International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 4 Issue 04, April-2015

Review of Modern Optimization Techniques

Shahbaz Khan
Mechanical Engineering
Jamia Millia Islamia
New Delhi (India)

Mohammad Asjad
Mechanical Engineering
Jamia Millia Islamia
New Delhi (India)

Akhlas Ahmad
Mechanical Engineering
IntegralUniversity
Lucknow (India)

***Abstract*—Many difficulties are associated with the optimization of large-scale problems. The major difficulties are multi-modality, dimensionality and differentiability. Traditional techniques generally fail to solve such large-scale problems especially with nonlinear objective functions. The main problem is to solve non-differentiable functions with the help of traditional techniques because most of the traditional techniques require gradient information and hence it is not possible. Moreover, such techniques often fail to solve optimization problems that have many local optima. To overcome these problems, there is a need to develop more powerful optimization techniques. These techniques are known as modern optimization technique. In this Paper, the theory needed to understand the modern optimization techniques are explained. These modern techniques are used to solve linear, nonlinear, differential and non-differential optimization problems. Although various optimization methods have been proposed in recent years, but some more popular optimization techniques such as Genetic Algorithm, Simulated Annealing Ant colony method, Honey Bee Algorithm are presented here. The methods were broadly reviewed.**

Keyword: Optimization, non-differentiable, nonlinear, Genetic Algorithm, Simulated Annealing, Ant colony method, Honey Bee Algorithm

I. INTRODUCTION

Optimization is a technique which is used everywhere, from engineering design to financial markets from fashion technology to mass communication and also in our daily activities. We always intend to maximize or minimize something which is simply known is the objective function. To determine the optimal solution for objective functions, there are various optimization tools like linear programming, simplex method, assignment model, transportation model, CPM and PERT are playing a vital role. Organizations are implementing these techniques to maximize their profits, minimizetheir costs,. Even when we plan our holidays, we want to maximize our enjoyment (objective function) with least cost (or ideally free). In fact, we are constantly searching for the optimal solutions to every problem we meet, though we are not necessarily able to find such solutions.

The complexity of the problem of interest makes it impossible to search every possible solution or combination, the aim is to find good, feasible solutions in an acceptable timescale. There is no guarantee that the best solutions can be found, and we even do not know whether an algorithm will work and why if it does work.

As this paper is mainly about the introduction to metaheuristic algorithms, techniques such as Genetic Algorithm, Simulated Annealing Ant colony method, Honey Bee Algorithm In fact, quite a significant number of new algorithms in optimization are primarily metaheuristics.

II. LITERATURE REVIEW

Alan Turing was probably the first to use heuristic algorithms during the Second World War, when he was breaking German Enigma ciphers at Bletchley Park, where Turing, together with British mathematician Gordon Welchman, designed in 1940 a cryptanalytic electromechanical machine, the *Bombe*, to aid their code-breaking work. The bomb used a heuristic algorithm, as Turing called, to search, among about 1022 potential combinations, the possibly correct setting coded in an Enigma message. Turing called his search method heuristic search, as it could be expected, it worked most of the time, but there was no guarantee to find the correct solution, but it was a tremendous success [3].

The next significant step is the development of evolutionary algorithms in the 1960s and 1970s. First, John Holland and his collaborators at the University of Michigan developed the genetic algorithms in the 1960s and 1970s. As early as 1962, Holland studied the adaptive system and was the first to use crossover and recombination manipulations for modeling such systems. His seminal book summarizing the development of genetic algorithms was published in 1975 [4].

The decades of the 1980s and 1990s were the most exciting time for metaheuristic algorithms. The next big step is the development of simulated annealing (SA) in 1983, an optimization technique, pioneered by S. Kirkpatrick, C. D. Gellat and M. P. Vecchi, inspired by the annealing process of metals. It is a trajectory-based search algorithm starting with an initial guess solution at a high temperature, and gradually cooling down the system. A move or new solution is accepted if it is better; otherwise, it is accepted with a probability, which makes it possible for the system to escape any local optima. It is then expected that if the system is cooled slowly enough, the global optimal solution can be reached [1].

In 1992, Marco Dorigo finished his PhD thesis on optimization and natural algorithms, in which he described his innovative work on ant colony optimization (ACO). This search technique was inspired by the swarm

intelligence of social ants using pheromone as a chemical messenger [5].

In 1997, the publication of the 'no free lunch theorems for optimization' by D. H. Wolpert and W. G. Macready sent out a shock wave to the optimization community. Researchers have always been trying to find better algorithms, or even universally robust algorithms, for optimization, especially for tough NPhard optimization problems. However, these theorems state that if algorithm A performs better than algorithm B for some optimization functions, then B will outperform A for other functions. That is to say, if averaged over all possible function space, both algorithms A and B will perform on average equally well. Alternatively, there is no universally better algorithms exist [2].

III. OPTIMIZATION TECHNIQUES

There are different optimization techniques for diverse troubleshoots or bottlenecks that we face in industries or in our daily life. For these optimization techniques, there are always a best suited solution in case of various assorted intermingled issues.

Metaheuristics

Most metaheuristic algorithms are nature-inspired as they have been developed based on some abstraction of nature. Nature has evolved over millions of years and has found perfect solutions to almost all the problems she met." No one manufactures a lock without a key, thus it is learning process of success of problem-solving from nature and develop naturally-inspired heuristic and/or metaheuristic algorithms. More specifically, some nature-inspired algorithms are inspired by Darwin's evolutionary theory. Consequently, they are said to be biologically-inspired or simply bio-inspired [6].

Two major components of any metaheuristic algorithms are: selection of the best solutions and randomization. The selection of the best ensures that the solutions will lead to the optimality, while the randomization ignores local optimal solution and enhances the diversity of the solutions. Effective utilization of these two components will usually ensure that the global optimality is achievable.

A. Simulated annealing (SA)

Simulated annealing (SA) is a random search technique for global optimization problems, and it reveals the annealing process in material processing when a metal cools and freezes into a crystalline state with the minimum energy and larger crystal size so as to reduce the defects in metallic micro structures. The annealing process involves the strict control of temperature and cooling rate called annealing schedule [7].

Metaphorically speaking, this is similar to dropping some bouncing balls over a landscape, and as the balls bounce and lose energy, they settle down to some local minima. If the balls are allowed to bounce enough times and lose energy slowly enough, some of the balls will eventually fall into the globally lowest locations; hence, the global minimum will be reached.

The basic idea of the simulated annealing algorithm is to use random search in terms of a Markov chain, which not only permits changes that improve the objective function, but also keeps some changes that are not suitable. In a minimization problem, for example, any better moves or changes that decrease the value of the objective function / will be permitted ; however, some changes that enhancement will also be accepted with a probability p. This probability p, also called the transition probability, is determined by

$$P\{\text{accept } \omega' \text{ as next solution}\} = \begin{cases} \exp\left[-\frac{f(\omega')-f(\omega)}{tk}\right] & if\ f(\omega')-f(\omega) > 0 \\ 1 & if\ f(\omega')-f(\omega) \le 0 \end{cases}$$

Where

ω = initial solution

ω' = neighboring solution

tk = as the temperature parameter at iteration *k*

The working procedure of SA is easily understand by following steps

Step 1: Initialize – Start with a random initial placement. Initialize a very high "temperature".

Step 2: Move – Perturb the placement through a defined move.

Step 3: Calculate score – calculate the change in the score due to the move made.

Step 4: Choose – Depending on the change in score, accept or reject the move. The probe of acceptance depending on the current "temperature".

Step 5: Update and repeat– Update the temperature value by lowering the temperature. Go back to Step 2.

The process is done until "stop condition" is reached. There are several stop condition of the algorithm. Some examples are:

a) Maximum number of iterations;
b) Minimum temperature value;
c) Minimum value of objective function;
d) Minimum value of acceptance rate.

Pseudo code

Objective function *f(x)*, $x = (x_1, x_2, \dots\dots\dots\dots\dots x_p)^T$

Initialize initial temperature T_0 and initial guess $x^{(0)}$
Set final temperature T_f and max number of iterations *N*
Define cooling schedule T → αT, (0 <a <1)
while($T > T_f$ and $n < N$)
  Move randomly to new locations: $x_{n+1} = x_n +$ randn
  Calculate Δf = $f_{x+1}(x_{n+1}) - f_n(x_n)$
  Accept the new solution if better
  if not improved
    Generate a random number *r*
    Accept if $p = exp\ [-\Delta f/T] > r$
  endif
  Update the best x_* and f_*
  $n = n + 1$
end while

B.Genetic Algorithm

Holland was first presented systematically Genetic Algorithm [7], the basic ideas of analysis and design based on the concepts of biological evolution can be found in the work of Rechenberg [8]. Philosophically, GAs are based on Darwin's theory of survival of the fittest. Genetic algorithms are based on the principles of natural genetics and natural selection. The basic elements of natural genetics reproduction, crossover, and mutation—are used in the procedure of genetic algorithm.

The solution of an optimization problem by GAs commences with a population of random strings denoting different (population of) design vectors. The population size in GAs (n) is generally fixed. Each string (or design vector) is evaluated to find its fitness value. The population is operated by three operators—reproduction, crossover, and mutation—to produce a new population of points. The new population is further evaluated to find the fitness values and examined for the convergence of the process. One cycle of reproduction, crossover, and mutation and the evaluation of the fitness values is known as a generation in GAs. If the convergence criterion is not satisfied, the population is iteratively operated by the three operators and the resulting new population is evaluated for the fitness values. The procedure is continued through different generations until the convergence criterion is satisfied and the process is terminated. The details of the three operations of GAs are given below.

Reproduction It is the first operation applied to the population to select good strings (designs) of the population to form a mating pool. The reproduction operator is also called the selection operator because it opts good strings of the population. The reproduction operator is used to choose above-average strings from the recent population and apply their numerous copies in the mating pool based on a probabilistic procedure. In a commonly used reproduction operator, a string is chosen from the mating pool with a probability proportional to its fitness [10].

*Crossover*After reproduction, the crossover operator is implemented. The aim of crossover is to form new strings by exchanging information among the strings of the mating pool. Numerous crossover operators have been used in the literature of GAs. In most crossover operators, two individual strings (designs) are chosen (or selected) at random from the mating pool generated by the reproduction operator and some part of the string are interchanged between the strings. In the commonly used process, known as a single-point crossover operator, a crossover site is selected at random along the string length, and the binary digits (alleles) lying on the right side of the crossover site are swapped (exchanged) between the two strings. The two strings involve in the crossover operators are known as parent strings and the strings generated by the crossover operator are known as child strings [10].

For example, if two design vectors (parents), each with a string length of 10, are given by

(Parent 1) **X**1 = {1 1 0 | 1 0 1 0 0 1 1}

(Parent 2) **X**2 = {0 0 1 | 0 1 0 1 1 0 1}

The result of crossover, when the crossover site is is given by

(Offspring 1) **X**3 = {1 1 0 | 0 1 0 1 1 0 1}
(Offspring 2) **X**4 = {0 0 1 | 1 0 1 0 0 1 1}

Since the crossover operator combines substrings from parent strings (which have good fitness values), the resulting child strings created are expected to have better fitness values provided an appropriate (suitable) crossover site is selected

*Mutation*The crossover is the important operator by which new strings with better fitness values are created for the new generations. The mutation operator is used in the new strings with a particular small mutation probability, pm. The mutation operator changes the binary digit (allele's value) 1 to 0 and vice versa. Different methods can be utilized for implementing the mutation operator. In the single-point mutation, a mutation site is selected at random along the string length and the binary digit at that site is then changed from 1 to 0 or 0 to 1 with a probability of pm. In the bit-wise mutation, each bit (binary digit) in the string is considered one at a time in sequence, and the digit is changed from 1 to 0 or 0 to 1 with a probability P_m Numerically, the process can be implemented as follows. A random number between 0 and 1 is generated/chosen. If the random number is smaller than pm, then the binary digit is changed. Otherwise, the binary digit is not changed. The Aim of mutation is (1) to generate a string (design point) in the adjacent of the current string, therefore accomplishing a local search around the current solution, (2) to prevent against a pre-mature loss of important genetic material at a particular position, and (3) to maintain diversity in the population [10].

Pseudo code

Objective function $f(x)$, $x = (x_1, x_2, \ldots\ldots\ldots\ldots\ldots x_n)^T$

Encode the solution into binary strings (chromosomes)
Define fitness F (eg, F α f(x) for maximization)
Generate the initial population
Initial probabilities of crossover (P_c) and mutation (P_m)
While(t<Max number of generations)
Generate new solution by crossover and mutation
ifP_c>rand, Crossover; end if
ifP_m>rand, Mutate; end if
Accept the new solutions if their fitness increase
Select the current best for new generation
end while
Decode the results and visualization

C. Ant Colony Method

In this, we will discuss the nature-inspired ant colony optimization (ACO), which is a metaheuristic method. Ants are social insects in habit and they hold out together in organized colonies whose population size can range from around 2 to 25 million. When foraging, a swarm of ants or mobile agents interact or communicate with their local surroundings. Each ant can lay scent chemicals or pheromone so as to communicate with others, and each

unit is likewise able to travel along the road marked with pheromone laid by other ants. When ants find a food source, they will mark it with pheromone and also mark the trails to and from it [11]. From the initial random foraging route, the pheromone concentration varies and the ants follow the route with higher pheromone concentration, and the pheromone is enhanced by the increasing number of ants. As more and more ants follow the same route, it becomes the favored path. Thus, some favorite routes emerge, often the shortest or more efficient. This is a positive feedback mechanism.

Based on these characteristics of ant behavior, scientists have developed a number of powerful ant colony algorithms with important progress made in recent years. Marco Dorigo pioneered the research in this area in 1992 [12]. Many different variants have appeared since then.

If we just apply some of the foraging behavior of ants and add some new features, we can organize a category of new algorithms. Two significant issues here: the probability of choosing a route, and the dehydration rate of pheromone [13]. There are a few ways of solving these problems, although it is still an area of active research. Here we introduce the current best method.

For a network routing problem, the probability of ants at a particular node *to* choose the route from node *to* node *j*, among n<j nodes, is given by

$$p_{ij} = \frac{\Phi_{ij}^{\alpha}\, d_{ij}^{\beta}}{\sum_{i,j=1}^{nd} \Phi_{ij}^{\alpha}\, d_{ij}^{\beta}} \qquad \text{(a)}$$

Whereα >0 and β > 0 are the influence parameters
Φ = pheromone concentration

The pheromone concentration can change with time due to the evaporation of pheromone. Furthermore, the advantage of pheromone evaporation is that the system could avoid being trapped in local optima. If there is no evaporation, then the path randomly chosen by the first ants will become the preferred path as the attraction of other ants by their pheromone. For a constant rate γ of pheromone decay or evaporation, the pheromone concentration usually varies with time exponentially

$$\Phi(t) = \Phi_0 e^{-\gamma t}$$

Where$\Phi_0$the initial concentration of pheromone and t Is time. If $\gamma t \ll 1$ then we have $\Phi(t) \approx (1-\gamma t)\Phi_o \cdot$ For the unitary time increment Δt = 1, the evaporation can be approximated by $\Phi^{t+1} \leftarrow (1-\gamma)\Phi^t$. Therefore, we have the simplified pheromone update formula:

$$\Phi_{ij}^{t+1} = (1-\gamma)\Phi_{ij}^{t} + \Phi_{ij}^{t}$$

γ Pheromone decay or evaporation
The increment $\delta\Phi_{ij}^{t}$ is the amount of pheromone deposited at time *t* along route *i*to *j*

Pseudo code
Objective function *f(x)*, $x = (x_1, x_2, \ldots\ldots\ldots\ldots\ldots x_n)^T$
[or$f(x_{ij})$ for routing problem where *(I,j)* ϵ {1 .,n}]
Define pheromone evaporation rate γ
While(criterion)
for loop over all *n* dimensions (or nodes)
Generate new solutions
Evaluate the new solutions
Mark better locations/routes with pheromone $\delta\Phi_{ij}$
Update pheromone: $\Phi_{ij} = (1-\gamma)\Phi_{ij} + \delta\Phi_{ij}$
end for
Daemon actions such as finding the current best
end while
Output the best results and pheromone distribution

D. Honey Bee Algorithms.

Bee algorithms form another class of algorithms which are closely related to the ant colony optimization. Bee algorithms are inspired by the foraging behavior of honey bees. Honey bees live in a colony and they forage and store honey in their constructed colony. Honey bees can communicate by pheromone and 'waggle dance'. For example, an alarming bee may release a chemical message (pheromone) to stimulate an attack response in other bees. Furthermore, when bees find a good food source and bring some nectar back to the hive, they will communicate the location of the food source by performing the so-called waggle dances as a signal system. Such signaling dances vary from species to species, however, they will try to recruit more bees by using directional dancing with varying strength so as to communicate the direction and distance of the found food resource.

From the literature survey, it seems that the Honey Bee Algorithm (HBA) was first formulated in around 2004 by Craig A Tovey at Georgia Tech in collaboration with Sunil Nakrani then at Oxford University to study a method to allocate computers among different clients and web-hosting servers [14]. In the honey bee algorithm, forager bees are allocated to different food sources (or flower patches) so as to maximize the total nectar intake. The colony has to 'optimize' the overall efficiency of nectar collection; the allocation of the bees is thus depending on many factors such as the nectar richness and the proximity to the hive. The probability of an observer bee following the dancing bee to forage can be determined in many ways depending on the actual variant of algorithms. A simple way is given by Guijano and Passino

$$p_i = \frac{w_i^j}{\sum_{i=1}^{n_f} w_i^j}$$

Where w_i^j be the strength of the waggle dance of bee *i*at time step *t = j*,

n_f is the number of bees

In addition, the rating of each route is ranked dynamically and the path with the highest number of bees become the preferred path. For a routing problem, the probability of selecting a path between any two nodes can hold the frame similar to the equation (a).

It is really effective in dealing with discrete optimization problems such as routing and scheduling. When dealing with continuous optimization problems, it is not straightforward, and some modifications are needed. They possess the advantages over genetic algorithms and simulated annealing in dealing with dynamical network routing and task scheduling problems.

Pseudo code
Objective function *f(x)*, $x = (x_1, x_2, \ldots\ldots\ldots\ldots\ldots x_n)^T$

Encode *f(x)* into virtual nectar levels
Define dance routine (strength, direction) or protocol
while (criterion)
for loop over all *n* dimensions
(or nodes for routing and scheduling problems)
Generate new solutions
Evaluate the new solutions
end for
Communicate and update the optimal solution set
end while
Decode and output the best results.

IV.CONCLUSIONS

In this paper modern optimization techniques are explained in detail. These include: Genetic Algorithm (GA), Simulated Annealing (SA), Ant Colony optimization (ACO) and Honey Bee Algorithm (HBO). A brief description of each method is presented along with a pseudo code to facilitate their implementation. Modern optimization technique is used to solve Non Linear and non-differentiable optimization problems which are not possible to solve by traditional optimization methods.

REFERENCES

(1) S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing", *Science*, 220 (4598), 671- 680 (1983).
(2) D. H. Owlet and W. G. Macready, "No free lunch theorems for optimizaiton", *IEEE Transaction on Evolutionary Computation*, 1, 67-82 (1997).
(3) B. J. Copeland, *Alan Turing's Automatic Computing Engine*, Oxford University Press, 2005.
(4) K. De Jong, *Analysis of the Behaviour of a Class of Genetic Adaptive Systems*, PhD thesis, University of Michigan, Ann Anbor, 1975.
(5) M. Dorigo, *Optimization, Learning and Natural Algorithms*, PhD thesis, Politecnicodi Milano, Italy, 1992
(6) X. S. Yang, *Nature-Inspired Metaheuristic Algorithms*, Luniver Press, 2008
(7) G. W. Flake, *The Computational Beauty of Nature: Computer Explorations of Fractals, Chaos, Complex Systems, and Adaptation*, Cambridge, Mass.: MITPress, 1998.
(8) J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.
(9) I. Rechenberg, *Cybernetic Solution Path of an Experimental Problem*, Library Translation1122, Royal Aircraft Establishment, Farnborough, Hampshire, UK, 1965.
(10) Engineering Optimization: Theory and Practice, Fourth Edition by Singiresu S. Rao697-699, 2009.
(11) C. Blum, "Ant colony optimization: introduction and recent trends", *Physics of Life Review*, 2, 353-373 (2005).
(12) M. Dorigo, *Optimization, Learning and Natural Algorithms*, PhD thesis, Politencnicodi Milano, Italy, 1992.
(13) M. Dorigo and T. Stiitzle, *Ant Colony Optimization*, MIT Press, Cambridge, 2004.
(14) S. Nakrani and C. Tovey, "On honey bees and dynamic server allocation in Internet hosting centers", *Adaptive Behaviour*, 12, 223-240 (2004).