smoothness. In [189] multi-objective hybrid algorithm based on gravitational search and PSO algorithm is proposed. The algorithm uses two criteria functions to generate optimal paths in static environment which contains obstacles and threat sources.

Ju et al. [190] proposed hybrid algorithm which combines evolutionary approach and PSO. Binary tree based coding is used for scalable representation of the path, and algorithm adds dummy nodes into the tree in order to solve problems of combining two algorithms. Algorithm works in such a way that first initial paths are generated with corresponding binary trees. Each path is then evaluated by its fitness value, and the paths are sorted from best to worst. The first half of elite paths is kept, while others are deleted. Afterwards, the elite paths are improved using PSO algorithm, and new paths are created with crossover operator, where the parents are from the elite path group. Combination of cuckoo search and differential evolution is proposed for UAV path planning in 3D environment [191]. The algorithm has two objectives: minimizing fuel consumption and avoiding threats, while the space is modelled with a grid. Cuckoo search is used for finding optimal path, while mutation and crossover operators from DE are used instead of the original method of selecting cuckoo from CS algorithm. In this way, integration of DE into CS brings diversity into population and improves search efficiency.

6 ROBOT PATH PLANNING BY BRAIN STORM ALGORITHM

6.1 UCAV path planning by brain storm algorithm

Brain storm optimization algorithm is applied in [192] to solve formation flight problem for multiple UAV vehicles. The proposed solution uses modified BSO and non-linear mode of UAV receding horizon control to determine RHC control parameters for UAV formation flight. RHC is a control method based on optimization which is used for formation flight and other cooperative tasks. The basic idea of RHC is online recede/move which applies iterative procedure to solve the optimal control problem and update the states, after the initial input of optimal command sequence. Using this approach the global control problem with constraints is efficiently divided into smaller local optimization problems, which decreases computing complexity of the entire procedure. In the modified BSO new method for clustering is used, that uses fitness value sorting and two possible solutions for the implementation of replacement of old solution with a better one: probabilistic approach and application of chaos theory. Besides that, cluster centres have more important role of guiding other individuals.

UAV vehicles equipped with optical sensors, which have the possibility of limited turn so they can cover part of the space, can be used for reconnaissance and monitoring, i.e. searching one or more targets in some 2D region. The search problem can be formulated as a multi-objective optimization problem with constraints, where position and UAV and its optical sensor should be determined so the costs were minimized. In [193] decoupling receding horizon search approach and BSO algorithm are proposed as a solution to this problem.

In this chapter we propose solution for UCAV path planning problem based on brain storm optimization algorithm. The quality of the proposed solution is tested by comparing obtained results with other well known metaheuristics.

UCAV are often used in hostile environments, so the control parameters for them are very demanding. Path planning problem of unmanned combat aerial vehicles is one of the most important parts of autonomous control model for UCAV. It refers to the problem of finding optimal path from starting point to final destination, considering several objectives and constraints. The aim of UCAV path planning is finding, i.e.

calculating suboptimal flight route so the vehicle can finish its flight for the appropriate time, avoid all possible threats, survives and successfully completes a mission.

6.1.1 Mathematical model

Unmanned combat aerial vehicles path planning represents an active research topic and various path modelings were proposed. UCAV path planning problem can be described as an optimization problem where the goal is to find the optimal route from the start point (S) to the target (T), according to some metrics. Optimal path can be defined in numerous ways since different desirable features can be considered such as the shortest, the smoothest path, minimal fuel consumption, the safest path, etc. In this paper two different criteria were used, fuel consumption and safety degree. Unmanned combat aerial vehicles are moving in three dimensional space, but in this paper altitude was not considered which means that two dimensional space is used for path planning problem. This simplification was also considered in other papers from the literature [171], [185]. For the solution modelling we adopted the method which is commonly used not only for the UCAV path planning [147], [174], but also for robot path planning [148]. The used model is rather simple but efficient. The main idea is to ensure UCAV's moving toward the target position. Path model used in this paper transforms path planning problem into D-dimensional optimization problem in the following way. The first step is to transform coordinate system so the horizontal axis (x-axis) is the line that connects the start and the target position. In this way, start point becomes $(0, 0)$ point which is accomplished by the following transformation:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} x_s \\ y_s \end{bmatrix} \tag{6.1}$$

where (x, y) represent the original coordinates, (x', y') are the new coordinates in the transformed coordinate system, (x_s, y_s) are coordinates of the start position S and ϕ is anti-clockwise angle from x-axis to the vector $\overrightarrow{ST}$:

$$\phi = \arcsin \frac{|y_T - y_S|}{\|\overrightarrow{ST}\|} \tag{6.2}$$

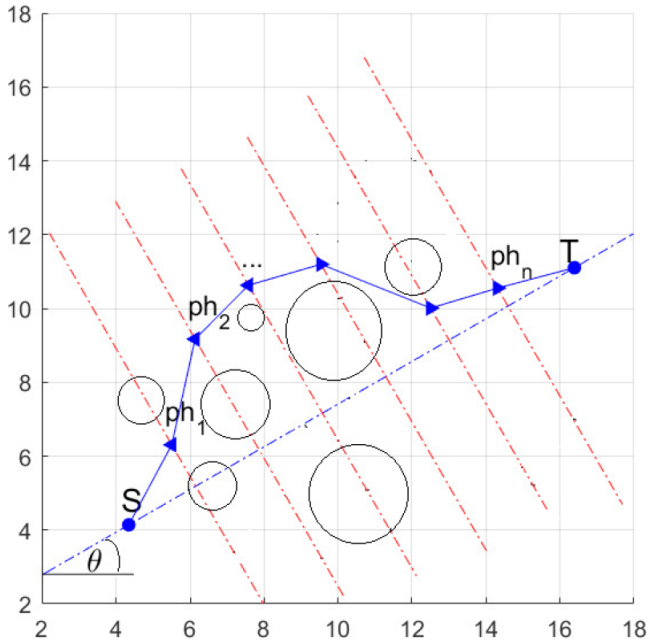


Figure 6.1 Path modelling

Next, the path is modeled by dividing the horizontal axis, the line S-T, into $n+1$ equal segments by n points. The x-coordinate of each path point is determined and fixed as it is shown in Fig. 6.1. One path point is located in each red line presented in Fig. 6.1. The y-coordinate for each point needs to be found for the optimal path. In Fig. 6.1 path points are marked by $ph_1, ph_2, \ldots, ph_n$ and n determines the optimization problem dimensionality, i.e. $D = n$. The complete path of the UCAV is represented as:

$$PH = (S, ph_1, ph_2, \ldots, ph_n, T) \tag{6.3}$$

6.1.2 Performance criteria

The goal of the path planning methods is to find the optimal route for an UCAV where the criteria for the optimality can be different. In this paper we will use two criteria: fuel consumption and safety degree.

Fuel consumption is proportional to the path length thus it can be represented as

a function of the length. Total fuel consumption is defined as:

$$F_{fuel} = \int_0^L w_f dl \tag{6.4}$$

where L is the path curve and w_f represents fuel cost for each path point. In this paper we used $w_f = 1$ thus the fuel cost is equal to the path length which is the sum of distances between two neighbour points in the path. If the start position S is denoted as ph_0 and the target position T as ph_{n+1}, path length can be calculated as:

$$L(PH) = \sum_{i=0}^{n} d(ph_i, ph_{i+1}) \tag{6.5}$$

where $d(a, b)$ is the Euclidean distance between a and b.

The second criterion considered in this paper is safety. UCAV during their flight can be faced with certain threats such as radars, radiation, missiles, antiaircraft artillery and similar. These threats can be represented as circle areas with different radii and threat degree or weight [171]. If the path goes through that danger areas, UCAV is exposed to the threat of same degree. The longer the path passes through the danger area, the probability of UCAV being damaged or destroyed is larger. On the other hand, probability to be damaged if the path is outside of the threat circle is equal to 0. Similarly to the fuel consumption, threat cost can be calculated as:

$$F_{threat} = \int_0^L w_t dl \tag{6.6}$$

where again L is the path curve and w_t represents the threat weight for each path point. In order to determine the total threat cost it is necessary to include all threats. Each threat has a circle range where inside that circle the threat degree is constant. The total threat cost produced by N_t threats while UCAV flies along the path segment L_{ij} is defined by the following equation [171]:

$$w_{t,L_{ij}} = \int_0^L \sum_{k=1}^{N_t} \frac{t_k}{((x - x_k)^2 + (y - y_k)^2)^2} dl \tag{6.7}$$

where t_k represents the danger degree of the threat k with the center at (x_k, y_k). In order to simplify calculation of the threat or safety degree, Eq. 6.7 can be converted into discreet model. Each path segment can be divided into $m+1$ equal subsegment by m points and the threat degree of the path segment L_{ij} is represented by the average of the danger degrees in these m points. In this paper we used the same model presented in [171], [174] i [185] where each path segment was divided by 5 points thus the threat produced by N_t threats while UCAV flies along the path segment L_{ij} was calculated by the following equation:

$$w_{t,L_{ij}} = \frac{L_{ij}^5}{5} \sum_{k=1}^{N_t} t_k \left(\frac{1}{d_{1,k}^4} + \frac{1}{d_{2,k}^4} + \frac{1}{d_{3,k}^4} + \frac{1}{d_{4,k}^4} + \frac{1}{d_{5,k}^4}\right) \tag{6.8}$$

where L_{ij} represents the length of the path segment between ph_i and ph_j , $d_{m,k}$ is the Euclidean distance between the m-th subsegment point of the segment L_{ij} and threat k, while t_k represents the threat level of the k-th threat.

In this paper we propose the BSO for unmanned combat aerial vehicle path planning problem. During the flight UCAV should avoid threat areas while minimizing the fuel consumption. By minimizing the fuel consumption, threat degree can be increased and vice verse. In order to avoid threat areas path length can be increased which results in larger fuel consumption. Since these two objectives cannot be minimized at the same time, we used parameter λ that determines the influence of each of them. Final objective function that need to be minimized by brain storm optimization algorithm is:

$$F(PH) = \lambda F_{fuel} + (1 - \lambda) F_{threat} \tag{6.9}$$

6.1.3 Simulation results

Our proposed method for the UCAV path planning was implemented in Matlab R2016a and all simulations were performed on the platform with Intel R CoreTM i7-3770K CPU at 4GHz, 8GB RAM, Windows 10 Professional OS. In order to test the quality of our proposed method we compared it with the methods proposed in [171] and [185]. In [171] modified firefly algorithm (MFA) was used for solving unmanned combat aerial vehicle path planning problem. Modifications includes adding Levy flight with dynamic parameter for generating new positions of the fireflies and adding

information exchange between the fireflies with the best fitness function values. These modifications increased local search and convergence speed. In [185] bat algorithm with mutation (BAM) was proposed for the same problem as the one considered in this paper. Differential evolution mutation operator was added into the original bat algorithm with the aim to speed up convergence of the algorithm. Besides mutation operator, another change was to fix dynamic parameters of the bat algorithms.

In this paper we organized simulations the same way as in [171] and [185] where one flight environment was considered. Start position was set to (10, 10) while the target point has coordinate (55, 100). Fitness function parameter λ was set to 0.5. Five threat zones are known in the field. All information, i.e. coordinates, threat radii and threat grades are listed in Table 6.1. In [171] and [185] the proposed methods

Table 6.1 Information about threats in the field

No.	Location(km)	Threat radius(km)	Threat grade
1	(45,50)	10	2
2	(12,40)	10	10
3	(32,68)	8	1
4	(36,26)	12	2
5	(55,80)	9	3

were compared with numerous nature-inspired algorithms such as genetic algorithm (GA), stud genetic algorithm (SGA), differential evolution (DE), evolutionary strategy (ES), particle swarm optimization (PSO), ant colony optimization (ACO), original bat algorithm (BA) and the original firefly algorithm (FA). In this paper we included those results. Simulation results include performance analysis in two different cases: when maximal number of generation was changed and when different dimensionality of the problem was considered. In this paper, the reported results were normalized the same way as it was done in [171] and [185]. Obtained fitness function values were reduced by 50 which means that if the reported result is 2.9036, actual fitness function value was 52.9036.

Parameters for the brain storm optimization algorithm were set by conducting preliminary computational experiments. Probability for generating a new random solution p_{5a} was set to 0.2. Parameter for selecting one cluster p_{6b} was 0.8. In the

case that one cluster is chosen, probability of using its center p_{6bii} was 0.4, which is the same as the probability of combining cluster centers p_{6c}. Probability p_{6bi} refers to the probability of the cluster to be chosen and it is proportional to the number of individuals in it.

The first set of simulations includes changing the maximal number of fitness function evaluations. For these experiments, dimension was set to 20 and maximal number of fitness function evaluation was 1500, 3000, 4500, 6000 and 7500, same as in [171] and [185]. Population size for the BSO was 20. Comparison of the results for the best, the worst and average fitness function values obtained by methods from [171], [185] and our proposed method are presented in Table 6.2, Table 6.3 and Table 6.4, respectively.

Table 6.2 The best normalized fitness function values obtained in 100 runs for different maximal number of generations

No.eval.	GA	SGA	DE	ES	PSO	ACO	FA	MFA	BA	BAM	BSO
1500	1.2604	1.7370	2.4179	9.6276	2.7827	10.7202	1.4713	0.7030	4.0662	0.6208	**0.6064**
3000	1.5073	1.3218	0.8503	10.6318	2.3469	10.8912	0.6577	0.5382	4.7582	**0.4900**	0.5314
4500	1.0991	1.1559	0.5319	11.1469	2.3738	9.9096	0.5459	0.4857	4.1112	0.4724	**0.4112**
6000	1.0792	0.7595	0.5047	11.2403	3.4276	12.3080	0.4931	0.4661	3.1463	0.4590	**0.4541**
7500	1.0640	1.0166	0.4792	12.3745	2.5221	7.1358	0.4753	0.4508	4.4072	0.4636	**0.3744**

Our proposed algorithm found the best solution in all cases, except for the 3000 objective function evaluations. Bat algorithm with mutation found better path for the UCAV for that case and also better solutions than all other compared algorithms. Good result were also achieved by MFA, FA and DE, but with higher number of objective function evaluations. Other algorithms were inferior. Our proposed algorithm has shown constant improvements with maximal number of generations growth. It is very important that when other algorithms started stagnation, BSO made significant improvement when increasing the number of objective function evaluations to 7500.

Based on the worst fitness function values obtained in 100 runs presented in Table 6.3 it can be concluded that our proposed method found relatively good solutions in all cases. The worst solution was found by 1500 fitness function evaluations and it was 8.7150 but when the number of fitness function evaluation is increased to 3000, the worst solution was significantly improved and it was 3.9941. All other algorithms, except MFA for 1500 evaluations and BAM for 7500 evaluations, had larger difference

Table 6.3 The worst normalized fitness function values obtained in 100 runs for different maximal number of generations

No.eval.	GA	SGA	DE	ES	PSO	ACO	FA	MFA	BA	BAM	BSO
1500	10.2501	13.0102	25.3999	41.9676	28.6115	18.7099	28.0425	**4.6726**	39.0832	11.7494	8.7150
3000	8.2047	11.0529	18.6288	38.5875	25.7065	18.4316	29.3022	4.5749	29.9962	9.7666	**3.9941**
4500	10.5257	13.3517	13.8150	46.0828	29.6341	17.4223	27.8480	4.9631	31.1293	7.6952	**2.9817**
6000	6.7466	7.5385	10.4226	31.3944	33.0709	17.2147	26.5768	9.1502	24.9732	6.7334	**2.1708**
7500	8.9162	13.5830	8.9560	34.8908	27.3858	16.9896	26.3005	3.6783	24.7175	**3.3564**	3.4103

Table 6.4 Mean normalized fitness function values obtained in 100 runs for different maximal number of generations

No.eval.	GA	SGA	DE	ES	PSO	ACO	FA	MFA	BA	BAM	BSO
1500	4.2541	4.5491	12.3797	20.5653	10.076	16.3819	6.2034	1.9576	16.6782	**1.4842**	2.7838
3000	3.5523	3.4353	6.0887	20.6706	9.1725	16.2884	4.3526	1.3048	14.9048	**0.9337**	1.2222
4500	3.4269	3.1636	3.7267	20.1996	9.5459	16.1408	4.1809	0.9933	14.4874	0.9123	**0.9062**
6000	3.0080	2.6434	2.6358	20.8610	8.9917	16.3976	2.2791	0.8984	12.4323	0.8000	**0.7533**
7500	2.9160	3.1409	1.9715	20.7600	7.8005	16.1958	2.2064	0.7025	11.4213	0.7422	**0.7014**

between the best and the worst solution which proves the robustness of our proposed BSO method.

Average of fitness function values obtained in 100 runs are presented in Table 6.4. Our proposed BSO algorithm had the best mean values for 4500, 6000 and 7500 fitness function evaluations, while for the 1500 and 3000 BAM obtained better results. BSO needed more generation to find the optimal solution, i.e. it had slower convergence but excellent ability of finding better solution.

In [171] and [185] standard deviation was not reported which would reveal the true robustness of the algorithms. Without that data we can just make rough conclusion based on the best, the worst and mean results. Since our proposed algorithm had the smaller difference between the best and the worst solution, it can be concluded that compared to the other mentioned nature-inspired algorithms, our proposed BSO algorithm was the most stable one.

The second set of experiments included testing the algorithm with different dimensions. Maximal number of fitness function evaluations was set to 6000 to make it comparable with other mentioned algorithms. Algorithms were tested for dimensions 5, 10, 15, 20, 25, 30, 35 and 40. Obtained results are presented in Table 6.5, Table 6.6 and Table 6.7. The best results obtained in 100 runs with different dimensions are

listed in Table 6.5.

Table 6.5 The best results obtained in 100 runs with different problem dimensions

D	GA	SGA	DE	ES	PSO	ACO	FA	MFA	BA	BAM	BSO
5	5.2471	9.9596	4.3568	12.3746	5.6082	10.1164	4.3585	4.3573	10.6909	4.3575	**0.3843**
10	1.5716	1.5498	1.3952	8.0656	2.1101	7.4746	1.3990	1.3966	2.3600	1.3953	**0.3690**
15	0.8299	0.9700	0.6204	7.7408	3.2257	9.8297	0.6172	0.6115	3.0757	0.6094	**0.3774**
20	0.8600	0.8426	0.4913	9.6276	2.3738	10.0836	0.4626	0.4552	2.3950	0.4679	**0.4541**
25	1.5243	1.3743	0.6265	12.3169	2.3740	11.5490	0.4908	0.4571	5.0173	0.4484	**0.3929**
30	1.7026	1.5147	1.1301	18.0090	3.6751	13.8615	0.6828	0.5160	7.2470	0.4671	**0.4411**
35	2.1602	1.5319	1.2849	16.8613	5.4765	16.9476	1.0829	**0.4709**	7.4484	0.4795	0.5979
40	2.4178	1.9406	3.9617	19.8244	5.5384	17.6142	1.5225	**0.4506**	8.6500	0.6028	0.6003

Our proposed method has found best solutions for dimensions less than 35, i.e. dimensions 5, 10, 15, 20, 25 and 30. When problem dimensions was set to 35 and 40 MFA achieved best solutions. However, our proposed BSO algorithm outperformed even the MFS when the maximal number of fitness function evaluations was increased to 10,000. In that case, the proposed BSO algorithm found the best solutions 0.4418 and 0.4502, for dimensions 35 and 40, respectively (the mean solutions were 1.1495 for dimension 35 and 1.4210 for dimension 40). This is the consequence of the nature of the BSO algorithm which has the ability to work very well with multi-objective and large optimization problems, but it needs certain number of iterations which does not necessarily means to increase in computational time.

In most cases, our proposed algorithm had the smallest worst solution in 100 runs (see Table 6.6). All algorithms except MFA, BAM and our BSO have very large worst solutions. For the dimensions 5, 10 and 15, the proposed BSO algorithm had significantly smaller worst solutions compared to BAM, while in other cases the

Table 6.6 The worst results obtained in 100 runs with different problem dimensions

D	GA	SGA	DE	ES	PSO	ACO	FA	MFA	BA	BAM	BSO
5	20.1888	22.6326	9.7959	62.1765	13.3267	12.6928	15.7395	12.4186	295.2557	10.2403	**0.3873**
10	6.3799	5.7899	12.4821	74.6665	23.2604	18.2565	6.7095	3.7858	58.7386	10.7242	**0.4372**
15	8.1499	9.9385	12.5250	50.3214	28.0228	10.9917	44.2763	3.8319	35.7454	10.1928	**0.6771**
20	9.4820	11.6024	18.8897	38.7234	34.7133	17.0266	28.9142	2.0279	33.7068	3.7420	**2.0108**
25	12.7971	16.0736	17.1415	33.4598	31.6741	12.2373	16.4518	3.7043	24.9265	**3.5192**	4.0914
30	22.1291	14.0512	29.6529	37.4566	35.6656	14.4647	15.9757	8.3364	30.0844	10.2851	**8.0677**
35	24.4790	15.6693	39.4435	46.6475	38.0578	18.7271	33.8871	**5.8830**	32.7374	8.8193	8.2009
40	19.2098	22.5022	45.4130	44.3624	35.5090	27.0641	36.6626	**7.7236**	33.2634	8.4273	8.2159

Table 6.7 The mean results obtained in 100 runs with different problem dimensions

D	GA	SGA	DE	ES	PSO	ACO	FA	MFA	BA	BAM	BSO
5	10.5709	10.8836	8.0557	31.8202	10.0765	11.4856	8.7499	9.1673	56.4830	9.0542	**0.3856**
10	2.3722	2.2813	3.1206	27.2252	7.2212	12.5333	2.1801	1.5740	19.4251	2.7075	**0.4304**
15	2.1136	1.8973	2.3737	22.0792	7.7362	10.2484	2.8217	0.8967	13.6018	1.2318	**0.4305**
20	2.9612	2.8621	3.0044	20.4717	9.9091	16.3303	3.7327	0.7004	13.6305	0.7609	**0.7003**
25	3.7244	3.7238	4.6029	22.7244	10.3315	11.5842	3.9039	0.9987	14.9017	0.7093	**0.6851**
30	5.3097	4.3798	11.4103	25.4016	12.7964	13.9422	4.9621	1.3568	16.6162	1.1067	**1.0530**
35	6.0765	5.4943	19.1074	27.2172	13.8799	18.3452	5.9955	1.6009	17.7033	1.4617	**1.4535**
40	7.6989	7.4237	28.7062	30.0177	15.1555	24.7642	7.8558	2.1978	19.9737	**1.8769**	2.0039

difference was not so substantial. Compared to the MFA, the proposed BSO had significantly smaller worst solution just for the dimension 5, while for the dimension 35 and 40, the MFA obtained better worst solutions. In other cases, worst solutions were similar.

On average, in 100 runs our proposed method was better for all the problem dimensions except in the case of the problem dimension 40 where the bat algorithm with modification had better performance. When for that problem dimension the number of fitness function evaluations was increased to 7000, the proposed BSO algorithm found better solution compared to BAM, 1.6892 versus 1.8769, which again showed its ability to improve when other algorithms stagnate. That result proves the quality of the proposed brain storm optimization algorithm for the UCAV path planning problem even for the larger dimensions. In these experiments, we fixed the number of objective function evaluations in order to fairly compare the results but it is well known that for larger problem dimensions more iterations are needed.

Additionally, we qualitatively compared our proposed BSO method with the method proposed in [194] where the artificial neural network (ANN) trained by imperialist competitive algorithm (ICA) was used for the UCAV path planning. That method was tested for two different environments and compared with the artificial bee colony algorithm (ABC). Simulation results showed that the proposed ICA-ANN algorithm was superior to the ABC algorithm. Optimal path was defined by the same fitness function as in this paper. Two test environments were used in [194] with 7 and 8 danger areas, but without exact coordinates reported so we approximated them from the picture. In both cases our proposed BSO algorithm found path as straight as

possible while the ICAANN algorithm had some unnecessary turns and arcs during the avoidance of the danger zones.

6.2 Robot path planning in uncertain environment using brain storm algorithm

6.2.1 Mathematical model

In this research movement in the two dimensional space is considered. Path of the robot is searched from the start position to the target where static obstacles and uncertain danger sources need to be avoided. More precisely, robot path planning problem can be defined as finding an optimal collision-free route from the start position (S) to the target (T) according to some metric. Optimal path can be the shortest or the smoothest one, path that spends the least time or energy of robot, or some other criteria can be used. In this paper length of the path and safety degree are considered in search for optimal robot route.

For modeling the solution we adopted method used in chapter 6.1 because it is rather simple and it has small sensitivity to the shape of obstacles. The described model is presented in Fig. 6.2.

6.2.2 Performance criteria

The goal of the path planning methods is to find the optimal route for a robot where optimal can mean different things. In this paper we will use two criteria: path length and safety.

Path length can be calculated as the sum of distances between two neighbor points on the path. Calculation of path lenght can be done exactly like in chapter 6.1.

$$L(PH) = \sum_{i=0}^{n} d(ph_i, ph_{i+1}) \tag{6.10}$$

where $d(a, b)$ is the Euclidean distance between a and b.

Path length is rather clear and simple. There are no unknown or uncertain factors, however safety degree depends on several uncertain parameters. In this research we developed probabilistic model for calculating safety degree in uncertain environment

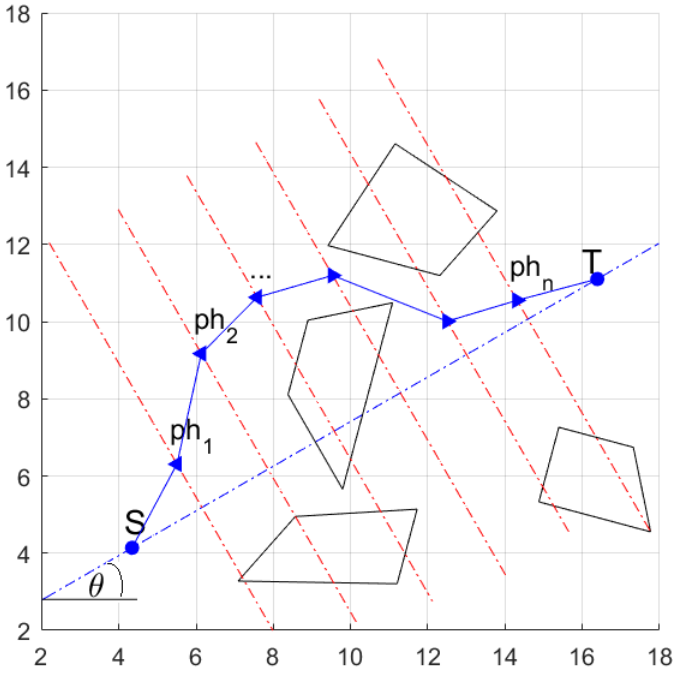


Figure 6.2 Illustration for the path model

that can be more precise than the fuzzy model in [148]. Statical obstacles are fixed and the path only needs to go around them because robot cannot go throw them. On the other hand, there are danger zones that are not physical obstacles but they can damage or brake the robot. For example danger zones can be zones with high radiation, heat, cold, exposed to enemy fire or similar. Danger zones have their sources and closer to the source the danger is higher and it drops down with the distance. Different sources can have different functions of their danger strength. In this paper we used function for danger degree that was proposed in [155] and used in [148]. Depending on the shortest distance $d(DS_i)$ between the danger source i and the path, danger degree dan_i is defined as follows:

$$dan_i = \begin{cases} 1, if \quad d(DS_i) \leq d^i_{min} \\ \frac{d^i_{max} - d(DS_i)}{d^i_{max} - d^i_{min}}, if \quad d^i_{min} < d(DS_i) < d^i_{max} \\ 0, if \quad d(DS_i) \geq d^i_{max} \end{cases} \tag{6.11}$$

In the previous function d^i_{min} and d^i_{max} are defined for each danger source and they rep-

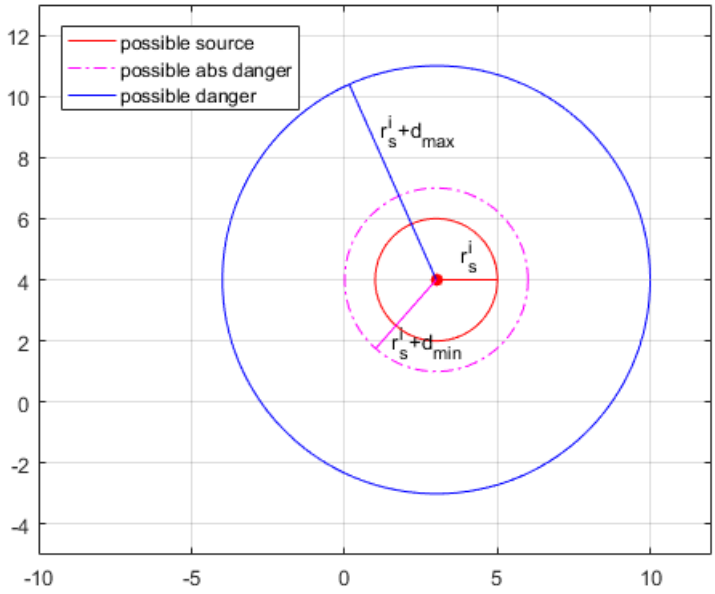


Figure 6.3 Danger area around uncertain danger source

resent borders of absolute danger and danger free zones, respectively. If the shortest distance between the path and danger source is larger than d^i_{max} then robot is completely safe, while if that distance is less than d^i_{min} the robot is in absolute danger. In the zone between danger degree is calculated based on the distance and difference between d^i_{min} and d^i_{max}.

Exact position of the danger source is not always known but zone where the source is can be usually determined. In this paper the model for path planning where position of the danger source is inside circle area is discussed. In [148] fuzzy model for calculating danger degree in uncertain environment was proposed. In the path planning problems like this the probability that a point on the path is in danger can be directly calculated and it is more accurate than the fuzzy model. If it is known that the danger source is somewhere inside the circle of the radius r^i_s and the danger degree is calculated by the Eq. 6.11 it can be seen that the path can be in danger if it is inside the big circle with the radius $r^i_s + d^i_{max}$ and center is the same as the center of the possible source position circle (Fig. 6.3).

Danger zone represents the geometrical place of all the circles with the radius d^i_{max} and centers in the circle with the radius r^i_s and center in (x_c, y_c). In Fig. 6.3 red circle is the area where the danger source can be. Outside blue circle in Fig. 6.3 a path is

completely safe regardless of the position of that danger source. Danger degree for a point inside the blue circle depends on the distance and position of the danger source. It is defined as integral of danger degree function over all possible source positions (for all possible x and all possible y coordinates within the red circle). Thus, for calculating danger degree solving Stieltjes integral is needed since the position possibilities of the danger source inside given area have uniform distribution. In the case of different danger source position distribution inside the red circle, Lebesgue integral with that distribution function as a measure instead of Stieltjes integral would be used. Danger degree at point (x_0, y_0) is obtained by solving the following integral:

$$\begin{aligned} &dan^i(x_0, y_0) \\ &= I_{\sqrt{(x-x_0)^2+(y-y_0)^2} \leq d^i_{min}} \int_{a_x}^{b_x} \int_{a_y}^{b_y} dydx \\ &\quad + I_{d^i_{min} \leq \sqrt{(x-x_0)^2+(y-y_0)^2} \leq d^i_{max}} \int_{a_x}^{b_x} \int_{a_y}^{b_y} \frac{d^i_{max} - \sqrt{(x-x_0)^2+(y-y_0)^2}}{d^i_{max} - d^i_{min}} dydx \end{aligned} \tag{6.12}$$

Two integrals are needed since the model for danger degree calculation has three cases but the last one is equal to zero so it will not have contribute to the sum. By introducing indicators I only appropriate integral will not be set to zero, so one integral will always be zero. Integrals limits were written as a_x, b_x, a_y and b_y where a_x and b_x is used to limit values of x coordinate while a_y and b_y depends on x and they limit y coordinate. Limits for x are set to:

$$\begin{aligned} a_x &= x_c - r^i_s \\ b_x &= x_c + r^i_s \end{aligned} \tag{6.13}$$

where x_c is x coordinate of the center of the area where source can be. Since area for integration is a circle whose center is not in $(0, 0)$ but in (x_c, y_c) and it is defined as:

$$(x - x_c)^2 + (y - y_c)^2 = (r^i_s)^2 \tag{6.14}$$

Expressing y from the previous equation determines limits a_y and b_y:

$$\begin{aligned} a_y &= \sqrt{(r^i_s)^2 - (x - x_c)^2} + y_c \\ b_y &= -\sqrt{(r^i_s)^2 - (x - x_c)^2} - y_c \end{aligned} \tag{6.15}$$

By using Eq. 6.12 danger degree is determined by our probabilistic model. Since probabilistic model was used for calculation of danger degree caused by one danger source, combining them is easy based on the probability theory. If one point is affected by two or more overlapping danger zones, danger degree is calculated using that the probability that the point is not affected by any danger zone is equal to the product of probabilities that it is not affected by each of the zones individually, if they are independent:

$$D(x_0, y_0) = 1 - (1 - dan^i(x_0, y_0))(1 - dan^j(x_0, y_0)) \tag{6.16}$$

The case with any number of danger zones with intersection can be analogously calculated.

Depending on the particular danger source the risk degree has to be calculated differently. In some environments the risk increases with the time spent in the danger zone, while in others the length of the path that goes through the risk area should be considered. For example, in the case of radiation the time spent in radiated areas is determining factor, together with the strength of the radiation. In our model the strength of the radiation is represented by the danger probability and the total risk can be calculated by integration along the path with regard to the speed. If the robot goes through dust or some other pollutant, then only the length of the path, but not the speed, may be significant.

In this research we used the point closest to the danger source as a danger estimate to make it comparable with the model in [148]. This model is suitable for the situations similar to "single missile" threat, that is the length of the path through the danger zone does not influence the level of the threat, missile is fired only once and with given probability (depending on the closeness) it will hit or miss. Unlike the model in [148], our proposed model can be easily adjusted to other described situations.

6.2.3 The proposed algorithm

In this research we propose a method for robot path planning in uncertain environment. Besides static obstacles, danger sources with uncertain position are included into path planning problem. Different objectives and metrics for optimal path that include path length as one of the most important, but also safety, smoothness of the

path, time of arrival, etc. were considered. In this research we designed a method for path planning that minimizes path length and maximizes safety. This two objectives are in most cases contradictory, i.e. if the path is minimal it goes right through danger zone so in order to increase safety (or reduce danger degree) path length will have be increased too. For multiobjective optimization problems different approaches have been proposed in the past. In this research our objective function includes parameter w that controls the influence of each criteria. Since they are opposing, controlling parameter gives more importance to one or the another objective.

Besides these two objectives, the solution must be feasible. In order to implement this constraint we introduced penalty into the objective function that has larger value compared to the values of path length and danger degree. In order to reduce the number of iterations needed to find a feasible solution and to prevent trapping inside local optima that is obtained by infeasible solution, we implemented more exploration if the infeasible solution is the global best in several generations. Usually, methods for robot path planning introduce some mechanisms to generate feasible solutions based on infeasible solutions. In [148] limited loop for updating solutions was proposed. New solution from the infeasible solution is generated by using the method based on random sampling and the uniform mutation. This mechanism can reduce the number of iterations, but it increases the computational time which is an important recourse in optimization algorithms. On the other hand, our proposed brain storm optimization approach just increases explorations which does not affect iteration computational time or algorithm's complexity.

We empirically determined the following parameters. If the solution is infeasible in more then 5 generations probability p_{5a} that controls changing cluster center by a new random solution is increased from 0.2 to 0.5. Moreover, if a feasible solution is not found in more then 10 generations, p_{5a} is set to 0.9 which means that in 90% of cases the best solution in one cluster will be replaced by a random solution. This emphasized exploration significantly and reduced the number of generations needed to find a feasible solution. Without this part in some cases the BSO was not able to find feasible solution in 500 iterations while with this mechanism no more then 100 iteration were needed to find a feasible solution. In most cases feasible solutions are found in the first 50 iterations.

Objective function contains normalized path length defined by Eq. 6.10, danger degree from Eq. 6.12 and penalty for infeasible solutions. Danger degree is in range [0,1] thus path length need to be normalized. Since the largest path cannot be calculated we used the following approximation. It was considered that the largest path is two times larger of shortest path. With that assumption path length was normalized as follows:

$$L(PH)_{normalized} = \frac{L(PH) - d(S,T)}{d(S,T)} \tag{6.17}$$

where $d(S,T)$ represents Euclidean distance between start point S and target point T. By combining previously mentioned criteria the following object function was used in brain storm optimization algorithm:

$$F(PH) = wL(PH) + (1-w)D(PH) + penalty \tag{6.18}$$

Robot path is defined by $n+2$ points including start and target positions. Brain storm algorithm dimension is n since only the offset from x-axis is needed to determine path point ph_i. The first coordinates of ph_i are calculated once at the beginning and the neighbor points are equidistant.

6.2.4 Simulation results

The proposed method for robot path planning in uncertain environment was implemented in Matlab version R2016b. All simulations were performed on Intel CoreTM i7-3770K CPU at 4GHz, 8GB RAM computer with Windows 10 Professional OS.

In the literature numerous test examples for path planning can be found. In this paper we used examples proposed in [148] where four different environments were created for testing robot path planning algorithm. Test examples contain different number of static obstacles, as well as danger sources. For all danger sources d^i_{min} and d^i_{max} were set to 1 and 10, respectively. The size of the search area was set to be 50×50. For all test examples our proposed method was run 30 times. Detailed information about each test problem is given bellow. Brain storm algorithm parameters were set empirically. Number of agents was set to 100 and number of the clusters was 5. Probability p_{5a} initially was set to 0.2 but it can be changed as it was described in

Section 6.2.3. Probabilities p_{6b}, p_{6bi} and p_{6c} were set to 0.8, 0.4 and 0.5, respectively. Penalty for infeasible solution was set to 20. Maximal number of iterations was 500.

Our proposed algorithm is compared to [148] where Pareto front was used for solving multi-objective optimization problem. Fuzzy model was used in [148] for danger degree and it is not represented as a single value but as an interval. In this research we used our proposed probabilistic model, thus danger degree is uniquely determined for each path which more precisely describes exactly the same situation. In [148] besides path length and danger degree, interval improved hypervolume metric designed to deal with imprecise optimization problems was used. Since we do not have imprecise optimization due to the proposed probabilistic model for danger degree, this metric was not included.

Test example 1

The first test example contains three statical obstacles and one uncertain danger source. Start position is (7.2399, 38.9460) while the target coordinates are (41.3780, 9.4897). Position of the danger source is in the circle with radius 2 and center (21.4110, 26.0000). Obstacles' coordinates are set as follows:

$$ob1 = \begin{bmatrix} 23.987 & 25.250 \\ 18.271 & 18.964 \\ 18.834 & 15.869 \\ 26.161 & 23.092 \end{bmatrix} \quad ob2 = \begin{bmatrix} 25.275 & 26.750 \\ 31.233 & 40.54 \\ 35.662 & 38.664 \\ 28.254 & 26.938 \end{bmatrix} \quad ob3 = \begin{bmatrix} 16.338 & 16.994 \\ 9.3333 & 10.897 \\ 9.6554 & 8.833 \\ 15.936 & 14.555 \end{bmatrix}$$

Number of the path segments was set in all test problems to be the same as in [148]. In the first test example it was set to be 6, which means that the problem dimension is 5. In Fig. 6.4 the results for different values of parameter w are shown.

As it can be seen, if the objective function's parameter w is set to 1, which means that only path length is considered, our proposed method finds a straight line (blue path in Fig. 6.4) from start to the target point, which is the shortest path. On the other hand, if w is set to 0, i.e. only danger degree needs to be minimized, it can be seen that the danger zone is completely avoided (red path in Fig. 6.4), but the length of the path is unnecessary long due the fact that it was not even considered. If w is set to 0.1 which means that only when danger zone is almost completely avoided,

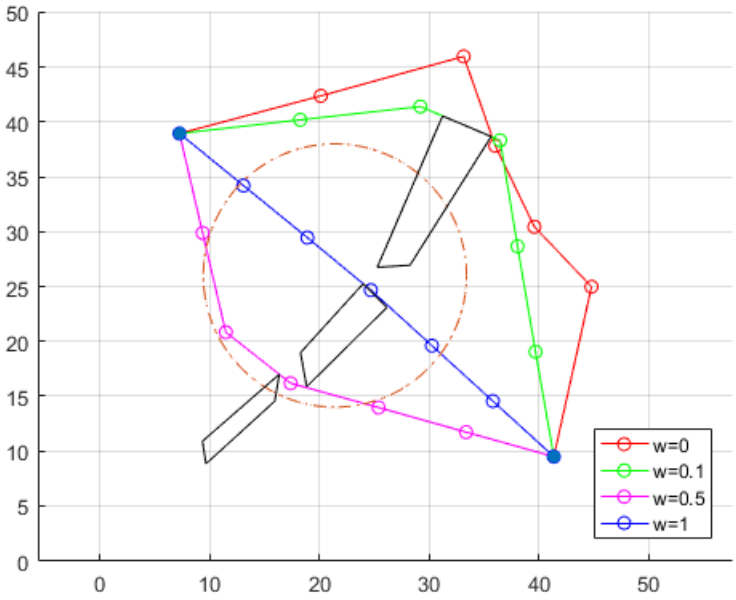


Figure 6.4 Optimal paths for the first test problem obtained by BSO with different values for parameter w

minimization of the path length is attempted, the proposed algorithm finds green path in Fig. 6.4. This path avoided danger zone and reduced path length compared to the previous case. When parameter w is increased to 0.5 the path length and danger degree are equally considered and path that was obtained by our proposed method is shown in magenta in Fig. 6.4. This path is shorter compared to the previous one, but still the danger zone is avoided. This test problem has good properties to illustrate the quality of the proposed method. Depending on the needs, length or safety can be considered more or less important and the proposed method will give the best solution in given situation.

Results reported in [148] for the first test problem have some inconsistency. It is reported in the text that the line between start and target position is divided into equal segments, hence the solution in the figure that has points at the corners of the obstacle is not possible. This solution is reported as the solution with minimal danger degree and it is equivalent to our solution when $w = 0.1$.

In Table 6.8 path lengths and danger degrees for 10 different values of parameter w, forming Pareto front, are reported. Following the usual practice for swarm intelligence

algorithms, mean, standard deviation and best solutions are listed.

Table 6.8 Path lengths and danger degrees for the first test example when different values for parameter w are used

	L(PH)			D(PH)		
w	mean	std	best	mean	std	best
0.0	79.868	13.566	68.174	0.000	0.000	0.000
0.1	59.241	0.000	59.241	0.000	0.000	0.000
0.2	58.465	2.453	51.483	0.003	0.009	0.000
0.3	52.901	3.342	51.303	0.023	0.012	0.000
0.4	51.177	0.004	51.170	0.030	0.000	0.030
0.5	51.044	0.002	51.042	0.033	0.000	0.032
0.6	50.902	0.001	50.900	0.036	0.000	0.036
0.7	50.748	0.000	50.748	0.043	0.000	0.043
0.8	50.358	0.000	50.358	0.067	0.000	0.067
0.9	45.482	0.000	45.481	0.743	0.000	0.743
1.0	45.108	0.000	45.108	0.926	0.000	0.926

Our shortest path ($w = 1$) is the same as in [148], straight line from the start to the target point. On the other hand, when w is zero, then the path length is not considered at all. In that case it easy to make the risk equal to zero wandering arbitrarily around search area, only avoiding danger zone, hence large standard deviation is expected. When w increases to 0.1, the path stabilizes keeping the risk at zero. From the results presented in Table 6.8 it can be concluded that our proposed method builds good Pareto front and the algorithm is robust with small standard deviation.

In [148] path lengths of 58.5849 and 45.1079 with danger degrees of 0.0 and 0.7356 to 1.0, respectively, were reported. Our best path with the risk equal to zero is the same one that goes above all the obstacles (green path in Fig. 6.4). For the second case, our solution is also the straight line (blue path in Fig 6.4), only the risk is calculated as 0.926 according to our probabilistic model.

Test example 2

The second test example contains three convex statical obstacles and two non-convex where one is U-shaped and the other is V-shaped obstacle. Test also has one uncertain danger source. Start and target positions are (6.8374, 30.409) and (46.451,

28.814), respectively. Position of the danger source is in the circle with radius 2 and the center is in (20.928, 21.779). Obstacles' coordinates are set as follows:

$$ob1 = \begin{bmatrix} 13.601 & 21.652 & 21.652 & 13.520 & 13.681 & 20.122 & 20.122 & 13.52 \\ 40.071 & 40.071 & 25.156 & 25.250 & 27.407 & 27.313 & 38.195 & 38.00 \end{bmatrix}$$

$$ob2 = \begin{bmatrix} 47.981 & 38.721 & 38.319 & 40.010 & 40.251 & 48.061 \\ 37.257 & 37.163 & 24.968 & 25.062 & 35.287 & 35.287 \end{bmatrix}$$

$$ob3 = \begin{bmatrix} 26.564 & 48.139 \\ 27.047 & 35.099 \\ 28.899 & 35.099 \\ 29.140 & 48.233 \end{bmatrix} \quad ob4 = \begin{bmatrix} 26.886 & 32.660 \\ 26.322 & 21.779 \\ 28.738 & 21.591 \\ 29.301 & 32.473 \end{bmatrix} \quad ob5 = \begin{bmatrix} 26.161 & 17.839 \\ 25.517 & 6.9568 \\ 28.818 & 6.7692 \\ 28.738 & 17.557 \end{bmatrix}$$

In the second example the path was divided into 10 segments, thus problem dimension was 9. Similarly to the first example, in Fig. 6.5 paths obtained by our proposed method are presented when different values for parameter w were used. The blue path represents the shortest path found by our proposed method when danger degree was neglected. In this example straight line is not a feasible solution but it can be seen that the obtained path follows the edges of obstacles and between them the path is a straight line. The green and the red paths are for $w = 0.1$. The green path follows the edge of the danger zone avoiding it as well as the edge of the obstacle from below. On the other hand other solution for $w = 0.1$ was found and it avoid danger zone from above. Optimal path obtained for $w = 0.8$ goes through danger zone with a little turn in order to avoid the most dangerous area.

In Table 6.9 path lengths and danger degrees for the second test example when $w \in \{0, 0.1, 0.2, \ldots, 1\}$ are presented.

Since our proposed method finds minimal path when $w = 1$ with standard deviation 0, we can conclude that our proposed algorithm is capable of finding the shortest path very robustly. In this example standard deviation when w is 0.9, 0.8 and 0.7 is large for both, risk degree and path length. This is due to the nature of this test example where the shortest path goes through the area where the center of danger source is located and this cause high value for risk degree of 0.921. On the other hand, the only

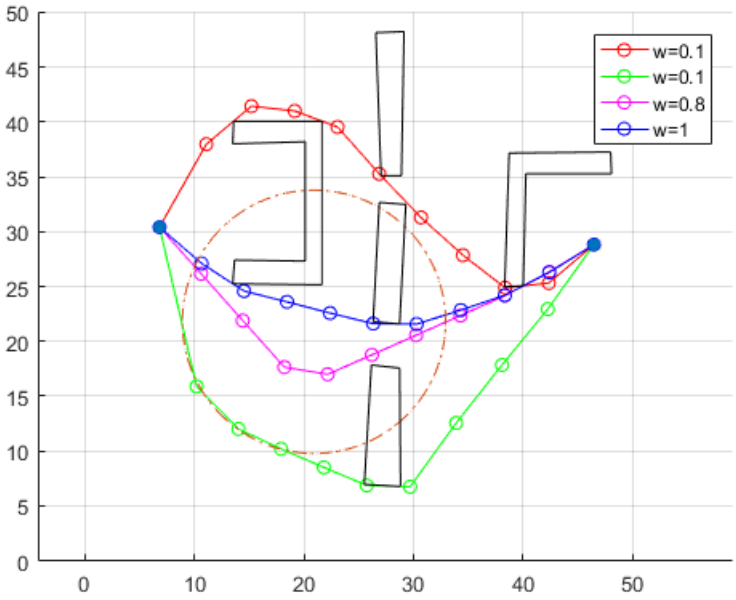


Figure 6.5 Optimal paths for the second test problem obtained by BSO with different values for parameter w

Table 6.9 Path lengths and danger degrees for the second test example when different values for parameter w are used

	L(PH)			D(PH)		
w	mean	std	best	mean	std	best
0.0	85.1425	15.942	72.226	0.000	0.000	0.000
0.1	63.617	3.787	52.838	0.000	0.000	0.000
0.2	64.914	3.333	52.804	0.000	0.000	0.000
0.3	64.634	3.321	52.509	0.001	0.000	0.000
0.4	63.404	3.767	52.732	0.001	0.001	0.000
0.5	63.245	3.710	52.745	0.004	0.002	0.000
0.6	64.195	0.496	64.000	0.010	0.004	0.000
0.7	58.245	7.012	50.073	0.191	0.211	0.026
0.8	54.142	7.839	47.941	0.386	0.282	0.056
0.9	52.162	10.908	43.666	0.529	0.426	0.000
1.0	43.378	0.001	43.378	0.921	0.003	0.917

feasible paths that are not so close to the danger center are the ones that completely avoid risk area but are much longer. Hence, two classes of non-dominated solutions

have very similar objective function values and in 30 runs mix of these appears. In [148] path length of 51.4852 was found as the safest one with the danger degree equal to 0.0. Our proposed algorithm found the path length of 52.509 and danger degree 0 (similar to red path in Fig. 6.5). Again, in this example the solution reported in [148] goes exactly along the obstacles' edges which is not possible if the path is divided into equal segments as required. The shortest path in [148] was with length of 43.3782 and danger degree from 0.7329 to 1.0. Our proposed algorithm found the same optimal path when risk degree is ignored (the blue path in Fig. 6.5). In this case path risk obtained by our proposed method was 0.921.

Test example 3

The third test example contains three statical obstacles and three uncertain danger sources. Coordinates of the start position are (5.066, 40.071) while the target position is (45.163, 15.212). Radius of the possible positions is 2 for all three danger sources. Center of the first danger source is in (20.042, 13.148), of the second is (20.928, 21.779) and finally the third danger source is in the circle with center in (34.213, 18.871). Obstacles' positions are set as follows:

$$ob1 = \begin{bmatrix} 23.021 & 37.069 \\ 11.910 & 32.754 \\ 17.546 & 19.809 \end{bmatrix} \quad ob2 = \begin{bmatrix} 26.886 & 37.069 \\ 38.077 & 43.730 \\ 38.882 & 29.565 \\ 29.704 & 28.908 \end{bmatrix} \quad ob3 = \begin{bmatrix} 26.000 & 6.394 \\ 20.767 & 15.118 \\ 25.034 & 22.529 \\ 33.649 & 22.529 \end{bmatrix}$$

In Fig. 6.6 paths obtained by our proposed method with different w values are presented. Number of segments in this example was again 10 with the dimension of the problem 9. This example has only three obstacles and the shortest path is found without any problems. When $w = 0.9$ path is similar to the shortest one but slightly increasing the path by going farther from the danger source. Target point is inside dangerous zone, thus small part of the path has to be inside. Paths when $w = 0.1$ and $w = 0.5$ are very similar. The only difference is in the last four segments. When safety is considered more important, the last segment of the path tries to enter danger zone later.

Table 6.10 contains obtained results of our proposed method for the third test

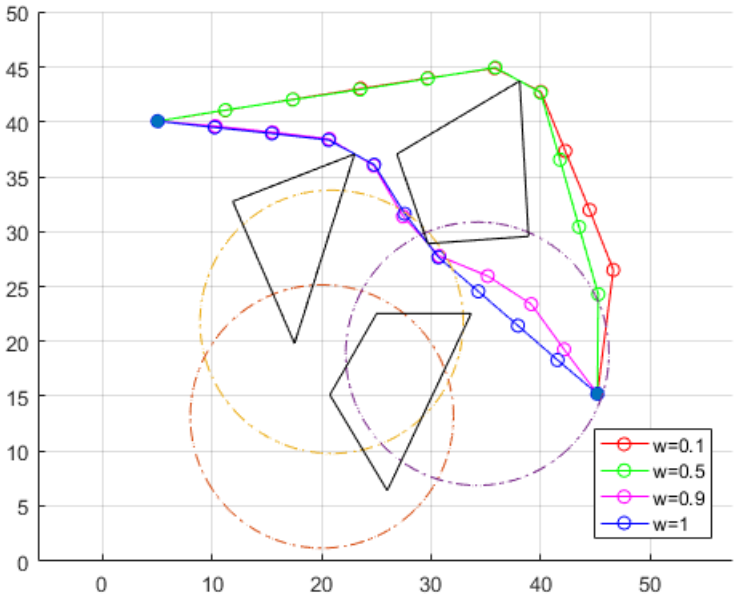


Figure 6.6 Optimal paths for the third test problem obtained by BSO with different values for parameter w

problem.

Table 6.10 Path lengths and danger degrees for the third test example when different values for parameter w are used

	L(PH)			D(PH)		
w	mean	std	best	mean	std	best
0.0	86.0588	19.091	72.516	0.001	0.000	0.001
0.1	66.116	4.298	64.756	0.003	0.000	0.003
0.2	64.461	0.005	64.458	0.004	0.000	0.004
0.3	64.385	0.443	64.289	0.006	0.000	0.005
0.4	64.190	0.002	64.188	0.008	0.000	0.006
0.5	64.102	0.009	64.091	0.008	0.000	0.008
0.6	65.681	3.451	64.035	0.027	0.038	0.009
0.7	63.019	6.035	52.675	0.064	0.071	0.011
0.8	61.942	5.052	52.188	0.080	0.128	0.014
0.9	56.983	6.942	50.356	0.296	0.300	0.020
1.0	55.240	7.036	49.830	0.486	0.330	0.037

In [148] the shortest path was 49.8304 and the same path was found by our proposed

method. Danger degree reported in [148] was in the range from 0.8359 to 1, while our proposed method reported risk degree of 0.486 (the blue path in Fig. 6.6). This huge difference is due to the fact that in [148] second danger source is listed as we reported here at (20.928, 21.779). However, from the Figure 11 in [148] it seems that coordinates around (25, 35) were used for the second danger source. In that case our algorithm would determine risk as 0.983. On the other hand, when risk degree need to be minimized path of 64.9518 with risk between 0 and 0.0406 was found in [148] and another path with the same danger degree was found with length of 65.985. Our proposed algorithm found path with risk degree 0.008 and length of 64.091 (the green path in Fig. 6.6) which is better both, in length and safety, compared to the one reported in [148] and moreover path with risk degree 0.003 and length of 64.756 was also found (the red path in Fig. 6.6).

Test example 4

The last test example has thirteen statical obstacles and two uncertain danger sources. In [148] coordinates for this test problem were not given, thus we approximated coordinates by measuring them from the image. We set the start position to be (7, 21) while the target coordinates are (44, 29). Positions of the danger source are in the circle with radius 2 while the centers are in (23, 25) and (35,33). Obstacles' coordinates are set as follows:

$$ob1 = \begin{bmatrix} 7.50 & 13.00 \\ 12.50 & 12.00 \\ 13.75 & 15.10 \\ 10.00 & 16.80 \end{bmatrix} \quad ob2 = \begin{bmatrix} 10.00 & 22.50 \\ 10.00 & 27.00 \\ 14.70 & 26.50 \\ 14.70 & 22.50 \end{bmatrix} \quad ob3 = \begin{bmatrix} 10.90 & 32.00 \\ 11.50 & 35.00 \\ 17.00 & 37.00 \\ 17.00 & 31.00 \end{bmatrix}$$

$$ob4 = \begin{bmatrix} 18.00 & 26.00 \\ 18.00 & 30.00 \\ 22.00 & 30.00 \\ 22.00 & 25.90 \end{bmatrix} \quad ob5 = \begin{bmatrix} 22.00 & 37.00 \\ 25.30 & 37.50 \\ 27.30 & 29.70 \\ 24.70 & 30.00 \end{bmatrix} \quad ob6 = \begin{bmatrix} 23.50 & 20.00 \\ 23.50 & 25.00 \\ 25.80 & 25.00 \\ 25.80 & 20.00 \end{bmatrix}$$

$$ob7 = \begin{bmatrix} 22.00 & 11.00 \\ 26.00 & 11.00 \\ 26.70 & 16.00 \end{bmatrix} \quad ob8 = \begin{bmatrix} 32.50 & 10.00 \\ 41.50 & 10.00 \\ 41.50 & 15.00 \end{bmatrix} \quad ob9 = \begin{bmatrix} 30.50 & 15.50 \\ 30.50 & 20.00 \\ 34.30 & 19.00 \\ 34.90 & 15.50 \end{bmatrix}$$

$$ob10 = \begin{bmatrix} 20.00 & 15.00 \\ 20.50 & 18.50 \\ 17.50 & 21.30 \\ 15.70 & 16.30 \end{bmatrix} \quad ob11 = \begin{bmatrix} 34.00 & 23.50 \\ 37.50 & 21.50 \\ 40.00 & 25.00 \\ 35.20 & 26.70 \end{bmatrix}$$

$$ob12 = \begin{bmatrix} 29.80 & 26.90 \\ 29.80 & 32.00 \\ 34.00 & 31.00 \\ 32.00 & 27.00 \end{bmatrix} \quad ob13 = \begin{bmatrix} 36.30 & 30.20 \\ 36.30 & 36.00 \\ 40.50 & 35.00 \\ 40.50 & 30.20 \end{bmatrix}$$

Path was segmented into 13 parts that leads to the 12-dimensional problem. This last example is rather complex. It contains 13 static obstacles concentrated in the center of the field. One danger source is also in the center and the other one cause the danger zone that contains target. The shortest path that goes through danger zones was found and in Fig. 6.7 is drawn by blue line. By adding a little influence of the path length in order to prevent wide walk through search space, the path marked by green line was obtained. If the influence of both criteria is the same, i.e. $w = 0.5$, obtained path goes above the obstacles in order to reduce the path length, but it also makes a larger turn in order to reduce danger degree. Path for $w = 0.3$ is marked red. This example is a good illustration of the influence of the parameter w. By increasing the parameter shortest path were made by reducing the arc but entering the danger zone more.

In Table 6.11 path lengths and danger degrees for the paths obtained by our proposed method for the fourth test example are presented.

For this test case the results are not exactly comparable to the results from [148] since the exact positions of statical obstacles and danger sources' areas are not known. Shortest path reported in [148] was 38.2251 and it goes through both danger sources

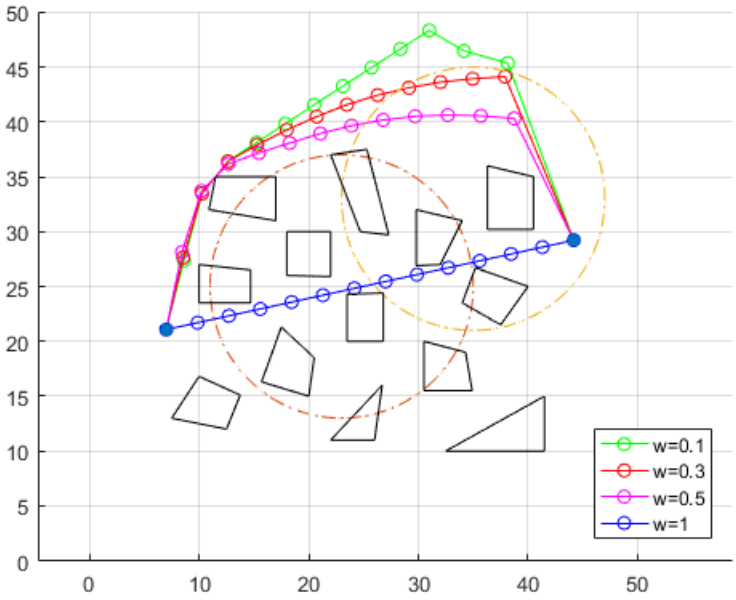


Figure 6.7 Optimal paths for the fourth test problem obtained by BSO with different values for parameter w

Table 6.11 Path lengths and danger degrees for the fourth test example when different values for parameter w are used

	L(PH)			D(PH)		
w	mean	std	best	mean	std	best
0.0	91.809	22.931	76.052	0.050	0.008	0.048
0.1	69.197	5.970	65.146	0.052	0.009	0.048
0.2	64.225	5.100	61.232	0.052	0.009	0.048
0.3	61.242	3.900	57.964	0.281	0.002	0.278
0.4	57.417	8.352	49.019	0.296	0.281	0.019
0.5	56.825	6.607	43.897	0.310	0.101	0.164
0.6	52.077	6.892	48.923	0.337	0.216	0.121
0.7	51.126	3.772	48.032	0.383	0.127	0.297
0.8	49.132	3.851	46.221	0.523	0.132	0.449
0.9	42.059	3.593	39.112	0.821	0.216	0.716
1.0	38.584	0.000	38.584	0.998	0.000	0.998

thus the risk degree reported in [148] was between 0.9087 and 1. Our proposed algorithm found similar shortest path and the danger degree was 0.998 (the blue path in

Fig. 6.7). Based on the results obtained for this complex test example, again it can be concluded that our proposed method builts good Pareto front. The path length decreases when w increases and the danger degree decreases along with w.

Low or equal to zero standard deviation obtained in most cases by our proposed algorithm confirms its quality and robustness. In some cases larger standard deviation is due the fact that objective function can be improved by either shortening the path or avoiding the danger zones. For some configurations of obstacles and danger sources, it is possible to have two non-dominated solutions of almost the same quality (objective function value) but very different, one short but dangerous, the other safe but long. In repeated runs, good solution considering objective function will always be found, but these solutions will be a mix of two mentioned solutions, thus increasing the variance.

6.3 Mobile robot path planning by improved brain storm optimization algorithm

6.3.1 Mathematical model

Due to the variety of the objectives, environment properties and conditions, numerous research papers propose rather different path models, optimization methods and include various constraints [183], [195], [196].

In this research, two dimensional mobile path planning in grid-based environment with static obstacles was considered. Environment model used in this research is rather often used [109], [197].

The area of interest, i.e. search space, is rectangular space with x_{max} width and y_{max} height. It is divided into equal sized square cells, grid cells. If the initial area of the interest does not have rectangular shape, it can be expended to the proper shape and the added cells are considered as obstacles. Obstacles O_j where $j = 1, 2, \ldots, M$ are placed inside the search space. Each obstacle is defined by one or more grid cells. Environment is implemented as $x_{max} \times y_{max}$ matrix where zero elements represent free space and elements whose value is equal to 1 define obstacles. For example, the following matrix represents the environment presented in Fig. 6.8.

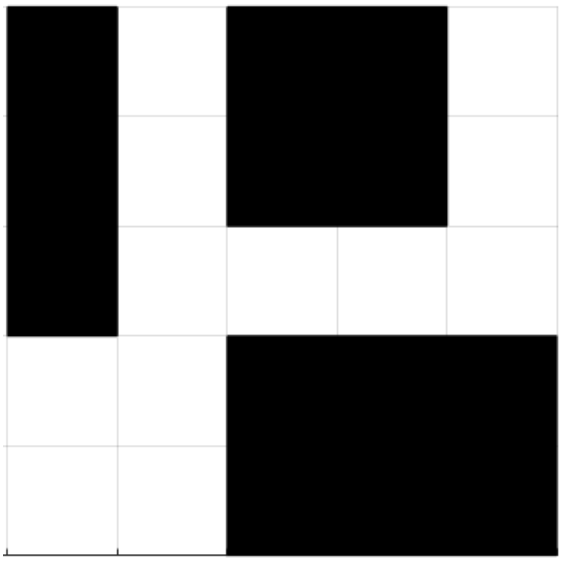


Figure 6.8 Environment defined by Eq. 6.19

$$E = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} \tag{6.19}$$

The path PH is defined by starting point S, target point T and n_{path} points between them marked as $ph_1, ph_2, \ldots, ph_{n_{path}}$:

$$PH = (S, ph_1, ph_2, \ldots, ph_{n_{path}}, T) \tag{6.20}$$

Each point is defined by its grid coordinates (x, y), i.e. $x_{min} \leq x \in N \leq x_{max}$ and $y_{min} \leq y \in N \leq y_{max}$. Grid cell center was considered to be a grid point.

Path length is defined by the sum of distances between two neighbour points on the path:

$$L(PH) = \sum_{i=0}^{n} d(ph_i, ph_{i+1}) \tag{6.21}$$

where $d(a, b)$ is the Euclidean distance between a and b:

$$d(a,b) = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2} \tag{6.22}$$

6.3.2 Our proposed algorithm

Our proposed method combines local search procedure for finding the shortest path in a graph with the brain storm optimization algorithm.

The first step in finding the optimal path is to find the feasible one since the unfeasible solutions are not usable in real situations. In order to improve chances of finding the optimal path by BSO algorithm we used the idea of generating feasible paths similar as it was proposed in [196].

In [196], path was built incrementally where firstly N random points (all in free space, outside the obstacles) are generated. The path is then built starting from the start point S and the next path point is chosen from the N generated random points. Closest point that built feasible solution is used. After that the same procedure was done for the chosen points until the target is reached. This is basically Dijkstra's algorithm for finding the shortest path in a graph where the nodes are starting, target and randomly generated points and the edges are their connection. In [196], edges that have intersection with the obstacles were not considered.

In this research, we use local search method for generating the initial population for the brain storm algorithm. The proposed algorithm starts by generating N random grid points in free space (outside the obstacles). These points are considered to be graph nodes. Euclidean distance was calculated between each pair of nodes and distance matrix is created. In order to ensure that only feasible solution can be found as the shortest path, we set the distance for nodes whose connection would lead to unfeasible solution, to be infinite. Local search procedure similar to the one proposed in [196] is used to find the shortest path in that graph where the nodes are generated points along with the starting and target point. Each point is directly connected to all others. Since unfeasible edges have infinite length, they will not be a part of the shortest path.

Since the points are generated randomly, in the case of a complex environment with large number of obstacles, it can happen that feasible path obtained by connecting

the generated points does not exist. This probability decreases with the number of generated points, but the complexity increases. In such cases, if the shortest path length found by the local search method is infinite, we generate new set of random point and repeat procedure until the feasible solution is found.

Described method is used for setting the initial population for the BSO algorithm hence it is repeated n times, where n is the size of the population. Dimension of the problem is equal to $2 * n_{path}$ where n_{path} is the number of path points between the start and target and it is multiplied by 2 because points are defined in two dimensional space. Solutions obtained by local search algorithm do not necessary have the same size. In order to make that all solutions have the same dimension, first the solution with the highest dimension is found. Solutions that have less points are expanded by dividing the longest path segment into as many equal parts as necessary.

When the initial solutions are found by the proposed local search algorithm and after their dimensions are adjusted, BSO algorithm is used to optimize them. Instead of using random solutions for the first generation of the BSO algorithm, obtained feasible solutions are used. Since the proposed environment is grid based and the brain storm optimization algorithm works with the real numbers, we rounded each obtained coordinate to the nearest integer.

Solutions found by the BSO are evaluated by the following fitness function. The objective of the proposed method is to find the shortest feasible path. We ensured that initial population in BSO algorithm contains only feasible solutions, but during the search process, some of the solutions can be changed in a way that path goes through the obstacle. In some papers, solution correction methods are proposed in order to obtain the most similar feasible solution to the generated one. In this research, unfeasible solutions are allowed but it was reflected in the value of fitness function. If the generated solution is unfeasible, penalty will be added to the fitness function value. Path length $L(PH)$ is determined by the Eq. (6.21) and the fitness function is calculated as:

$$fitness(PH) = L(PH) + penalty \tag{6.23}$$

where

$$penalty = \begin{cases} \frac{L(PH)}{2}, & \text{if the path is unfeasible} \\ 0, & \text{otherwise} \end{cases} \tag{6.24}$$

By introducing the penalty into the fitness function, BSO will eventually discard unfeasible solutions since they will have larger fitness function value compared to the similar feasible solutions. The proposed method for mobile robot path planning in static environment is summarized in Algorithm 6.1.

6.3.3 Simulation results

The proposed brain storm optimization algorithm combined with local search algorithm for mobile robot path planning was implemented using Matlab R2016a. Experiments were performed on the platform with Intel Core i7-3770K CPU at 4GHz, 8GB RAM, Windows 10 Professional OS.

Parameters for the proposed methods were set empirically by conducting several pre-tests. For finding the feasible paths by the local search procedure, 50 random points were generated ($N = 30$). Population size for the BSO algorithm n was set to 20 and the number of the clusters m was 4. Probability of changing the solution randomly p_{5a} was set to 0.2. Probabilities p_{6b}, p_{6bi} and p_{6c} were set to 0.8, 0.4 and 0.5, respectively. Maximal number of iterations was 500.

Grid based model for path planning problem was considered in [109]. For solving the path planning problem, membrane inspired algorithm based on the particle swarm optimization (mPSO) was proposed. We tested our proposed BSO combined with local search procedure in three environments used in [109]. Search space was 20×20 grid with 6, 8 and 10 obstacles. Population size was set to 100 and the maximal number of iterations was 2000, which are both significantly larger compared to parameters of our proposed algorithm. The proposed method was run 30 times for each test environment and the same path was found in each run for the first two test examples, with 6 and 8 obstacles, while for the environment with 10 obstacles two different paths were found in 30 runs. This proves the robustness of the brain storm optimization algorithm combined with Dijkstra's algorithm.

The first test environment, with 6 static obstacles, as well as the obtained path by our proposed method is shown in Fig. 6.9. Starting and target point are set in all three environments to be in the lower left and upper right corner, respectively. Path that was found by our proposed method was the same as the path presented in [109]. It contains 4 points between the start and target. These 4 points are needed in order

Algorithm 6.1: Pseudo-code of the proposed method for path planning problem

```
Initialization by local search procedure;
for i=1 to n do
    repeat
        Generate N random point in obstacle free grid cells;
        Calculate the matrix of distances between generated, starting and target points, setting the
          distances for unfeasible path segments to infinite;
        Find the shortest path by local search based on the matrix of distances;
    until shortest path is less then infinitive;
end
Find the highest solution dimension, d;
for all solutions s_i with dimension less then d do
    Find the largest path segment of s_i and divided equally with d − dim(s_i) points;
end
repeat
    Cluster n solutions into m clusters by k-means algorithm;
    Rank solutions in each cluster and set the best one as cluster;
    Randomly generate a value r between 0 and 1.;
    if r < p_5a then
        Randomly select a cluster center;
        Randomly generate a solution to replace the selected cluster;
    end
    repeat
        Randomly generate a value r between 0 and 1;
        if r < p_6b then
            Randomly select a cluster with probability p_6bi;
            Randomly generate a value r_1 between 0 and 1;
            if r_1 < p_6bii then
                Select the cluster center and add random values to it to generate new individual;
            else
                Randomly select a solution from the chosen cluster and add random value to the
                  solution to generate new one;
            end
        else
            Randomly select two clusters;
            Generate random value r_2 between 0 and 1;
            if r_2 < p_6c then
                Two cluster centers are combined to generate new solution;
            else
                Two solutions from each selected cluster are randomly chosen to be combined to
                  generate new individual.;
            end
        end
        The newly generated solution is compared with the same solution index and the better one is kept;
    until n new solution is generated;
until Maximal iteration number is reached;
return the best solution among all population
```

to avoid two obstacles that are disallowing to go straight from the start to the target point.

The second test environment is similar to the previous one with two extra obstacles. In order to reach the target point from the start, path has to go around the added obstacles. All path models that ensure moving forward in each step are not able to find feasible solution for this environment. Path founded by our proposed BSO method is

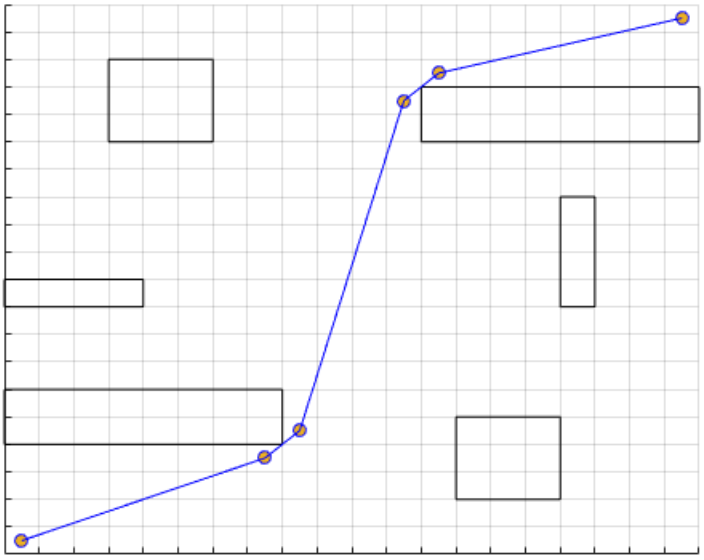


Figure 6.9 The first test environment, 6 obstacles

presented in Fig. 6.10. The path has also four points between the start and target points, as in the previous example. In [109], mPSO found a path with 6 points in between.

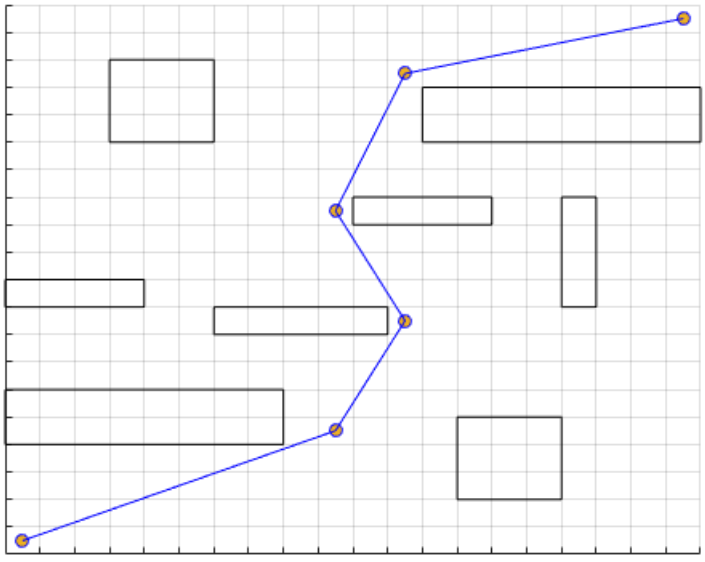


Figure 6.10 The second test environment, 8 obstacles

The third test environment is the most complicated. Into the previous test example two new obstacles were introduced. In Fig. 6.11 environment with the found solution is

presented. The optimal path for this example was built by 7 points that are necessary to avoid the obstacles precisely around the edges. In Fig. 6.12 is presented the second solution that was obtained by our proposed method. Path length for the this solution was 35.1172, while the path length of the first solution shown in Fig. 6.11 was 34.7999.

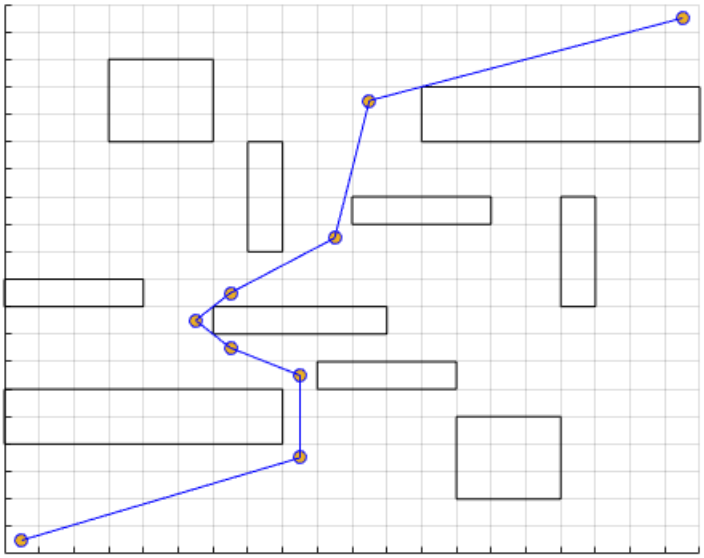


Figure 6.11 The third test environment, 10 obstacles

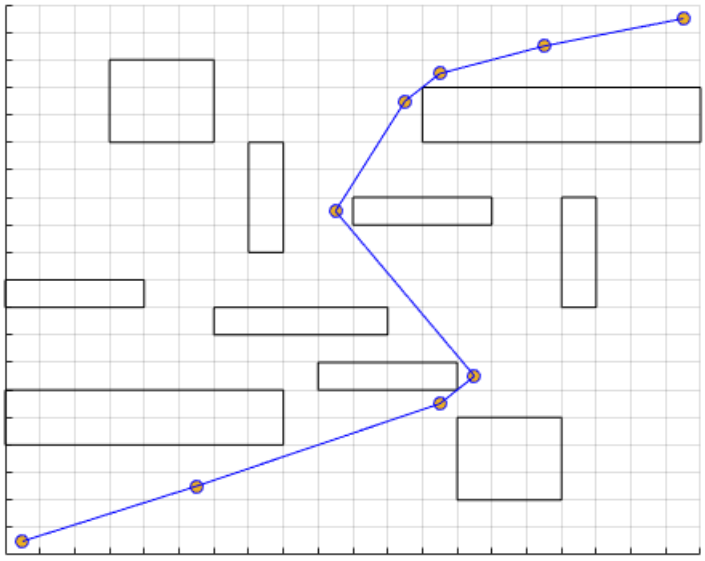


Figure 6.12 The third test environment, 10 obstacles

In addition, we compared our proposed method with the methods proposed in [197] where two improved particle swarm optimization (PSO) methods were proposed for robot path planning problem. In the first variant, nonlinear inertia weight coefficients were introduced to PSO in order to control local and global search ability. The second proposed method combined PSO by simulated annealing which prevents PSO from being trapped into local optima. The proposed method was tested on two different environments. Size of the field was 20×20 and the first environment contains 9 obstacles while the second one is more complex and contains 17 static obstacles. The proposed PSO methods were compared with the original PSO.

For the first environment in [197] (fourth in our research), basic PSO obtained a solution were the path length was 28.5006, nonlinear inertia weight PSO found a path of length 27.6853, while the simulated annealing PSO obtained a solution whose length is 28.1973. Our proposed BSO method found the shortest path compared to the all three PSO methods, 27.2442. Obtained solution is presented in Fig. 6.13.

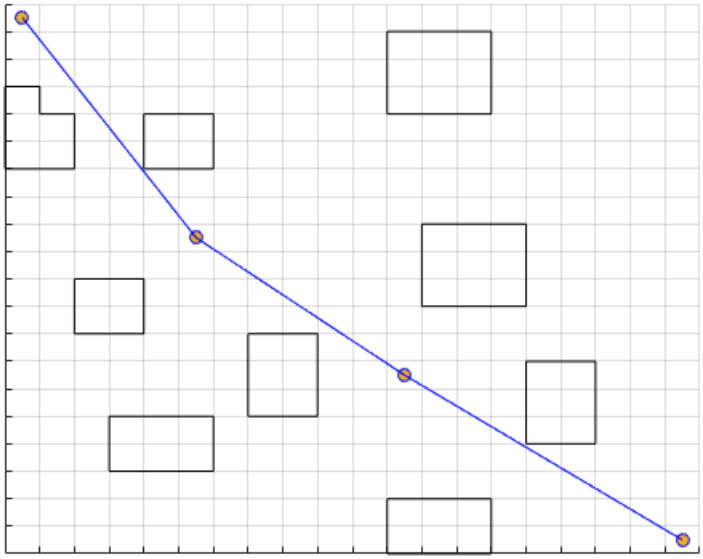


Figure 6.13 The fourth test environment, 9 obstacles

The second test performed in [197] represents path search in rather complex environment with 17 static obstacles in the grid 20×20. The test environment along with the path obtained by our proposed algorithm are shown in Fig. 6.14. Path length obtained by the BSO method is 28.3802. We calculated path lengths for the paths presented in [197] and the obtained values were 32.0153, 30.3623 and 28.7831 obtained

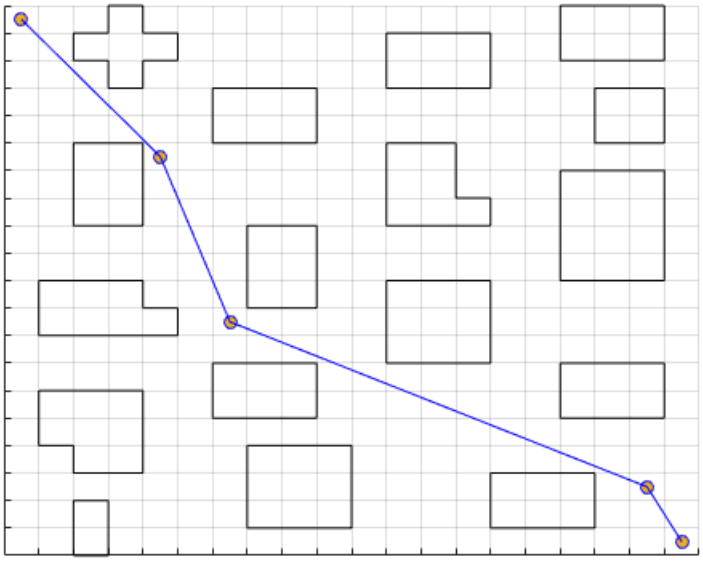


Figure 6.14 The fifth test environment, 17 obstacles

Table 6.12 PATH LENGTHS OBTAINED BY PSO, NONLINEAR INERTIA WEIGHT PSO (NLI-PSO) AND SIMULATED ANNEALING PSO (SA-PSO) [197] AND THE PROPOSED BSO METHOD

Environment	PSO	NLI-PSO	SA-PSO	BSO
Test 4	28.5006	27.6853	28.1973	**27.2442**
Test 5	32.0153	30.3623	28.7831	**28.3802**

by basic PSO, nonlinear inertia weight PSO and simulated annealing PSO methods, respectively (which is different from the path lengths reported along with the figures). Again, our proposed method found better solution compared to all three variants of the PSO algorithm. Described results are presented in Table 6.12.

Conclusion

In this dissertation we analysed method for solving robot path planning problem by using nature-inspired metaheuristics, with emphasis on swarm intelligence algorithms. We adjusted and applied brain storm optimization algorithm for solving robot path planning problem.

In the first part of the research we applied basic version of BSO algorithm on UCAV path planning problem. Fuel consumption and safety are considered as criteria for path optimality. The proposed method is tested in test environment from the literature, with circular threat zones and different threat degrees. Our proposed method was compared with ten other methods from the literature. It can be concluded, based on simulation results, that proposed brain storm optimization algorithm exhibits very promising features. It showed better performances for smaller problem dimensions, while for higher dimensions more iterations were needed, however the results for that case also showed improvement compared to other tester algorithms.

In the second part of the research we applied basic brain storm optimization algorithm on robot path planning problem in uncertain environment with static obstacles. We proposed probabilistic model for determining thereat degree, for threat sources with unknown exact positions. Two contradictory criteria, path length and safety, are connected by introducing a control parameter in the objective function. The proposed method solves the problem of unfeasible solutions by adding penalty value into objective function. Combination between penalty and increased exploration provided that feasible solutions were always generated. Comparison with competing algorithms proved that our proposed method, although simpler, was more efficient and robust because better solutions were obtained in all test cases.

Finally, mobile robot path planning problem in two-dimensional environment is investigated using improved BSO algorithm. The proposed method combines BSO algorithm with local search method in order to find the shortest path in a graph, with the aim of searching for optimal path in the environment with static obstacles. Path length is used as the only objective. The initial feasible paths for BSO are generated using deterministic local search procedure, and BSO is used to further optimize the path. The proposed method is tested for five different scenarios, and it is proved that it can find optimal feasible solutions.

Future research can include hybridization and modification of brain storm optimization algorithm in order to improve its convergence speed and adjust the algorithm to solving larger dimensional problems. Since in our research we used two-dimensional space for path planning problem description, future research can include third dimension, i.e UCAV altitude. BSO algorithm showed robustness and superior performances u test cases. However, its potential is even greater having in mind its unique feature of results clustering, that can be used for more complex, multi-objective formulations of UCAV path planning problem. Current research are focused on solving self-adaptive path planning problem for one UCAV vehicle, as well as on collaborative path planning, if the fleet of more aircrafts is considered. Adaptive path planning should have a possibility of data analysis in real time, in the case of uncertain and dynamic environment, and path replanning. Challenges of collaborative path planning refer to coordination between multiple UCAV vehicles, including flight formation, constraints with respect to arrival on destination, avoiding conflicts etc.

Future research can also include numerous improvements. Since paths are far from random collection of points, but have many inherent relations and dependencies, it may be possible to exploit that to make more efficient algorithms. In future research, real search spaces can be used instead of grid based models, and the initial points can be obtained using some guidance instead of using randomly deployed points in search space.

References

[1] P. Pedregal, Introduction to Optimization, Springer-Verlag, New York, 2004.

[2] XS. Yang, Nature-Inspired Metaheuristic Algorithms: Second Edition, Luniver Press, UK, 2010.

[3] E. Chong, S. Zak, An Introduction to Optimization, John Wiley and Sons, USA, 2001.

[4] B. Korte, J. Vygen, Combinatorial Optimization, Theory and Algorithms, Third Edition, Springer-Verlag, 2005.

[5] E. L. Lawler, Combinatorial Optimization:Networks and Matroids, Holt, Rinehart and Winston, USA, 1976.

[6] M. Gestal, M.P. Gomez-Carracedo, Finding Multiple Solutions with GA in Multimodal Problems, Encyclopedia of Artificial Intelligence, pp. 647-653, doi: 10.4018/9781599048499.ch098, 2009.

[7] T. H. Cormen, C. E. Leiserson,R. L. Rivest, C. Stein, Introduction to Algorithms, Second Edition, The MIT Press, USA, 2001.

[8] O. Bozorg-Haddad (ed.), Advanced Optimization by Nature-inspired Algorithms, Springer Nature Singapore Pte Ltd, Singapore, 2018.

[9] M. Jamil, XS. Yang, A literature survey of benchmark functions for global optimization problems, Int. Journal of Mathematical Modelling and Numerical Optimisation, Vol. 4, No. 2, pp. 150-194, 2013.

[10] Fister Jr., XS. Yang, I. Fister, J. Brest, D. Fister, A Brief Review of Nature-Inspired Algorithms for Optimization, Elektrotehnicki Vestnik, Vol. 80, No. 3, pp. 1-7, 2013.

[11] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi, Complexity and Approximation, Combinatorial optimization problems and their approximability properties, Springer-Verlag, Germany, 1999.

[12] T. El-Ghazali, Metaheuristics: From Design to Implementation, John Wiley and Sons, USA, 2009.

[13] I. Boussaid, J. Lepagnot, P. Siarry, A survey on optimization metaheuristics, Information Sciences, Vol. 237, pp. 82-117, 2013.

[14] T.O. Ting, XS. Yang, S. Cheng, K. Huang, Hybrid Metaheuristic Algorithms: Past, Present, and Future. In: Yang XS. (ed.) Recent Advances in Swarm Intelligence and Evolutionary Computation. Studies in Computational Intelligence, Vol. 585, Springer, Cham, Switzerland, 2015.

[15] A. Colorni, M. Dorigo, F. Maoli, V. Maniezzo, G. Righini, M. Trubian, Heuristics from nature for hard combinatorial optimization problems, International Transactions in Operational Research, Vol. 3, No. 1, pp. 1-21, 1996.

[16] XS. Yang (ed.), Nature-Inspired Computation in Engineering, Springer International Publishing, Switzerland, 2016.

[17] XS. Yang, Nature-Inspired Optimization Algorithms, Elsevier, USA, 2014.

[18] K. L. Du, M. N. S. Swamy, Search and Optimization by Metaheuristics, Techniques and Algorithms Inspired by Nature, Springer International Publishing, Switzerland, 2016.

[19] XS. Yang, X. He, Swarm Intelligence and Evolutionary Computation: Overview and Analysis. In: XS. Yang (eds.) Recent Advances in Swarm Intelligence and Evolutionary Computation. Studies in Computational Intelligence, Vol. 585, Springer, Cham, Switzerland, 2015.

[20] S. Kirkpatrick, C. D. Gelatt, Jr., M. P. Vecchi, Optimization by Simulated Annealing, Science, Vol. 220, Issue 4598, pp. 671-680, 1983.

[21] E. Aarts, J. Korst, W. Michiels, Simulated Annealing. In: Burke E.K., Kendall G. (eds.) Search Methodologies, Springer, Boston, MA, 2005.

[22] B Suman, P. Kumar, A survey of simulated annealing as a tool for single and multiobjective optimization, Journal of the Operational Research Society, Vol. 57, No. 10, pp. 1143-1160, 2006.

[23] M. Melanie, An Introduction to Genetic Algorithms, The MIT Press, USA, 1999.

[24] M. Srinivas, L. M. Patnaik, Genetic algorithms: a survey, Computer, Vol. 27, No. 6, pp. 17-26, 1994.

[25] R. C. Eberhart, Y. Shi, Comparison between Genetic Algorithms and Particle Swarm Optimization, Proceedings of the 7th International Conference on Evolutionary Programming, pp. 611-616, San Diego, California, USA, 1998.

[26] S. Koziel, XS. Yang (eds.), Computational Optimization, Methods and Algorithms, Springer-Verlag, Germany, 2011.

[27] R. Storn, K. Price, Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces, Journal of Global Optimization, Vol. 11, pp. 341-359, 1997.

[28] A.K. Qin, P.N. Suganthan, Self-adaptive differential evolution algorithm for numerical optimization, 2005 IEEE Congress on Evolutionary Computation, pp. 1785-1791, Vol. 2., Scotland, UK, 2005.

[29] F. Neri, V. Tirronen, Recent advances in differential evolution: a survey and experimental analysis, Artificial Intelligence Review, Vol. 33, Issue 1-2, pp. 61-106, 2010.

[30] S. Das, P. N. Suganthan, Differential Evolution: A Survey of the State-of-the-Art, IEEE Transactions on Evolutionary Computation, Vol. 15, Issue 1, pp. 4-31, 2011.

[31] E. Dolicanin, I. Fetahovic, Monte Carlo Optimization of Redundancy of Nanotechnology Computer Memories in the Conditions of Background Radiation, Nuclear Technology and Radiation Protection, Vol. XXXIII, No. 2, 2018.

[32] L. Fisher, The Perfect Swarm: The Science of Complexity in Everday Life, Basic Books, USA, 2009.

[33] E.F. Keller, Organisms, Machines, and Thunderstorms: A History of Self-Organization, Part Two: Complexity, Emergence, and Stable Attractors, Historical Studies in the Natural Sciences, Vol. 39, No. 1, pp. 1-31, 2009.

[34] E. Bonabeau, M. Dorigo,G. Theraulaz, Swarm intelligence: from natural to artificial systems, Oxford University Press Inc, New York, NY, USA , 1999.

[35] Y. Shi, Developmental swarm intelligence: developmental learning perspective of swarm intelligence algorithms, Int J Swarm Intell Res, Vol. 5, No. 1, pp. 36-54, 2014.

[36] M. Dorigo, G. Di Caro, L. M. Gambardella, Ant Algorithms for Discrete Optimization, Artificial Life, Vol. 5, No. 2, pp. 137-172, 1999.

[37] M. Dorigo, T. Stutzle, The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances. In: F. Glover, G.A. Kochenberger (eds) Handbook of Metaheuristics. International Series in Operations Research and Management Science, Vol 57. Springer, Boston, MA, USA, 2003.

[38] M. Dorigo, G. Di Caro, Ant Colony Optimization: A New Meta-Heuristic, Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), pp. 1477 , Vol. 2., Washington, DC, USA, 1999.

[39] M. Dorigo, C. Blum, Ant colony optimization theory: A survey, Theoretical Computer Science, Vol. 344, Issues 2-3, pp. 243-278, 2005.

[40] M. Gendreau, JY. Potvin (eds.), Handbook of Metaheuristics Second Edition, Springer, USA, 2010. M. Dorigo, V. Maniezzo, A. Colorni, Ant system: optimization by a colony of cooperating agents, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), Vol. 26, No. 1, pp. 29-41, 1996.

[41] C. Blum, Ant colony optimization: Introduction and recent trends, Physics of Life Reviews, Vol. 2, Issue 4, pp. 353-373, 2005.

[42] M. Dorigo, G. Di Caro, The ant colony optimization meta-heuristic. In: D. Corne, M. Dorigo, F. Glover, (eds.) New Ideas in Optimization, pp. 11-32, McGraw Hill, London, UK, 1999.

[43] C.W. Reynolds, Flocks, herds, and schools: a distributed behavioral model, Comput Graph 21(4):25-34, 1987.

[44] F. Heppner, U. Grenander, A stochastic nonlinear model for coordinated bird flocks, In: S. Krasner (ed.), The Ubiquity of Chaos, AAAS Publications, Washington DC, USA, 1990.

[45] J. Kennedy, R.C. Eberhart, Particle swarm optimization, Proceedings of the IEEE international conference on neural networks, pp. 1942-1948, Perth, Australia, 1995.

[46] Y. Shi, R.C. Eberhart, A modified particle swarm optimizer, Proceedings of the IEEE international conference on evolutionary computation, pp. 69-73, Anchorage, Alaska, USA, 1998.

[47] Y. Zhang, S. Wang, G. Ji, A Comprehensive Survey on Particle Swarm Optimization Algorithm and Its Applications, Mathematical Problems in Engineering, Vol. 2015, Article ID 931256, 38 pages, 2015.

[48] T. D. Seeley, The wisdom of the hive : the social physiology of honey bee colonies, Harvard University Press, Cambridge, Massachusetts, USA, 1995.

[49] D. Karaboga, B. Gorkemli, C. Ozturk, N. Karaboga, A comprehensive survey: artificial bee colony (ABC) algorithm and applications, Artificial Intelligence Review, Vol. 42, Issue 1, pp. 21-57, 2014.

[50] J. C. Bansal, H. Sharma, S. S. Jadon, Artificial bee colony algorithm: a survey, International Journal of Advanced Intelligence Paradigms, Vol. 5, Issue 1-2, pp. 123-159, 2013.

[51] F. S. Abu-Mouti, M. E. El-Hawary, Overview of Artificial Bee Colony (ABC) Algorithm and Its Applications, IEEE International Systems Conference SysCon 2012, pp. 1-6, Vancouver, Canada, 2012.

[52] D. Karaboga, B. Akay, A comparative study of Artificial Bee Colony algorithm, Applied Mathematics and Computation, Vol. 214, Issue 1, pp. 108-132, 2009.

[53] D. Karaboga, An Idea based on Honey Bee Swarm for Numerical Optimization, Techn.Rep. TR06, Erciyes Univ. Press, Erciyes, Turkey, 2005.

[54] B. Akay, D. Karaboga, A modified Artificial Bee Colony algorithm for real-parameter optimization, Information Sciences, Vol. 192, pp. 120-142, 2012.

[55] XS. Yang, Nature-Inspired Metaheuristic Algorithms, Luniver Press, UK, 2008.

[56] O. Watanabe, T. Zeugmann (eds.), Stochastic Algorithms: Foundations and Applications, Proceedings of the 5th International Symposium, SAGA 2009, Sapporo, Japan, 2009.

[57] M. Bramer, R. Ellis, M. Petridis (eds.), Research and Development in Intelligent Systems XXVI Incorporating Applications and Innovations in Intelligent Systems XVII, Springer-Verlag London, UK, 2010.

[58] XS. Yang, Engineering Optimization: An Introduction with Metaheuristic Applications, John Wiley and Sons, USA, 2010.

[59] XS. Yang (ed.), Cuckoo Search and Firefly Algorithm Theory and Applications, Springer International Publishing, Switzerland, 2014.

[60] XS. Yang, Firefly Algorithms for Multimodal Optimization. In: O. Watanabe, T. Zeugmann (eds.) Stochastic Algorithms: Foundations and Applications SAGA 2009, Lecture Notes in Computer Science, Vol. 5792, Springer, Berlin, Germany,2009.

[61] R.B. Payne, M.D. Sorenson, K. Klitz, The Cuckoos, Oxford University Press, UK, 2005.

[62] XS. Yang, S. Deb, Engineering optimisation by cuckoo search, Int. J. Mathematical Modelling and Numerical Optimisation, Vol. 1, No. 4, pp. 330-343, 2010.

[63] XS. Yang, S. Deb, Cuckoo Search via LBẏevy Flights, 2009 World Congress on Nature and Biologically Inspired Computing (NaBIC), pp. 210-214, Coimbatore, India, 2009.

[64] XS. Yang, A New Metaheuristic Bat-Inspired Algorithm, In: J.R. Gonzalez, D.A. Pelta, C. Cruz, G. Terrazas, N. Krasnogor (eds.) Nature Inspired Coop-

erative Strategies for Optimization (NICSO 2010). Studies in Computational Intelligence, Vol. 284, Springer, Berlin, Germany, 2010.

[65] A.H. Gandomi, XS. Yang, A.H. Alavi, S. Talatahari, Bat algorithm for constrained optimization tasks, Neural Computing and Applications, Vol. 22, Issue 6, pp. 1239-1255, 2013.

[66] XS. Yang, A.H. Gandomi, Bat algorithm: a novel approach for global engineering optimization, Engineering Computations, Vol. 29, Issue 5, pp. 464-483, 2012.

[67] Y. Shi, Brainstorm optimization algorithm, In: Y. Tan, Y. Shi, Y. Chai, G. Wang (eds.), Advances in swarm intelligence, lecture notes in computer science, Vol. 6728, pp. 303-309, Springer, Berlin, 2011.

[68] Y. Shi, An optimization algorithm based on brainstorming process, Int J Swarm Intell Res, Vol. 2, No. 4, pp. 35-62, 2011.

[69] Z. Cao, Y. Shi, X. Rong, B. Liu, Z. Du, B. Yang, Random Grouping Brain Storm Optimization Algorithm with a New Dynamically Changing Step Size, In: Y. Tan, Y. Shi, F. Buarque, A. Gelbukh, S. Das, A. Engelbrecht (eds.), Advances in Swarm and Computational Intelligence ICSI 2015, Lecture Notes in Computer Science, Vol. 9140, Springer, Cham, Switzerland, 2015.

[70] Z. Zhan, J. Zhang, Y. Shi, H. Liu, A modified brain storm optimization, 2012 IEEE Congress on Evolutionary Computation, pp. 1-8, Brisbane, QLD, Australia, 2012.

[71] M. El-Abd, Brain storm optimization algorithm with re-initialized ideas and adaptive step size, 2016 IEEE Congress on Evolutionary Computation (CEC), , pp. 2682-2686, Vancouver, Canada, 2016.

[72] M. El-Abd, Global-best brain storm optimization algorithm, Swarm and Evolutionary Computation, Vol. 37, pp. 27-44, 2017.

[73] D. Zhou, Y. Shi, S. Cheng, Brain Storm Optimization Algorithm with Modified Step-Size and Individual Generation, In: Y.Tan, Y.Shi, Z. Ji (eds.), Advances in

Swarm Intelligence, ICSI 2012, Lecture Notes in Computer Science, Vol. 7331, Springer, Berlin, Heidelberg, Germany, 2012.

[74] J. Xue, Y. Wu,Y. Shi, S. Cheng, Brain Storm Optimization Algorithm for Multi-objective Optimization Problems, In: Y.Tan, Y.Shi, Z. Ji (eds.), Advances in Swarm Intelligence, ICSI 2012, Lecture Notes in Computer Science, Vol. 7331, Springer, Berlin, Heidelberg, Germany, 2012.

[75] J. Chen, Y. Xie, J. Ni, Brain Storm Optimization Model Based on Uncertainty Information, 2014 Tenth International Conference on Computational Intelligence and Security, pp. 99-103, Kunming, China, 2014.

[76] J. Chen, S. Cheng,Y. Chen, Y. Xie, Y. Shi, Enhanced Brain Storm Optimization Algorithm for Wireless Sensor Networks Deployment. In: Y. Tan, Y. Shi, F. Buarque, A. Gelbukh, S. Das, A. Engelbrecht (eds) Advances in Swarm and Computational Intelligence, ICSI 2015, Lecture Notes in Computer Science, Vol. 9140, Springer, Cham, Switzerland, 2015.

[77] H. Zhu, Y. Shi, Brain storm optimization algorithms with k-medians clustering algorithms, 2015 Seventh International Conference on Advanced Computational Intelligence (ICACI), pp. 107-110, Wuyi, China, 2015.

[78] Y. Shi, Brain storm optimization algorithm in objective space, 2015 IEEE Congress on Evolutionary Computation (CEC), pp. 1227-1234, Sendai, Japan, 2015.

[79] S. Cheng, Y. Shi, Q. Qin, T. O. Ting, R. Bai, Maintaining population diversity in brain storm optimization algorithm, 2014 IEEE Congress on Evolutionary Computation (CEC), pp. 3230-3237, Beijing, China, 2014.

[80] Z. Yang, Y. Shi, Brain storm optimization with chaotic operation, 2015 Seventh International Conference on Advanced Computational Intelligence (ICACI), pp. 111-115, Wuyi, China, 2015.

[81] XS. Yang, Chaos-Enhanced Firefly Algorithm with Automatic Parameter Tuning, International Journal of Swarm Intelligence Research (IJSIR), Vol. 2, No. 4, 2011.

[82] Z. Jia, H. Duan, Hybrid brain storm optimisation and simulated annealing algorithm for continuous optimisation problems, Int. J. Bio-Inspired Computation, Vol. 8, No. 2, pp. 109-121, 2016.

[83] K.R. Krishnanand, S.M.F. Hasani, B.K. Panigrahi, S.K. Panda, Optimal Power Flow Solution Using Self-Evolving Brain-Storming Inclusive Teaching-Learning-Based Algorithm, In: Y. Tan, Y. Shi, H. Mo (eds) Advances in Swarm Intelligence, ICSI 2013, Lecture Notes in Computer Science, Vol. 7928, Springer, Berlin, Heidelberg, Germany, 2013.

[84] Z. Cao, X. Hei, L. Wang, Y. Shi, X. Rong, An Improved Brain Storm Optimization with Differential Evolution Strategy for Applications of ANNs, Mathematical Problems in Engineering, Vol. 2015, Article ID 923698, 18 pages, 2015.

[85] Y. Wu, L. Xie, Q. Liu, Multi-objective Brain Storm Optimization Based on Estimating in Knee Region and Clustering in Objective-Space, In: Y. Tan, Y. Shi, B. Niu (eds.), Advances in Swarm Intelligence, 7th International Conference, ICSI 2016, Bali, Indonesia, Lecture Notes in Computer Science, Vol. 9713, Springer, Berlin, Heidelberg, Germany, 2016.

[86] L. Xie, Y. Wu, A Modified Multi-Objective Optimization Based on Brain Storm Optimization Algorithm. In: Y. Tan, Y. Shi, C.A.C. Coello (eds), Advances in Swarm Intelligence, ICSI 2014, Lecture Notes in Computer Science, Vol. 8795, Springer, Cham, Switzerland, 2014.

[87] M. Ester, H.P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, Proceedings of the Second International Conference on Knowledge Discovery and Data Mining KDD'96, pp. 226-231, Portland, Oregon, USA, 1996.

[88] X. Guo, Y. Wu, L. Xie, Modified Brain Storm Optimization Algorithm for Multimodal Optimization In: Y. Tan, Y. Shi, C.A.C. Coello (eds), Advances in Swarm Intelligence, ICSI 2014, Lecture Notes in Computer Science, Vol. 8795, Springer, Cham, Switzerland, 2014.

[89] Z. Zhan, W. Chen, Y. Lin, Y. Gong, Y. Li, J. Zhang, Parameter investigation in brain storm optimization,2013 IEEE Symposium on Swarm Intelligence (SIS), pp. 103-110, Singapore, 2013.

[90] S. Cheng, Y. Shi, Q. Qin, S. Gao, Solution clustering analysis in brain storm optimization algorithm, 2013 IEEE Symposium on Swarm Intelligence (SIS), pp. 111-118, Singapore, 2013.

[91] H. Jadhav, U. Sharma, J. Patel, R. Roy, Brain storm optimization algorithm based economic dispatch considering wind power, In: Proceedings of the 2012 IEEE international conference on power and energy (PECon 2012), pp 588-593, Kota Kinabalu, Malaysia, 2012.

[92] K. Ramanand, K. Krishnanand, BK. Panigrahi, MK. Mallick, Brain storming incorporated teaching learning-based algorithm with application to electric power dispatch, In: BK. Panigrahi, S. Das, PN. Suganthan, PK. Nanda (eds), Swarm, evolutionary, and memetic computing, Vol. 7677, Lecture Notes in Computer Science, pp. 476-483, Springer, Berlin, Germany, 2012.

[93] AR. Jordehi, Brainstorm optimisation algorithm (BSOA): an efficient algorithm for finding optimal location and setting of facts devices in electric power systems, Electr Power Energy Syst, Vol. 69, pp. 48-57, 2015.

[94] GW. Zhang, ZH. Zhan, KJ. Du, WN. Chen, Normalization group brain storm optimization for power electronic circuit optimization, In: Proceedings of the 2014 conference companion on genetic and evolutionary computation companion, GECCO Comp '14ACM, pp. 183-184, New York, NY, USA, 2014.

[95] K. Lenin, BR. Reddy, MS. Kalavathi, Brain storm optimization algorithm for solving optimal reactive power dispatch problem, Int J Res Electron Commun Technol, Vol. 1, No. 3, pp. 25-30, 2014.

[96] X. Zhao, Research and application of brain storm optimization algorithm, Master's thesis, Xi'an University of Technology, 2013.

[97] M. Arsuaga-Rios, MA. Vega-Rodriguez, Cost optimization based on brain storming for grid scheduling, Fourth edition of the International Conference on the Innovative Computing Technology (INTECH 2014), pp. 31-36, Luton, UK, 2014.

[98] Y. Sun, A hybrid approach by integrating brain storm optimization algorithm with grey neural network for stock index forecasting, Abstract Appl Anal, Vol. 2014, pp. 1-10, 2014.

[99] C. Sun, H. Duan, Y. Shi, Optimal satellite formation reconfiguration based on closed-loop brain storm optimization, IEEE Comput Intell Mag, Vol. 8, Issue 4, pp 39-51, 2013.

[100] J. Li, H. Duan, Simplified brain storm optimization approach to control parameter optimization in F/A-18 automatic carrier landing system, Aerosp Sci Technol, Vol. 42, pp. 187-195, 2015.

[101] H. Duan, C. Li, Quantum-behaved brain storm optimization approach to solving loney's solenoid problem, IEEE Trans Magn, Vol. 51, Issue 1, pp. 1-7, 2015.

[102] H. Duan, S. Li, Y. Shi, Predator-prey brain storm optimization for DC brushless motor, IEEE TransMagn, Vol. 49, Issue 10, pp. 5336-5340, 2013.

[103] K. M. Lynch, F. C. Park, Modern Robotics: Mechanics, Planning, and Control 1st Edition, Cambridge University Press, UK, 2017.

[104] K. Nonami, F. Kendoul, S. Suzuki, W. Wang, D. Nakazawa, Autonomous Flying Robots: Unmanned Aerial Vehicles and Micro Aerial Vehicles, Springer, 2010.

[105] C. Goerzen, Z. Kong, B. Mettler, A Survey of Motion Planning Algorithms from the Perspective of Autonomous UAV Guidance, J Intell Robot Syst, Vol. 57, pp. 65-100, 2010.

[106] K. Hwang Yong, A. Narendra, Gross Motion Planning- A Survey, ACM Computing Surveys, Vol. 24, No. 3, pp. 219-291, 1992.

[107] T. Lozano-Perez, M.A. Wesley, An algorithm for planning collision-free paths among polyhedral obstacles, Communications of the ACM, Vol. 22, Issue 10, pp. 560-570, 1979.

[108] V. Girbes, L. Armesto, J. Tornero, Path following hybrid control for vehicle stability applied to industrial forklifts, Robotics and Autonomous Systems, Vol. 62, Issue 6, pp. 910-922, 2014.

[109] X.Y. Wang, G.X. Zhang, J.B. Zhao, H.N. Rong, F. Ipate, R. Lefticaru, A Modified Membrane-Inspired Algorithm Based on Particle Swarm Optimization for Mobile Robot Path Planning, International Journal of Computers Communications and Control, Vol. 10, No. 5, pp. 732-745, 2015.

[110] XN. Bui, P. Soueres, JD. Boissonnat, JP. Laumond, The Shortest path synthesis for non-holonomic robots moving forwards, [Research Report] RR-2153, INRIA, 1994.

[111] E.P. Anderson, R.W. Beard, T.W. McLain, Real-time dynamic trajectory smoothing for unmanned air vehicles, IEEE Transactions on Control Systems Technology, Vol. 13, Issue 3, pp. 471-477, 2005.

[112] M. Hwangbo, J. Kuffner, T. Kanade, Efficient Two-phase 3D Motion Planning for Small Fixed-wing UAVs, Proceedings 2007 IEEE International Conference on Robotics and Automation, pp. 1035-1041, Roma, Italy, 2007.

[113] M. Elbanhawi, M. Simic, R.N. Jazar, Continuous Path Smoothing for Car-Like Robots Using B-Spline Curves, Journal of Intelligent and Robotic Systems, Vol. 80, Supplement 1, pp. 23-56, 2015.

[114] J. Tang, J. Zhu, Z. Sun, A Novel Path Planning Approach Based on AppART and Particle Swarm Optimization, In: J. Wang, XF. Liao, Z. Yi (eds) Advances in Neural Networks вЃ Y ISNN 2005, ISNN 2005, Lecture Notes in Computer Science, Vol. 3498, Springer, Berlin, Heidelberg, Germany, 2005.

[115] S. H. Tang, W. Khaksar, N. B. Ismail, M. K. A. Ariffin, A Review on Robot Motion Planning Approaches, Pertanika J. Sci. and Technol., Vol. 20, No. 1, pp. 15-29, 2012.

[116] L. Yang, J. Qi, D. Song, J. Xiao, J. Han, Y. Xia, Survey of Robot 3D Path Planning Algorithms, Journal of Control Science and Engineering, Vol. 2016, Article ID 7426913, 22 pages, 2016.

[117] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, J.P. How, Real-Time Motion Planning With Applications to Autonomous Urban Driving, IEEE Transactions on Control Systems Technology, Vol. 17, Issue 5, pp. 1105-1118 , 2009.

[118] M. K. Habib, H. Asama, Efficient method to generate collision free paths for an autonomous mobile robot based on new free space structuring approach, Intelligent Robots and Systems '91. 'Intelligence for Mechanical Systems, Proceedings IROS '91. IEEE/RSJ International Workshop on, pp. 563-567 Vol.2, Osaka, Japan, 1991.

[119] S. E. Muldoon, C. Luo, F. Shen, H. Mo, Naturally inspired optimization algorithms as applied to mobile robotic path planning, 2014 IEEE Symposium on Swarm Intelligence, pp. 1-6, Orlando, FL, USA, 2014.

[120] R.S. Tavares, T.C. Martins, M.S.G. Tsuzuki, Simulated annealing with adaptive neighborhood: A case study in off-line robot path planning, Expert Systems with Applications, Vol. 38, pp. 2951-2965, 2011.

[121] M. G. Park, M. C. Lee, Experimental evaluation of robot path planning by artificial potential field approach with simulated annealing, Proceedings of the 41st SICE Annual Conference, SICE 2002, pp. 2190-2195, Vol.4, Osaka, Japan, 2002.

[122] F. Janabi-Sharifi, D. Vinke, Integration of the artificial potential field approach with simulated annealing for robot path planning, Proceedings of 8th IEEE International Symposium on Intelligent Control, pp. 536-541, Chicago, IL, USA, 1993.

[123] C. Zheng, L. Li, F. Xu, F. Sun, M. Ding, Evolutionary Route Planner for Unmanned Air Vehicles, IEEE Transactions on Robotics, Vol. 21, No. 4, pp. 609-620, 2005.

[124] E. Besada-Portas, L. de la Torre, J. M. de la Cruz, B. de AndrГкs-Toro, Evolutionary Trajectory Planner for Multiple UAVs in Realistic Scenarios, IEEE Transactions on Robotics, Vol. 26, No. 4, pp. 619-634, 2010.

[125] A. Hosseinzadeh, H. Izadkhah, Evolutionary Approach for Mobile Robot Path Planning in Complex environment, International Journal of Computer Science Issues, Vol. 7, Issue 4, No. 8, pp. 1-9, 2010.

[126] M. Davoodi, F. Panahi, A. Mohades, S. N. Hashemi, Clear and smooth path planning, Applied Soft Computing, Vol. 32, pp. 568-579, 2015.

[127] Y. V. Pehlivanoglu, A new vibrational genetic algorithm enhanced with a Voronoi diagram for path planning of autonomous UAV, Aerospace Science and Technology, Vol. 16, pp. 47-55, 2012.

[128] Q. Li, W. Zhang, Y. Yin, Z. Wang, G. Liu, An Improved Genetic Algorithm of Optimum Path Planning for Mobile Robots, Sixth International Conference on Intelligent Systems Design and Applications, pp. 637-642, Jinan, China, 2006.

[129] Tuncer, M. Yildirim, Dynamic path planning of mobile robots with improved genetic algorithm, Computers and Electrical Engineering, Vol. 38, pp. 1564-1572, 2012.

[130] M. Chen, A. M. S. Zalzala, Safety considerations in the optimisation of paths for mobile robots using genetic algorithms, First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, pp. 299-306, Sheffield, UK, 1995.

[131] AL-Taharwa, A. Sheta, M. Al-Weshah, A Mobile Robot Path Planning Using Genetic Algorithm in Static Environment, Journal of Computer Science, Vol. 4, No. 4, pp. 341-344, 2008.

[132] F. Ahmed, K. Deb, Multi-objective optimal path planning using elitist non-dominated sorting genetic algorithms, Soft Computing, Vol. 17, Issue 7, pp. 1283-1299, 2013.

[133] Z. Cheng, Y. Sun, Y. Liu, Path planning based on immune genetic algorithm for UAV, 2011 International Conference on Electric Information and Control Engineering, pp. 590-593, Wuhan, China, 2011.

[134] K. Nikolos, K. P. Valavanis, N. C. Tsourveloudis, A. N. Kostaras, Evolutionary algorithm based offline/online path planner for UAV navigation, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), Vol. 33, No. 6, pp. 898-912, 2003.

[135] S. Mittal, K. Deb, Three-dimensional offline path planning for UAVs using multiobjective evolutionary algorithms, 2007 IEEE Congress on Evolutionary Computation, pp. 3195-3202, Singapore, 2007.

[136] A. Elshamli, H. A. Abdullah, S. Areibi, Genetic algorithm for dynamic path planning, Canadian Conference on Electrical and Computer Engineering 2004 (IEEE Cat. No.04CH37513), pp. 677-680, Vol.2, Ontario, Canada, 2004.

[137] Z. Yao, L. Ma, A Static Environment-Based Path Planning Method by Using Genetic Algorithm, 2010 International Conference on Computing, Control and Industrial Engineering, pp. 405-407, Wuhan, China, 2010.

[138] J. Cao, Y. Li, S. Zhao, X. Bi, Genetic-Algorithm-Based Global Path Planning for AUV, 2016 9th International Symposium on Computational Intelligence and Design (ISCID), pp. 79-82, Hangzhou, China, 2016.

[139] D. M. Pierre, N. Zakaria, A. J. Pal, Master-Slave Parallel Vector-Evaluated Genetic Algorithm for Unmanned Aerial Vehicle's path planning, 2011 11th International Conference on Hybrid Intelligent Systems (HIS), pp. 517-521, Melacca, Malaysia, 2011.

[140] V. Roberge, M. Tarbouchi, G. Labonte, Comparison of Parallel Genetic Algorithm and Particle Swarm Optimization for Real-Time UAV Path Planning, IEEE Transactions on Industrial Informatics, Vol. 9, No. 1, pp. 132-141, 2013.

[141] X. Zhang, H. Duan, An improved constrained differential evolution algorithm for unmanned aerial vehicle global route planning planning, Applied Soft Computing, Vol. 26, pp. 270-284, 2015.

[142] Z. Zhou, H. Duan, P. Li, B. Di, Chaotic differential evolution approach for 3D trajectory planning of unmanned aerial vehicle, 2013 10th IEEE International

Conference on Control and Automation (ICCA), pp. 368-372, Hangzhou, China, 2013.

[143] A. N. Brintaki, I.K. Nikolos, Coordinated UAV path planning using Differential Evolution, Operational Research, Vol. 5, Issue 3, pp. 487-502, 2005.

[144] J. Chakraborty, A. Konar, L. C. Jain, U. K. Chakraborty, Cooperative multi-robot path planning using differential evolution, Journal of Intelligent and Fuzzy Systems: Applications in Engineering and Technology - Theoretical advances of intelligent paradigms, Vol. 20, Issue 1,2, pp. 13-27, 2009.

[145] C. L. Huo, T. Y. Lai, T. Y. Sun, The preliminary study on multi-swarm sharing particle swarm optimization: Applied to UAV path planning problem, 2011 IEEE Congress of Evolutionary Computation (CEC), pp. 1770-1776, New Orleans, LA, USA, 2011.

[146] G. Han, W. Fu, W. Wang, The Study of Intelligent Vehicle Navigation Path Based on Behavior Coordination of Particle Swarm, Computational Intelligence and Neuroscience, Vol. 2016, Article ID 6540807, 10 pages, 2016.

[147] Y. Liu, X. Zhang, X. Guan, D. Delahaye, Adaptive sensitivity decision based path planning algorithm for unmanned aerial vehicle with improved particle swarm optimization, Aerospace Science and Technology, Vol. 58, pp. 92-102, 2016.

[148] Y. Zhang, D. Gong, J. Zhang, Robot path planning in uncertain environment using multi-objective particle swarm optimization, Neurocomputing, Vol. 103, pp. 172-185, 2013.

[149] Y. Fu, M. Ding, C. Zhou, Phase Angle-Encoded and Quantum-Behaved Particle Swarm Optimization Applied to Three-Dimensional Route Planning for UAV, IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, Vol. 42, No. 2, pp. 511-526, 2012.

[150] Y. Bao, X. Fu, X. Gao, Path planning for reconnaissance UAV based on Particle Swarm Optimization, 2010 Second International Conference on Computational Intelligence and Natural Computing, pp. 28-32, Wuhan, China, 2010.

[151] L. Guangshun, S. Hongbo, Path planning for mobile robot based on particle swarm optimization, 2008 Chinese Control and Decision Conference, Yantai, pp. 3290-3294, Shandong, China, 2008.

[152] M. Li, D. B. Wang, T. T. Bai, S. Z. Sheng, Route planning based on particle swarm optimization with threat heuristic, Electronics Optics and Control, Vol. 18, No. 12, pp. 1-4, 2011.

[153] C. Xin,Y.M. Li, Smooth path planning of a mobile robot using stochastic particle swarm optimization, Proceedings of the IEEE International Conference on Mechatronics and Automation, pp.1722-1727, Luoyang, China, 2006.

[154] E. Masehian, D.Sedighizadeh, Multi-objective PSO and NPSO based algorithms for robot path planning, Adv.Electr.Comput.Eng., Vol. 10, pp. 69-76, 2010.

[155] D. Gong, J. Zhang, Y. Zhang, Multi-objective particles warm optimization for robot path planning in environment with danger sources, J. Comput., Vol. 6, No. 8, pp. 1554-1561, 2011.

[156] Q.R. Zhang, G.C. Gu, Path planning based on improved binary particle swarm optimization algorithm, Proceedings of IEEE International Conference on Robotics, Automation and Mechatronics, pp. 462-466, Chendu, China, 2008.

[157] Z. C. Fan, Path planning method based on the algorithm of PSO and elastic rope for underwater vehicle in three-dimensional space [Master thesis], Harbin Engineering University, China, 2013.

[158] M. Saska, M. Macas, L. Preucil, L. Lhotska, Robot Path Planning using Particle Swarm Optimization of Ferguson Splines, 2006 IEEE Conference on Emerging Technologies and Factory Automation, pp. 833-839, Prague, Czech Republic, 2006.

[159] N. Geng, D. W. Gong, Y. Zhang, PSO-Based Robot Path Planning for Multisurvivor Rescue in Limited Survival Time, Mathematical Problems in Engineering, Vol. 2014, Article ID 187370, 10 pages, 2014.

[160] L. Lu, D. Gong, Robot Path Planning in Unknown Environments Using Particle Swarm Optimization, 2008 Fourth International Conference on Natural Computation, pp. 422-426, Jinan, China, 2008.

[161] GZ. Tan, H. He, A. Sloman, Ant Colony System Algorithm for Real-Time Globally Optimal Path Planning of Mobile Robots, Acta Automatica Sinica, Vol. 33, Issue 3, pp. 279-285, 2007.

[162] H. Wang, W. Xiong, Research on global path planning based on ant colony optimization for AUV, Journal of Marine Science and Application, Vol. 8, No. 1, pp. 58-64, 2009.

[163] L. Yang, K.S. Li, W.S. Zhang, Y. Wang, Y. Chen, L.F. Zheng, Three-dimensional Path Planning for Underwater Vehicles Based on an Improved Ant Colony Optimization Algorithm, Journal of Engineering Science and Technology Review, Vol. 8, No. 5, pp.24-33, 2015.

[164] G. Zhang, H. Jia, Global path planning of AUV based on improved ant colony optimization algorithm, 2012 IEEE International Conference on Automation and Logistics, pp. 606-610, Zhengzhou, China, 2012.

[165] G. Zhang, H. Jia, 3D path planning of AUV based on improved ant colony optimization, Proceedings of the 32nd Chinese Control Conference, pp. 5017-5022, Xi'an, China, 2013.

[166] W. Ye, D. Ma, H. Fan, Algorithm for Low Altitude Penetration Aircraft Path Planning with Improved Ant Colony Algorithm, Chinese Journal of Aeronautics, Vol. 18, No. 4, pp. 304-309, 2005.

[167] H. Duan, X. Zhang, J. Wu, G. Ma, Max-Min Adaptive Ant Colony Optimization Approach to Multi-UAVs Coordinated Trajectory Replanning in Dynamic and Uncertain Environments, Journal of Bionic Engineering, Vol. 6, No. 2, pp. 161-173, 2009.

[168] M. Chen, Q. Wu, C. Jiang, A modified ant optimization algorithm for path planning of UCAV, Applied Soft Computing, Vol., No. 4, pp. 1712-1718, 2008.

[169] M.A. Porta Garcia, O. Montiel, O. Castillo, R. SepΓəlveda, P. Melin, Path planning for autonomous mobile robot navigation with ant colony optimization and fuzzy cost function evaluation, Applied Soft Computing, Vol. 9, No. 3, pp. 1102-1110, 2009.

[170] X. Fan, X. Luo, S. Yi, S. Yang, H. Zhang, Optimal path planning for mobile robots based on intensified ant colony optimization algorithm, IEEE International Conference on Robotics, Intelligent Systems and Signal Processing, 2003, Proceedings 2003, pp. 131-136, Vol.1, Changsha, Hunan, China, 2003.

[171] G. Wang, L. Guo, H. Duan L. Liu, H. Wang, A Modified Firefly Algorithm for UCAV Path Planning, International Journal of Hybrid Information Technology, Vol.5, No.3, pp. 123-144, 2012.

[172] C. Liu, Y. Zhao, F. Gao, L. Liu ,Three-Dimensional Path Planning Method for Autonomous Underwater Vehicle Based on Modified Firefly Algorithm, Mathematical Problems in Engineering, Vol. 2015, Article ID 561394, 10 pages, 2015.

[173] P.K. Mohanty, D.R. Parhi, Optimal path planning for a mobile robot using cuckoo search algorithm, Journal of Experimental and Theoretical Artificial Intelligence, Vol. 28, No. 1-2, pp. 35-52, 2016.

[174] B. Li, L. Gong, W. Yang, An Improved Artificial Bee Colony Algorithm Based on Balance-Evolution Strategy for Unmanned Combat Aerial Vehicle Path Planning, The Scientific World Journal, Vol. 2014, Article ID 232704, 10 pages, 2014.

[175] C. Xu, H Duan, F. Liu, Chaotic artificial bee colony approach to Uninhabited Combat Air Vehicle (UCAV) path planning, Aerospace Science and Technology, Vol. 14, pp. 535-541, 2010.

[176] B. Zhang, H. Duan, Three-Dimensional Path Planning for Uninhabited Combat Aerial Vehicle Based on Predator-Prey Pigeon-Inspired Optimization in Dynamic Environment, IEEE/ACM Transactions on Computational Biology and Bioinformatics, Vol. 14, No. 1, pp. 97-107, 2017.

[177] C. Purcaru, R.E. Precup, D. Iercan, L.O. Fedorovici, R.C. David, F.Dragan, Optimal Robot Path Planning Using Gravitational Search Algorithm, International Journal of Artificial Intelligence, Vol. 10, No. S13, pp. 1-20, 2013.

[178] P. Li, H. Duan, Path planning of unmanned aerial vehicle based on improved gravitational search algorithm, Science China Technological Sciences, Vol. 55, No. 10, pp. 2712-2719, 2012.

[179] H. Duan, S. Liu, J. Wu, Novel intelligent water drops optimization approach to single UCAV smooth trajectory planning, Aerospace Science and Technology, Vol. 13, No. 8, pp. 442-449, 2009.

[180] Y. Zhang, G. Guan, X. Pu, The Robot Path Planning Based on Improved Artificial Fish Swarm Algorithm, Mathematical Problems in Engineering, Vol. 2016, Article ID 3297585, 11 pages, 2016.

[181] Y. Fu, M. Ding, C. Zhou, H. Hu, Route Planning for Unmanned Aerial Vehicle (UAV) on the Sea Using Hybrid Differential Evolution and Quantum-Behaved Particle Swarm Optimization, IEEE Transactions on Systems, Man, and Cybernetics: Systems, Vol. 43, No. 6, pp. 1451-1465, 2013.

[182] B. Tang, Z. Zhu, J. Luo, Hybridizing Particle Swarm Optimization and Differential Evolution for the Mobile Robot Global Path Planning, International Journal of Advanced Robotic Systems, Vol. 13, No. 3, pp. 1-17, 2016.

[183] H. Mo, L. Xu, Research of biogeography particle swarm optimization for robot path planning, Neurocomputing, Vol. 148, pp. 91-99, 2015.

[184] P.K.Das, H.S.Behera, S. Das, H.K.Tripathy, B.K.Panigrahi, S.K.Pradhan, A hybrid improved PSO-DV algorithm for multi-robot path planning in a clutter environment, Neurocomputing, Vol. 207, pp. 735-753, 2016.

[185] G. Wang, L. Guo, H. Duan, L. Liu, H. Wang, A Bat Algorithm with Mutation for UCAV Path Planning, The ScientificWorld Journal, Vol. 2012, Article ID 418946, 15 pages, 2012.

[186] S. Piao, B. Hong, Path planning of robot using genetic annealing algorithm, Proceedings of the 4th World Congress on Intelligent Control and Automation (Cat. No.02EX527), pp. 493-495, Vol.1, Shanghai, China, 2002.

[187] H. Duan, Y. Yu, X. Zhang, and S. Shao, Three-dimension path planning for UCAV using hybrid meta-heuristic ACODE algorithm, Simulation Modelling Practice and Theory, Vol. 18, No. 8, pp. 1104-1115, 2010.

[188] B.K. Oleiwi, H.Roth, B.I.Kazem, A hybrid approach based on ACO and GA for multi-objective mobile robot path planning, Appl.Mech.Mater., Vol. 527, pp. 203-212, 2014.

[189] C. Purcaru, R. E. Precup, D. Iercan, L. O. Fedorovici, R. C. David, Hybrid PSO-GSA robot path planning algorithm in static environments with danger zones, 2013 17th International Conference on System Theory, Control and Computing (ICSTCC), pp. 434-439, Sinaiapp, Romania, 2013.

[190] M. Ju, S. Wang, J. Guo, Path Planning Using a Hybrid Evolutionary Algorithm Based on Tree Structure Encoding, The Scientific World Journal, Vol. 2014, Article ID 746260, 8 pages, 2014.

[191] G. Wang, L. Guo, H. Duan, H. Wang, L. Liu, and M. Shao, A hybrid meta-heuristic DE/CS algorithm for UCAV three dimension path planning, The Scientific World Journal, Vol.2012, Article ID 583973, 11 pages, 2012.

[192] H. Qiu, H. Duan Receding horizon control for multiple UAV formation flight based on modified brain storm optimization, Nonlinear Dyn, Vol. 78, pp. 1973-1988, 2014.

[193] H. Qiu, H. Duan, Y. Shi, A decoupling receding horizon search approach to agent routing andoptical sensor tasking based on brain storm optimization, Optik, Vol. 126 pp. 690-696, 2015.

[194] H. Duan, L. Huang, Imperialist competitive algorithm optimized artificial neural networks for UCAV global path planning, Neurocomputing, Vol 125, pp. 166-171, 2014.

[195] A. Alihodzic, E. Tuba, R. Capor-Hrosik, E. Dolicanin, M. Tuba, Unmanned aerial vehicle path planning problem by adjusted elephant herding optimization, in Proceedings of the 25th Telecommunications Forum (TELFOR), pp. 804-807, 2017.

[196] M. A. Contreras-Cruz, V. Ayala-Ramirez, U. H. Hernandez-Belmonte, Mobile robot path planning using artificial bee colony and evolutionary programming, Applied Soft Computing, vol. 30, pp. 319-328, 2015.

[197] Z. Nie, X. Yang, S. Gao, Y. Zheng, J. Wang, and Z. Wang, Research on autonomous moving robot path planning based on improved particle swarm optimization, in IEEE Congress on Evolutionary Computation (CEC), pp. 2532-2536, IEEE, 2016.