Practical Optimization Using Evolutionary Methods

Kalyanmoy Deb
Kanpur Genetic Algorithms Laboratory (KanGAL)
Department of Mechanical Engineering
Indian Institute of Technology Kanpur
Kanpur, PIN 208016, India
deb@iitk.ac.in
http://www.iitk.ac.in/kangal

KanGAL Report Number 2005008

Abstract

Many real-world problem solving tasks, including CFD problems, involve posing and solving optimization problems, which are usually non-linear, non-differentiable, multi-dimensional, multi-modal, stochastic, and computationally time-consuming. In this paper, we discuss a number of such practical problems which are, in essence, optimization problems and review the classical optimization methods to show that they are not adequate in solving such demanding tasks. On the other hand, in the past couple of decades, new yet practical optimization methods, based on natural evolutionary techniques, are increasingly found to be useful in meeting the challenges. These methods are population based, stochastic, and flexible, thereby providing an ideal platform to modify them to suit to solve most optimization problems. The remainder of the paper illustrates the working principles of such evolutionary optimization methods and presents some results in support of their efficacy. The breadth of their application domain and ease and efficiency of their working make evolutionary optimization methods promising for taking up the challenges offered by the vagaries of various practical optimization problems.

1 INTRODUCTION

Optimization is an activity which does not belong to any particular discipline and is routinely used in almost all fields of science, engineering and commerce. Chambers dictionary describes optimization as an act of 'making the most or best of anything'. Theoretically speaking, performing an optimization task in a problem means finding the most or best suitable solution of the problem. Mathematical optimization studies spend a great deal of effort in trying to describe the properties of such an ideal solution. Engineering or practical optimization studies, on the other hand, thrive to look for a solution which is as similar to such an ideal solution as possible. Although the ideal optimal solution is desired, the restrictions on computing power and time often make the practitioners happy with an *approximate* solution.

Serious studies on practical optimization begun as early as the second World war, when the need for efficient deployment and resource allocation of military personnel and accessories became important. Most development in the so-called 'classical' optimization field was made by developing step-by-step procedures for solving a particular type of an optimization problem. Often fundamental ideas from geometry and calculus were borrowed to reach the optimum in an iterative manner. Such optimization procedures have

enjoyed a good 50 years of research and applications and are still going strong. However, around the middle of eighties, completely unorthodox and less-mathematical yet intriguing optimization procedures have been suggested mostly by computer scientists. It is not surprising because these 'non-traditional' optimization methods exploit the fast and distributed computing machines which are getting increasingly available and affordable like slide-rules of sixties.

In this paper, we focus on one such non-traditional optimization method which takes the lion's share of all non-traditional optimization methods. This so-called 'evolutionary optimization (EO)' mimics the natural evolutionary principles on randomly-picked solutions from the search space of the problem and iteratively progresses towards the optimum point. Nature's ruthless selective advantage to *fittest* individuals and creation of new and more fit individuals using recombinative and mutative genetic processing with generations is well-mimicked artificially in a computer algorithm to be played on a search space where good and bad solutions to the underlying problem coexist. The task of an evolutionary optimization algorithm is then to avoid the bad solutions in the search space, take clues from good solutions and eventually reach close to the best solution, similar to the genetic processing in natural systems.

Like the existence of various natural evolutionary principles applied to lower and higher level species, researchers have developed different kinds of evolutionary plans resulting in a gamut of evolutionary algorithms (EAs) – some emphasizing the recombination procedure, some emphasizing the mutation operation and some using a niching strategy, whereas some using a mating restriction strategy. In this article, we only give a description of a popular approach – genetic algorithm (GA). Other approaches are also well-established and can be found from the EA literature [42, 1, 19, 29, 44, 27].

In the remainder of the paper, we describe an optimization problem and cite a number of commonly-used practical problems, including a number of CFD problems, which are in essence optimization problems. A clear look at the properties of such practical optimization problems and a description of the working principles of classical optimization methods reveal that completely different optimization procedures are in order for such problem solving. Thereafter, the evolutionary optimization procedure is described and its suitability in meeting the challenges offered by various practical optimization problems is demonstrated. The final section concludes the study.

2 AN OPTIMIZATION PROBLEM AND ITS NOTATIONS

Throughout this paper, we describe procedures for finding the optimum solution of a problem of the following type:

$$\begin{array}{rl} \text{Minimize} & f(\mathbf{x}), \\ \text{Subject to} & g_j(\mathbf{x}) \geq 0, \\ & h_k(\mathbf{x}) = 0, \\ & \mathbf{x}^{\min} \leq \mathbf{x} \leq \mathbf{x}^{\max}. \end{array} \tag{1}$$

Here, $f(\mathbf{x})$ is the objective function (of n variables) which is to be minimized. A maximization problem can be converted to a minimization problem by multiplying the function by -1. The inequality constraints g and equality constraints h demand a solution $\mathbf{x}$ to be *feasible* only if all constraints are satisfied. Many problems also require that the search is restricted within a prescribed hypervolume defined by lower and upper bound on each variable. This region is called the *search space* and the set of all feasible solutions is called *feasible space*. Usually, there exists at least one solution $\mathbf{x}^*$ in the feasible space which corresponds to the minimum objective value. This solution is called the *optimum* solution.

Thus, the task in an optimization process is to start from one or a few random solutions in the search space and utilize the function and constraint functions to drive its search towards the feasible region and finally reach near the optimum solution by exploring as small as a set of solutions as possible.

3 SCOPE OF OPTIMIZATION IN PRACTICE

Many researchers and practitioners may not know, but they often either use or required to use an optimization method. For example, in fitting a linear relationship $y = mx + c$ between an input parameter x and output parameter y through a set of n data points, we often almost blindly use the following relationships:

$$m = \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - (\sum x)^2}, \quad (2)$$

$$c = \frac{\sum y - m\sum x}{n}. \quad (3)$$

It is interesting to know that the above relationship has been derived by solving a unconstrained quadratic optimization problem of minimizing the overall vertical error between the actual and the predicted output values. In this section, we briefly discuss different practical problem solving tasks in which an optimization procedure is usually used.

Optimal design: Instead of arriving at a solution which is simply a functionally satisfactory one, an optimization technique can be used to find an optimal design which will minimize or maximize a design goal. This activity of optimization probably takes the lion's share of all optimization activities and is most routinely used. Here the decision variables can be dimensions, shapes, materials etc., which describe a design and objective function can be cost of production, energy consumption, drag, lift, reliability, stability margin etc. For example, in the optimal design of an airfoil shape for minimum drag (objective function), the shape of an airfoil can be represented by a few control parameters which can be considered as the decision variables. A minimum limit on the lift can be kept as a constraint and the airfoil area can be bounded within certain limits. Such a task will not only produce an airfoil shape providing the minimum desired lift and area, but will also cause as small a drag as possible.

Optimal control: In this activity, control parameters, as functions of time, distance, etc., are decision variables. The objective functions are usually some estimates computed at the end of process, such as the quality of end-product, time-averaged drag or lift, and stability or some other performance measure across the entire process. For example, in the design of a varying nozzle shape for achieving minimum average drag from launch to full throttle, parameters describing the shape of the nozzle as a function of time are the decision variables. To evaluate a particular solution, the CFD system must have to be solved in a time sequence from launch to full throttle and the drag experienced at every time step must have to be recorded for the computation of the average drag coefficient for the entire flight. Stability or other performance indicators must also be computed and recorded at every time step for checking if the system is safe and perform satisfactorily over the entire flight.

Modeling: Often in practice, systems involve complex processes which are difficult to express by using exact mathematical relationships. However, in such cases, either through pilot case studies or through actual plant operation, numerous input-output data are available. To understand the system better and to improve the system, it is necessary to find the relationships between input and output parameters in a process known as 'modeling'. Such an optimization task involves minimizing the error between the predicted output obtained by the developed model and actual output. To compute the predicted output, a mathematical parameterized model can be assumed based on some information about the system, such as $y = a_1 \exp(-a_2 x)$, where a_1 and a_2 are the parameters to be optimized for modeling the exponential relationship between the input x and output y. If a mathematical relationship is not known at all, a relationship can be found by a sophisticated optimization method (such as genetic programming method [29]) or by using an artificial neural network.

Scheduling: Many practical problems involve finding a permutation which causes certain objectives optimum. In these problems, the sequence of operations are of importance such as the absolute location of the operation on the permutation. For example, in a machining sequencing problem, it is important to know in what order a job will flow from one machine to the next, instead of knowing when exactly a job has to be made on a particular machine. Such optimization tasks appear in time-tabling, planning, resource allocation problems and others, and are usually known as combinatorial optimization problems.

Prediction and forecasting: Many time-varying real-world problems are often periodic and predictable. Past data for such problems can be analyzed for finding the nature of periodicity so that a better understanding of the problem can be achieved. Using such a model, future forecasts can also be made judiciously. Although such applications are plenty in financial domain, periodicities in fluidic systems, such as repeated wake formation of a particular type and its prediction of where and when will it reappear, are also of importance.

Data mining: A major task in any real-world problem solving today is to analyze multi-dimensional data and discover the hidden useful information they carry. This is by no means is an easy task, simply because it is not known what kind of information they would carry and in most problems it is not even known what kind of information one should look for. A major task in such problems is to cluster functionally similar data together and functionally dissimilar data in separate clusters. Once the entire data set is clustered, useful properties of iso-cluster data can be deciphered for a better understanding of the problem. The clustering task is an optimization problem in which the objective is usually to maximize inter-cluster distance between any pair of clusters and simultaneously minimize their intra-cluster distances. Another task often required to be performed is the identification of smallest size classifiers which would be able correctly classify most samples of the data set into various classes. Such problems often arise in bioinformatics problems [15] and can also be prevalent in other data-driven engineering tasks. In a CFD problem such a classification task should be able to find classifiers by which an investigation of a flow pattern should reveal the type of flow (laminar or turbulent) or type of fluid or kind of geometry associated with the problem.

Machine learning: In the age of automation, many real-world systems are equipped with automated and intelligent subsystems which can make decisions and adjust the system optimally, as and when an unforeseen scenario happens. In the design of such intelligent subsystems, often machine learning techniques involving different soft-computing techniques and artificial intelligence techniques are combined. To make such a system operate with a minimum change or with minimum energy requirement, an optimization procedure is needed. Since such subsystems are to be used on-line and since on-line optimization is a difficult proposition due to time restrictions, such problems are often posed as an offline optimization problem and solved by trying them on a number of synthetic (or real) scenarios [32, 33]. In such optimization tasks, an optimal rule base is learned which works the best on the chosen test scenarios. It is then hypothesized that such an optimal rule base will also work well in real scenarios. To make the optimal rule base reliable in real scenarios, such an optimization procedure can be used repeatedly with the new scenarios and new optimal rule base can be learned.

3.1 Properties of Practical Optimization Problems

Based on the above discussion, we observe that the practical optimization problems usually have the following properties:

1. They are non-smooth problems having their objectives and constraints are most likely to be non-differentiable and discontinuous.

2. Often, the decision variables are discrete making the search space discrete as well.

3. The problems may have mixed types (real, discrete, Boolean, permutation, etc.) of variables.

4. They may have highly non-linear objective and constraint functions due to complicated relationships and equations which the decision variables must form and satisfy. This makes the problems non-linear optimization problems.

5. There are uncertainties associated with decision variables, due to which the the true optimum solution may not of much importance to a practitioner.

6. The objective and constraint functions may also be noisy and non-deterministic.

7. The evaluation of objective and constraint functions is computationally expensive.

8. The problems give rise to multiple optimal solutions, of which some are globally best and many others are locally optimal.

9. The problems involve multiple conflicting objectives, for which no one solution is best with respect to all chosen objectives.

4 CLASSICAL OPTIMIZATION METHODS

Most classical point-by-point algorithms use a deterministic procedure for approaching the optimum solution. Such algorithms start from a random guess solution. Thereafter, based on a pre-specified transition rule, the algorithm suggests a search direction, which is often arrived at by considering local information. A one-dimensional search is then performed along the search direction to find the best solution. This best solution becomes the new solution and the above procedure is continued for a number of times. Figure 1 illustrates this procedure. Algorithms vary mostly in the way the search directions are defined at each intermediate solution.

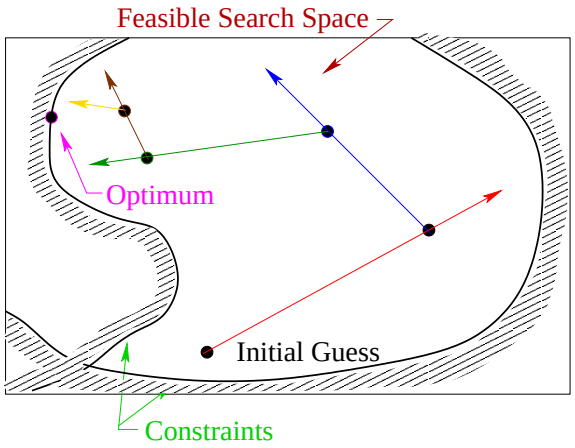

Figure 1: Most classical methods use a point-by-point approach.

Classical search and optimization methods can be classified into two distinct groups mostly in the way the directions are chosen: Direct and gradient-based methods [6, 35, 38]. In direct methods, only objective function and constraints are used to guide the search strategy, whereas gradient-based methods use the first and/or second-order derivatives of the objective function and/or constraints to guide the search process. Since derivative information is not used, the direct search methods are usually slow, requiring many function evaluations for convergence. For the same reason, they can be applied to many problems without a major change of the algorithm. On the other hand, gradient-based methods quickly converge to an optimal solution, but are not efficient in non-differentiable or discontinuous problems. In addition, there are some common difficulties with most of the traditional direct and gradient-based techniques:

- Convergence to an optimal solution depends on the chosen initial solution.
- Most algorithms are prone to get *stuck* to a sub-optimal solution.
- An algorithm efficient in solving one problem

may not be efficient in solving a different problem.

- Algorithms are not efficient in handling problems having discrete variables or highly nonlinear and many constraints.
- Algorithms cannot be efficiently used on a parallel computer.

5 MOTIVATION FROM NATURE AND EVOLUTIONARY OPTIMIZATION

It is commonly believed that the main driving principle behind the natural evolutionary process is the Darwin's survival-of-the-fittest principle [3, 18]. In most scenarios, nature ruthlessly follows two simple principles:

1. If by genetic processing an above-average offspring is created, it usually survives longer than an average individual and thus have more opportunities to produce offspring having some of its traits than an average individual.
2. If, on the other hand, a below-average offspring is created, it usually does not survive longer and thus gets eliminated quickly from the population.

The principle of emphasizing good solutions and eliminating bad solutions seems to dovetail well with desired properties of a good optimization algorithm. But one may wonder about the real connection between an optimization procedure and natural evolution! Has the natural evolutionary process tried to maximize a utility function of some sort? Truly speaking, one can imagine a number of such functions which the nature may have been thriving to maximize: life span of a species, quality of life of a species, physical growth, and others. However, any of these functions is non-stationary in nature and largely depends on the evolution of other related species. Thus, in essence, the nature has been optimizing a much more complicated objective function by means of natural genetics and natural selection than the search and optimization problems we are interested in solving in practice. Thus, it is not surprising that the computerized evolutionary optimization (EO) algorithm is not as complex as the natural genetics and selection procedures, rather it is an abstraction of the complex natural evolutionary process. Although an EO is a simple abstraction, it is robust and has been found to solve various search and optimization problems of science, engineering, and commerce.

5.1 Evolutionary Optimization

The idea of using evolutionary principles to constitute a computerized optimization algorithm was suggested by a number of researchers located in geographically distant places across the globe. The field now known as 'genetic algorithms' was originated by John Holland of University of Michigan [24], a contemporary field 'evolution strategy' was originated by Ingo Rechenberg and Hans-Paul Schwefel of Technical University of Berlin [37, 41], the field of 'evolutionary programming' was originated by Larry Fogel [20], and others. Figure 2 provides an overview of computational intelligence and its components. Also, different subfields of evolutionary computing are also shown. In this paper, we mainly discuss the principles of a genetic algorithm and its use in different optimization problem solving.

Genetic algorithm (GA) is an iterative optimization procedure. Instead of working with a single solution in each iteration, a GA works with a number of solutions (collectively known as a population) in each iteration. A flowchart of the working principle of a simple GA is shown in Figure 3. In the absence of any knowledge of the problem domain, a GA begins its search from a random population of solutions. If a termination criterion is not satisfied, three different operators – reproduction and variation operators (crossover and mutation, and others) – are applied to update the population of solutions. One iteration of these operators is known as a *generation* in the parlance of GAs. Since the representation of a solution in a GA is similar to a natural chromosome and GA operators are similar to genetic operators, the above

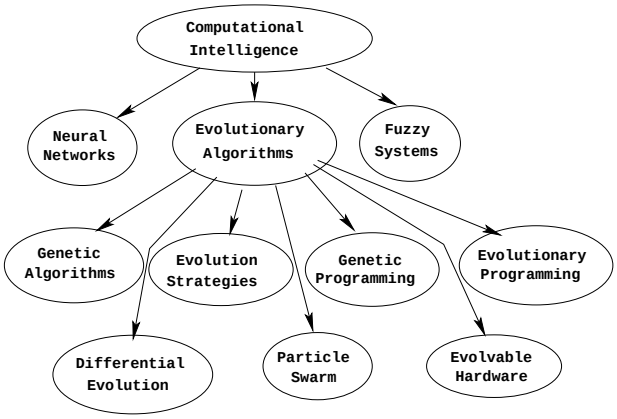

Figure 2: Computational intelligence and evolutionary computation.

procedure is called a *genetic algorithm*.

5.1.1 Step 1: Representation of a solution

Representation of a solution describing the problem is an important first step in a GA. The solution vector $\mathbf{x}$ can be represented as a vector of real numbers, discrete numbers, a permutation of entities or others or a combination, as suitable to the underlying problem. If a problem demands mixed real and discrete variables, a GA allows such a representation of a solution.

5.1.2 Step 2: Initialization of a population of solutions

Usually, a set of random solutions (of size N) are initialized in a pre-defined search space ($x_i^{(L)} \leq x_i \leq x_i^{(U)}$). However, it is not necessary that the subsequent GA operations are confined to create solutions in the above range. To make sure that initial population is well-distributed, any space-filling or Latin-hypercube method can also be used. It is also not necessary that the initial population is created randomly on the search space. If some problem information is available (such as knowledge about good solutions), a biased distribution can be created in any suitable portion of the search space. In difficult problems with a known good solutions, the population can be created around the good solution by randomly perturbing it [5].

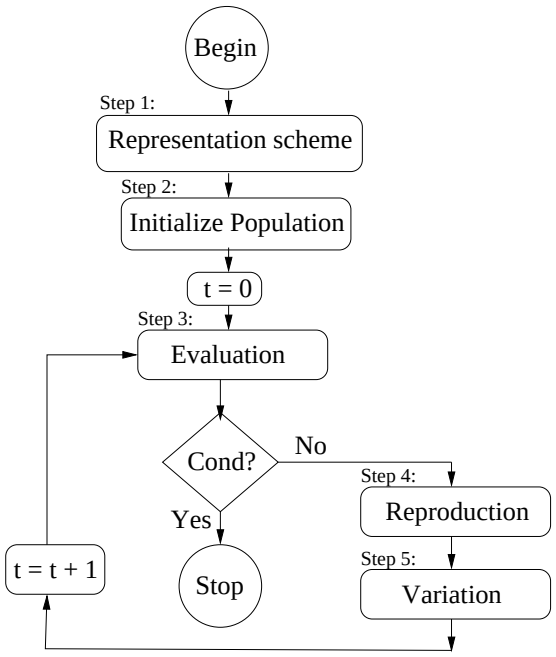

Figure 3: A flowchart of working principle of a genetic algorithm.

5.1.3 Step 3: Evaluation of a solution

In this step, every solution is evaluated and a fitness value is assigned to the solution. This is where the solution is checked for its feasibility (by computing and checking constraint functions g_j and h_k). By simply assigning a suitable fitness measure, a feasible can be given more importance over an infeasible solution or a better feasible solution can be given more importance over a worse feasible solution. This is also the place where a single-objective GA dealing with a single objective function can be converted to a multi-objective GA in which multiple conflicting objectives can be dealt with. In most practical optimization problems, this is also the most time-consuming step. Hence,

any effort to complete this step quickly (either by using distributed computers or approximately) would provide a substantial saving in the overall procedure.

The evaluation step requires to first decipher the solution vector from the chosen representation scheme. If the variables are represented directly as real or discrete numbers, they are already directly available. However, if a binary substring is used to represent some discrete variables, first the exact variable value must be computed from the substring. For example, the following is a string, representing n variables:

$$\underbrace{11010}_{x_1}\ \underbrace{1001001}_{x_2}\ \underbrace{010}_{x_3}\ \ldots\ \underbrace{0010}_{x_n}$$

The i-th problem variable is coded in a binary substring of length ℓ_i, so that the total number of alternatives allowed in that variable is 2^{ℓ_i}. The lower bound solution $x_i^{\min}$ is represented by the solution `(00...0)` and the upper bound solution $x_i^{\max}$ is represented by the solution `(11...1)`. Any other substring s_i decodes to a solution x_i as follows:

$$x_i = x_i^{\min} + \frac{x_i^{\max} - x_i^{\min}}{2^{\ell_i} - 1}\mathrm{DV}(s_i), \qquad (4)$$

where $\mathrm{DV}(s_i)$ is the decoded value of the substring s_i. Such a representation of discrete variables (even real-parameter variables) has a traditional root and also nature root. A binary string resembles a chromosome comprising of a number of genes taking one of two values – one or zero. Treating these strings as individuals, we can then think of mimicking natural crossover and mutation operators similar to their natural counterparts.

To represent a permutation (of n integers), an innovative coded representation procedure can be adopted so that simple genetic operators can be applied on the coding. In the proposed representation [13], at the i-th position from the left of the string, an integer between zero and $(i-1)$ is allowed. The string is decoded to obtain the permutation as follows. The i-th position denotes the placement of component i in the permutation. The decoding starts from the left-most position and proceeds serially towards right. While decoding the i-th position, the first $(i-1)$ components are already placed, thereby providing with i place-holders to position the i-th component. We illustrate this coding procedure by using an example having six components (`a` to `f`) using six integers (0 to 5). Let us say that we have a string

```
(0 0 2 1 3 2)
```

and would like to decipher the corresponding permutation. The second component ($i = 2$) has two places to be positioned – (i) before the component `a` or (ii) after the component `a`. The first case is denoted by a `0` and the second case is denoted by a `1` in the string. Since the place-holder value for $i = 2$ (component `b`) is zero in the above string, the component `b` appears before the component `a`. Similarly, with the first two components placed as `(b a)`, there are three place-holders for the component `c` – (i) before `b` (with a value `0`), (ii) between `b` and `a` (with a value `1`), and (iii) after `a` (with a value `2`). Since the string has a value `2` in the third place, component `c` is placed after `a`. Continuing in this fashion, we obtain the permutation

```
(b d f a e c)
```

corresponding to the above string representation. The advantage of this coding is that simple crossover and mutation operators (to be discussed later) can be applied on the coding to create valid permutations.

5.1.4 Step 4: Reproduction operator

Reproduction (or selection) is usually the first operator applied to the population. Reproduction selects good strings in a population and forms a mating pool. The essential idea is to emphasize above-average strings in the population. The so-called binary *tournament* reproduction operator picks two solutions at random from the population and the better of the two is chosen according to its fitness value. Although a deterministic procedure can be chosen for this purpose (for example, the best 50% population members can be duplicated), usually a stochastic procedure is adopted in an EA to reduce the chance of getting trapped into a local optimal solution.

5.1.5 Step 5: Variation operators

During the selection operator, good population members are emphasized at the expense of bad population members, but no new solution is created. The purpose of variation operators is just to create new and hopefully improved solutions by using the mating pool. A number of successive operations can be used here. However, there are two main operators which are mostly used.

In a *crossover* operator, two solutions are picked from the mating pool at random and an information exchange is made between the solutions to create one or more offspring solutions. In a single-point crossover operator applied to binary strings, both strings are cut at an arbitrary place and the right-side portion of both strings are swapped among themselves to create two new strings:

$$\begin{array}{cc|ccc} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{array} \quad \Rightarrow \quad \begin{array}{ccccc} 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \end{array}$$

It is true that every crossover between any two solutions from the new population is not likely to find offspring better than both parent solutions, but the chance of creating better solutions is far better than random [21]. This is because the parent strings being crossed are not any two arbitrary random strings. These strings have survived tournaments played with other solutions during the earlier reproduction phase. Thus, they are expected to have some good bit combinations in their string representations. Since, a single-point crossover on a pair of parent strings can only create ℓ different string pairs (instead of all $2^{\ell-1}$ possible string-pairs) with bit combinations from either strings, the created offspring are also *likely* to be good strings. To reduce the chance of losing too many good strings by this process, usually a pair of solutions are participated in a crossover operator with a crossover probability, p_c. Usually, a large value within $[0.7, 1]$ is chosen for this parameter. A value of $p_c = 0.8$ means that 80% population members participate in crossovers to create new offspring and the 20% parents are directly accepted as offspring.

The *mutation* operator perturbs a solution to its vicinity with a small mutation probability, p_m. For example, in a binary string of length ℓ, a `1` is changed to a `0` and vice versa, as happened in the fourth bit of the following example:

$$0 \quad 0 \quad 1 \quad \mathbf{1} \quad 1 \quad \Rightarrow \quad 0 \quad 0 \quad 1 \quad \mathbf{0} \quad 1$$

Usually, a small value of $p_m \approx 1/\ell$ or $1/n$ (n is the number of variables) is used. Once again, the mutation operation uses random numbers but it is not an entirely random process, as from a particular string it is not possible to move to any other string in one mutation event. Mutation uses a biased distribution to be able to move to a solution *close* to the original solution. The basic difference between the crossover and mutation operations is that in the former more than one parent solutions are needed, whereas in the later only one solution is used directly or indirectly.

After reproduction and variation operators are applied to the whole population, one generation of a GA is completed. These operators are simple and straightforward. Reproduction operator selects good strings and crossover operator recombines good substrings from two good strings together to hopefully form a better substring. Mutation operator alters a string locally to hopefully create a better string. Even though none of these claims are guaranteed and/or tested while creating a new population of strings, it is expected that if bad strings are created they will be eliminated by the reproduction operator in the next generation and if good strings are created, they will be retained and emphasized.

Although the above illustrative discussion on the working of a genetic algorithm may seem to be a computer-savvy, nature-hacker's game-play, there exists rigorous mathematical studies where a complete processing of a finite population of solutions under GA operators is modeled by using Markov chains [45], using statistical mechanics approaches [34], and using approximate analysis [22]. On different classes of functions, the recent dynamical systems analysis [46] treating a GA as a random heuristic search reveals the complex dynamical behavior of a binary-coded GA with many meta-stable attractors. The interesting outcome is the effect of population size on the degree of *meta-stability*. In the statistical mechanics approach, instead of considering microscopic details of the evolving system, several macroscopic variables

describing the system, such as mean, standard deviation, skewness and kurtosis of the fitness distribution, are modeled. Analyzing different GA implementations with the help of these cumulants provides interesting insights into the complex interactions among GA operators [39]. Rudolph has shown that a simple GA with an elite-preserving operator and nonzero mutation probability converges to the global optimum solution to any optimization problem [40]. Leaving the details of the theoretical studies (which can be found from the growing EA literature), here we shall illustrate the robustness (combined breadth and efficiency) of a GA-based optimization strategy in solving different types of optimization problems commonly encountered in practice.

5.2 Real-Parameter Genetic Algorithms

When decision variables in an optimization problem are real-valued, they can be directly represented in a GA as mentioned above, instead of a binary representation. Such real-parameter GAs came to light in nineties, when engineering applications of GAs gained prominence. Although the GA flowchart remains the same, the crossover and mutation operations suitable to real-valued parameters were needed to be developed.

Most studies used variable-wise crossover and mutation operators in the past. Recently, vector-wise operators are suggested. Mutation operator perturbs a parent solution in its vicinity by using a Gaussian-like probability distribution. However, sophisticated crossover operators are also suggested. We proposed a parent-centric crossover (PCX) operator [9] which uses three or more parents and construct a probability distribution for creating offspring solutions. The interesting feature of the probability distribution is that it is defined based on the vector-difference of parent solutions, instead of parent solutions themselves. Figure 4 shows the distribution of offspring around parents under the PCX operator.

Since offspring solutions result in a diversity proportional to that in parent population, such a crossover operator introduces a self-adaptive feature, which allows a broad search early on and a focused search later in an automatic manner. Such a property is desired in an efficient optimization procedure and is present in other real-parameter EAs, such as evolution strategy (ES) [42], differential evolution [44] and particle swarm optimization [27].

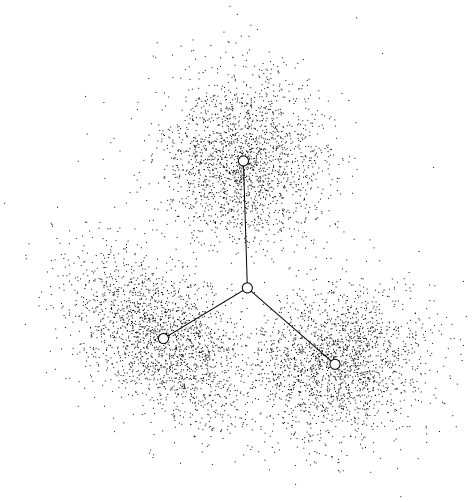

Figure 4: Offspring created using the PCX operator.

On a 20-variable non-linear unconstrained minimization problem

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} \left(100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2\right), \qquad (5)$$

a GA with the PCX operator [9] is reported to find the optimum solution with an error of 10^{-20} in function value in the smallest number of function evaluations compared to a number of other real-parameter EAs ((μ, λ)-ES, CMA-ES [23] and differential evolution (DE)) and the classical quasi-Newton (BFGS) approach (which got stuck to a solution having $f = 6.077(10^{-17})$).

Method	Best	Median	Worst
GA+PCX	16,508	21,452	25,520
$(1, 10)$-ES	591,400	803,800	997,500
CMA-ES	29,208	33,048	41,076
DE	243,800	587,920	942,040
BFGS	26,000		

In the following sections, we discuss a few practical optimization problems and how their (approximate) solution can be made easier with an evolutionary optimization strategy.

6 NON-SMOOTH OPTIMIZATION PROBLEMS

Many practical problems are non-smooth in nature, introducing non-differentiabilities and discontinuities in the objective and constraint functions. In such special points of non-smoothness, the exact derivatives do not usually exist. Although numerical gradients can be computed, they can often be meaningless, particularly if such non-smoothness occurs on or near optimum points. Thus, it becomes difficult for gradient-based optimization methods to work well in such scenarios. In a GA, such non-smoothness is not a matter, as a GA does not use any gradient information during its search towards the optimum. As shown in Figure 5, as long as a descent information is provided by the function values (by the way of comparing function values of two solutions), a GA is ready to work its way towards the optimum.

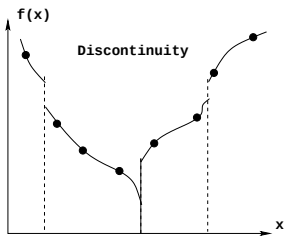

Figure 5: Discontinuous search space.

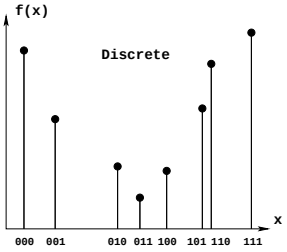

Figure 6: Discrete search space.

In many problems, variables take discrete values, thereby providing no meaningful way of computing gradients and using the gradient-based classical methods. Once again, the sole purpose of arriving at descent information using function value comparisons allows a GA to be used in such discrete search space as well (Figure 6).

6.1 MIXED (REAL AND DISCRETE) OPTIMIZATION PROBLEMS

In an EO, decision variables can be coded in either binary strings or directly depending on the type of the variable. A zero-one variable can be coded using a single bit (`0` or `1`). A discrete variable can be coded in either a binary string or directly (if the total number of permissible choices for the variable is not 2^k, where k is an integer). A continuous variable can be coded directly. This coding allows a natural way to code different variables, as depicted in the following solution representing a complete design of a cantilever beam having four design variables:

$$((1) \quad 15 \quad 23.457 \quad (1011))$$

The first variable represents the shape of the cross-section of the beam. There are two options—circular (a `1`) or square (a `0`). Thus, its a zero-one variable. The second variable represents the diameter of the circular section if the first variable is a `1` or the side of the square if the first variable is a `0`. This variable takes only one of a few pre-specified values. Thus, this variable is a discrete variable coded directly. The third variable represents the length of the cantilever beam, which can take any real value. Thus, it is a continuous variable. The fourth variable is a discrete variable representing the material of the cantilever beam. This material takes one of 16 pre-specified materials. Thus, a four-bit substring is required to code this variable. The above solution represent the 12th material from a pre-specified list. Thus, the above string represents a cantilever beam made of the 12th material from a prescribed list of 16 materials having a circular cross-section with a diameter 15 mm and having a length of 23.457 mm. With the above coding, any combination of cross sectional shape and size, material specifications, and length of the cantilever beam can be represented. This flexibility in the representation of a design solution is not possible with traditional optimization methods. This flexibility makes an EO efficient to be used in many engineering design problems [11, 16], including CFD problems involving shapes etc.

7 CONSTRAINT OPTIMIZATION

Constraints are inevitable in real-world (practical) problems. The classical penalty function approach of penalizing an infeasible solution is sensitive to a penalty parameter associated with the technique and often requires a trial-and-error method. In the penalty function method for handling inequality constraints in minimization problems, the fitness function $F(\vec{x})$ is defined as the sum of the objective function $f(\vec{x})$ and a penalty term which depends on the constraint violation $\langle g_j(\vec{x})\rangle$:

$$F(\vec{x}) = f(\vec{x}) + \sum_{j=1}^{J} R_j \langle g_j(\vec{x})\rangle^2, \tag{6}$$

where $\langle\ \rangle$ denotes the absolute value of the operand, if the operand is negative and returns a value zero, otherwise. The parameter R_j is the penalty parameter of the j-th inequality constraint. The purpose of a penalty parameter R_j is to make the constraint violation $g_j(\vec{x})$ of the same order of magnitude as the objective function value $f(\vec{x})$. Equality constraints are usually handled by converting them into inequality constraints as follows:

$$g_{k+J}(\vec{x}) \equiv \delta - |h_k(\vec{x})| \geq 0,$$

where δ is a small positive value.

In order to investigate the effect of the penalty parameter R_j (or R) on the performance of GAs, we consider a well-studied welded beam design problem [38]. The resulting optimization problem has four design variables $\vec{x} = (h, \ell, t, b)$ and five inequality constraints:

$$\begin{array}{ll} \text{Min.} & f_w(\vec{x}) = 1.10471h^2\ell + 0.04811tb(14.0+\ell), \\ \text{s.t.} & g_1(\vec{x}) \equiv 13,600 - \tau(\vec{x}) \geq 0, \\ & g_2(\vec{x}) \equiv 30,000 - \sigma(\vec{x}) \geq 0, \\ & g_3(\vec{x}) \equiv b - h \geq 0, \\ & g_4(\vec{x}) \equiv P_c(\vec{x}) - 6,000 \geq 0, \\ & g_5(\vec{x}) \equiv 0.25 - \delta(\vec{x}) \geq 0, \\ & 0.125 \leq h \leq 10, \\ & 0.1 \leq \ell, t, b \leq 10. \end{array} \tag{7}$$

The terms $\tau(\vec{x})$, $\sigma(\vec{x})$, $P_c(\vec{x})$, and $\delta(\vec{x})$ are given below:

$$\begin{aligned} \tau(\vec{x}) &= \sqrt{\left[(\tau'(\vec{x}))^2 + (\tau''(\vec{x}))^2 + \right.} \\ &\qquad \left. \ell\tau'(\vec{x})\tau''(\vec{x})/\sqrt{0.25(\ell^2+(h+t)^2)}\right], \\ \sigma(\vec{x}) &= \frac{504,000}{t^2 b}, \\ P_c(\vec{x}) &= 64,746.022(1-0.0282346t)tb^3, \\ \delta(\vec{x}) &= \frac{2.1952}{t^3 b}, \end{aligned}$$

where

$$\begin{aligned} \tau'(\vec{x}) &= \frac{6,000}{\sqrt{2}h\ell}, \\ \tau''(\vec{x}) &= \frac{6,000(14+0.5\ell)\sqrt{0.25(\ell^2+(h+t)^2)}}{2\left\{0.707h\ell(\ell^2/12+0.25(h+t)^2)\right\}}. \end{aligned}$$

The optimized solution reported in the literature [38] is $h^* = 0.2444$, $\ell^* = 6.2187$, $t^* = 8.2915$, and $b^* = 0.2444$ with a function value equal to $f^* = 2.38116$. Binary GAs are applied on this problem in an earlier study [4] and the solution $\vec{x} = (0.2489, 6.1730, 8.1789, 0.2533)$ with $f = 2.43$ (within 2% of the above best solution) was obtained with a population size of 100. However, it was observed that the performance of GAs largely dependent on the chosen penalty parameter values, as shown in Table 1. With $R = 1$ (small values of R), although 12 out of 50 runs have found a solution within 150% of the best-known solution, 13 EO runs have not been able to find a single feasible solution in 40,080 function evaluations. This happens because with small R there is not much pressure for the solutions to become feasible. With larger penalty parameters, the pressure for solutions to become feasible is more and all 50 runs found feasible solutions. However, because of larger emphasis of solutions to become feasible, when a particular solution becomes feasible it has a large selective advantage over other solutions (which are infeasible) in the population and the EO is unable to reach near to the true optimal solution.

In the recent past, a parameter-less penalty approach is suggested by the author [7], which uses the following fitness function derived from the objective

Table 1: Number of runs (out of 50 runs) converged within ϵ% of the best-known solution using an EO with different penalty parameter values and using the proposed approach on the welded beam design problem.

	ϵ			Optimized $f_w(\vec{x})$		
R	$\leq$ 150%	$>$ 150%	Infes.	Best	Med.	Worst
10^0	12	25	13	2.413	7.625	483.502
10^1	12	38	0	3.142	4.335	7.455
10^3	1	49	0	3.382	5.971	10.659
10^6	0	50	0	3.729	5.877	9.424
Prop.	50	0	0	2.381	2.383	2.384

and constraint values:

$$F(\mathbf{x}) = \begin{cases} f(\mathbf{x}), & \text{if } \mathbf{x} \text{ is feasible;} \\ f_{\max} + \sum_{j=1}^{J} \langle g_j(\mathbf{x}) \rangle + \sum_{k=1}^{K} |h_k(\mathbf{x})|, & \text{else.} \end{cases} \quad (8)$$

In a tournament between two feasible solutions, the first clause ensures that the one with a better function value wins. The quantity $f_{\max}$ is the objective function value of the worst feasible solution in the population. The addition of these quantity to the constraint violation ensures that a feasible solution is always better than any infeasible solution. Moreover, since this quantity is constant in any generation, between two infeasible solutions the one with smaller constraint violation is judged better. Since the objective function value is *never* compared with a constraint violation amount, there is no need of any penalty parameter with such an approach.

When the proposed constraint handling EO is applied to the welded beam design problem, all 50 simulations (out of 50 runs) are able to find a solution very close to the true optimal solution, as shown in the last row of Table 1. This means that with the proposed GAs, one run is enough to find a satisfactory solution close to the true optimal solution.

8 ROBUST AND RELIABILITY-BASED OPTIMIZATION

For practical optimization studies, robust and reliability-based techniques are commonplace and are getting increasingly popular. Often in practice, the mathematical optimum solution is not desired, due to its sensitivity to the parameter fluctuations and inaccuracy in the formulation of the problem. Consider Figure 7, in which although the global minimum is at B, this solution is very sensitive to parameter fluctuations. A small error in implementing solution B will result in a large deterioration in the function value. On the other hand, solution A is less sensitive and more suitable as the desired solution to the problem. The dashed line is an average of the function around

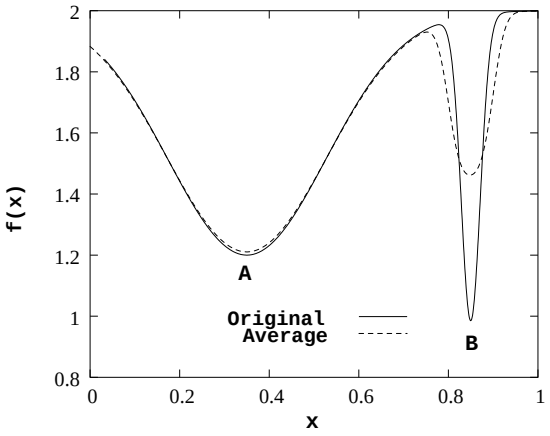

Figure 7: Although solution B is the global minimum, it is a robust solution. Solution A is the robust solution.

a small region near a solution. If the function shown in dashed line is optimized the robust solution can be achieved [2, 26, 12].

For a canonical deterministic optimization task, the optimum solution usually lies on a constraint surface or at the intersection of more than one constraint surfaces. However, if the design variables or some system parameters cannot be achieved exactly and

are uncertain with a known probability distribution of variation, the so-called deterministic optimum (lying on one or more constraint surface) will fail to remain feasible in many occasions (Figure 8). In such

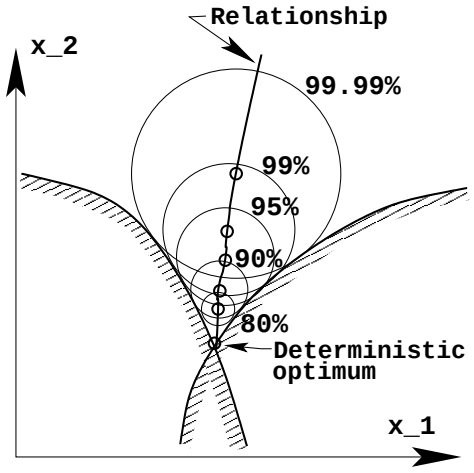

Figure 8: The constrained minimum is not reliable. With higher desired reliability, the corresponding solutions moves inside the feasible region.

scenarios, a stochastic optimization problem is usually formed and solved, in which the constraints are converted into probabilistic constraints meaning that probability of failures (of being a feasible solution) is limited to a pre-specified value (say ϵ) [36, 17]. The advantage of using an EO is that a global robust solution can be obtained and the method can be extended for finding multi-objective reliable solutions easily [28].

9 OPTIMIZATION WITH DISTRIBUTED COMPUTING

One way to beat the difficulties with problems having computationally expensive evaluation procedures is a distributed computing environment. This way the evaluation procedure can be parallelized and overall computational time will reduce. There is an additional advantage with using an EO. Since an EO deals with a population of solutions in every generation and their evaluations are independent to each other, each population member can be evaluated on a different processor independently and parallely. Homogeneous or heterogeneous processors on a master-slave configuration can be used for such a purpose equally well to any other arrangements. Since the computational time for solution evaluation is comparatively much more than the time taken by the genetic operators, a fast input-output processing is also not a requirement. Thus, a hand-made Beowulf cluster formed with a number of fast processors can be used efficiently to run an EO. This feature makes an EO suitable for parallel processors compared to their classical counterparts.

Ideally, the evaluation of each solution must also be parallelized, if deemed necessary. For this purpose, several clusters can be combined together and a GA population member can be sent to each cluster for its efficient evaluation, as shown in Figure 9.

10 COMPUTATIONALLY DIFFICULT OPTIMIZATION PROBLEMS

Many engineering optimization problems involve objectives and constraints which require an enormous computational time. For example, in CFD control problems involving complicated shapes and geometries, every solution requires mesh generation, automated node-numbering, and solution of the finite difference equations. The computational time in such problems can easily consume a few minutes to days for evaluating a single solution on a parallel computer, depending the rigor of the problem. Clearly, optimization of such problems is still out of scope in its traditional sense and often innovations in using the optimization methodology must be devised. Here, we suggest a couple of approaches:

1. Approximate models for evaluation instead of

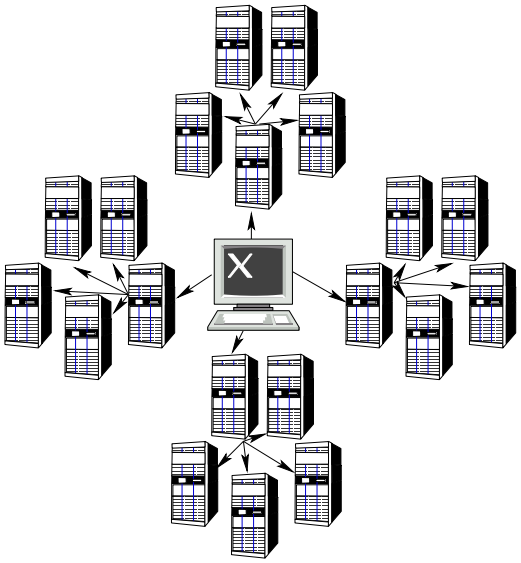

Figure 9: A parallel computing environment for an ideal EO.

exact objective and constraint functions can be used.

2. For time-varying control problems, a fast yet approximate optimization principle can be used.

10.1 Approximate Models

In this approach, an approximate model of the objective and constraint functions are first developed using a handful of solutions evaluated exactly. In this venture, a response surface methodology (RSM), a kriging methodology or an ANN approach can all be used. Since the model is at best assumed to be an approximate one, it must be used for a while and discarded when solutions begins to crowd around the best region of the approximate model. At this stage, another approximate model can be developed near the reduced search region dictated by the current population. One such coarse-to-fine grained approximate modeling technique (with an ANN approach) has been recently developed for multi-objective optimization [31] and a number of test problems and two practical problems are solved. A saving of 30 to 80% savings in the computational time has been recorded by using the proposed approach.

10.2 Approximate Optimization Principle

In practical optimal control problems involving time-varying decision variables, the objective and constraint functions need to be computed when the whole process is simulated for a time period. In CFD problems, such control problems often arise in which control parameters such as velocities and shapes need to be changed over a time period in order to obtain a minimum drag or a maximum lift or a maximally stable system. The computation of such objectives is a non-linear process of solving a system of governing partial differential equations sequentially from the initial time to a final time. When such a series of finite difference equations are solved for a particular solution describing the system, one evaluation is over. Often, such an exact evaluation procedure may take a few hours to a few days. Even if only a few hundred solutions are needed to be evaluated to come any where closer to the optimum, this may take a few months on even a moderately fast parallel computer. For such problems, an approximate optimization procedure can be used as follows [43].

The optimal control strategy for the entire time span $[t_i, t_f]$ is divided into K time intervals $[t_k, t_k + \Delta t]$, such that $t_0 = t_i$ and $t_K = t_{K-1} + \Delta t = t_f$. At the k-th time span, the GA is initialized by mutating the best solution found in $(k-1)$-th time span. Thereafter, the GA is run for τ generations and the best solution is recorded. The control strategy corresponding to this best solution is assigned as the overall optimized control strategy for this time span. This is how the optimized control strategy gets formed as the time span increases from t_0 to t_f. This proposed genetic search procedure is fast ($O(K)$ compared to $O(K^2)$) and allows an approximate way to handle computationally expensive time-varying optimization problems, such as the CFD problem solved elsewhere [43]. Although the procedure is fast, on the flip side, the proposed search procedure constitutes an approximate search and the resulting optimized

solution need not be the true optimum of the optimal control problem. But with the computing resources available today, demanding CFD simulations prohibit the use of a standard optimization algorithm in practice. The above approximate optimization procedure allows a viable way to apply optimization algorithms in computationally demanding CFD problems.

11 MULTI-MODAL OPTIMIZATION

Many practical optimization problems possess a multitude of optima – some global and some local. In such problems, it may be desirable to find as many optima as possible to have a better understanding of the problem. Due to the population approach, GAs can be used to find multiple optimal solutions in one simulation of a single GA run. In one implementation [10], the fitness of a solution is degraded with its *niche count*, an estimate of the number of neighboring solutions. It has been shown that if the reproduction operator is performed with the degraded fitness values, stable subpopulations can be maintained at various optima of the objective function. Figure 10 shows that a niched-GA can find all five optima (albeit a mix of local and global optima) in one single simulation run.

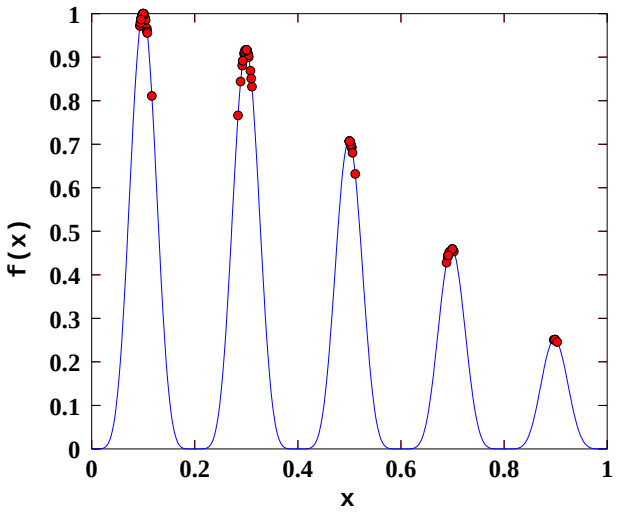

Figure 10: A niched-GA finds multiple optimal solutions.

12 MULTI-OBJECTIVE EVOLUTIONARY OPTIMIZATION

Most real-world search and optimization problems involve multiple conflicting objectives, of which the user is unable to establish a relative preference. Such considerations give rise to a set of multiple optimal solutions, commonly known as the Pareto-optimal solutions [8]. Classical approaches to solve these problems concentrate mainly in developing a single composite objective function from multiple objectives and in using a single-objective optimizer to find a particular optimum solution [30]. Such procedures are subjective to the user, as the optimized solution depends on the chosen scalarization scheme. Once again, due to the population approach of a GA, multiple Pareto-optimal solutions are found simultaneously in a single simulation run, making it an unique way to handle multi-objective optimization problems.

To convert a single-objective GA to a multi-objective optimizer, three aspects are kept in mind: (i) *non-dominated* solutions are emphasized for progressing towards the optimal front, (ii) *less-crowded* solutions are emphasized to maintain a diverse set of solutions, and (iii) *elite* or best solutions in a population are emphasized for a quicker convergence near the true Pareto-optimal front. In one such implementation – elitist non-dominated sorting GA or NSGA-II (which received the ESI Web of Science's recognition as the Fast Breaking Paper in Engineering in February 2004), parent and offspring populations are combined together and non-dominated solutions are hierarchically selected starting from the best solutions. A crowding principle is used to select the remaining solutions from the last front which could not be selected in total. These operations follow the above three aspects and the algorithm has been successful in solving a wide variety of problems. A sketch

of the algorithm is shown in Figure 11.

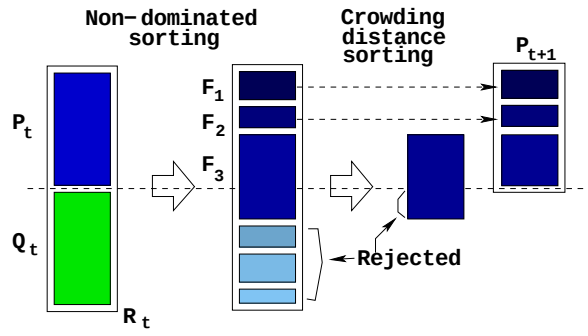

Figure 11: An iteration of the NSGA-II procedure.

On a 18-speed gearbox design problem having 28 mixed decision variables, three objectives, and 101 nonlinear constraints, the NSGA-II is able to find as many as 300 different trade-off solutions including individual minimum solutions (Figure 12). It is clear from the figure that if a large error (ϵ) in the output shaft speeds from the ideal is permitted, better non-dominated solutions are expected. Because of the mixed nature of variables and multiple objectives, a previous study [25] based on classical methods had to divide the problem in two subproblems and even then it failed to find a wide spectrum of solutions as found by NSGA-II. Since the number of teeth in gears are also kept as variables here, a more flexible search is permitted, thereby obtaining a better set of non-dominated solutions.

12.1 INNOVIZATION: Understanding the Optimization Problem Better

Besides finding the trade-off optimal solutions, evolutionary multi-objective optimization (EMO) studies are getting useful for another reason. When a set of trade-off solutions are found, they can be analyzed to discover any useful similarities and dissimilarities among them. Such information, particularly the common principles, if any, should provide useful insight to a designer, user, or a decision-maker.

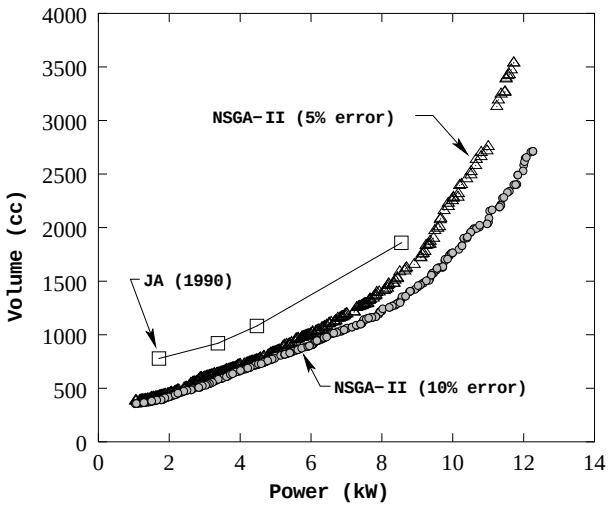

Figure 12: NSGA-II solutions are shown to outperform a classical approach with two limiting errors in output speeds.

For example, for the gearbox design problem described above, when we analyze all 300 solutions, we find that the only way these optimal solutions vary from each other is by having a drastically different value in only one of the variables (module of the gear) and all other variables (thickness and number of teeth of each gear) remain more or less the same for all solutions. Figure 13 shows how module (m) is changing with one of the objectives (power delivered (p) by the gearbox). When a curve is fitted through these points, the following mathematical relationship emerges: $m = \sqrt{p}$. This is an useful information for a designer to have. What this means is that if a gearbox is designed today for $p = 1$ kW power requirement and tomorrow if a gearbox is needed to be designed for another application requiring $p = 4$ kW the only change needed in the gearbox is an increase of module by a factor of two. A two-fold increase in module will increase the diameter of gears and hence the size of the gearbox. Although a complete re-optimization can be performed from scratch and there may be other ways the original gearbox can be modified for the new requirement, but such a

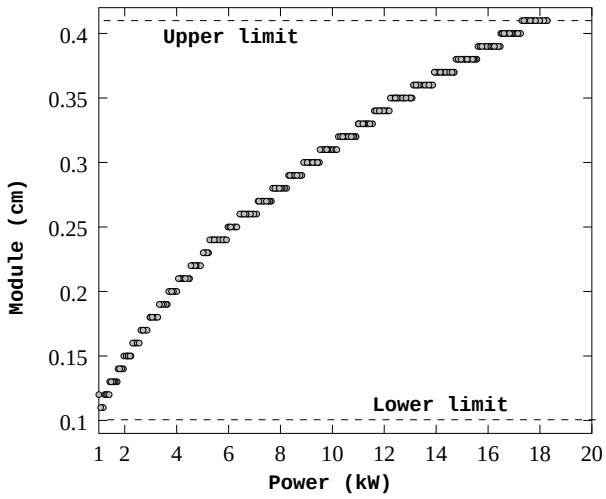

Figure 13: An optimal relationship between module and power is discovered by the NSGA-II.

simple change in module alone will mean a redesign of the gearbox for the new requirement in an optimal manner and importantly not requiring any further computation [14]. Such a task brings out a 'recipe' or 'blue-print' of solving the problem optimally. It is not clear how such useful information can be achieved by any other means.

13 CONCLUSIONS

In this paper, we have presented a number of commonly-used practical problems which are, in essence, optimization problems. A closer look at these problems has revealed that such optimization problems are complex, non-linear, and multi-dimensional, thereby making the classical optimization procedures inadequate to be used with any reliability and confidence. We have also presented the evolutionary optimization procedure – a population-based iterative procedure mimicking natural evolution and genetics, which has demonstrated its suitability and immense applicability in solving various types of optimization problems. A particular implementation – genetic algorithm (GA) – is different from classical optimization methods in a number of ways: (i) it does not use gradient information, (ii) it works with a set of solutions instead of one solution in each iteration, (iii) it is a stochastic search and optimization procedure and (iv) it is highly parallelizable.

Besides GAs, there exist a number of other implementations of the evolutionary idea, such as evolution strategy [42, 1], evolutionary programming [19], genetic programming [29], differential evolution [44], particle swarm optimization [27] and others. Due to the flexibilities in their search, these evolutionary algorithms are found to be quite useful as a search tool in studies on artificial neural networks, fuzzy logic computing, data mining, and other machine learning activities.

Acknowledgment

The author acknowledges the efforts of all his collaborators during the past 13 years of research and application in developing efficient evolutionary optimization methods for practical problem solving.

References

[1] H.-G. Beyer. *The theory of evolution strategies*. Berlin, Germany: Springer, 2001.

[2] J. Branke. Creating robust solutions by means of an evolutionary algorithm. In *Proceedings of the Parallel Problem Solving from Nature (PPSN-V)*, pages 119–128, 1998.

[3] R. Dawkins. *The Selfish Gene*. New York: Oxford University Press, 1976.

[4] K. Deb. Optimal design of a welded beam structure via genetic algorithms. *AIAA Journal*, 29(11):2013–2015, 1991.

[5] K. Deb. Genetic algorithms in optimal optical filter design. In *Proceedings of the International Conference on Computing Congress*, pages 29–36, 1993.

[6] K. Deb. *Optimization for Engineering Design: Algorithms and Examples*. New Delhi: Prentice-Hall, 1995.

[7] K. Deb. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2–4):311–338, 2000.

[8] K. Deb. *Multi-objective optimization using evolutionary algorithms*. Chichester, UK: Wiley, 2001.

[9] K. Deb, A. Anand, and D. Joshi. A computationally efficient evolutionary algorithm for real-parameter optimization. *Evolutionary Computation Journal*, 10(4):371–395, 2002.

[10] K. Deb and D. E. Goldberg. An investigation of niche and species formation in genetic function optimization. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 42–50, 1989.

[11] K. Deb and M. Goyal. A robust optimization procedure for mechanical component design based on genetic adaptive search. *Transactions of the ASME: Journal of Mechanical Design*, 120(2):162–164, 1998.

[12] K. Deb and H. Gupta. Searching for robust Pareto-optimal solutions in multi-objective optimization. In *Proceedings of the Third Evolutionary Multi-Criteria Optimization (EMO-05) Conference (Also Lecture Notes on Computer Science 3410)*, pages 150–164, 2005.

[13] K. Deb, P. Jain, N. Gupta, and H. Maji. Multi-objective placement of VLSI components using evolutionary algorithms. *To appear in IEEE Transactions on Components and Packaging Technologies*, Also KanGAL Report No. 2002006, 2002.

[14] K. Deb and S. Jain. Multi-speed gearbox design using multi-objective evolutionary algorithms. *ASME Transactions on Mechanical Design*, 125(3):609–619, 2003.

[15] K. Deb and A. R. Reddy. Classification of two-class cancer data reliably using evolutionary algorithms. *BioSystems*, 72(1-2):111–129, 2003.

[16] K. Deb and A. Srinivasan. Innovization: Innovation through optimization. Technical Report KanGAL Report Number 2005007, Kanpur Genetic Algorithms Laboratory, IIT Kanpur, PIN 208016, 2005.

[17] O. Ditlevsen and H. O. Madsen. *Structural Reliability Methods*. New York: Wiley, 1996.

[18] N. Eldredge. *Macro-Evolutionary Dynamics: Species, Niches, and Adaptive Peaks*. New York: McGraw-Hill, 1989.

[19] D. B. Fogel. *Evolutionary Computation*. Piscataway, NY: IEEE Press, 1995.

[20] L. J. Fogel. Autonomous automata. *Industrial Research*, 4(1):14–19, 1962.

[21] D. E. Goldberg. *Genetic Algorithms for Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[22] D. E. Goldberg. *The design of innovation: Lessons from and for Competent genetic algorithms*. Kluwer Academic Publishers, 2002.

[23] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strageties: The covariance matrix adaptation. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 312–317, 1996.

[24] J. H. Holland. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: MIT Press, 1975.

[25] P. Jain and A. M. Agogino. Theory of design: An optimization perspective. *Mech. Mach. Theory*, 25(3):287–303, 1990.

[26] Yaochu Jin and Bernhard Sendhoff. Trade-off between performance and robustness: An evolutionary multiobjective approach. In *Proceedings of the Evolutionary Multi-Criterion Optimization (EMO-2003)*, pages 237–251, 2003.

[27] James Kennedy and Russell C. Eberhart. *Swarm intelligence*. Morgan Kaufmann, 2001.

[28] R. T. F. King, H. C. S. Rughooputh, and K. Deb. Evolutionary multi-objective environmental/economic dispatch: Stochastic versus deterministic approaches. In *Proceedings of the Third International Conference on Evolutionary Multi-Criterion Optimization (EMO-2005)*, pages 677–691. Lecture Notes on Computer Science 3410, 2005.

[29] J. R. Koza. *Genetic Programming : On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press, 1992.

[30] K. Miettinen. *Nonlinear Multiobjective Optimization*. Kluwer, Boston, 1999.

[31] P. K. S. Nain and K. Deb. Computationally effective search and optimization procedure using coarse to fine approximations. In *Proceedings of the Congress on Evolutionary Computation (CEC-2003)*, pages 2081–2088, 2003.

[32] D. Pratihar, K. Deb, and A. Ghosh. Fuzzy-genetic algorithms and time-optimal obstacle-free path generation for mobile robots. *Engineering Optimization*, 32:117–142, 1999.

[33] D. K. Pratihar, K. Deb, and A. Ghosh. Optimal path and gait generations simultaneously of a six-legged robot using a ga-fuzzy approach. *Robotics and Autonomous Systems*, 41:1–21, 2002.

[34] A. Prügel-Bennett and J. L. Shapiro. An analysis of genetic algorithms using statistical mechanics. *Physics Review Letters*, 72(9):1305–1309, 1994.

[35] S. S. Rao. *Optimization: Theory and Applications*. Wiley, New York, 1984.

[36] S. S. Rao. Genetic algorithmic approach for multiobjective optimization of structures. In *Proceedings of the ASME Annual Winter Meeting on Structures and Controls Optimization*, volume 38, pages 29–38, 1993.

[37] I. Rechenberg. Cybernetic solution path of an experimental problem, 1965. Royal Aircraft Establishment, Library Translation Number 1122, Farnborough, UK.

[38] G. V. Reklaitis, A. Ravindran, and K. M. Ragsdell. *Engineering Optimization Methods and Applications*. New York : Wiley, 1983.

[39] A. Rogers and A. Prügel-Bennett. Modelling the dynamics of steady-state genetic algorithms. In *Foundations of Genetic Algorithms 5 (FOGA-5)*, pages 57–68, 1998.

[40] G. Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Network*, 5(1):96–101, 1994.

[41] H.-P. Schwefel. Projekt MHD-Staustrahlrohr: Experimentelle optimierung einer zweiphasendüse, teil I. Technical Report 11.034/68, 35, AEG Forschungsinstitut, Berlin, 1968.

[42] H.-P. Schwefel. *Numerical Optimization of Computer Models*. Chichester, UK: Wiley, 1981.

[43] T. K. Sengupta, K. Deb, and S. B. Talla. Drag optimization for a circular cylinder at high reynolds number by rotary oscillation using genetic algorithms. Technical Report KanGAL Report No. 2004018, Mechanical Engineering Department, IIT Kanpur, India, 2004.

[44] R. Storn and K. Price. Differential evolution – A fast and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[45] M. D. Vose. *Simple Genetic Algorithm: Foundation and Theory*. Ann Arbor, MI: MIT Press, 1999.

[46] M. D. Vose and J. E. Rowe. Random heuristic search: Applications to gas and functions of unitation. *Computer Methods and Applied Mechanics and Engineering*, 186(2–4):195–220, 2000.