# A STUDY ON OPTIMIZATION TECHNIQUES FOR SOLVING CONSTRAINED NON LINEAR PROBLEMS

M.M.Janolkar[1], K.A.Joshi[2], K.L.Bondar[3]
[1]First Year Engineering Department
Prof Ram Meghe College of Engineering & Management, Amravati-India
[2]Electrical Engineering Department, Prof Ram Meghe College of Engineering & Management Amravati-India
[3]P.G. Department of Mathematics, N.E.S. Science College, Nanded-India

**Abstract**

**In real world, we usually come across the constrained non-linear optimization problems (CNLP) which bounds on the decision variables and the objective functions are either maximized or minimized under some constraint. In recent past, there have been many traditional and heuristic algorithms to solve CNLP. Lately, the Excel Solver and MATLAB® toolboxes have made use of Genetic Algorithms as an inbuilt function in order to solve CNLP. This paper studies various optimization techniques which are in use in order to solve the constrained non-linear problems.**

**Keywords: constrained non-linear problem; heuristic algorithms; evolutionary algorithms**

## I. INTRODUCTION

The non-linear problems are more frequently encountered while modeling the real time systems and hence the non-linear problems have attracted researchers. As the nature of non-linear problems is complex, they can be encountered in our day to day life. Manually solving the non-linear is next to impossible as it involves the mathematical rigidity of the properties that are to be satisfied. When compared to the linear problems, non-linear are far more difficult to be solved. Hence making use of computing resources could be of great help to the researchers those who are willing t work in the domain of non-linear problems. Many a times, in order to make a non-linear into a linear problems, many approximations are done and thereby reducing the efficiency and could be of no help in the real life problem.

General form of a non-linear problem is: Optimize $f(\mathrm{X})$ Subject to constraints $g_j(\mathrm{X})(\leq=\geq)\,\mathrm{b}_j \;\; j=1,2,...,m$

Where $X=(\mathrm{x}_1,\mathrm{x}_2,\mathrm{x}_3,.....,\mathrm{x}_n)$

There have been many evolutionary algorithms and techniques that are available for the purpose of solving the constrained non-linear problem. The major problem faced in this domain of non-linear problems is that, a technique that is suitable for solving a particular non-linear problem would be highly inefficient for some other non-linear problem and hence there is a high risk in choosing any particular technique in order to solve a particular non-linear problem.

Starting with the Random Search Technique (RST) which was developed in 1965 by Prince and later it was improved by C Mohan and Kusum Deep as they made use of FORTRAN to solve the problems. Also since then, there have been many versions of the random search technique which have been based on higher level programming languages like C, C++ [1]. After the RST came the Genetic Algorithms (GA) and their hybrids [2], [3] and [4] which are majorly based on evolution and this got accepted by many researchers round the world in order to solve some complex non-linear problems. There have also been many heuristic algorithms and their hybrids which have been used in the non-linear problem solving. Particle Swarm Optimization [5], [6], [7], Ant Colony Optimization [8], and many more are the well-

known techniques for the non-linear problem solving.

## II. METHODS OF OPTIMIZATION

### A. *Particle Swarm Optimization (PSO)*

The Particle Swarm Optimization is an evolutionary search and optimization technique that has been designed and developed by Kennedy and Eberhart. Lately, the concept of PSO has been growing rapidly and this can be claimed from the number of applications that make use of the PSO. The major concept on which the PSO works is the behavior of some of the animal societies that have no leader like the fish or the birds. Basically, a group of animals that do not have any leader will some or the other way find some food for themselves at random or they could also follow a member from their group who is very much near to finding the food that is nothing but the possible solution to the problem. The groups of animals do achieve best of the results by communicating between the other animals that are in a better position of finding food. The animal who has found the source of food will inform the remaining animals in the group and hence the others follow simultaneously to that particular place. The concept of PSO follows the work of such animal communities in order to find the optimal values. Particle swarm optimization basically comprises of a swarm of particles, where in the particles represent a potential solution.

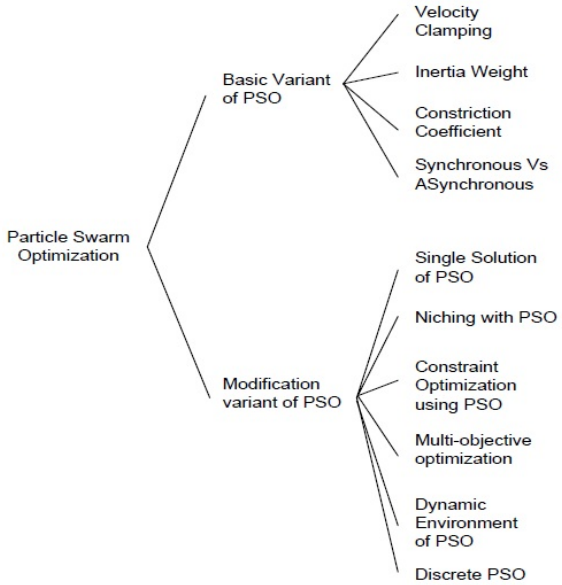


Fig. 1. Variants of Particle Swarm Optimization

Particle swarm optimization is one of the rare biologically inspired algorithms, which was proposed by Kennedy and Eberhart in the year 1995. The PSO algorithm was based on the behavior of birds and fishes. Lately, there have been many variants of the PSO technique as can be seen from figure 1. The basic variants of the PSO are velocity clamping, inertia weight, construction coefficient, synchronous and asynchronous updates. Each of the variant of the PSO is proposed recently and hence, the shortfall caused by the previous PSO shall be covered in the most recent variant of the PSO.

Let $x$ and $v$ denote a particle coordinates (position) and its corresponding flight speed (velocity) in a search space, respectively. Therefore, the $i$-th particle is represented as $x_i = (x_{i1},x_{i2},\ldots x_{id})$ in the $d$-dimensional space. The best previous position of the $i$-th particle is recorded and represented as $pbest_i = (pbest_{i1}, pbest_{i2}, \ldots pbest_{id})$The index of the best particle among all the particles in the group is represented by the $gbest_d$. The rate of the velocity for particle $i$ is represented as $v_i = (v_{i1}, v_{i2}, \ldots v_{id})$ . The modified velocity and position of each particle can be calculated using the current velocity and the distance from $pbest_i$ to $gbest_d$ as shown in the following formulas:

$$v_{id}^{(t+1)} = w.v_{id}^{(t)} + c_1 * rand() * (pbest_{id} - x_{id}^{(t)}) \quad (1)$$

$$+c_2 * \text{Rand}() * (g\,best_d - x_{id}^{(t)})$$

$$x_{id}^{(t+1)} = x_{id}^{(t)} + v_{id}^{(t+1)}, i = 1,2,\ldots,n \quad (2)$$

$$d = 1,2,\ldots,\text{m}$$

Where,

$n$ number of particles in a group;

$m$ number of members in a particle;

$t$ pointer to iterations (generations);

$w$ inertia weight factor;

$c_1,c_2$ acceleration constant;

rand(), Rand() uniform random value in the range [0,1];

$v_i^{(t)}$ velocity of particle $I$ at iteration $t$,

$$V_d^{\min} \le v_{id}^{(t)} \le V_d^{\max}$$

$x_i^{(t)}$ current position of particle $i$ at iteration $t$.

In the above procedures, the parameter $V^{max}$ determined the resolution, or fitness, with which regions are to be searched between the present position and the target position. If $V^{max}$ is too high, particles might fly past good solutions. If

$V^{max}$ is too small, particles may not explore sufficiently beyond local solutions. In many experiences with PSO, $V^{max}$ was often set at 10–20% of the dynamic range of the variable on each dimension.

The constants $c_1$ and $c_2$ and represent the weighting of the stochastic acceleration terms that pull each particle toward the *pbest* and *gbest* positions. Low values allow particles to roam far from the target regions before being tugged back. On the other hand, high values result in abrupt movement toward, or past, target regions. Hence, the acceleration constants $c_1$ and $c_2$ were often set to be 2.0 according to past experiences.

Suitable selection of inertia weight $w$ in (3) provides a balance between global and local explorations, thus requiring less iteration on average to find a sufficiently optimal solution. As originally developed, $w$ often decreases linearly from about 0.9 to 0.4 during a run. In general, the inertia weight $w$ is set according to the following equation:

$$w = w_{\max} - \frac{w_{\max} - w_{\min}}{iter_{\max}} * iter \qquad (3)$$

where $iter_{max}$ is the maximum number of iterations and *iter* is the current number of iterations.

### B. *Genetic Algorithms (GA)*

The famous naturalist Charles Darwin defined natural selection or survival of the fittest in his book (Darwin, 1929) as the preservation of favourable individual differences and variations, and the destruction of those that are injurious. In nature, individuals have to adapt to their environment in order to survive in a process called evolution, in which those features that make an individual more suitable to compete are preserved when it reproduces, and those features that make it weaker are eliminated. Such features are controlled by units called genes, which form sets known as chromosomes. Over subsequent generations not only the fittest individuals survive, but also their genes which are transmitted to their descendants during the sexual recombination process, which is called crossover.

Basically, the central thrust of the research on genetic algorithms (GAs) has been due to its robustness and the balance between efficiency and efficacy necessary for survival in many different environments. GAs are search algorithms, which are based on the mechanics of natural selection and survival of the fittest, and unlike many mathematical programming algorithms they do not require the evaluation of gradients of the objective function and constraints.

The Genetic Algorithms are heuristic search approaches that are applicable and valid for a large range of optimization problems and hence this flexibility makes the genetic algorithms very attractive for many of the optimization problems. The concept on which the genetic algorithms work is evolution. The current variety and success of species is a good reason for believing in the power of evolution. Species are able to adapt to their environment. They have developed to complex structures that allow the survival in different kinds of environments. Mating and getting offspring to evolve belong to the main principles of the success of evolution. These are good reasons for adapting evolutionary principles to solving optimization problems.

The classic Genetic Algorithm is based on a set of candidate solutions that represent a solution to the optimization problem we want to solve. A solution is a potential candidate for an optimum of the optimization problem. Its representation plays an important role, as the representation determines the choice of the genetic operators. Representations are usually lists of values and are more generally based on sets of symbols. If they are continuous, they are called vectors, if they consist of bits, they are called bit strings. In case of combinatorial problems the solutions often consist of symbols that appear in a list. An example is the representation of a tour in case of the traveling salesman problem. Genetic operators produce new solutions in the chosen representation and allow the walk in solution space.

Figure 2 illustrates the flowchart for a simple GA linked to a structural design problem. At the beginning, all the necessary data – GA parameters and structural geometry – will be read and the process of the GA will start for the first generation. The initial population will be generated randomly. Then, the objective function regarded as the weight of the structure as well as the constraint functions, which are reflected on the design criteria requested by BS

5950, are computed. At this stage, the average, maximum and the fittest design are obtained. Convergence criteria described later are also checked. The GA process is terminated if the convergence is achieved. Otherwise, the GA process resumes. By creating the mating pool and applying the GA operators, the next population is created. The GA process will proceed until either the convergence is achieved or the maximum number of generations is reached.

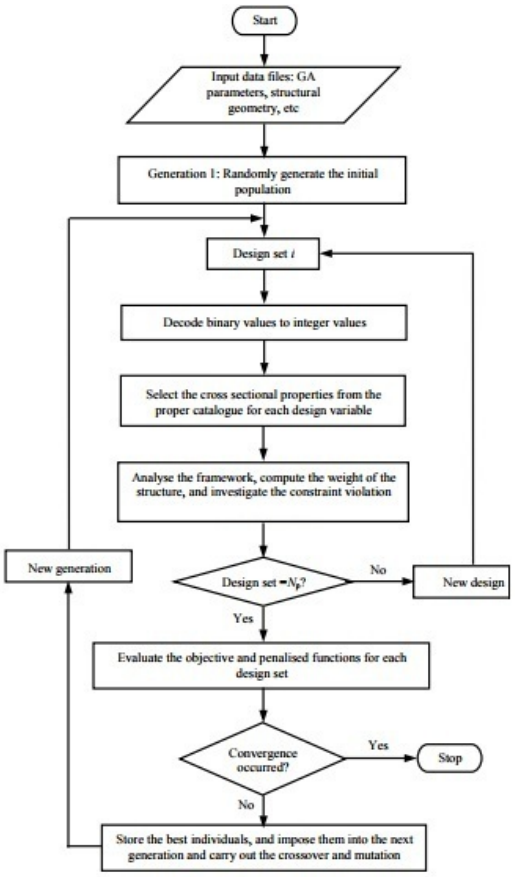


Fig. 2. Flowchart for GA linked to design problem

### C. *MATLAB®*

MATLAB® basically stands for Matrix Laboratory. It is a language having high performance capability used for technical computing. MATLAB® provides two different ways of solving problems, one is by the method of coding and the other is by making use of a toolbox. In the toolbox, there are plenty of tools to solve the optimization problems that are computer oriented and thereby making the process of solving such types of problems at a very high speed and making it comparatively less tedious and increasing the accuracy.

MATLAB® is also very well known for its graphics. There are many plotting functions that are available in MATLAB® toolbox which help in providing a good visual representation of the solutions of the various problems. The main purpose of studying the MATLAB® toolboxes is to get the basic idea of the optimization toolbox that is present in MATLAB®. The optimization toolbox is used to find the minimum of constrained non-linear multivariable function. Basically, it finds the minimum of a constrained non-linear multivariable function, and by default it is based on the Sequential Quadratic Programming algorithm. This algorithm repeatedly modifies a population of individual solutions.

Let us consider an example of a YALMIP MATLAB® toolbox that is used for the purpose of optimization. Once all variables and constraints have been defined, the optimization problem can be solved. Let us for simplicity assume that we have matrices c, A and b and we wish to minimize $c^Tx$ subject to the constraints $Ax \leq b$ and $\sum x_i = 1$. The YALMIP (Optimization toolbox in MATLAB®) code to define and solve this problem is extremely intuitive and is essentially a one-to-one mapping from the mathematical description. The command solvesdp3 is used for all4 optimization problems and typically take two arguments, a set of constraints and the objective function.

```
>> x = sdpvar(length(c),1);
>> F = set(A*x < b)+set(sum(x)==1);
>> solvesdp(F,c'*x);
```

YALMIP will automatically categorize this as a linear programming problem and call a suitable solver. The optimal solution can be extracted with the command double(x). A third argument can be used to guide YALMIP in the selection of solver, setting display levels and change solver specific options etc.

```
>> ops = sdpsettings('solver','glpk');
>> ops = sdpsettings(ops,'glpk.dual',0);
>> ops = sdpsettings(ops,'verbose',1);
>> solvesdp(F,c'*x,ops);
```

### D. *Branch and Bound*

The branch and bound method BBM is perhaps the most widely known–method for mixed–discrete optimization problems. The

method was originally developed for LP, however it is quite general and can be applied to nonlinear discrete and mixed variable problems. It is basically an enumeration method where one first obtains a minimum point for the problem assuming all variables to be continuous. Then, each variable is assigned a discrete value in sequence and the problem is solved again in the remaining variables. The process of assigning discrete values to variables need not start from a continuous optimum point although this approach may reduce the number of times the problem needs to be re–solved to obtain a feasible discrete point and subsequently the optimum solution. It can be seen that the number of times the problem needs to be re–solved increases exponentially with the number of variables. Several procedures have been devised to reduce this number. The first use of the branch and bound method is attributed to Land and Doig for linear problem. Other attempts, to use branch and bound to solve integer LP problems related to the plastic design of frames, made by Reinschmidt.

Branch and bound was combined with exterior penalty functions and SQP methods to treat the mixed–discrete NLP problem. John et al. had combined branch and bound with sequential linearization for discrete optimal design of trusses. Hajela and Shih used branch and bound to solve multi–objective optimization problems with discrete and integer variables. Salajegheh and Vanderplaats [9] used branch and bound for optimizing trusses with discrete sizing and shape variables. Large storage space was needed and an exponential growth in computational effort limits the applicability of branch and bound to solve higher dimensional problems.

The usual BBM approach is to systematically search continuous solutions where the discrete variables are forced to take discrete values from the specified set. The logical structure for the set of solutions is that of a tree for each variable. Initially an optimum point is obtained by treating all design variables as continuous. If this solution is discrete, then the process is terminated. If one of the desired variables is not discrete, then its value lies between two discrete values:

$$d_{j,\lambda} < x_{i,j} < d_{j,\lambda+1}$$

Now two sub–problems are defined, one with the constraint $x_{i,j} \le d_{j,\lambda}$ and the other with $x_{i,j} \le d_{j,\lambda+1}$. This process called branching. It basically eliminates some portion of the continuous feasible region, which is not feasible for the discrete problem. It does not, however, eliminate any of the discrete feasible solutions. The two sub–problems are solved again, and the optimum solutions are stored as nodes of the tree containing optimum values of the variables, the objective function and the appropriate bounds on the variables. This process of branching and solving continuous problems is continued until a feasible solution is obtained. The cost function corresponding to this solution becomes an upper bound on the optimum solution. From this point, all the nodes of the tree that have cost function values higher than the established upper bound are eliminated from further consideration. Such nodes, known as fathomed nodes, when their lowest point is reached and no further branching is necessary from them. This process is known as bounding.

### *E. Simulated Annealing*

Simulated annealing (SA) is one of the techniques, which do not require derivatives of the problem functions because it does not use any gradient or Hessian information. The idea of SA originated in statistical mechanics by Metropolis et al. The approach is suited to solving combinatorial optimization problems, such as discrete–integer programming problems.

The use of simulated annealing for structural optimization is a quite recent occurrence. Elperin applied the SA to design a ten–bar truss where member cross–sectional areas were to be selected from a set of discrete values. Kincaid and Padula used SA for minimizing the distortion and internal forces in a truss structure. Balling obtained the optimum design of three–dimensional steel structures using SA. One year later, the same framework was studied using the filtered simulated annealing algorithm by May and Balling. Recently, Leite and Topping proposed a parallel simulated annealing model for structural optimization.

## III. CONCLUSION

The realistic problems that are seen are more often non-linear in nature and are also

constantly varying which brings in demand for improvised techniques that give better optimal results. Although there are many other techniques, there is still need for more robust techniques and also must be feasible. In this paper, a study of various optimization techniques that are in place for solving constrained non-linear problems. In the process of the study being done, a new technique can be found that can be useful to solve nonlinear problems. And also by testing the technique on certain benchmarking constrained non-linear problems, one can find whether the technique is useful and also whether it is robust by nature or not.

## REFERENCES


[1] C. Mohan and H.T. Nguyen, "A Controlled Random Search Technique Incorporating the Simulated Annealing Concept for Solving Integer and Mixed Integer Global Optimization Problems", Comput. Optim. Appl., 14, pp. 103-32, 1999.

[2] Kusum Deep, "Recent advances in optimization techniques and their applications", Department of Mathematics and Continuing Education Centre, Indian Institute of Technology, 2009.

[3] Richard Y. K. Fung, Jiafu Tang and Dingwei Wang, "Extension of a hybrid genetic algorithm for non-linear programming problems with equality and inequality constraints", Comput. Oper. Res., 29, pp. 261-74, 2002.

[4] Jiafu Tang and Dingwei Wang, "A Hybrid Genetic Algorithm for a type of Nonlinear Programming Problem", Comput. Math. Appl., 36(5), pp. 11-21, 1998.

[5] Mazdak Shokrian and Karen Ann High, "Application of a multi objective multi-leader particle swarm optimization algorithm on NLP and MINLP problems", Comput. Chem. Eng., 60, pp. 57-75, 2014.

[6] Yang Sun, Lingbo Zhang and XingshengGu, "A hybrid co-evolutionary cultural algorithm based on particle swwarm optimization for solving global optimization problems", Neurocomputing, 98, pp. 76-89, 2012.

[7] Ying Dong, Jiafu Tang, Baodong Xu and Dingwei Wang, "An Application of Swarm Optimization to Nonlinear Programming", Comput. Math. Appl., 49, pp. 1655-68, 2005.

[8] Christian Blum, "Ant Colony Optimization: Introduction and recent trends", Phys. Life.Rev., 2, pp. 353-73, 2005.

[9] Salajegheh, E. & Vanderplaats, G.N. "Structural Optimization", pp. 6: 79. , 1993