Chapter 1

Introduction

1.1 Optimization

Various kinds of optimization problems occupy a large portion of research in applied science and engineering. Mathematically, an optimization problem involves finding

$$\min_{x \in S} f(x)$$

where $x$ is a vector of $n$ decision variables, $S$ is an $n$-dimensional region (assumed to be nonempty), and $f$ is a real valued function defined over $S$. The problem is to find a value of $x$ contained in $S$ that minimizes $f$. $S$ is called the feasible set which may be represented by a system of linear or non-linear equations or inequalities.

Classical optimization deals with finding optimal solutions, mostly by analytical methods, of differentiable functions of one or more variables subject to one or more constraints. Although the classical methods are limited in their scope, their study has led to the development of various numerical and probabilistic techniques for optimization which are more suitable for today’s practical problems.

Random search algorithms refer to algorithms which employ probabilistic methods for arriving at the optimal solution. Some of the random search algorithms include genetic algorithms, simulated annealing, tabu search, evolutionary programming,
1.2 Evolutionary algorithms

particle swarm optimization, ant colony optimization, cross-entropy, stochastic approximation, multi-start and clustering algorithms. Random search algorithms have wider applicability than classical methods because they can optimize functions which are not necessarily differentiable or convex over regions which are possibly discontinuous or discrete. Random search algorithms are relatively faster and have the potential to solve large-scale problems involving several variables and constraints which deterministic algorithms cannot handle.

1.2 Evolutionary algorithms

The term evolutionary algorithm is used to refer to a class of probabilistic algorithms designed by observing and mimicking the intelligent mechanism behind biological evolutions. The emerging field of studying and applying evolutionary algorithms is called evolutionary computation (EC) having its specific research goals and aims. The concept of accuracy in a representation of historical events is often relative. And computation is no exception. Our historical reach of evolutionary computation is limited by lack of an authentic body of facts and hence it is convenient to trace its history through the available source of data from the mid-1950’s onwards [9, 31].

The middle of the twentieth century witnessed significant changes in the world of computers chiefly due to the fact that digital computer models were introduced and widely used during this period. This enables us to get an idea about the natural process of evolution. Friedberg and Friedberg et al, in their articles published respectively in the year 1958 and 1959, describe the use of evolutionary process in computer problem solving. Based on the work that had already been done in Machine learning the authors gave a description of the use of an evolutionary algorithm for automatic programming. They dealt with the task of finding a programme calculating a given input-output function. There may be many more predecessors and one of the earliest names that comes to the mind is that of Fraser whose 1957 paper threw some light in this regard.
1.2 Evolutionary algorithms

It was when Bremermann came into the scene. In his 1962 paper [19] he presented some of the first attempts to apply simulated evolution to both numerical optimization problems and nonlinear simultaneous equations. He also developed some of the early evolutionary algorithm theory. It was further modified by Box [18] who in his evolutionary operation ideas provided an evolutionary technique for the design and analysis of industrial experiments. But his concepts were never considered as belonging to the field of computer algorithm. At the same time we should not forget that Spendley and others used them as the basis for their famous simplex design method. More over the attempts to introduce randomness into the evolutionary operations were frowned upon in the 1950's.

Scholars approached these early studies with a spirit of scepticism. By the mid 1960's the scenario changed, and clarity and lucidity was realized with the establishment of the three main forms of the evolutionary algorithms; evolutionary programming(EP), genetic algorithm(GA) and evolution strategies(ES). Lawrence Fogel [43] devoleped the principles of evolutionary programming in San Diego, California. Holland developed the ideas behind the genitic algorithms in 1967. The principles of evolution strategies were developed by the students Bienert, Rechenberg and Schwefel [92, 98] in Berlin in the year 1965. Over the next quarter of a century these branches developed independently of each other, contributing to a rich history.

Modelled on organic evolution, evolutionary algorithms share a lot of common characteristics. Evolution naturally results from the interplay between the formation of new genetic identity and its evolution and selection. One member of a population is influenced by the other members of the population and by the environment. Hence the better one performs under these conditions, the greater one’s change for survival and offspring generation will be. Evolutionary algorithms make use of the combined learning process of a population of individuals. Each member of the population comes to represent a search point in the space of solutions to a given problem. Additional information
like strategy parameters of the evolutionary algorithm may also be incorporated. Successes are the result of randomized process modelled on mutation and recombination. Mutation points to an error-stricken self-replication of individuals. On the other hand recombination exchanges information among individuals. Based on the evaluation of the individuals in their environment, a qualitative or fitness value can be given to them. As a primary necessity, a comparison of the fitness of the individuals is made. Based on which a decision regarding whether they are better or worse can be made. The conclusion of the process reveals that the individuals termed better reproduce more often and better than those who are termed worse.

1.3 Genetic algorithm

The evolution of genetic algorithm was inspired by closely examining and replicating biological evolutions such as reproduction, recombination and mutation. Over the years genetic algorithm has become one of the most important methods for obtaining approximate solutions of optimization problems. Genetic algorithm works by letting the competing variables interact with one another to evolve a potential solution naturally. A large number of optimization problems of various types spanning diverse areas of engineering and natural sciences have been solved by the application of genetic algorithm.

The basic principles of Genetic algorithms(GAs) were introduced, for the first time, by John Henry Holland [62] for solving practical optimization problems and are well documented [7, 12, 28, 29, 30, 49, 55, 79, 83]. The aim of a GA is to achieve better results through selection, crossover and mutation. The success of any GA depends on the design of its search operators as well as their appropriate integration. There is a lot of literature to improve the effectiveness of the GA operators. There are various approaches [10, 11, 20, 28, 36, 49, 117, 127, 128] suggested to select the parents string which may vary depending on the difficulty level of the problems. In a GA, maintaining population diversity is very important to reach to the global optimum [90]. It is necessary to select
the best solution of the current generation to direct a GA to reach to the global optimum. Goldberg and Deb [51] compare four different selection schemes: proportionate selection, fitness ranking, tournament selection and steady state selection. They conclude that by suitable adjustment of parameters, all these schemes can be made to give similar performances, so there is no absolute best method. Many different crossover algorithms are devised [36, 73, 118]. DeJong [33] investigated the effectiveness of multiple-point crossover. Besides, the initial population plays an important role in solving a problem by GA [4, 24, 104, 120]. Genetic algorithms have been successfully used in numerical function optimization [33], image processing [49], combinatorial optimization [50], machine learning [52, 53, 65], data mining, information retrieval and grammar induction.

A basic genetic algorithm comprises of three genetic operators, namely, selection, crossover and mutation. Reproduction (selection) is usually the first operator applied on a population. Reproduction selects good strings in a population and forms a mating pool. There exists a number of reproduction operators in GA literature, but the essential idea in all of them is that the above average strings are picked from the current population and their multiple copies are inserted in the mating pool in a probabilistic manner.

In the crossover operator, new strings are created by exchanging information among strings of the mating pool. Many crossover operators exist in the GA literature. In most crossover operators, two strings are picked from the mating pool at random and some portions of the strings are exchanged between the strings. The two strings participating in the crossover operation are known as parent strings and the resulting strings are known as offspring strings. In practice, all parents in the mating pool are not selected for crossover operation so that some of the good strings may be preserved. This is achieved by selecting a fixed percentage of parents from the mating pool and it is known as the crossover probability.

Mutation is applied to each child individually after crossover. This operator randomly flips or alters one or more bit values at randomly selected locations in a chromosome with a small probability.
The overall GA process steps are summarized as follows:

**STEP 1** Choose coding to represent problem parameters: a selection operator, a crossover operator, and mutation operator.

**STEP 2** Choose initial population size, length of a string, probabilities of crossover and mutation, search domain of the variables, termination criteria or maximum number of iteration.

**STEP 3** Set $T = 0$. Generate initial population from the search domains randomly.

**STEP 4** Evaluate each string of the population for fitness.

**STEP 5** If termination criteria is satisfied, then STOP

ELSE Jump to STEP 6.

**STEP 6** Perform selection on the population.

**STEP 7** Perform crossover on random pairs of strings.

**STEP 8** Perform mutation on every string.

**STEP 9** Evaluate the strings of the new population.

**STEP 10** Set $T = T + 1$, go to STEP 4.
1.3 Genetic algorithm

Fig. 1.1 shows the schematic of the application of GA algorithm.

Figure 1.1: Flowchart for GA algorithm
1.3 Genetic algorithm

1.3.1 Basic Principles of GA

The method of genetic algorithm begins with a population of possible solutions known as individuals. It gradually applies the Darwinian theory of evolution based on the survival of the fittest and moves towards the optimal solution. The initial population is randomly chosen. Before a GA can function one requires to devise an apt coding or representation for the problem. Each coded solution must be assigned with a figure of merit. For this we require a fitness function. During the process one must choose the parents for reproduction and recombine them to generate offspring. These stages are described below.

1. Coding

We are under the assumption that a valid solution to a problem may be shown as a set of parameters. Such parameters called genes, join together to bring out a chain of values commonly referred to as a chromosome or individual. Objects bringing out possible solution sets to the actual problem are called phenotypes. At the same time the encoding or representation of the individual in the GA is called genotype. In GA the variables are represented as strings of numbers, usually, binary in nature. If each design variable has a string length of $l$, and the number of variables be $n$ the string length of the design vector will be $nl$. For instance, suppose there are 2 design variables $x_1$ and $x_2$, and the string length be 8 for each variable. Let $x_1 = 4$ and $x_2 = 7$. Then the chromosome length of 16 is given below.

\[
\begin{array}{cccccccccccc}
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & : & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{c}
x_1 \\
: \\
x_2
\end{array}
\]

2. Fitness function

Every problem that require to be solved need to device a fitness function. For a given chromosome, the fitness function provides a single numerical fitness supposedly proportional to the utility or ability of the individual represented by the chromosome.
For problems on function optimization what the fitness function should measure is the value of the function.

3. Genetic Operations on Binary Strings

A basic genetic algorithm comprises of three genetic operators, namely, selection, crossover and mutation.

Selection

The first operator applied on a population is usually selection or reproduction. After fitness evaluation, individuals are classified based on their quality. Reproduction selects good strings in a population forming a mating pool. Roulette wheel selection, tournament selection, rank selection, Boltzman selection, steady state selection etc are some of the selection methods in GA literature. The essential idea in all of them is that the above average strings are picked from the current population and their multiple copies are inserted in the mating pool in a probabilistic manner.

Roulette wheel selection: Here an individual is chosen with a probability in proportion to its fitness value. It can be regarded as a random experiment with

\[ P(b_i \text{ selected }) = \frac{f(b_i)}{\sum_{i=1}^{m} f(b_i)} \]

where \( f \) is the function to be optimized, \( b_i \) is the \( i \)-th individual in the population, and \( m \) is the population size. The above equation is valid only when all the fitness values are positive. If this is not the case, a non-decreasing transformation \( \phi : \mathbb{R} \rightarrow \mathbb{R}^+ \) must be applied. Then the probabilities can be expressed

\[ P(b_i \text{ selected }) = \frac{\phi(f(b_i))}{\sum_{i=1}^{m} \phi(f(b_i))} \]
In order to implement this selection process we may imagine a roulette-wheel with its circumference marked for each string proportionate to the fitness of the string. The roulette wheel is spun \( m \) times, to select an instance of the string chosen by the roulette wheel pointing each time. Since the circumference of the wheel is marked according to a string’s fitness, this roulette wheel mechanism is expected to make \( \frac{f(b_i)}{\bar{f}} \) copies of the \( i \)-th string in the mating pool. The average fitness of the population is calculated as

\[
\bar{f} = \frac{1}{m} \sum_{i=1}^{m} f(b_i)
\]

<table>
<thead>
<tr>
<th>Chromosome</th>
<th>Fitness value</th>
<th>% of total fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10.36</td>
<td>45.08</td>
</tr>
<tr>
<td>2</td>
<td>3.42</td>
<td>14.88</td>
</tr>
<tr>
<td>3</td>
<td>1.16</td>
<td>5.05</td>
</tr>
<tr>
<td>4</td>
<td>5.39</td>
<td>23.46</td>
</tr>
<tr>
<td>5</td>
<td>2.65</td>
<td>11.53</td>
</tr>
<tr>
<td>Total</td>
<td>22.98</td>
<td>100</td>
</tr>
</tbody>
</table>

Figure 1.2: A graphical representation of roulette wheel selection

Figure 1.2 shows a roulette wheel for five individual having different fitness values. Depending on the actual problem, other selection schemes than the roulette wheel can be useful.
1.3 Genetic algorithm

**Tournament selection**: Here, a small group of individuals is taken from the population and the individual with best fitness is chosen for reproduction. This method is also suitable when the fitness function is given in implicit form, i.e., when we only have a comparison relation regarding the merit between the given individuals.

**Linear rank selection**: The potentially good individuals, in the beginning, often fill the population very fast leading to premature convergence into local maxima or minima. But refinement in the end phase may be slow due to the similar fitness values of the individuals. These problems can be avoided by taking the rank of the fitness values than the values themselves as the basis for selection.

**Crossover**

It is the exchange of genes between the chromosomes of the two parents. Several crossover methods are available. In most of them two strings are picked up randomly from the mating pool and some of their portions are exchanged between the strings. The two strings engaged in the crossover operator are called the parent strings and the resultant strings are called offspring(child) strings. All parents from the mating pool are not selected for crossover operation and hence some good strings are preserved. Crossover probability as it is known chooses a fixed percentage of parents from the mating pool.

**Single point crossover**: This process may be realized by putting two strings at a randomly chosen position and swapping the two tails. Termed as one point crossover in the following, this process is visualized in Figure 1.3.

**Two point crossover**: Here two crossover points are chosen and their contents are exchanged between two mated parents, as shown in Figure 1.4. In this respect one point crossover can be seen as two point crossover with a cut point fixed at the start of the string.
1.3 Genetic algorithm

**Figure 1.3**: Single point crossover

**Figure 1.4**: Two point crossover

**N-point crossover**: Here $N$ number of breaking points are randomly chosen. Section is swapped every second. Among these classes, two point crossover is particularly important.

**Uniform crossover**: It is radically different from the one point crossover. Each offspring gene is created by copying the corresponding gene from one or the other parent, chosen according to a randomly generated crossover mask. If there is a 1 in the crossover mask, the gene will be copied from the first parent, and if there is a 0 in the mask, the gene will be copied from the second parent, as shown in Figure 1.5. This process is repeated after exchanging the parents to produce the second offspring. A new crossover mask is randomly generated for each pair of parents. Offsprings hence contain a mixture of genes from each parent. In figure 1.5 new children are produced using uniform...
1.3 Genetic algorithm

Figure 1.5: Uniform crossover

crossover approach. It is noticed that while producing offspring 1, if there is a 1 in the mask, the gene will be copied from the parent 1 else from the parent 2. On producing offspring 2, if there is a 1 in the mask, the gene is copied from parent 2; if there is a 0 in the mask, the gene will be copied from the parent 1.

Mutation

It is applied to each offspring individually after crossover. It randomly flips or alters one or more bit values at locations randomly selected in a chromosome with a small probability. Here also, as in the case of crossover, the choice of the suitable mutation technique depends on the coding and the problem itself. Three choices are commonly available. They are

1. **Inversion of single bits**: With the probability $p_M$, one randomly chosen bit is negated.

2. **Bitwise inversion**: The whole string is inverted bit by bit with the probability $p_M$.

3. **Random selection**: With probability $p_M$, the string is replaced by a randomly chosen one.

1.3.2 Convergence of GA

With the proper implementation of the GA, the population evolves over successive generations so that the fitness of the best and average individual in each generation increases
1.3 Genetic algorithm

towards the global optimum. A gene is likely to converge when 95% of the population shares the same value [33]. The population will converge when all the genes have converged.

1.3.3 Schemata and the Schema theorem

Here we discuss the fundamental result on the behaviour of genetic algorithms—the so-called Schema Theorem. Although completely incomparable with convergence results for conventional optimization methods, it still provides valuable insight into the intrinsic principles of genetic algorithms.

Holland’s Schema theorem [62] provided the first rigorous explanation for how GAs work. A schema, a pattern of gene values, may be represented (in a binary coding) by a string of characters in the alphabet \{0, 1, *\}. A particular individual string is said to contain a particular schema if it matches that schemata with the * symbol matching anything. The order of a schema \(H\) is the number of non * symbols it contains and it is denoted by \(o(H)\). The length of a schema \(H\), denoted by \(\delta(H)\), is the distance between the first and last fixed positions. Also the fitness of a schema \(H\) at time (or generation) \(t\) is defined as the average fitness of all strings in the population matched by the schema \(H\) and is denoted by \(f(H)\).

**Schema Theorem:** For every schema \(H\), the following schema growth equation holds:

\[
m(H, t + 1) \geq m(H, t) \frac{f(H)}{\bar{f}} \left[ 1 - p_c \frac{\delta(H)}{l - 1} - o(H) p_M \right]
\]

where \(m(H, t)\) is the number of strings in the population matched by the schema \(H\) at the time \(t\), \(f(H)\) is the fitness of a schema \(H\) at time \(t\), \(\bar{f}\) is the average fitness of the population at time \(t\), \(o(H)\) is the order the schema \(H\), \(p_c\) is the probability for crossover and \(p_M\) is the probability for mutation.

Schema theorem interprets the power of the GA in terms of how schemata are processed. Individuals in the population are given opportunities to reproduce, often referred
1.4 Adaptation in Genetic algorithms

Adaptation in Genetic algorithms refers to the process where the number of opportunities an individual receives to produce offspring is in proportion to its fitness. Hence, the better individuals contribute more of their genes to the next generation. The high fitness of an individual is often equated with the good schemata it has. By transferring more of these good schemata to the next generation, we can increase the likelihood of finding even better solutions.

According to Holland, the optimum way to explore the search space is to allocate reproductive trails to individuals in proportion to their fitness relative to the best of the population. In this way, good schemata receive an exponentially increasing number of trails sequential generations.

According to Goldberg [49], the power of genetic algorithms lies in its ability to form good building blocks. Such schemata are of short defining length, consist of bits which work well together, and tend to lead to improved performance when incorporated into an individual. A successful coding scheme encourages the formation of building blocks. It ensures that the related genes are close together in the individual. It also ensures that there is little interaction between genes.

1.4 Adaptation in Genetic algorithms

Adaptive genetic algorithms are GAs whose parameters such as the string length, the population size, the crossover probability, or mutation probability are varied while the GA is running. The efficiency of a GA is often improved by incorporating some adaptive techniques in the basic GA operators. Many such adaptive techniques are available in the literature; and among them self-adaptation which is the method to adjust the setting of control parameters is the most widely used method of modifying GA algorithm at present. The use of self-adaptation in evolutionary algorithms has made them more flexible and closer to natural evolution. Among the evolutionary methods, self-adaptation properties have been explored with evolution strategies [8, 16, 58, 97]. In literature, so many adaptive techniques are available [35, 55, 72, 106, 108, 121]. Jinn-Moon Yang
1.4 Adaptation in Genetic algorithms

et al described an algorithm [129], known as the adaptive mutations genetic algorithm (AMGA), for training an Artificial Neural Networks. Wang Lei and Shen Tingzhi proposed an Improved Adaptive Genetic Algorithm and its application to image segmentation [126]. Adimurthy et al proposed a GA with Adaptive Bounds (GAAB), [3] and this algorithm is implemented successfully in obtaining precise lunar gravity assist transfers to Geostationary orbits. Deb K and Beyer H G [32] have developed a self-adaptive GA. In their paper, they demonstrated the self adaptive feature of real parameter genetic algorithms using a simulated binary crossover operator and without any mutation operator.

Generally, parameter setting is accomplished prior to running a Genetic Algorithm and this setting remains unchanged during execution. The GAs basic mechanism depends on the choice of several parameters such as string length, population size, mechanism of selection, crossover operators, mutation operators, crossover probability and mutation probability. All these parameters have a great impact on the GAs performance. As such, the selection of right parameters is critical in achieving the desired results for an optimal solution [35, 46, 54].

The lower and upper bounds for the optimal solution may not be known a priori in many problems; however information about the lower and upper bounds of the problem variables are important in many search algorithms such as binary-coded genetic algorithms, which are hard-coded to search within the chosen bounds. It is in such circumstances that a search algorithm, such as the one that we propose in this work, which adaptively changes the search limits in GA, would be found useful. This algorithm redefines the bounds for each problem variable adaptively by expanding or contracting the current limits or shifting the search to an interval outside the current limits.

During conceptual design, the problem space and solution space are not fixed yet and we can explore a range of possible problem solution scenarios. The initial limits assigned to the design variables in such a design framework are always subjected to changes. The objective of the research underlying this thesis is to develop an adaptive
search technique which will assist the researchers at the conceptual stage. This will give the designers an option to explore a range of possible solutions from a wide range of possible problem scenarios, rather than working towards a single unique solution which may not be optimal.

1.5 Evolutionary Programming

Lawrence J Fogel’s evolutionary programming [43, 44] provided us with a method for creating artificial intelligence. Ever since its introduction in the year 1960 the concept has undergone investigations and revision [41, 42, 45] and extended in Burgin [21, 22], Atmar [6]. From the realms of heuristics and the simulation of undeveloped neural networks the concept underwent drastic changes and came to accommodate behaviour changes to meet the requirements in a wide range of environments. Predicting possibilities on the foundation of scientific observation of sequences gave rise to the evolution of finite state machanics (FSM). Forecasting the variable time series in accordance with arbitrary standards.

Intelligent behaviour helps us to predict one’s environment and to convert the predictions into an appropriate response, required to achieve a given goal. The environment is defined as containing a sequence of symbols. An FSM successfully converts an input symbol sequence into an output symbol sequence. The algorithm that is evolved in the process operates on the sequence of symbols to effect an output symbol, optimizing the algorithm’s performance in terms of subsequent symbols and a well defined payoff function.

1.6 Evolution Strategies

Evolution strategies suggested by Rechenberg [92] and Schwefel [98, 99] deal with the object of solving difficult, discrete and continuous parameter optimization problems
and extended by Herdy[61], Kursawe [70], Ostermeier [88, 89], Rudolph [95], Schwegel [100]. The first version of evolutionary strategy, known as (1 + 1) ES focused on discrete and binomially scattered mutations. It dealt with one parent and one descent per generation. Evolution strategies modify real valued vectors. They stress the importance of mutation and recombination as essential operators for searching in the search space as well as strategy parameter space simultaneously.

1.7 Multiple Gravity Assist problems

Real spacecraft trajectory problems involve many complex models and highly nonlinear constraints. The solution to such optimization problems require enormous computational requirements and most important of all, a good initial guess to help achieve the solution quickly. This reduces the area of search domain, usually called the process of space pruning. Global optimization of such problems have been dealt with extensively in the literature. Hartmann et al [59] used a pareto genetic algorithm along with a calculus of variation based trajectory optimization algorithm to generate trajectories for Earth-Mars rendezvous missions. They have produced new and versatile class of trajectories with a continuous period of launch dates with this modification. Mortari [2] formulated an algorithm which requires only three design variables for a general transfer between two non coplanar orbits. Emphasis was given to spacecraft attaining the final orbit even if the solution was not optimal globally. Vasile and De Pascale [125] used a general global optimization approach employing a deterministic branching strategy which splits the solution space into a number of subdomains. Their algorithm is claimed to have solutions which are valid even in highly multi-modal domains.

Considering the wide variety of trajectory optimization problems of interest, the numerous physical models one could think of in approaching them and the different techniques used to solve them, it is nearly impossible to compare results available in literature. Hence the importance of analysing each problem, specifying the algorithms
and constraints used remains explicit. The yearly global trajectory optimisation competition [87] (GTOC) proposes every year a trajectory optimisation problem to be solved in this context.

1.8 Simulated annealing

Simulated annealing (SA) is a randomized perturbation technique, which often generates excellent solutions to various optimization problems. The main advantage of SA is the substantial reduction in computation time for most optimization problems. The basic strategy behind simulated annealing is to start from an arbitrarily selected solution and keep generating improved solutions by slightly perturbing it. Each time a solution is generated, a cost function evaluates its optimality, based on which it is accepted or allowed to move onto an improved solution according to an acceptance rule.

It is based on an analogy of thermodynamics with the way metals cool and anneal. If a liquid metal is cooled slowly, its atoms form a pure crystal corresponding to the state of minimum energy for the metal. The metal reaches a state with higher energy, if it is cooled quickly.

SA has received a lot of attention in the recent times to solve optimization problems. Kirkpatrick et al [68] and Cerny [23] showed a model for simulating the annealing of solids, based on the theory originally proposed by Metropolis et al [82]. This model is used for the optimization of problems, where the objective functions to be minimized corresponds to the energy states of the metal. SA has these days become one of the many heuristic approaches designed to give a good, not necessarily optimal solution. Since its introduction, several applications in different fields of engineering, such as integrated circuit placement, VLSI design, pattern recognition, optimal encoding, resource allocation, logic synthesis, have been deploying SA solution. At the same time, theoretical studies have been focusing on the reasons for the excellent behaviour of the algorithm.
1.8 Simulated annealing

Theoretical developments and researches in simulated annealing are well documented [1, 34, 38, 60, 71, 86, 94, 113].

Though, the majority of the theoretical developments and application work with simulated annealing have been for discrete optimization problems, simulated annealing has also been used as a tool to address problems in the continuous domain. We are observing an increased interest in using simulated annealing for global optimization over regions containing several local and global minima. Fabian [37] studies the performance of simulated annealing methods for finding a global minimum of a given objective function. Bohachevsky et al. [17] proposes a generalized simulated annealing algorithm of function optimization for the use in statistical applications and Locatelli [77] presents a proof of convergence for the algorithm. Belisle [13] presents a special simulated annealing algorithm for global optimization, which uses a heuristically motivated cooling schedule. This algorithm provides a reasonable alternative to existing methods and can be implemented easily. Suman [110, 111, 112] proposes different simulated annealing-based approach to address constrained multi-objective optimization problems. In [111], a comparative analysis of five simulated annealing algorithms is conducted for the system reliability optimization problem. These methods were aimed to generate a set of efficient solutions that are a good approximation to the entire set of efficient solutions over a fixed period of time.

Belisle et al. [14] discusses convergence properties of simulated annealing algorithms applied to continuous functions and apply these results to hit-and-run algorithms used in global optimization. The convergence properties identified are consistent with those presented in [56] and provide a good contrast between convergence in probability and almost sure convergence. This work is further extended in [132] to an improved hit-and-run algorithm used for global optimization. Several researchers have independently proven the convergence results [5, 47, 48, 56, 78, 84].

Fleischer and Jacobson [40] propose cybernetic optimization by simulated annealing as a method of parallel processing that accelerates the convergence of simulated
annealing to the global optima. This theory is extended by Fleischer [39] into the continuous domain by applying probabilistic feedback control to the generation of candidate solutions. The probabilistic feedback control method of generating candidate solutions effectively accelerates convergence to a global optimum using parallel simulated annealing on continuous variable problems. Locatelli [75] presents convergence properties for a class of simulated annealing algorithms for continuous global optimization. It was done by removing the restriction that the next candidate point must be generated according to a probability distribution whose support is the whole feasible set. A study on simulated annealing algorithms for globally minimizing functions of multiple continuous variables is conducted by Siarry et al. [105]. The study focuses on how high dimensionality can be addressed using variables discretization. It also addresses the design and implementation issues for several complementary stopping criteria. Convergence results and criteria for simulated annealing applied to continuous global optimization problems are also provided in [130] and [76]. A general-purpose simulated annealing algorithm that solves mixed integer linear programs is introduced by Kiatsupaibul and Smith [66]. The simulated annealing algorithm is constructed using a Markov chain sampling algorithm to generate uniformly distributed points on an arbitrary bounded region of a high-dimensional integer lattice. They show that the algorithm converges in probability to a global optimum. Romeijn et al. [93] study a simulated annealing algorithm that uses a reflection generator for mixed integer or continuous global optimization problems.

Simulated annealing has recently been adapted to address multi-objective problems by originally used as an optimization tool for combinatorial optimization problems (see [113]). Its framework is easy to implement and simulated annealing-based algorithms are capable of producing a Pareto set of solutions in a single run with very little computational cost. Additionally, its performance is not influenced by the shape of the Pareto set, which is a concern for mathematical programming techniques. The first multi-objective version of simulated annealing has been proposed by Serafini [101, 102]. The method
closely follows the guidelines of regular single objective simulated annealing and uses a modification of the solution acceptance criteria in the original algorithm. Various alternative studies have been conducted with the objective to increase the probability of accepting non-dominated solutions. A special selection rule produced by the combination of several criteria has been proposed in order to concentrate the search almost exclusively on the non-dominated solutions. This idea has also been used by Ulungu and Teghem [123] and Serafini [103], with the latter utilizing a target-vector approach to solve a bi-objective optimization problem. Ulungu et al. [122] proposes a complete MOSA algorithm, where a weighted aggregating function is used to evaluate and compare the obtained solutions. The MOSA algorithm works with only one current solution but keeps a record of the population of non-dominated solutions found during the search. A further improved, interactive version of MOSA is presented in [124] and is referred to as the UMOSA method. Suppapitnerm and Parks [114] propose a different simulated annealing-based approach proposed to tackle multi-objective problems (SMOSA method). At each iteration, the algorithm does the search based on one solution and the annealing process adjusts the temperature adaptively, using the objective function value of the obtained solution in each of the multiple objectives. An archive is created to store all the non-dominated solutions.

Integrated circuits (IC) layout is an area where SA is heavily used. The challenge in IC layout is placing components on the surface of an IC so as to optimize subsequent wirability. The IC itself is modelled as a grid, where each grid point can hold one module. Each set of module terminals, wired together, forms a net whose wirability is to be optimized. The chosen cost function is the estimated wire length. SA terminates when the cost improvement across three temperatures is very small [96]. It has been applied to the combined problems of the synthesis of structures or controls and the actuator-location problem for the design of intelligent structures. Multiple and conflicting design objectives such as vibration reduction, dissipated energy, power and a performance index are included by utilizing an efficient multi objective optimization formulation.
1.8 Simulated annealing

The adaptive simulated annealing (ASA) offers a viable optimization tool for tackling these difficult nonlinear optimization problems [26]. Optimization of batch distillation processes, widely used in chemical industry can be effectively solved using SA. SA’s potential for developing optimal operation strategies for batch chemical processes was illustrated by Hanke and Li [57].

Johnson et al. [63, 64] presents a series of articles on simulated annealing applied to certain well-studied discrete optimization. The first one in the series uses the graph partitioning problem to illustrate simulated annealing and highlight the effectiveness of several modifications to the basic simulated annealing algorithm. The second one focuses on applying lessons learned from the first article to the graph coloring and number partitioning problems. Local optimization techniques were previously thought to be unacceptable approaches to these two problems. Johnson et al. [64] also observes that for long run lengths, simulated annealing outperforms the traditional techniques used to solve graph coloring problems. Koulamas et al. [69] focuses specifically on simulated annealing applied to applications in production/operations management and operations research. They discuss traditional problems such as single machine, flow shop and job shop scheduling, lot sizing, and traveling salesman problems as well as non-traditional problems to include graph coloring and number partitioning. Many signal processing applications create optimization problems with multimodal and non-smooth cost functions. Gradient methods are ineffective in these situations because of multiple local minima and the requirement to calculate gradients. Chen and Luk [26] proposes an adaptive simulated annealing algorithm as a viable optimization tool for addressing such difficult non-linear optimization problems. The adaptive simulated annealing algorithm maintains the advantages of simulated annealing, but converges faster. Chen and Luk demonstrate the effectiveness of adaptive simulated annealing with three signal processing applications: maximum likelihood joint channel and data estimation, infinite-impulse-response filter design and evaluation of minimum symbol-error-rate decision
feedback equalizer. They conclude that the adaptive simulated annealing algorithm is a powerful global optimization tool for solving such signal processing problems.

SA has been greatly used in operational research problems. A new approach was proposed by Chen et al [25] to setup planning of prismatic parts using Hopfield neural net coupled with SA. Sridhar and Rajendran [107] have described three perturbation schemes to generate new sequences for solving the scheduling problem in cellular manufacturing system. Suresh and Sahu [115] have used SA for assembly line balancing. They only considered single objective problems. They have found that SA performed at least as well as the other approaches. Meller and Bozer [81] have applied SA to facility layout problems with single and multiple floors. The facility layout problem is highly combinatorial in nature and generally exhibits many local minima. SA achieves low-cost solutions that are much less dependent on the initial layout than other approaches. Mukhopadhyay et al [85] have used SA to solve the problem of Flexible Manufacturing system (FMS) machine loading with the objective of minimizing the system imbalance. Kim et al [67] have considered a multi-period, multi-stop transportation planning problem in a one-warehouse multi-retailer distribution system to determine the routes of vehicles and delivery quantities for each retailer. New job sequences have been generated with a proposed perturbation scheme named the 'modified insertion scheme' (MIS), which has been used in the proposed SA algorithm to arrive at a near global optimum solution. The SA algorithm using the proposed MIS gave substantial improvement in system imbalance. Its other applications have been presented by machine loading problem of FMS [116], part classification [119], resource constrained project scheduling [27], etc. McCormick and Powell [80] described a two stage SA algorithm to derive pump schedules for water distribution in a time short enough for routine operational use.

Bell et al [15] have used it to cluster tuples in databases. They have attempted to use SA in circuit board layout design and it suggests that it would be advantageously applied to clustering tuples in database in order to enhance responsiveness to queries.
1.8 Simulated annealing

Recently, it has not only been applied for optimization but also for recognition of patterns and object classification [74, 109, 131]. Liu and Huang [74] have proposed hybrid pattern recognition based on the evolutionary algorithms with fast SA that can recognize patterns deformed by transformation caused by rotation, scaling or translation, singly or in combination. A new method for online optimization of multiobjective optimization algorithm parameters was proposed by Suman.

The basic approach in SA considers the neighbouring state of the current state to reach a stable state for the application. The strategy consists of exploring the solution space starting from an arbitrary selected solution or state and generating a new one produced through conservatively modifying a given state. Every time a new solution is generated, its cost is evaluated to accept or reject the solution according to an acceptance rule. In SA, as we are looking for an actual best solution which would be a global optimum, the algorithm some times even accepts the worse move to avoid being trapped in a local minimum. The acceptance and rejection of the worse move is controlled by a probability function. The probability of accepting a move, which causes an increase $\Delta f$ in the objective function $f$, is called the acceptance function. It is normally set to $e^{-\Delta f / T}$, where $T$ is a control parameter, which corresponds to the temperature in analogy with the physical annealing. This acceptance function implies that the small increase in $f$ is more likely to be accepted than a large increase in $f$. When $T$ is high most uphill moves are accepted, but as $T$ approaches to zero, most uphill moves will be rejected. Therefore, SA algorithm starts with a high temperature to avoid being trapped in local minimum and proceeds by attempting a certain number of moves at each temperature by decreasing the temperature. Thus, the configuration decisions in SA proceed in a logical order.
The overall SA process steps are summarized as follows:

STEP 1 Select an initial temperature $T$.

STEP 2 Select a randomly generated initial solution vector, $X$, and generate the objective function.

STEP 3 Select the temperature change counter.

STEP 4 Select a temperature cooling schedule.

STEP 5 Select a repetition schedule that defines the number of iterations executed at each temperature.

STEP 6 Give a random perturbation on $X$ and generate a new solution vector, $Y$, in the neighbourhood of current solution vector, $X$, evaluate the objective function.

STEP 7 If $f(Y) < f(X)$, replace $X$ with $Y$. Update the existing optimal solution and go to Step 9.

STEP 8 Else accept $Y$ with the probability $p = e^{-\frac{\Delta f}{T}}$ where $\Delta f = f(Y) - f(X)$. If the solution is accepted, replace $X$ with $Y$. Update the existing optimal solution and go to Step 9.

STEP 9 Decrease the temperature periodically.

STEP 10 Repeat Steps 6–9 until stopping criterion is met.
1.8 Simulated annealing

Figure 1.6 shows the schematic of the application of SA algorithm.

**Figure 1.6**: Flowchart for SA algorithm
1.9 Overview of the thesis

Genetic algorithm is a partial search algorithm inspired by observing the biological evolution processes such as reproduction, recombination and mutation. The application of genetic algorithm has been successful in generating sufficiently close approximate solutions to various optimization problems. The genetic algorithm is designed in such a way that a solution of the problem naturally evolves when the parameters of the problem are suitably tweaked. The applications of genetic algorithm are wide and diverse and include areas such as engineering, art, biology, economics, marketing, genetics, operations research, robotics, social sciences, physics, politics and chemistry. Genetic algorithms have been successfully used in numerical function optimization [33], combinatorial optimization [50], image processing [49], machine learning [52, 53, 65], data mining, information retrieval and grammar induction.

The basic ideas of GA was proposed by John Henry Holland [62] in the 1970s for solving practical optimization problems and are well documented [7, 12, 28, 29, 49]. A basic genetic algorithm comprises of three genetic operators, namely, selection, crossover and mutation. GA aims to achieve quality results through selection, crossover and mutation. The effectiveness of GA can be improved by the correct design of the search operators and their appropriate integration into the GA. The literature contains a large number of methods to improve the effectiveness of the GA operators (see [10, 11, 20, 28, 33, 36, 36, 49, 51, 73, 117, 118, 127, 128]).

Simulated annealing (SA) is a randomized perturbation technique, which often generates excellent solutions to various optimization problems. The main advantage of SA is the substantial reduction in computation time for most optimization problems. The basic approach in SA considers the neighbouring state of the current state to reach a stable state for the application. The strategy consists of exploring the solution space starting from an arbitrary selected solution or state and generating a new one produced
through conservatively modifying a given state. Every time a new solution is generated, its cost is evaluated to accept or reject the solution according to an acceptance rule.

In SA, as we are looking for an actual best solution which would be a global optimum, the algorithm sometimes even accepts a worst move to avoid being trapped in a local minimum. The acceptance and rejection of the worse move is controlled by a probability function. The probability of accepting a move, which causes an increase $\Delta f$ in the objective function $f$, is called the acceptance function. It is normally set to $e^{-\frac{\Delta f}{T}}$, where $T$ is a control parameter, which corresponds to the temperature in analogy with the physical annealing. This acceptance function implies that the small increase in $f$ is more likely to be accepted than a large increase in $f$. When $T$ is high most uphill moves are accepted, but as $T$ approaches to zero, most uphill moves will be rejected. Therefore, SA algorithm starts with a high temperature parameter to avoid being trapped in local minimum and proceeds by attempting a certain number of moves at each temperature by decreasing the temperature. Thus, the configuration decisions in SA proceed in a logical order.

SA has received a lot of attention in the field of solving various optimization problems. Based on a theory proposed by Metropolis et al [82], Kirkpatrick et al [68] and Cerny [23] exhibited a model for simulating the annealing of solids. Many of the disciplines in Engineering such as integrated circuit placement, VLSI design, pattern recognition, optimal encoding, resource allocation, logic synthesis use SA solution. At the same time, theoretical studies have been focusing on the reasons for the excellent behaviour of the algorithm. There is an extensive literature on the strategy and its convergent properties, which can be found in [1, 5, 14, 17, 34, 37, 38, 47, 48, 56, 60, 71, 77, 78, 84, 86, 94, 113].

In this work, firstly, we propose a new operator which is based on imitating the biological cloning process. A simple genetic algorithm is combined with the new operator, leading to an integrated genetic algorithm. Validation of the new algorithm done with
standard test cases shows that the new operator helps in getting faster convergence and global optima.

Using some adaptive techniques in the basic GA operators, the efficiency of GA can be improved. Among the many adaptive techniques available, self adaptation which is the method to adjust the setting of control parameters, is the most popular technique for modifying GA algorithm. The use of self adaptation in evolutionary algorithms has made them more flexible and closer to natural evolution.

In this work, an innovative adaptive algorithm for changing the search limits of design variables in genetic algorithm is incorporated with the integrated genetic algorithm with clone operator. It proceeds by keeping tracks of the solution by expanding, shifting or shrinking the search limits as necessary, even if we start farther away from the actual solution space. Results obtained from testing the algorithm against various case studies are also presented. It is also demonstrated that the algorithm is robust and capable of obtaining optimal solution with any arbitrary search limit.

A widely accepted benchmark for testing global optimizers under the broad category of Space Mission Design are Multiple Gravity Assist problems. The new algorithm is also applied on two Multiple Gravity Assist with single Deep Space Manoeuvre problems namely Cassini2 and Messenger. The new algorithm helps in getting faster convergence and global optima.

Both genetic algorithm and simulated annealing have been used for solving numerous problems from a wide range of application domains and have been found to be very effective and robust throughout, and are even suitable for ill-posed problems where some of the parameters are not known beforehand. In this work we propose simulated annealing based probability conditions to generate a new population in genetic algorithm. The proposed algorithm does not make use of the mutation operation. The new algorithm is validated with standard test cases and the results are compared with genetic algorithm and simulated annealing algorithm. The results obtained indicate a promising performance of the new approach in genetic algorithm.
1.9 Overview of the thesis

Organization of the thesis

The thesis consists of 6 chapters.

The first chapter contains basics of the various topics covered in this thesis. A review of the literature in evolutionary algorithms is included with emphasis on the genetic algorithms particularly relevant to us. Self adaptation which is the method to adjust the setting of control parameters is the most widely used method of modifying GA algorithm at present. A literature survey of adaptation in genetic algorithms and simulated annealing are also included in this chapter.

We propose a new operator based on the concept of cloning and the details are provided in chapter 2. The proposed new operator imitates the biological cloning process. Based on a simple genetic algorithm and new operator, an integrated genetic algorithm is proposed. The new algorithm is validated with standard test cases and is shown that the new operator helps in getting faster convergence and global optima.

In chapter 3, an innovative adaptive algorithm for changing the search limits of design variables in genetic algorithm is incorporated with the integrated genetic algorithm with clone operator. The algorithm keeps track of the solution by expanding, shifting or shrinking the search limits as necessary, even if we start from an interval farther away from the original interval in which the solution lies. The proposed algorithm is tested with various case studies obtained from the literature and the results are presented. Also, the study demonstrates the robustness of the algorithm and the capability of the method to obtain the optimum solution with any arbitrary search limits.

In chapter 4, we apply the integrated genetic algorithm with clone operator on two Multiple Gravity Assist with single Deep Space Manoeuvre problems namely Cassini2 and Messenger.

Simulated annealing based probability condition to generate a new population in genetic algorithm is proposed in chapter 5. The proposed algorithm does not make use
of the mutation operation. The new algorithm is validated with standard test cases and the results are compared with genetic algorithm and simulated annealing.

Conclusions and future work are the contents of Chapler 6.