We are IntechOpen,
the world's leading publisher of
Open Access books
Built by scientists, for scientists

6,900
Open access books available

186,000
International authors and editors

200M
Downloads

154
Countries delivered to

Our authors are among the
TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS
BOOK CITATION INDEX
INDEXED

WEB OF SCIENCE™
Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

16

Heuristic Optimization Algorithms in Robotics

Pakize Erdogmus[1] and Metin Toz[2]
[1]*Duzce University, Engineering Faculty, Computer Engineering Department*
[2]*Duzce University, Technical Education Faculty, Computer Education Department, Duzce Turkey*

1. Introduction

Today, Robotic is an essential technology from the entertainment to the industry. Thousands of articles have been published on Robotic. There are various types of robots such as parallel robots, industrial robots, mobile robots, autonomous mobile robots, health-care robots, military robots, entertainment robots, nano robots and swarm robots. So, this variety brings a lot of problems in Robotic. Inverse kinematic for serial robots, forward kinematic for parallel robots, path planning for mobile robots and trajectory planning for industrial robots are some of the problems in Robotic studied a lot. Some of the problems are solved easily with some mathematical equations such as forward kinematic problem for serial robots and inverse kinematic problem for parallel robots. But the problems consisting of nonlinear equations and higher order terms can't be solved exactly with the classical methods. The forward kinematics problem of the 6 degrees of freedom (DOF) 6x6 type of Stewart Platform can be given as an example to such problems. It has been shown that there are 40 distinct solutions for this problem (Raghavan, 1993). Some of the unsolved problems with classical methods are optimization problems and heuristic optimization techniques are an alternative way for the solution of such problems. The heuristic optimization techniques produce good solutions for the higher order nonlinear problems in an acceptable solution time, when the problems aren't solved with the classical methods (Lee & El-Sharkawi, 2008).
In this chapter, first in Section 2, optimization is shortly described. The introduction and some well-known heuristic algorithms including, Genetic Algorithms(GA), Simulated Annealing(SA), Particle Swarm Optimization(PSO) and Gravitational Search Algorithm(GSA) are reviewed in Section 3. In Section 4, two well-known optimization problems in Robotic are solved with GA and PSO.

2. Optimization

Optimization is of great importance for the engineers, scientists and managers and it is an important part of the design process for all disciplines. The optimal design of a machine, the minimum path for a mobile robot and the optimal placement of a foundation are all optimization problems.
A constrained optimization problem has three main elements; design variables, constraints and objective function/functions. Design variables are independent variables of the

objective function and can take continuous or discrete values. The ranges of these variables are given for the problems. Constraints are the functions of design variables and limit the search space. The objective function is the main function dependent on the design variables. If there is more than one objective function, the problem is called multi-objective optimization problem.
Solving an optimization problem means finding the values of the design variables which minimize the objective function within given constraints. So if the objective function is a maximization problem, it is converted to minimization problem multiplying by -1 as seen in the Figure 1. The mathematical model of an optimization problem is given by equation 1.

$$\text{Minimize } f_{obj}(x_1,x_2,\ldots x_n)\text{, Subject to } \begin{array}{ll} g_i(x_1,x_2,\ldots x_n,a) \le 0, & i=1,2,\ldots,l \\ h_j(x_1,x_2,\ldots x_n, b)=0, & j=1,2,\ldots,m \\ x_k^l \le x_k \le x_k^u, & k=1,2,\ldots n \end{array} \quad (1)$$

Where, $\mathbf{x_1, x_2, \ldots x_n}$ are design variables, **n** is the number of design variables, **l** is the number of inequality constraints and **m** is the number of equality constraints, **a** and **b** are constant values. $\mathbf{x_k^l}$ and $\mathbf{x_k^u}$ are respectively lower and upper bounds of the design variables.

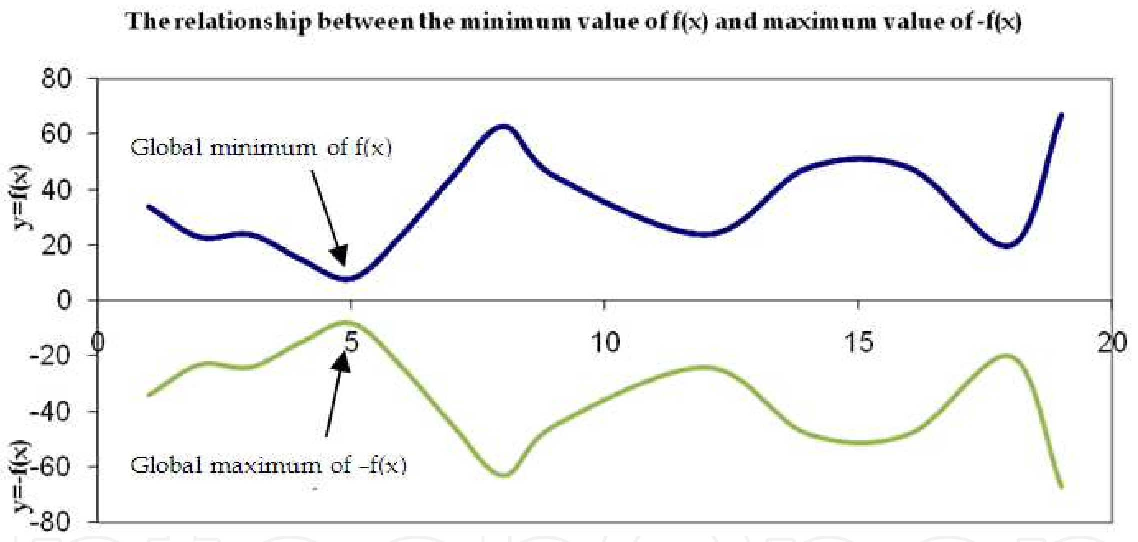


Fig. 1. An objective function conversion to minimization

There are a lot of applications in decision science, engineering, and operations research which can be formulated as constrained continuous optimization problems. These applications include path planning for mobile robots, trajectory planning for robot manipulators, engineering design and computer-aided-design (CAD). Optimal or good solutions to these applications are very important for the system performance, such as low-cost implementation and maintenance, fast execution and robust operation (Wang, 2001).
There are different methods for the solution of the optimization problems. Exact methods and heuristics are two main solution methods. Exact methods find certain solutions to a given problem. But, if the problem size increases, the solution time of the exact methods is unacceptable, because of the fact that solution time increases exponentially when the problem size increases. So, using the heuristics methods for the solution of optimization problems are more practical. (Rashedi et al., 2009). On the other hand, in last decades there

has been a great deal of interest on the applications of heuristic search algorithms to solve the such kind of problems.
The main difficulty encountered in the solution of the optimization problem is the local minimums. If there are a lot of local minimums, then both exact methods and heuristics can be trapped in to the local minimums. Since most of the heuristic algorithms developed a strategy to avoid the local minimums, they have been quite popular recently. The local minimums and global minimum of an objective function with one design variable are seen in the Figure 2.

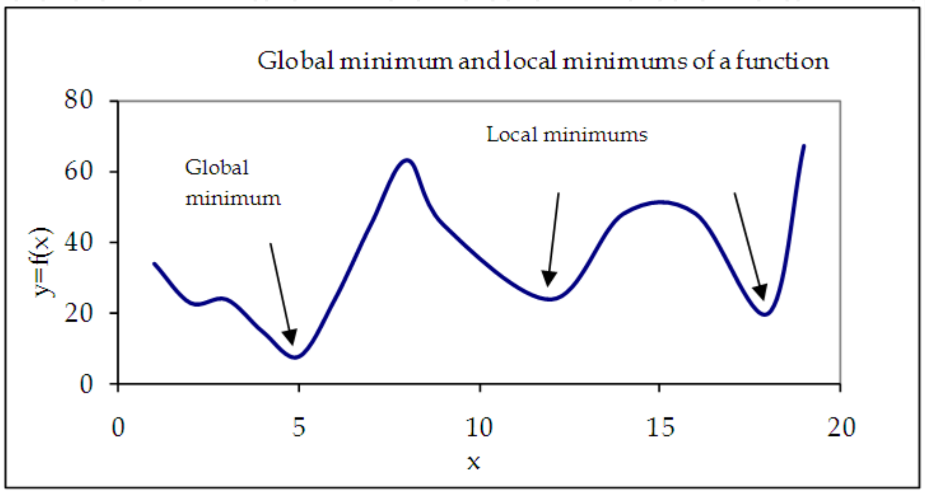


Fig. 2. An objective function with local minimums

3. Heuristic algorithms

Optimization studies drawn attention in 1960’s. It was developed a lot of algorithm based on the more sophisticated mathematical background. These algorithms were called exact methods. But, exact methods produced certain solutions only for the limited scope of application. As a result, attention turned to heuristic algorithms. (Fisher et al., 1998). Heuristics algorithms try to find acceptable solutions to the problems using some heuristic knowledge and most of them simulate real life. They use not only pure mathematics, but also algorithms using with basic formulations. While the most important property of heuristic algorithms is that they are designed for the unconstrained optimization problems, they can also be adapted to the constrained optimization problems. The algorithms are designed to find the minimum value of the objective function within the bounds of the constraints . If a solution doesn’t satisfy the constraints, this solution is not acceptable, even if the value of the objective function is minimum. So, if there are constraints in a minimization problem, penalty function is added to objective function. The total function is called as fitness function. Namely, if constraints are in feasible region, then there is no penalty and penalty function is equal to zero. If the constraints are not in feasible region, the fitness function is penalized by penalty function.
Heuristic algorithms don’t guarantee finding the optimal solutions. They try to find acceptable solutions near to optimum in a reasonable time. These algorithms were studied for both discrete and continuous optimization problems since the 1970's. Researchers have tried to develop adaptive and hybrid heuristics algorithms. By this aim, tens of algorithms were developed in last decade. Heuristic algorithms can be classified as single solution

algorithms and population based algorithms. The first category gathers Local Search (LS), Greedy Heuristic (GH) (Feo et al., 1995), Simulated Annealing (SA) (Kirkpatrick et al., 1983), Tabu Search (TS) (Laguna, 1994) and Iterated Local Search (ILS) (Glover & Garry, 2002). The second category, which is more and more studied, groups evolutionary algorithms such as Genetic Algorithms (GA)(Goldberg, 1989), Ant Colony Optimization (ACO) (Dorigo et al, 1996), Artificial Bee Colony (ABC)(Basturk & Karaboga, 2006), Scatter Search (SS) (Marti et al., 2006), Immune Systems (IS) (Farmer et al., 1986), Differential Evolution Algorithms (DEA) (Storn, 1996), Particle Swarm Optimization (PSO) (Keneddy & Eberhart, 1995), Harmony Search (HS) (Geem & Kim, 2001) and Gravitational Search Algorithm (GSA) (Rashedi et al., 2009). Evolutionary algorithms simulate natural phenomena or social behaviors of animals. So, the algorithms can be subcategorized to these criteria. While SA, GSA, HS simulates natural phenomena, ACO, ABC, PSO simulates the social behaviors of animals. The taxonomy of global optimization algorithms can be found in the related literature (Weise, 2008).

During the last years, hybrid optimization approaches have been popular. Firstly, cooperations have been realized between several heuristic algorithms. Later, cooperations between heuristic algorithms and exact methods have been realized. Since they gather the advantages of both types of algorithm, hybrid algorithms give more satisfied results. (Jourdan et al., 2009).

In single solution heuristic algorithms, the algorithm starts with an initial solution called current solution. This solution can be created randomly and consists of the values of design variables of the optimization problems. The objective function value is calculated with these initial values. If the initial solution satisfies the constraints, fitness function will be the same with the objective function. The second solution, called candidate solution, is created with random values close to the initial solution. Between the two fitness functions, the one which has a minimum value is selected as candidate solution. As long as the iteration proceeds, it is expected that algorithm converges to the global minimum value. But algorithms can be trapped to the local minimums. Heuristic algorithms develop different strategies to avoid to get trapped the local minimums. So, exploration and exploitation are of great importance for finding the global minimum value with the heuristics algorithms. In the first iterations, the algorithm searches the global solution space with big steps in order not to get trapped the local minimums. This is called exploration. In the next iterations, the algorithm searches the solution space with small steps in order to find best minimum. This is called exploitation. It is desired to have a balance between exploration and exploitation in the heuristic algorithms.

3.1 Genetic algorithms

GA was developed in Michigan University by Holland and later it got popularity with the efforts of Goldberg. GA is an adaptive global search method that mimics the metaphor of natural biological evolution. It operates on a population of potential solutions by applying the principle of survival of the fittest to achieve an optimal solution.

In the literature, GA is one of the most applied heuristic optimization algorithms. GA has been applied successfully to a huge variety of optimization problems, nearly in all disciplines, such as materials science, aircraft applications, chemistry, construction, seismology, medicine and web applications. GA has also been applied to the problems in Robotic such as the solution of multimodal inverse kinematics problem of industrial robots

(Kalra et al., 2006), trajectory generation for non-holonomic mobile manipulators (Chen & Zalzala, 1997), generation of walking periodic motions for a biped robot (Selene et al, 2011), the solution of the forward kinematics of 3RPR planar parallel manipulator(Chandra & Rolland, 2011), kinematic design optimization of a parallel ankle rehabilitation robot (Kumar et al., 2011).
GA starts to run with a lot of possible solutions according to the initial population which are randomly prepared. Each individual is encoded as fixed-length binary string, character-based or real-valued encodings. Encoding is an analogy with an actual chromosome (Lee & El-Sharkawi, 2008). A binary chromosome represents the design variables with a string containing 0s and 1s. Design variables are converted to binary arrays with a precision p. The bit number is calculated by equation 2 (Lin & Hajela, 1992).

$$B.\,N \geq \log_2(((x_k^u - X_k^l)/p)+1) \tag{2}$$

Where, **p** is a precision selected by the program designer. $\mathbf{x_k^u}$ and $\mathbf{x_k^l}$ are respectively upper and lower bounds of the design variables **B.N** is the number of bits representing design variables. If the number of bits is 4, the binary values of the design variables are represented as 0000, 0001, 0010, …, 1111.
Initial population is generated randomly after coding the variables. Initial population is represented as a matrix. Typical population size varies between 20 and 300. Each row of the population matrix is called as an individual.
Strings are evaluated through iterations, called generations. During each generation, the strings are evaluated using fitness function. Fitness function measures and evaluates the coded variables in order to select the best fitness strings (Saruhan, 2006). Then, it tries to find optimum solutions by using genetic operators. By this way, the best solutions are selected and worsts are eliminated. GA runs according to unconstrained optimization procedure. So, the constrained continuous optimization problems are transformed into unconstrained continuous optimization problem by penalizing the objective function value with the quadratic penalty function. The total function is called fitness function.
Fitness function (FF) values are calculated for each individual as it is seen in equation 3, 4 and 5. In a minimization problem, penalty function is added to objective function. The fitness function (FF), is the sum of objective function (OF) and the penalty function (PF) consisting constraint functions (CF). After elitism, selection, crossover, and mutation are operated, a new population is generated according to the fitness function values. The future of population depends on the evolutionary rules.

$$OF = f_{obj}(x_1, x_2, \ldots x_n) \tag{3}$$

$$PF = \sum_{i=1}^{l} R_i(\max[0, g_i(x_1, x_2, \ldots x_n, a)])^2 + \sum_{j=1}^{m} Rj(\max[0, \left|h_j(x_1, x_2, \ldots x_n, b)\right| - tol])^2 \tag{4}$$

$$FF = OF + PF \tag{5}$$

In the above equations, **a** and **b** are constant values**,** **l** is the number of inequality constraints, **m** is the number of equality constraints, **n** is the number of variables and **tol** is the tolerance value, for the equality constraints. If $\mathbf{h_j}$'s value is smaller than the tolerance value, it is accepted that equality constraints have been satisfied. If the problem variables satisfy the

constraints, g_i and (h_j –tol) values will be negative and PF will be zero. $\mathbf{R_i}$ and $\mathbf{R_j}$ are the penalty coefficients and these values affect the algorithm performance directly.
Selection is to choose the individuals for the generation based on the principle of survival of the best according to the Darwin's theory. The main idea in selection is to create parents from the chromosomes having better fitness function values. Namely the parents are selected from the chromosomes of the first population as directly proportional to their fitness function values. The better fitness function value of a chromosome, the more chance to be a parent. With the selection method, the individuals having worse fitness values disappear in the next generations. The most popular selection methods are roulette wheel and tournament rank. The successful individuals are called parents.
After selection, crossover, which represents mating of two individuals, is applied to the parents. The parents in GA create two children with crossover. There are various kinds of crossover process such as one point crossover, multiple point crossovers and uniform crossover. The crossover operator is applied with a certain probability, usually in the range 0.5-1. This operator allows the algorithm to search the solution space. Namely, exploration is performed in GA with crossover operator. In a basic crossover, the children called offspring are created from mother and father genes with a given crossover probability. Mutation is an another operation in GA and some gens are changed with this operation. But this is not common. Generally mutation rate of binary encoding is specified smaller than 0.1. In contrast to the crossover, mutation is used for the exploitation of the solution space. The last operation is the elitism. It transfers the best individual to the next generation. The evaluation of the previous population with the operators stated above, the new population is generated till the number of generation. Fitness function values are calculated in each new population and the best resulted ones are paid attention among these values. Until the stopping criteria are obtained, this process is repeated iteratively. The stopping criteria may be the running time of the algorithm, the number of generation and for fitness functions giving the same best possible values in a specified time. The pseudo codes of GA are given in Figure 3.

```
init_Popu← Produce initial solution ()
(while stopping criteria not true do)
    for i=1 to generation_number
        calculate fitness function values of init_popu
        choose parents with roulette wheel
        (Parent 1 and Parent 2 are chosen)
        Elitism(Select the best ones in the population)
        Crossover(Create two children for each parent)
        Mutation(Gens are changed with mutation_rate)
        The new ofspring is created
      Next
```

Fig. 3. The pseudo codes of GA

3.2 Simulated annealing

SA uses an analogy of physical annealing process of finding low energy states of solids and uses global optimization method. SA is inspired by Metropolis Algorithm and this approach was firstly submitted to an optimization by Kirkpatrick and his friends in 1983 (Kirkpatrick

et al, 1983). Because of the fact that SA is an effective algorithm and it is easy to implement for the difficult engineering optimization problems, it is also popular for last studies in several different engineering areas. SA is applied to various optimization problem in Robotics such as path planning problem solution(Gao & Tian,2007), the generation of the assembly sequences in robotic assembly(Hong & Cho, 1999), trajectory optimization of advanced launch system (Karsli & Tekinalp, 2005), optimal robot arm PID control, the placement of serial robot manipulator and collision avoidance planning in multi-robot.
SA combines local search and Metropolis Algorithm. Algorithm starts with a current solution at initial temperature. At each temperature, algorithm iterates n times, defined by the program designer. In the iterations for each temperature, a candidate solution that is close to the current solution is produced randomly. If candidate solution is better than the current solution, the candidate solution is replaced with the current solution. Otherwise, the candidate solution is not rejected at once. The random number is produced between 0-1 and compared with the acceptance probability P. P is an exponential function as given by equation 6. If produced random number is smaller than P, the worse solution is accepted as current solution for exploration. Search process progresses until stopping criteria is satisfied. Maximum run time, maximum iteration number, last temperature e.g. may be the stopping criteria.

$$P = e^{-\left(\frac{f(x\_cand) - f(x\_curr)}{T}\right)} \tag{6}$$

For an object function with one design variable, **x_cand** is candidate design variable, **x_curr** is current design variable and **f** is object function, **T** is temperature. At high temperatures, P is close to one. Since the most random number is smaller than one, the possibility of bad solution's acceptance is high for exploration in the first steps of the algorithm. T is decreased along the search process. At low temperatures, P is close to zero. Since the random numbers are bigger than zero, the possibility of bad solution's acceptance approach to zero. When P is equal to zero, the process converges to local search method for exploitation. The pseudo code of SA is given in Figure 4.

```
x_curr← Produce initial solution()
T←T0
while stopping criteria not true  do
    for i=1 to n
        x_cand←Produce a random solution
        if f(x_cand) < f(x_curr) then
            x_ curr ←x_cand
        else
            'Metropolis Algorithm
            x Produce a random number between (0,1)
            if   x < p ( T, x_curr, x_cand) then
                x_curr ← x_cand
            end if
        end if
    end
    update(T)
end while
```

Fig. 4. The pseudo codes of SA Algorithm

Annealing process simulates optimization. Annealing is realized slowly in order to keep the system of the melt in a thermodynamic equilibrium. Convergence to the optimal solution is controlled by the annealing process.
Proper cooling scheme is important for the performance of SA. The proper annealing process is related with the initial temperature, iteration for each temperature, temperature decrement coefficient and stopping criteria. All these criteria can be found in related article (Blum & Roli, 2001). In general, the temperature is updated according to equation 7.

$$T_{k+1} = \alpha T_k \tag{7}$$

Where, $\mathbf{T_{k+1}}$ is the next temperature, $\mathbf{T_k}$ is the current temperature and α is the temperature decrement coefficient, generally selected between 0.80 and 0.99.
Even if SA tries to find global minimum, it can be trapped by local minima. So, in order to overcome this drawback and develop the performance of SA, researchers have developed adaptive or hybridized SA with other heuristics, such as fuzzy logic, genetic algorithms, support vector machines, and distributed/parallel algorithms.

3.3 Particle swarm optimization

PSO was introduced by James Kennedy and Russell Eberhart in 1995 as an evolutionary computation technique (Kennedy & Eberhart, 1995) inspired by the metaphor of social interaction observed among insects or animals. It is simpler than GA because PSO has no evolution operators such as crossover and mutation (Lazinca, 2009).
PSO have many advantages such as simplicity, rapid convergence, a few parameters to be adjusted and no operators (Li & Xiao; 2008). So, PSO is used for solving discrete and continuous problems. It was applied to a wide range of applications such as function optimization, Electrical power system applications, neural network training, task assignment and scheduling problems in operations research, fuzzy system control and pattern identification. PSO was also applied to a lot of different Robotic applications. PSO was applied mobile robot navigation (Gueaieb & Miah, 2008), Robot Path Planning (Zargar & Javadi, 2009) and Swarm Robotic Applications and Robot Manipulators applications.
PSO algorithm is initialized with a group of particles. Each particle is a position vector in the search space and its dimension represents the number of design variables. PSO is started with random initial positions and velocities belong to each particle. For each particle, fitness function's value is computed. Global and local best values are updated. Local best value is the best value of the current particle found so far and the global best value is the best value of all local bests found so far. Velocity and positions are updated according to the global best and local bests. Each particle flies through the search space, according to the local and global best values. Along the iterations, particles converge to the global best. The convergence of the particles to the best solution shows the PSO performance. The rate of position change of the *i*th particle is given by its velocity. In the literature there are different velocity formulas. The first one proposed by Keneddy is given by equation 9. Particles' velocity and positions are updated according to equation 8 and 9.

$$x_i^{k+1} = x_i^k + v_i^{k+1} \tag{8}$$

$$v_i^{k+1} = v_i^k + \varphi_1 \mathrm{rand}()(p_{besti}{}^k - x_i^k) + \varphi_2 \mathrm{rand}()(g_{best} - x_i^k)) \tag{9}$$

Where, **k** is the iteration number, **rand** is a random number, **pbest$_i^k$** is the best value of *i*th particle and **gbest** is the global best value of all particles. **x$_i^k$** and **x$_i^{k+1}$** are respectively the current position and the next position of the *i*th particle. **v$_i^k$** and **v$_i^{k+1}$** are respectively the current velocity and the next velocity of the *i*th particle. The next position of a particle is specified by its current position and its velocity.
PSO simulates social swarm's behaviour. The velocity formula consists of three important components. First part presents the past velocity. The particle generally continues in the same direction. This is called habit. The second part **$\varphi_1$rand()(pbest$_i^k$-x$_i^k$)** presents private thinking. This part is also called self-knowledge and the last part **$\varphi_2$rand()(gbest-x$_i^k$)** presents the social part of the particle swarm(Kennedy, 1997). When implementing the particle swarm algorithm, some considerations must be paid attention in order to facilitate the convergence and prevent going to infinity of the swarm. These considerations are limiting the maximum velocity, selecting acceleration constants, the constriction factor, or the inertia constant (Valle et al., 2008).
The velocity and position formula with constriction factor (K) has been introduced by Clerc(Clerc, 1999) and Eberhart and Shi (Eberhart & Shi, 2001) as given by equation 10,11 and 12.

$$v_i^{k+1} = K(v_i^k + \varphi_1 \text{rand}()(p_{besti}{}^k - x_i^k) + \varphi_2 \text{rand}()(g_{best} - x_i^k)) \tag{10}$$

$$x_i^{k+1} = x_i^k + v_i^{k+1} \tag{11}$$

$$K = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|}, \quad \varphi = \varphi_1 + \varphi_2, \ \varphi > 4 \tag{12}$$

Where, **K** is the constriction factor, φ_1 and φ_2 represent the cognitive and social parameters, respectively. **rand** is uniformly distributed random number. φ is set to 4.1. The constriction factor produces a damping effect on the amplitude of an individual particle's oscillations, and as a result, the particle will converge over time. The pseudo code of PSO is given in Figure 5.

```
Generate initial P particle swarm's random positions and velocities
do
  for i=1:P
    Evaluate fitness function for each particle
    If fitness(Pi) <fitness(Gbest) then Gbest=Pi
    If fitness(Pi) <fitness(Pbest(i)) then Pbest(i)=Pi
    Update velocity of the particle i
    Update position of the particle i
  end
Until stopping criteria is not true
```

Fig. 5. The pseudo codes of PSO

3.4 Gravitational search algorithm

GSA was introduced by Esmat Rashedi and his friends in 2009 as an evolutionary algorithm. It is one of the recent optimization techniques inspired by the law of gravity. GSA was

originally designed for solving continuous problems. Since it is quite newly developed algorithm, it hasn't been applied to a lot of area. GSA was applied to filter modelling (Rashedi et al., 2011), Post-Outage Bus Voltage Magnitude Calculations (Ceylan et al., 2010), clustering (Yin et al., 2011), Scheduling in Grid Computing Systems (Barzegar et al., 2009).
GSA algorithm is based on the Newtonian gravity law: ''Every particle in the universe attracts every other particle with a force that is directly proportional to the product of their masses and inversely proportional to the square of the distance between them" (Rashedi et al., 2009).
In GSA, particles are considered as objects and their performances are measured by their masses. GSA is based on the two important formulas about Newton Gravity Laws given by the equation 13 and 14. The first one is as given by the equation 13. This equation is the gravitational force equation between the two particles, which is directly proportional to their masses and inversely proportional to the square of distance between them. But in GSA instead of the square of the distance, only the distance is used. The second one is the equation of accelaration of a particle when a force is applied to it. It is given by equation 14.

$$F = G\frac{M_1 M_2}{R^2} \tag{13}$$

$$a = \frac{F}{M} \tag{14}$$

G is gravitational constant, **M_1** and **M_2** are masses and **R** is distance, **F** is gravitational force, **a** is acceleration. Based on these formulas, the heavier object with more gravity force attracts the other objects as it is seen in Figure 6.

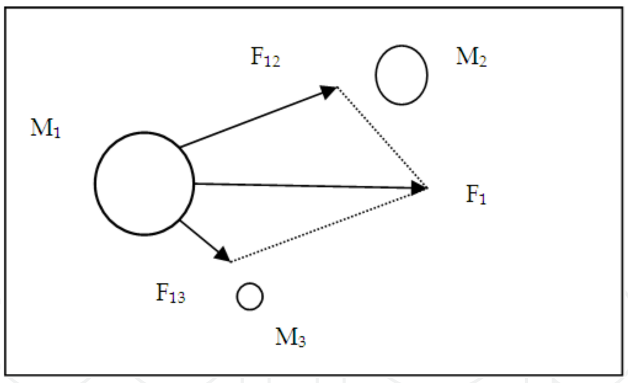


Fig. 6. The Newton Gravitational Force Representation

GSA algorithm initializes the position of the masses(X) given by equation 15.

$$X_i=(x_i^1, x_i^2, \dots , x_i^d, , x_i^n,) \quad i=1,2,3,\dots,N \tag{15}$$

Where, **X_i** is the position of ***i***th mass. **n** gives the dimension of the masses. N gives the number of particles. **x_i^d** represents ***i***th particle position in the ***d***th dimension.
The first velocities of the masses are accepted as zero. Until the stopping criterion (maximum iteration), algorithm evaluates the fitness functions of the objects. After the best and worst values are found for the current iteration, the masses of the particles are updated according to the equation 16, 17 and 18.

$$M_{ai} = M_{pi} = M_{ii} = M_i,\ i= 1, 2, \cdots ,N \tag{16}$$

$$m_i(t) = \frac{\text{fitness}_i(t) - \text{worst}(t)}{\text{best}(t) - \text{worst}(t)} \tag{17}$$

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)} \tag{18}$$

In the equation 17, ***fitness$_i$(t)*** represents the fitness value of the ***i***th mass at time t, ***worst(t)*** is the maximum value of all the fitness values and ***best(t)*** is the minimum value of the all fitness values for a minimization problem. For maximization problems, the opposite of this expression is applied. In GSA, active gravitational mass **Ma**, passive gravitational mass **Mp** and inertial mass **Mi** are accepted as they are equal to each other.
Gravitiational constant which is propotional with the time is also updated according to the iteration number given by the equation 19.

$$G(t)=G_0 e^{-\alpha \frac{t}{T}} \tag{19}$$

Where, **α** is a user specified constant, **T** is the total number of iterations, and **t** is the current iteration. This equation is similar to the SA acceptance probability equation given by the equation 6. According to the formula the gravity force is decreasing by the time. The graviational force and total forces are updated as follows.

$$F_{ij}^d(t) = G(t)\frac{M_{pi}(t)M_{aj}(t)}{R_{ij}(t) + \varepsilon}(x_j^d(t) - x_i^d(t)) \tag{20}$$

$$F_i^d(t) = \sum_{j \in Kbest, j \neq i}^{N} \text{rand}_j F_{ij}^d(t) \tag{21}$$

F$_{ij}^d$(t) is the gravity force acting on mass ***i*** from mass ***j*** at time t, **G(t)** is the gravitational constant at time t, **Mpi** is the passive gravitational mass of object i, **Maj** is the active gravitational mass of object j, ε is a small constant, **R$_{ij}$** is the Euclidean distance between two objects i and j . **F$_i^d$(t)** is the force that acts on particle ***i*** in dimension ***d***, **rand**$_j$ is a random number between 0 and 1, **Kbest** is the set of first K objects with the best fitness value and biggest mass.The acceleration of an object ***i*** in ***d***th dimension is as follows.

$$a_i^d(t) = \frac{F_i^d(t)}{M_i(t)} \tag{22}$$

The new positions and velocities of the particles are calculated by the equation 23 and 24.

$$v_i^d(t+1) = \text{rand}_i \times v_i^d(t) + a_i^d(t) \tag{23}$$

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t) \tag{24}$$

According to the law of motion, particles try to move towards the heavier objects. The heavier masses represent good solutions and they move slowly for the exploitation. The pseudo code of GSA is given in Figure 7.

```
Initialize the gravitational constant(G) and the positions of N masses($X_i$)
$x_i=(x_i^1, x_i^2, \ldots, x_i^d, , x_i^n,)$ i=1,2,3,…,N
for i=1:max_iteration
    Decrease gravitational constant with the equation 19.
    Evaluate the fitness of each object.
    Calculate the masses with the equation 17, 18.
    Compute the total forces with the equation 21.
    Find the accelerations with the equation 22.
    Compute the velocities with the equation 23.
    Compute the positions with the equation 24.
End
```

Fig. 7. The pseudo codes of GSA

4. PSO and GA applications in robotic

In this section some well-known problems in Robotic were solved with heuristic search algorithms. The first problem is the inverse kinematics problem for a nearly PUMA robot with 6 DOF, and the second problem is the path planning problem for mobile robots.

4.1 The inverse kinematics problem for PUMA robot

Robot kinematics refers to robot motions without consideration of the forces. The forward kinematics is finding the robot's end effector's position and orientation using the given robot parameters and the joint's variables. Also, the inverse kinematics is about finding the robot's joints variables while the robot's parameters and the desired position of the end effector are given (Kucuk & Bingul, 2006). The solution of the robot's forward kinematics is always possible and unique. On the other hand, generally, the inverse kinematics problem has several solutions (Kucuk & Bingul, 2006). The most known and used method for solving robot kinematics problems is Denavit-Hartenberg method. In this method several parameters, namely DH parameters, are defined according to robot's parameters and the replaced coordinate systems as given in Figure 8. These parameters are used to define transformations between the coordinate systems using 4x4 transformation matrices. The multiplication of all the matrices of the robot gives the final robot's end effectors' position and orientation (Kucuk & Bingul, 2006).

The selected robot for this example is a nearly PUMA type robot. Different from the general PUMA robot, the robot used in this example was equipped with an offset wrist. The inverse kinematics of the robot equipped with such a wrist is quite complicated and can be computationally cumbersome (Kucuk & Bingul, 2005). In this example, this problem was solved with GA and PSO. The robot's link parameters and coordinate systems replacements can be seen in Figure 9. Furthermore, according to Figure 9, the DH parameters of the robot can be defined as given in Table 1. Using the DH parameters, the forward kinematics of the robot can be solved easily using robot's transformation matrices. A transformation matrix is a 4x4 matrix and has the form as it is seen in equation 25.

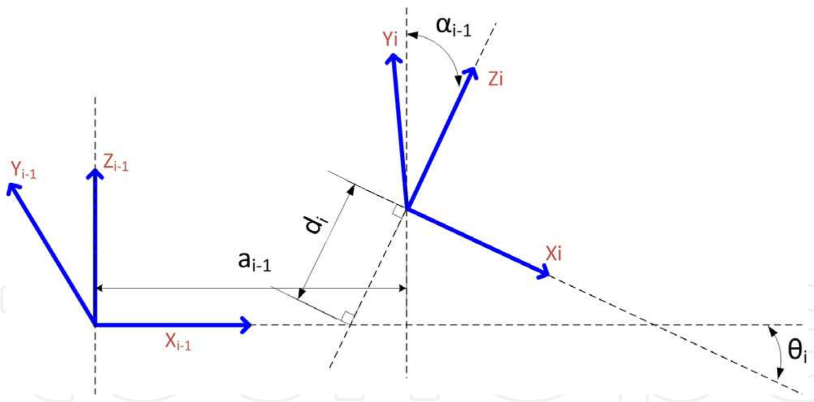


Fig. 8. DH parameters between two coordinate systems

$$T = \begin{bmatrix} & R & & P \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{25}$$

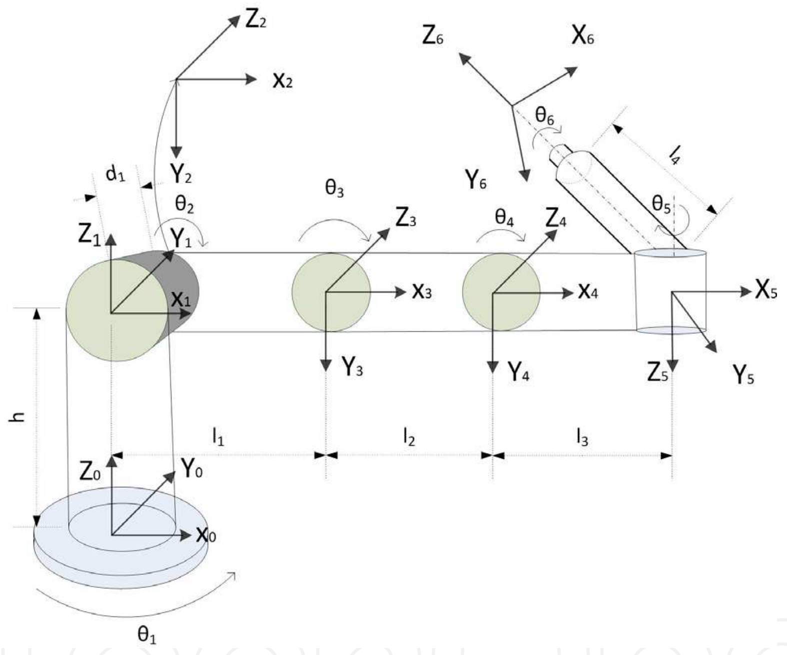


Fig. 9. The nearly Puma robot's link parameters and coordinate systems replacement

I	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	h	θ_1
2	$-\pi/2$	0	d_1	θ_2
3	0	l_1	0	θ_3
4	0	l_2	0	θ_4
5	$-\pi/2$	l_3	0	θ_5
6	$\pi/2$	0	l_4	θ_6

Table 1. The DH parameters of the robot

Where, T and R are 4x4 transformation and 3x3 rotation matrices between two consecutive links respectively while P is 3x1 position vector. Detailed information about defining this matrices and vectors can be found in (Kucuk & Bingul, 2006). The transformation matrices of the robot are as follows:

$$
{}_1^0T=\begin{bmatrix} C\theta_1 & -S\theta_1 & 0 & 0\\ S\theta_1 & C\theta_1 & 0 & 0\\ 0 & 0 & 1 & h\\ 0 & 0 & 0 & 1\end{bmatrix}\quad
{}_2^1T=\begin{bmatrix} C\theta_2 & -S\theta_2 & 0 & 0\\ 0 & 0 & 1 & d_1\\ -S\theta_2 & -C\theta_2 & 0 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\quad
{}_3^2T=\begin{bmatrix} C\theta_3 & -S\theta_3 & 0 & l_1\\ S\theta_3 & C\theta_3 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}
$$
$$
{}_4^3T=\begin{bmatrix} C\theta_4 & -S\theta_4 & 0 & l_2\\ S\theta_4 & C\theta_4 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\quad
{}_5^4T=\begin{bmatrix} C\theta_5 & -S\theta_5 & 0 & l_3\\ 0 & 0 & 1 & 0\\ -S\theta_5 & -C\theta_5 & 0 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\quad
{}_6^5T=\begin{bmatrix} C\theta_6 & -S\theta_6 & 0 & 0\\ 0 & 0 & -1 & -l_4\\ S\theta_6 & C\theta_6 & 0 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}
\tag{26}
$$

In the above matrices $C\theta_i$ and $S\theta_i$ (i=1,2,…,6) indicate $\cos\theta_i$ and $\sin\theta_i$ respectively. The forward kinematics of the robot is the forward multiplication of these matrices given by equation 27. The end effector's position in 3D space is defined with the last column of the result matrix.

$$
{}_6^0T={}_1^0T\,{}_2^1T\,{}_3^2T\,{}_4^3T\,{}_5^4T\,{}_6^5T=T=\begin{bmatrix} & R & & P\\ 0 & 0 & 0 & 1\end{bmatrix}\tag{27}
$$

In this example, the inverse kinematics of the robot was solved using the most general type of GA and PSO. Before starting to solve the problem with an optimization algorithm, it is needed to define the problem, exactly. The problem presented in this example is finding the robot joint angle values while the robot's parameters and the end effector's position are given. According to problem it can be seen that one individual of the population should have six joint's constraints. They are θ_1, θ_2, θ_3, θ_4, θ_5 and θ_6. So, an individual can be a 1x6 vector. Each component of the individual is for one of the joint's constraints. The range of the each constraint can be defined separately. However, for the sake of simplicity the ranges of the joints were defined between [-pi, pi]. A sample individual can be seen in Figure 10.

π	-π/4	π/2	0	π/5	0

Fig. 10. A sample individual

The most important part of the solution is the object function, since the fitness value of an individual can be defined with the help of this function. In the example, the object function was defined using the desired position values of the end effector of the robot and the obtained position values from individuals using forward kinematic equations of the robot. The Euler distance between the desired position of the end-effector and the results obtained from an individual can be defined as follows.

Let the desired position of the end effector be $P_d=\begin{bmatrix}x_d & y_d & z_d\end{bmatrix}$ and the obtained position knowledge using an individual be $P_i=\begin{bmatrix}x_i & y_i & z_i\end{bmatrix}$. The Euler distance between these two points in 3D space can be obtained as given in equation 28.

$$d_i = \sqrt{(x_d - x_i)^2 + (y_d - y_i)^2 + (z_d - z_i)^2} \quad (28)$$

The individual that offer the smallest distance is the most convenient candidate of the solution. The object function for the problem can be formulated like in equation 29.

$$O_i = p{d_i}^2 \quad (29)$$

O$_i$ is object function value, **p** is penalty constant and **d$_i$** is the Euler distance calculated by equation 28, for i’th individual.
The problem was solved firstly using GA. The features of the GA was defined as follows: The gene coding type was determined as real-value coding and the selection type was determined as roulette wheel technique while the mutation and the crossover types were determined as one-point crossover and one-point mutation. The GA starts with defining the needed parameters, like crossover and mutation rates, number of individuals, number of iterations (stopping criteria). Then the initial population is defined randomly in the ranges of the constraints and the object and the fitness values of this population are calculated. In each iteration, the elitism, crossover and mutation operations are performed. The new population is generated after these operations and it masked to comply with the range of the constraints. And finally, the new generation’s object and fitness values are calculated and the iterations proceeds until to reach the stopping criteria. In the end of the algorithm, the results are presented. The flowchart of the algorithm can be seen in Figure 11.

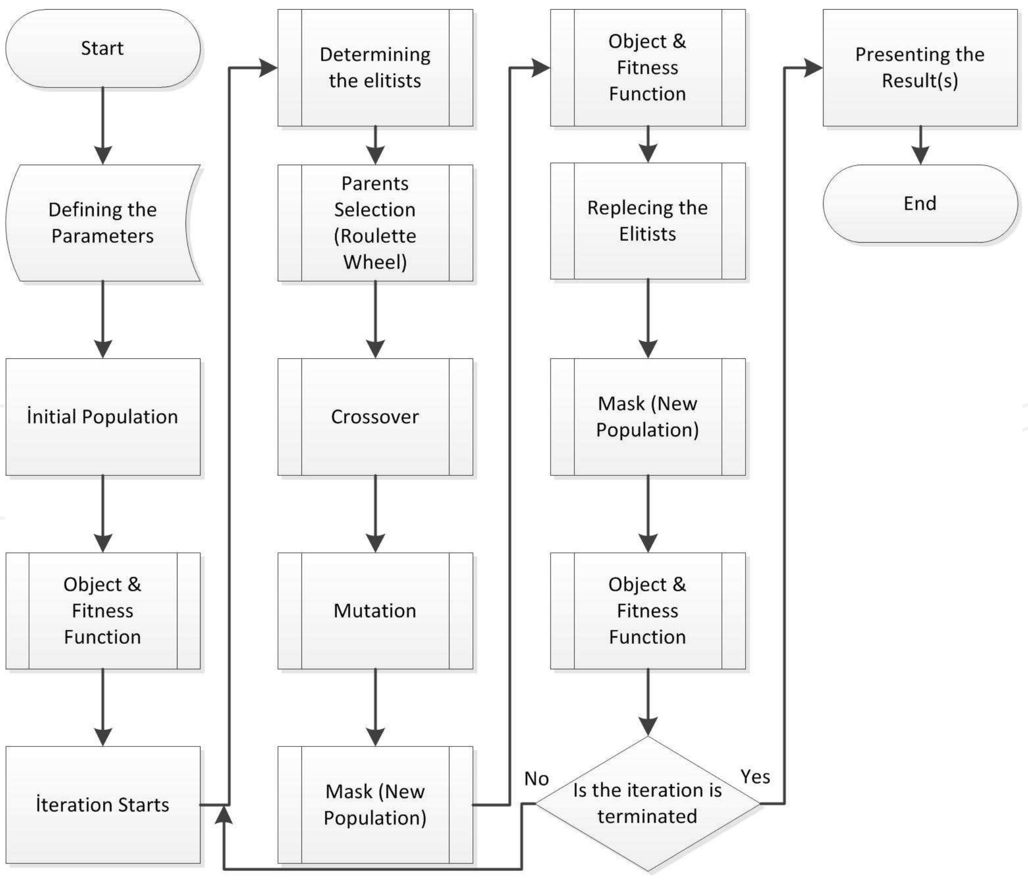


Fig. 11. The flowchart of the GA

The numerical example given below is simply the realization of the algorithm. The robot's first link was replaced in the [0 0 0] coordinates in 3D space and the other link's coordinates were defined according to these coordinates. The aim of the example is to find the robot's joint angles (constraints) that are needed to replace the end-effector of the robot in the goal coordinates. The parameters used in the algorithms were given in Table 2. According to defined parameters, the zero position of the robot's end effector was calculated using zero thetas as [30 15 30]. The solution is the smallest Euler distance between the goal coordinates, [10 5 60], and the each of the individuals. One of the real solutions for this goal coordinates is [0 -pi/2 pi/2 –pi/2 pi/2 0]. The algorithm continues until the stopping criterion is met. In this study, the generation number is the stopping criterion. After running the GA, the constraints' values were found as [-1.0278 -0.5997 -0.6820 -0.8940 0.9335 0] as radian. The coordinates of the end effector can be calculated using these constraints as [9.8143 4.9310 60.0652]. In Figure 12, the two solutions were depicted, the blue one is the sample solution, [0 -pi/2 pi/2 –pi/2 pi/2 0] and the red is the found solution [-1.4238 -1.9043 -0.7396 1.5808 0.9328 0]. The error between the desired and the found end effector's position was [0.1857 0.0690 -0.0652] units.

World Coordinates	Goal Point	Number of Population	Number of Generation	Mutation Rate	Crossover Rate	Penalty	Link Lengths	Zero Position Thetas
[0 0 0]	[10 5 60]	50	2000	0.1	0.9	10	h=30; d1=5; l1=l2=l3=l4=10	[0 0 0 0 0 0]

Table 2. The defined parameters for GA

In this example the most general type of GA was used to solve the problem. It is obvious that more acceptable solutions can be found using different techniques and/or object functions in GA.

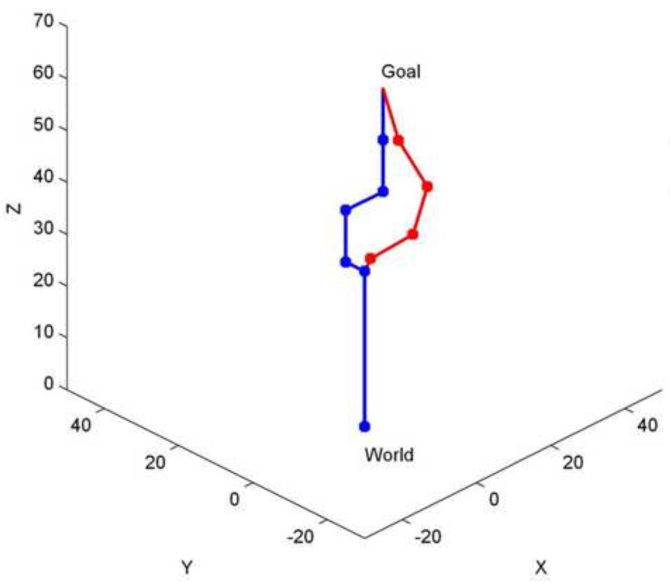


Fig. 12. The sample solution and the solution found with GA

The same problem was solved with PSO algorithm and the solutions were compared with GA. The flowchart of PSO algorithm that was used to solve the problem can be seen in Figure 13 and, the parameters defined for PSO can be seen in Table 3.

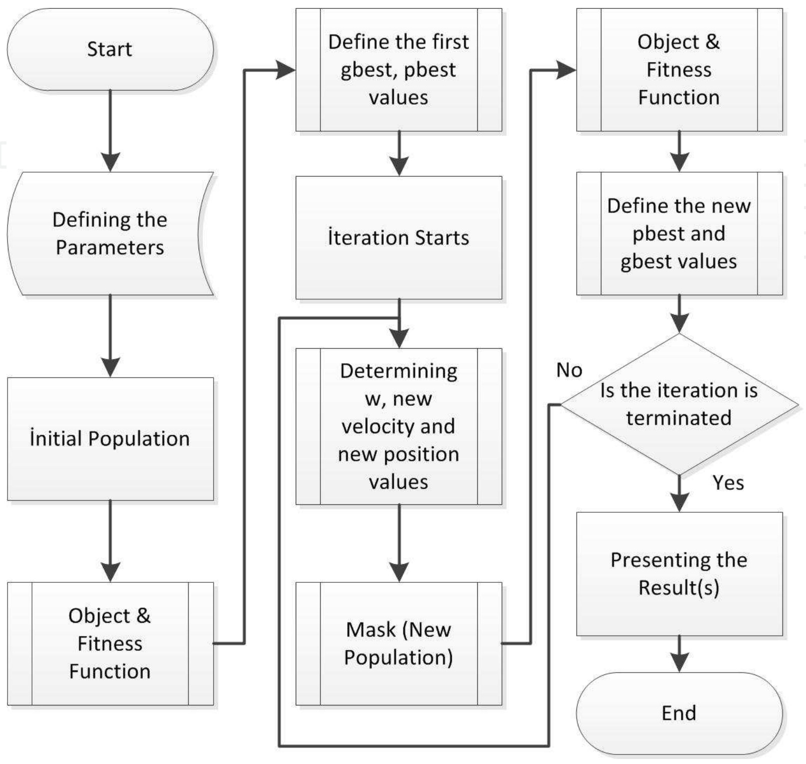

Start
Defining the Parameters
İnitial Population
Object & Fitness Function
Define the first gbest, pbest values
İteration Starts
Determining w, new velocity and new position values
Mask (New Population)
Object & Fitness Function
Define the new pbest and gbest values
Is the iteration is terminated
No
Yes
Presenting the Result(s)
End

Fig. 13. The flowchart of the PSO algorithm

World Coordinates	Goal Point	Number of Population	Number of Generation	Cognitive Component	Social Component	Penalty	Min. inertia weight	Max. inertia weight	Link Lengths	Zero Position Thetas
[0 0 0]	[10 5 60]	50	2000	2	2	10	0.4	0.4	h=30; d1=5; l1=l2=l3=l4=10	[0 0 0 0 0 0]

Table 3. The defined parameters for PSO

When the algorithm was run, the constraints were found in radian as [-1.3725 -2.5852 1.8546 -0.9442 0.9538 0] and the coordinates of the end effector calculated using these constraints were [9.9979 4.9941 60.0125]. According to these results it can be seen that PSO is found as it serves more convenient solution than GA's solution. In the Figure 14 the two solutions were depicted the blue one is the sample solution, [0 -pi/2 pi/2 –pi/2 pi/2 0] and the red is the found solution [-1.3725 -2.5852 1.8546 -0.9442 0.9538 0].

The convenience of an optimization algorithm to a problem differs from one problem to another and it can only be determined by doing several trials and comparisons. There is no specific algorithm to achieve the best solution for all optimization problems. The comparisons are mainly related to the algorithm's final object function values and the execution time of the algorithm. In the final part of the example a comparison between GA

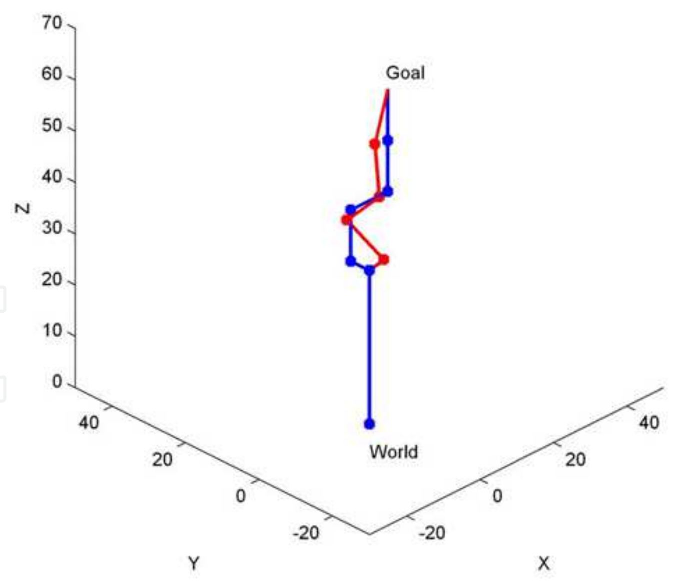


Fig. 14. The sample solution and the found solution using PSO

and PSO for the inverse kinematics problem of the 6 DOF nearly puma robot was presented. For each problem, 100 runs were simulated using the same parameters. The final object function's values and execution times of the two algorithms were presented in a comparative manner using graphs and Tables. The defined parameters for both of the algorithms can be seen in Table 4.

Parameters	**GA**	**PSO**
Number of Individuals	100	100
Number of Iterations	1000	1000
Penalty	10	10
World Coordinates	[0 0 0]	[0 0 0]
Goal Coordinates	[10 5 60]	[10 5 60]
Gene Code Type	Real Values	Real Values
Mutation Rate	0.2	***
Crossover Rate	0.9	***
Selection Type	R. Wheel	***
Mutation Type	One Point	***
c1	**	2
c2	**	2
Wmin	**	0.4
Wmax	**	0.9

Table 4. The parameters for GA and PSO for the inverse kinematic problem

Each algorithm was executed under the same condition and on the same computer, and according to the two criterions; the comparison was made between the two algorithms. The first criterion is about the algorithms' final object function values presented in Figure 15. The Figure indicates that both metaheuristic algorithms found the nearly optimum solutions to the problem. On the other hand it can be seen that PSO had more acceptable results than GA. In almost all executions PSO find the exact solutions. However, in 6 executions GA

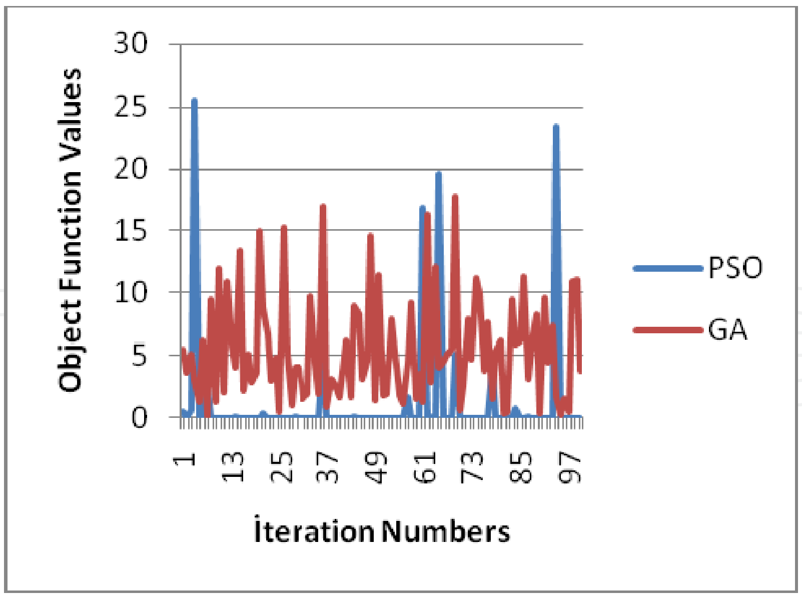


Fig. 15. The object function values of the two algorithms

outperforms PSO. The second criterion is for the execution times of the algorithms. The obtained results, in seconds, were presented in Figure 16. In the Figure, it is obvious that the elapsed time of the PSO is much less than the GA's. As a result, it can be said that PSO produced better results than GA in terms of both final object function's values and the execution times.

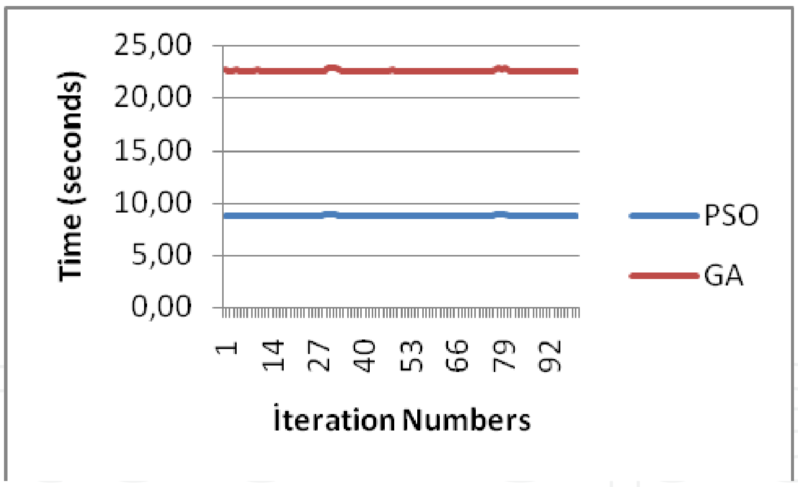


Fig. 16. The execution times of the two algorithms

4.2 Path planning problem for mobile robots

The second example is about path planning for mobile robots. Path planning is one of the most studied topics related to mobile robots. It works for finding the shortest path between the start and goal points while avoiding the collisions with any obstacles. In the example, this problem was solved using GA and PSO.

Path planning can be classified in two categories, global and local path planning. The global type path planning is made for a static and completely known environment while the local path planning is required if the environment is dynamic (Sedighi et al., 2004). In this example, global path planning was performed. The robot's environment was defined as a 10x10 unit size square in 2D coordinate system. In the environment, it was defined several obstacles that have different shapes. The problem is finding the shortest path between pre-defined start and goal points while the coordinates of these points and of the obstacle vertexes are known. A sample environment including obstacles and a sample path can be seen in the Figure 17. In the Figure there are six different shaped obstacles and the path is composed from four points. Two of these points are via points while the others are start and goal points. The mobile robot can reach the goal with tracing the path from the start point. The main object of this problem is to find the shortest path that is not crosses with any obstacles. According to these two constraints, the object function can be defined composing of two parts. The first one is the length of the path and the second one is the penalty function for collisions. The total length of the path can be calculated with the Euler distance between the points on the path if there are two via points that means the path is composed of three parts. The Euler distance between the two points can be simply calculated in 2D environment as in equation 30.

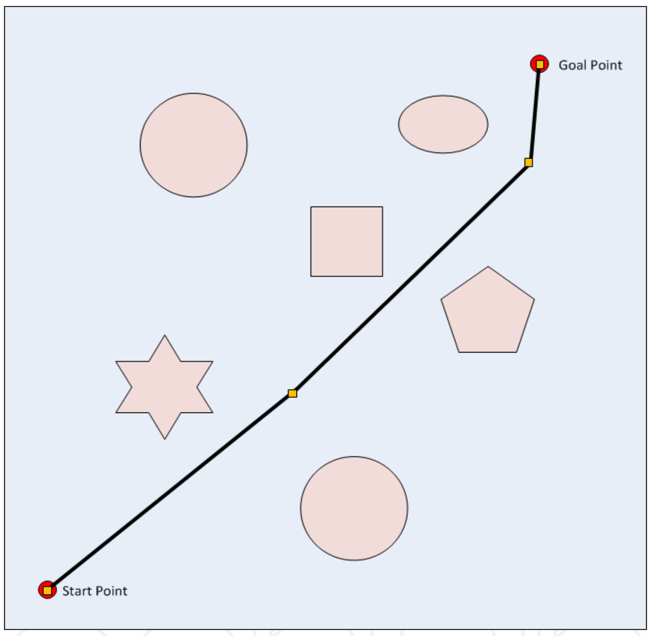


Fig. 17. A sample environment and a path

$$d_i = \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \quad i = 1,2,..,n \tag{30}$$

Where, **n** is the number of points on the path. The total length of the path is simply the sum of the distances, equation 31. In the equation, **L** is the total length of the path.

$$L = \sum_{i=1}^{n-1} d_i \quad i = 1,2,..,n \tag{31}$$

The second part of the object function determines the penalty for collision with the obstacles. Determining the collision with an obstacle is quite complicated. The obstacle avoidance technique, used in this study, can be briefly described using Figure 18.

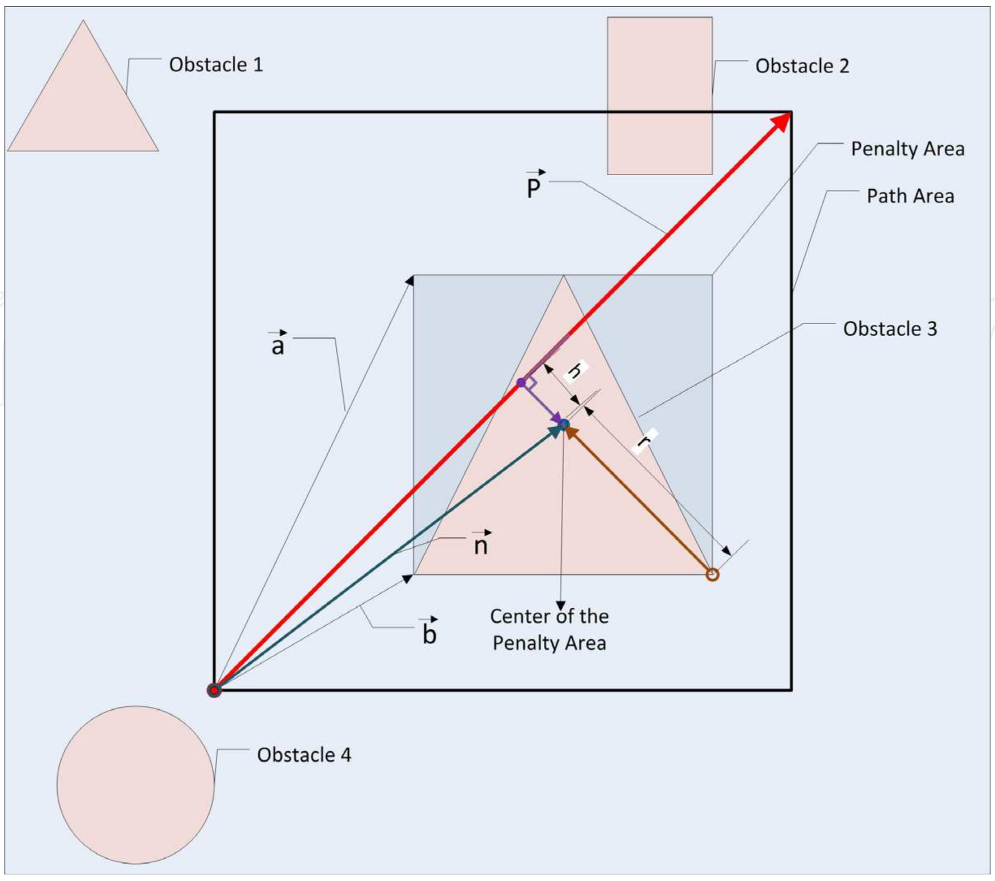


Fig. 18. The penalty calculation for a collision with an obstacle

In the Figure;
Path Area: The rectangular area defined using the maximum and minimum coordinate values of the two points on the path.
Penalty Area: The rectangular area defined using the maximum and minimum coordinate values of the obstacle's vertexes.
$\vec{P}$: The vector between two points of the path, from the first one to the next.
$\vec{a}$ and $\vec{b}$: The vectors from the first point of the $\vec{P}$ to the two vertexes of the penalty area. (There are two additional vectors from the first point of the $\vec{P}$ to the other two vertexes of the penalty area)
$\vec{n}$: The vector from the first point of the $\vec{P}$ to the center of the penalty area.
r : The distance between one vertex and the center of the penalty area.
$h = \frac{|\vec{n} \times \vec{P}|}{|\vec{P}|}$: The distance between the center of the penalty area and the $\vec{P}$ vector.

The collision detection procedure starts with defining the path area, and then the obstacles which are totally or partially located in the path are determined like obstacle 2 and obstacle 3 in the Figure 18. After that, penalty areas for each of the determined obstacle are defined; the penalty area for obstacle 3 is shown in the Figure 18. The following processes are achieved for the determined penalty areas. First, the $\vec{P}$ vector between the two points of the path, and four vectors from the first point of the $\vec{P}$ vector to the vertexes of the penalty area are defined. Afterward, the cross products of the $\vec{P}$ with each of these vectors are achieved and the sign of the each product is determined. If the all cross products have the same sign, this means that the part of the path doesn't cross with obstacle. On the other hand if there are different signs obtained from the cross products, then it means that the part of the path crosses with the obstacle. In Figure 18, the signs for the obstacle 2 are all same while not for the obstacle 3 and that means the part of the path crosses with the obstacle 3. Finally the penalty value is calculated for the obstacles which cross with the part of the path. Euler distance between the center of the penalty area and the $\vec{P}$ vector, and the distance between the center and one vertexes of the penalty area are used to calculate the penalty value as defined in equation 32. In the equation, c_p is the penalty constant for the collisions.

$$P = c_p(1 + r - h)^2 \tag{32}$$

As a result, the object function can be formulated completely as in the equation 33.

$$f = L + \sum_{i=1}^{k} P_i \tag{33}$$

Where, **f** is the final object function value, **L** is the total length of the path, **P** is the total penalty value for the collisions and **k** is the number of the collisions for an individual.
The path planning problem for mobile robots was solved using GA and PSO and a comparison between these two algorithms was presented as it is in the first example. The algorithm's flow charts and other details were not given here since they presented in the chapter and also in the first example. The parameters for the problem were defined as in Table 5, and the problem was firstly solved with GA. The GA's parameters were also defined as in Table 6.

Environment	A 10x10 unit area in 2D space
Obstacles	Seven different shape obstacles
Number of Points of an individual	There are four points. Two of them are via points while the others are start and goal points.
Start Point	[0.2, 0.2]
Goal Point	[8.5, 7.5]

Table 5. The parameters for the path planning problem for mobile robots

Number of Individuals	Number of Iterations	Mutation rate	Crossover Rate	Penalty constant
100	200	0.2	0.9	1000

Table 6. GA parameters

The obtained result can be seen in the Figure 19. It can be seen that GA can find nearly optimum solution to the path planning problem for mobile robots.
The second algorithm is the most general type of the PSO like used in first example. The defined parameters for the algorithm are in the Table 7, and the obtained result from PSO is in the Figure 20, and it is obvious that PSO can find nearly optimum solution to the path planning problem for mobile robots.

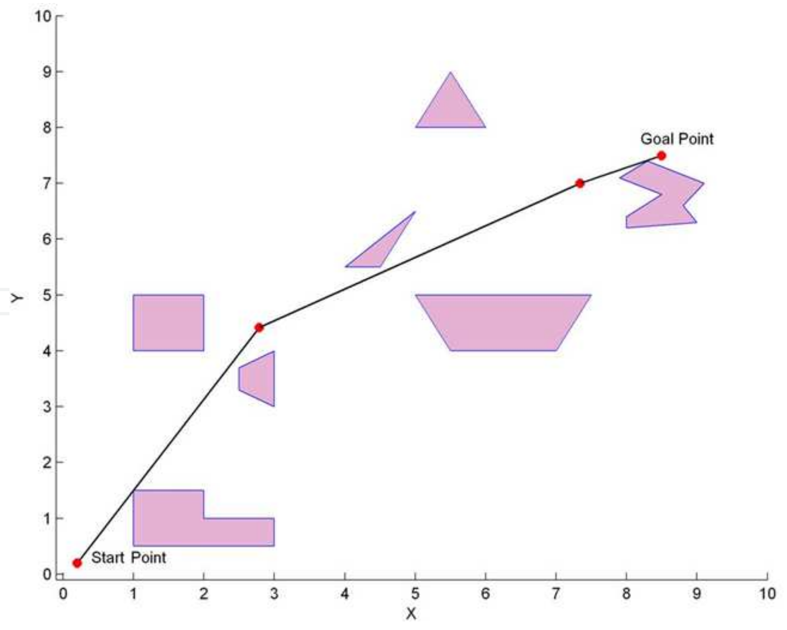


Fig. 19. GA solution

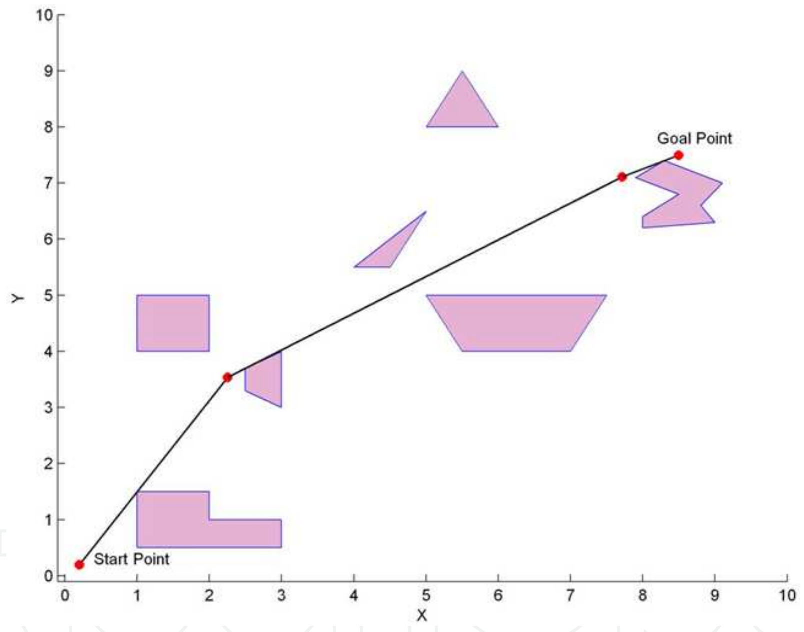


Fig. 20. PSO solution

Number of Individuals	Number of Iterations	Cognitive Component	Social Component	Min. inertia weight	Max. inertia weight	Penalty constant
100	200	2	2	0.4	0.9	1000

Table 7. PSO parameters

For the comparison the same parameters were defined for the two algorithms and each algorithm was run 100 times on the same computer under the same conditions. The parameters were defined as in Table 8.

Parameters	**GA**	**PSO**
Number of Individuals	100	100
Number of Iterations	200	200
Penalty	1000	1000
Start Point	[0.2, 0.2]	[0.2, 0.2]
Goal Point	[8.5, 7.5]	[8.5, 7.5]
Number of Points of an individual	3	3
Gene Code Type	Real Values	Real Values
Mutation Rate	0.2	***
Crossover Rate	0.9	***
Selection Type	R. Wheel	***
Mutation Type	One Point	***
c1	**	2
c2	**	2
Wmin	**	0.4
Wmax	**	0.9

Table 8. The parameters for GA and PSO for the path planning problem

The comparison results for the both algorithms were drawn on the two Figures. In Figure 21, there is a graph for the comparison of the final object function values of the algorithms. Lastly, the graph for the comparison of the algorithms' execution times is in the Figure 22.

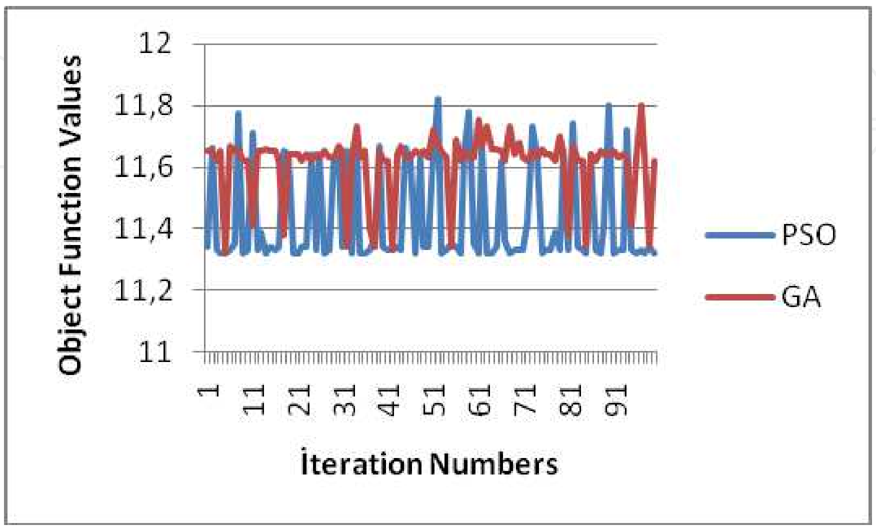


Fig. 21. The object function values of the two algorithms

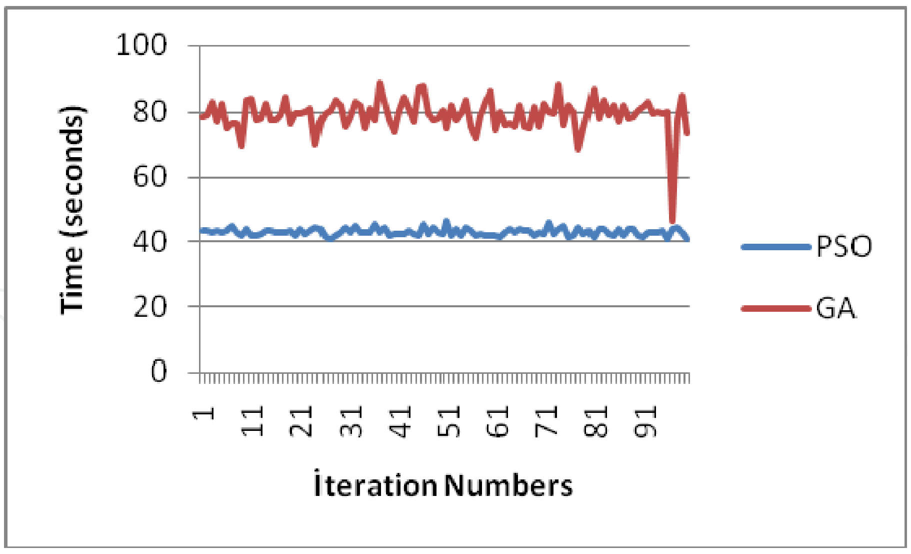


Fig. 22. The execution times of the two algorithms

The perpendicular distance between given start and goal points can be calculated as in equation 34.

$$d = \sqrt{(8.5-0.2)^2 + (7.5-0.2)^2} = 11.0535 \tag{34}$$

The perpendicular distance was calculated directly from start point to goal point and the obstacles were not considered. If the obstacles were considered that means the distance should be longer. The object function values that were calculated from the algorithms are between 11.3 and 11.82, and this means that these results are nearly optimum results. In terms of the comparison, there is no a significant difference about the final object function values. On the other hand, there is a remarkable difference between the two algorithms' execution times. GA's execution time is nearly twice of the PSO's execution times. So, it can be concluded by saying that the both PSO and GA can be used to solve the path planning of the mobile robots, but if the execution time is important, PSO should be preferred.

5. Conclusion

Heuristic algorithms are an alternative way for the solution of optimization problems. Their implementations are easy. Even if they couldn't find the exact optimum point, they find satisfactory results, in a reasonable solution time. In this chapter, two well-known optimization problems in Robotic were solved with PSO and GA. The first problem is the inverse kinematics of the near PUMA robot while the second problem is the path planning problem for mobile robots. These problems were solved with PSO and GA. It has seen that both algorithm give satisfactory results. But PSO outperforms GA in terms of the solution time, because of the fact that it uses little parameters.

6. References

Barzegar, B.; Rahmani, A.M.; Zamanifar, K. & Divsalar, A. (2009). Gravitational emulation local search algorithm for advanced reservation and scheduling in grid computing systems, *Computer Sciences and Convergence Information Technology,. ICCIT '09. Fourth International Conference on*, Vol., No., pp.1240-1245.

Basturk, B. & Karaboga, D. (2006). An Artificial Bee Colony (ABC) Algorithm for Numeric Function Optimization. *IEEE Swarm Intelligence Symposium 2006,* Vol. 8, No. 1, pp. 687-697.

Blum, C. & Roli, A., Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparasion, Technical Report, *TR/IRIDIA*/2001-13, October 2001.

Ceylan, O.; Ozdemir, A. & Dag, H.(2010). Gravitational search algorithm for post-outage bus voltage magnitude calculations, *Universities Power Engineering Conference (UPEC), 2010 45th International* , Vol., No., pp.1-6.

Chandra, R. & Rolland, L.(2011). On solving the forward kinematics of 3RPR planar parallel manipulator using hybrid metaheuristics, *Applied Mathematics and Computation*, Vol. 217, No. 22, pp. 8997-9008

Chen, M. W. & Zalzala, A. M. S.(1997). Dynamic modelling and genetic-based trajectory generation for non-holonomic mobile manipulators, Original Research Article *Control Engineering Practice*, Vol.5, No.1, pp. 39-48

Clerc, M.(1999). The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization. *Proc. Congress on Evolutionary Computation.*, pp 1951-1957.

Dorigo, M.; Maniezzo, V.; Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents, Systems, *Man, and Cybernetics*, Part B: Cybernetics, IEEE Transactions on , Vol.26, No.1, pp.29-41

Eberhart, R & Shi, Y.(2001). Particle swarm optimization: Developments, applications and resources, *in Proc. IEEE Congr. Evol. Comput.*, Vol. 1,No., pp. 81–86

Farmer, D.; Packard, N. & Perelson, A. (1986). The immune system, adaptation and machine learning, *Physica D,* Vol. 2, pp. 187–204

Feo, T. A. & Resende, M. G.C. (1995). Greedy randomized adaptive search procedures, *Journal of Global Optimization*, Vol. 6, No. 2, pp. 109–133

Fisher, M.L.; Alexander, H. G. & Rinnooy, K. (1998) The Design, analysis and implementation of heuristics, *Management Science*, Vol. 34., No.3, pp 263-265.

Gao,M. & Jingwen, T.(2007), Path planning for mobile robot based on improved simulated annealing artificial neural network, *icnc*, *Third International Conference on Natural Computation 2007*, Vol. 3, No., pp.8-12.

Geem, Z.W.; Kim, J.H. & Loganathan, G.V. (2001). A new heuristic optimization algorithm: harmony search, *Simulation*, Vol. 76 No. 2 pp. 60–68

Glover, F. & Kochenberger, Gary A. (2002). *Handbook of Metaheuristics,* Kluwer Academic Publishers, Norwell, MA.

Goldberg, D. E. (1989). *Genetic Algorithms in Search Optimization and Machine Learning*. Addison Wesley. pp. 41. ISBN 0201157675.

Gueaieb, W. & Miah, M.S.(2008). Mobile robot navigation using particle swarm optimization and noisy RFID communication, *Computational Intelligence for Measurement Systems and Applications, 2008. CIMSA 2008.* Vol. , No., pp. 111 - 116

Hong, D.S. & Cho, H.S.(1999). Generation of robotic assembly sequences using a simulated annealing, *Intelligent Robots and Systems*, IROS '99 Proceedings. *1999 IEEE/RSJ International Conference on* , Vol.2, No., pp.1247-1252.

Jourdan, L.; Basseur, M. & Talbi, E.-G.(2009). Hybridizing exact methods and metaheuristics: A taxonomy, *European Journal of Operational Research*, Vol. 199, No. 3, pp. 620-629.

Kalra, P.; Mahapatra, P.B. & Aggarwal, D.K. (2006). An evolutionary approach for solving the multimodal inverse kinematics, *Mechanism and Machine Theory,* Vol. 41, No.10, pp. 1213–1229

Karsli, G. & Tekinalp, O. (2005). Trajectory optimization of advanced launch system, *Recent Advances in Space Technologies*, RAST 2005. *Proceedings of 2nd International Conference on* ,Vol., No., pp. 374- 378

Kennedy, J.(1997). The particle swarm: Social adaptation of knowledge, *in Proc. IEEE Int. Conf. Evol. Comput.*, Vol. ,No., pp. 303–308.

Kennedy, J.; Eberhart, R. (1995). Particle Swarm Optimization. *Proceedings of IEEE International Conference on Neural Networks,* Vol.4, No., IV. pp. 1942–1948

Kirkpatrick, S.; Gelatt, D.C & Vechhi, M.P. (1983) Optimization by simulated annealing, *Science* Vol.220, No.4598, pp. 671–680

Kucuk, Serdar, & Bingul, Zafer. (2005). The Inverse Kinematics Solutions of Fundamental Robot Manipulators with Offset Wrist, Proceedings of the 2005 IEEE International Conference on Mechatronics, Taipei, Taiwan, July 2005

Kucuk, Serdar, & Bingul, Zafer. (2006). Robot Kinematics: Forward and Inverse Kinematics, In: Industrial Robotics: Theory, Modelling and Control, Sam Cubero, pp. (117-148), InTech - Open Access, Retrieved from < http://www.intechopen.com/ articles/show/title/robot_kinematics__forward_and_inverse_kinematics >

Kumar, J.; Xie, S. & Kean, C. A.(2009). Kinematic design optimization of a parallel ankle rehabilitation robot using modified genetic algorithm, *Robotics and Autonomous Systems*, Vol. 57,No. 10, pp 1018-1027

Laguna, M.(1994). A guide to implementing tabu search. *Investigacion Operative*, Vol.4, No.1, pp 5-25

Lazinca, A. (2009). *Particle Swarm Optimization*, InTech, ISBN 978-953-7619-48-0.

Lee, K. Y. & El-Sharkawi, M. A. (2008). *Modern Heuristic Optimization Techniques Theory and Applications to Power Systems*, IEEE Press, 445 Hoes Lane Piscataway, NJ 08854

Li, J. & Xiao, X.(2008). Multi- Swarm and Multi- Best particle swarm optimization algorithm. *Intelligent Control and Automation, 2008, WCICA 2008, 7th World Congress on* , Vol., No., pp.6281-6286.

Lin, CY & Hajela P.(1992). Genetic algorithms in optimization problems with discrete and integer design variables. *Engineering Optimization* ,Vol. 19, No.4 , pp.309–327.

Martı´, R.; Laguna, M. & Glover F. (2006). Principles of scatter search, *European Journal of Operational Research(EJOR),* Vol. 169, No. 2, pp. 359–372.

Raghavan, M. (1993). The Stewart Platform of General Geometry Has 40 Configurations. *Journal of Mechanical Design*, Vol. 115, pp. 277-282

Rashedi, E.; Nezamabadi-pour, H. & Saryazdi, S.(2009). GSA: a gravitational search algorithm. *Information Science*, Vol. 179, No.13, pp. 2232–2248

Saruhan, H.(2006). Optimum design of rotor-bearing system stability performance comparing an evolutionary algorithm versus a conventional method. *International Journal of Mechanical Sciences*, Vol. 48, No 12. pp 1341-1351

Sedighi , Kamran. H., Ashenayi , Kaveh., Manikas, Theodore. W., Wainwright, Roger L., & Tai, Heng Ming (2004). Autonomous Local Path Planning for a Mobile Robot Using a Genetic Algorithm, The Congress on Evolutionary Computation, Oregon, Portland, June 2004.

Selene, L. C-M.;Castillo, O. & Luis T. A.(2011). Generation of walking periodic motions for a biped robot via genetic algorithms, *Applied Soft Computing*, In Press, Corrected Proof, Available online 20 May 2011.

Storn, R.(1996). On the usage of differential evolution for function optimization, *Fuzzy Information Processing Society*, NAFIPS, Biennial Conference of the North American , vol. , No., pp.519-523

Valle, Y.; Venayagamoorthy, G. K.; Mohagheghi, S.; Hernandez, J.C. & Harley, R. G.(2008). Particle swarm optimization: Basic Concepts,Variants and Applications in Power Systems, *IEEE Transactions On Evolutionary Computation*, Vol. 12, No. 2, pp 171-195.

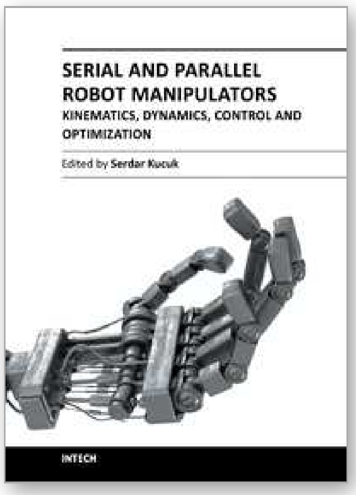
Wang, T. (2001). *Global Optimization For Constrained Nonlinear Programming*, Doctor of Philosophy in Computer Science in the Graduate College of the University of Illinois at Urbana-Champaign, Urbana, Illinois

Weise, T. (2008). *Global Optimization Algorithms – Theory and Application*, Online as e-book, University of Kassel, Distributed Systems Group, Copyright (c) 2006-2008 Thomas Weise, licensed under GNU, online available http://www.it-weise.de/

Yin, M.; Hu, Y., Yang, F.; Li, X. & Gu, W. (2011). A novel hybrid K-harmonic means and gravitational search algorithm approach for clustering, *Expert Systems with Applications* Vol.38, No. , pp. 9319–9324.

Zargar, A. N. & Javadi, H.(2009). Using particle swarm optimization for robot path planning in dynamic environments with moving obstacles and target, *Third UKSim European Symposium on Computer Modeling and Simulation*, Vol. , No., pp.60-65

IntechOpen



Serial and Parallel Robot Manipulators - Kinematics, Dynamics, Control and Optimization
Edited by Dr. Serdar Kucuk

ISBN 978-953-51-0437-7
Hard cover, 458 pages
Publisher InTech
Published online 30, March, 2012
Published in print edition March, 2012

The robotics is an important part of modern engineering and is related to a group of branches such as electric & electronics, computer, mathematics and mechanism design. The interest in robotics has been steadily increasing during the last decades. This concern has directly impacted the development of the novel theoretical research areas and products. This new book provides information about fundamental topics of serial and parallel manipulators such as kinematics & dynamics modeling, optimization, control algorithms and design strategies. I would like to thank all authors who have contributed the book chapters with their valuable novel ideas and current developments.

How to reference
In order to correctly reference this scholarly work, feel free to copy and paste the following:

Pakize Erdogmus and Metin Toz (2012). Heuristic Optimization Algorithms in Robotics, Serial and Parallel Robot Manipulators - Kinematics, Dynamics, Control and Optimization, Dr. Serdar Kucuk (Ed.), ISBN: 978-953-51-0437-7, InTech, Available from: http://www.intechopen.com/books/serial-and-parallel-robot-manipulators-kinematics-dynamics-control-and-optimization/heuristic-optimization-algorithms-in-robotic

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

© 2012 The Author(s). Licensee IntechOpen. This is an open access article distributed under the terms of the Creative Commons Attribution 3.0 License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

IntechOpen

IntechOpen