CHAPTER 8

Non-traditional optimization techniques

8.1 Introduction

In this chapter, we will look at two optimization techniques namely (a) Genetic algorithms (GA) and (b) Simulated annealing (SA), both of which are unconventional, yet powerful optimization techniques. Both use only the information on the objective function and not any auxiliary information like derivatives. GA and SA are both search techniques that also employ probabilistic laws in the search process and have gained immense popularity in the recent past.

8.2 Genetic Algorithms (GA)

A very powerful non traditional optimization technique is the class of algorithms which mimics the process of evolution. We look at the features of biological systems and see if these can be implemented in engineering optimization. Anything that looks at the process of evolution comes under the class of **evolutionary optimization** techniques. There are several of them, of which the most important is the Genetic Algorithms or GA. This is relatively new, developed in 1975 and is gaining popularity in the last 20 years or so.

Genetic algorithms are basically non-calculus based search algorithms. Every technique has a search logic and this varies. Previously, for a 2 dimensional search, we searched around a particular point in the east,

west , north, south, north east, north west, south east and south west direction. We then proceeded in one direction. That was "hill climbing". Or based on a 2 point test we eliminated a portion of the interval. If it is a 2 variable problem, we can work with one variable at a time. So we can have an efficient, single variable search technique, which could be broadly based on either a hill climbing or a region elimination method.

So what is the logic behind GA?

8.2.1 Principle behind GA

The logic, as far as genetic algorithms is concerned, is based on the mechanics of natural selection and natural genetics. The central idea is the *survival of the fittest* concept, which stems from the Darwinian theory, wherein, the fittest will survive and procreate, such that, successive generations become better and better progressively. This is true if we look at any parameter. For example, the average life expectancy of an Indian male is now 68 years. It was only 40 years at the time of independence. The life expectancy in Japan is about 88 years. The probability that a child born in Germany today will live for 100 years is more than 0.5.

We cannot say that all the diseases have been conquered. Several diseases have been conquered, but new diseases have come and medical research is going on at a frenetic pace and intensity. However, it cannot be denied that a lot of people live longer with and inspite of diseases. So outside of Darwinian evolution, there is intervention by man also. Successive generations are becoming better!.

So, basically genetic algorithms simulate or mimic the process of evolution. The key idea here is that evolution is an optimizing process. If evolution is an optimizing process and successive generations are becoming better and better, each generation is like an iteration in numerical methods. So if we have 5 generations, it is like having 5 iterations in a numerical method. So with each iteration, there is a progressive improvement in the objective function. So it means that more or less, GA is like a hill climbing technique. For example if y is a function of x_1, x_2, x_3, we start with some combination of values of x_1, x_2, x_3, and we get a value of y. As we apply the genetic algorithm, with successive iterations, the y will keep increasing (for a maximization problem) based on a suitable convergence criterion, we stop the iterations (generations).

traditional optimization techniques

h west, south east and south west
lirection. That was "hill climbing".
nated a portion of the interval. If
ork with one variable at a time. So
le search technique, which could be
ıg or a region elimination method.

ıms is concerned, is based on the
ıatural genetics. The central idea
which stems from the Darwinian
ırvive and procreate, such that,
:r and better progressively. This
r. For example, the average life
w 68 years. It was only 40 years
e expectancy in Japan is about 88
orn in Germany today will live for

ve been conquered. Several diseases
es have come and medical research
nsity. However, it cannot be denied
and inspite of diseases. So outside
:rvention by man also. Successive

mulate or mimic the process of
evolution is an optimizing process.
ss and successive generations are
generation is like an iteration in
5 generations, it is like having 5
So with each iteration, there is a
:ctive function. So it means that
ng technique. For example if y is
ith some combination of values of
As we apply the genetic algorithm,
:eep increasing (for a maximization
nce criterion, we stop the iterations

8.2.2 Origin of GA

Prof. John Holland, along with his colleagues and students developed this method at the University of Michigan around 1975. Prof. David Goldberg, an illustrious student of Holland, is the author of the book "Genetic Algorithms in search, optimization and machine learning"(1989). Goldberg is a civil engineer who did his Ph.D with Prof. Holland and is now an independent consultant. He also worked with Prof. Holland on genetic algorithms and has contributed significantly to the spectacular growth of this field along with a few other key scientists.

8.2.3 Robustness of GA

The central theme of research on Genetic Algorithms has been robustness. If we say something is robust, it means it can survive even in hostile environments. Robustness in optimization refers to a fine balance between efficiency and efficacy that is essential for survival in varied environments. Robustness of a species or the robustness of human beings, as a race, can mean many things. How much temperature can a man withstand? How long can we go on without food and water? How long can we survive under hostile conditions? Equivalently from an optimization point of view, we are looking at how many different classes of problems are amenable to genetic algorithms.

We do not claim that for all the optimization problems, genetic algorithms will solve it better than other optimization techniques. But the point is that for a wide class of problems, genetic algorithms work very well and that is what we mean by robustness. For a specific 2 variable, that is continuous and differentiable, where the first and second derivatives can be obtained, the steepest descent or the conjugate gradient method will work very fast and will defeat genetic algorithms hands down. There is a class of problems where specialized techniques will be superior to GA.

However, if we apply the conjugate gradient method for a function that has a lot of local minima and maxima and is oscillating wildly, the conjugate gradient will just get choked. In the class of problems, where getting the derivatives is messy and is computationally expensive, GA can be very potent and useful.

8.2.4 Features of biological systems that are outstanding

1. Expensive redesigns can be substantially reduced or even eliminated if systems can be made more robust. This is easy to understand because even for the slightest deviation from design conditions, if a system does not work, it is not robust. If we have a very specialized air conditioner, and the ambient temperature goes above $40°C$, it could possibly switch off. That is why some companies advertise that they use a tropicalised compressor. For example, an airconditioner that works in Europe may not work in Saudi Arabia. The maximum outside temperature in Saudi Arabia can reach $50°C$ and so the compressor has to be redesigned. The same will work in Europe, though this design may not be required there. So we say that the airconditioner is robust if it can work in different types of climatic conditions. So systems should not be finicky and fail if the design conditions change even slightly.

2. The adjustability, robustness and efficiency of biological systems are all amazing.

3. Self repair, self guidance and reproduction in biological systems are also amazing. If we take animals in the wild like tigers and elephants, they do not have doctors and hospitals, but they also get injured and carry out self repair. So if an animal species gets diseased or injured, some will die, while the best will survive. So the ones that survive are already immune to this disease. The weaklings are taken care of or consumed by the disease. In human beings too, it used to be the same case but now we have started intervening medically. For example, if we get a fracture, we do not remove that part and send it to a service centre. Alongwith our daily functions, if we just use a sling and take pain killers, the fracture gets healed automatically in due course of time, while the other processes go on as usual. So self repair is an outstanding feature of biological systems. Self repair is almost impossible in artificial systems.

8.2.5 Conclusions about living and artificial systems

1. If robustness is the objective, compared to man made designs, natural designs are far superior.

2. Biological examples or evolution can provide wonderful insights on survival and adaptation and help get the idea of how species survive and adapt to changing circumstances and how we can incorporate

ems that are outstanding

ıbstantially reduced or even
: more robust. This is easy to
slightest deviation from design
ork, it is not robust. If we have
and the ambient temperature
switch off. That is why some
a tropicalised compressor. For
rks in Europe may not work in
de temperature in Saudi Arabia
ssor has to be redesigned. The
his design may not be required
itioner is robust if it can work
ions. So systems should not be
ions change even slightly.

efficiency of biological systems

oduction in biological systems
ıals in the wild like tigers and
rs and hospitals, but they also
ir. So if an animal species gets
while the best will survive. So
immune to this disease. The
ımed by the disease. In human
case but now we have started
e, if we get a fracture, we do
to a service centre. Alongwith
sling and take pain killers, the
n due course of time, while the
ɔ self repair is an outstanding
repair is almost impossible in

ınd artificial systems

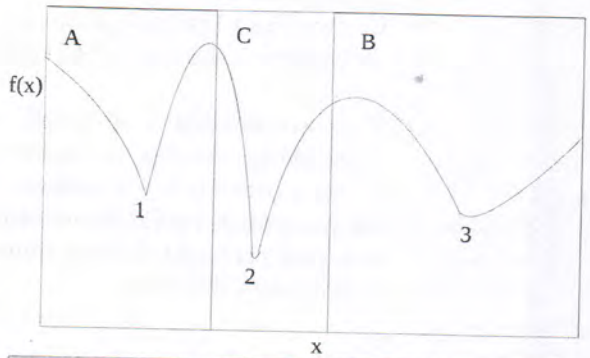
ıpared to man made designs,

ı provide wonderful insights on
the idea of how species survive
es and how we can incorporate

it in engineering problems. This has opened up a new field called biomimetics wherein we try to mimic features of biological systems in engineering design.

8.2.6 Why would we want to use Genetic Algorithms?

For example, we consider a one variable problem in x, as shown in Figure 8.1, let us look at a function f(x) that has 2 local minima and one global minimum. The two local minima are 1 and 3 while the global minimum is 2. A robust optimization algorithm is one, which, regardless of the starting point, will always converge to point 2, the global minimum.



Initial guess	Traditional optimization	Genetic Algorithm
A	Converges to 1	Converges to 2
B	Converges to 3	Converges to 2
C	Converges to 2	Converges to 2

Figure 8.1: Concept of local and global minima in an optimization problem

Let us look at it this way. If the initial guess is in A region, traditional algorithms like steepest ascent/descent or conjugate gradient method will converge the solution to 1. They will declare that 1 is the solution because if an iterate goes to the right of 1, the algorithm will push it to 1 and when the iterate goes to the left of 1, it will push it to 1. Therefore, the algorithm will aver that 1 is the true solution. If we start in B region, traditional optimization algorithms will converge it to 3.

However, if we use an evolutionary optimization technique or a global

optimization technique, regardless of whether we start in region A or B, we get the final answer as 2. If we start in region C, the regular algorithm and GA will both give the same answer. However, if one knows this much, then he/she may does not need an optimization tool. A robust optimization technique is one which gives us the solution regardless of the starting point. So the GA is one algorithm that can be used to get the correct solution, if f(x) looks as shown here.

Several engineering problems may have local minima like this and just one global optimum and we want to determine the global optimum. Now this picture should give us additional ideas. If we look at whether the genetic algorithm will be fast or slow compared to the conjugate gradient method, we see that there is a disadvantage of GA, that it converges slowly. The advantage is that it converges to the global optimum. Can we combine the speed with the robustness? How do we do this?

We start initially with genetic algorithms and zero in on region C. Once we get to this region, we quickly go to the conjugate gradient or some other faster traditional technique. So this is called a **hybrid optimization technique**. So, here we start with GA and switch later to a gradient based method. This is routinely used nowadays for complex optimization problems.

8.2.7 What are the mathematical examples for multi modal functions?

The Rastrigin function is one such example. The function is given here (Eqn. 8.1). It really tortures y and has several peaks and valleys but the minimum of the function is 0. The global minimum occurs at (0,0).

$$Ras(x_1, x_2) = 20 + x_1^2 + x_2^2 - 10(cos(2\pi x_1) + cos(2\pi x_2)) \tag{8.1}$$

$$Ban(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2 \tag{8.2}$$

The Rosenbrock function is called the Banana function, $Ban(x_1, x_2)$ (Eqn. 8.2). The global minimum occurs at (1,1). If somebody is interested in working on new optimization methods, one needs to first test his/her algorithm against standard functions like this. We have to benchmark and find out on a particular computer, how many iterations does this function take using the new algorithm and what level of accuracy it reaches. First it should reach the global optimum, then the speed and time are tested. These are all considered standard.

traditional optimization techniques

whether we start in region A or B,
rt in region C, the regular algorithm
ıswer. However, if one knows this
ed an optimization tool. A robust
gives us the solution regardless of
algorithm that can be used to get
:hown here.

ave local minima like this and just
letermine the global optimum. Now
ıl ideas. If we look at whether the
compared to the conjugate gradient
dvantage of GA, that it converges
verges to the global optimum. Can
stness? How do we do this?

›rithms and zero in on region C.
ckly go to the conjugate gradient
ınique. So this is called a **hybrid**
we start with GA and switch later
›utinely used nowadays for complex

ıtical examples for multi

xample. The function is given here
has several peaks and valleys but
ıe global minimum occurs at (0,0).

$$)(cos(2\pi x_1) + cos(2\pi x_2)) \quad (8.1)$$

$$_1)^2 + 100(x_2 - x_1^2)^2 \quad (8.2)$$

the Banana function, $Ban(x_1, x_2)$
occurs at (1,1). If somebody is
zation methods, one needs to first
ırd functions like this. We have to
lar computer, how many iterations
ıew algorithm and what level of
l reach the global optimum, then
are all considered standard.

8.2.8 Efficiency of the optimization technique versus the kind of problem

An optimization problem falls under one of the three categories listed below

1. Unimodal problems are those that have got only one peak or valley.
2. A combinatorial problem is one that permits only a finite number of combinations of the variables under question. For example, there are 5 machines and each can do 4 types of operations. There are various combinations of which machine will do which job, so that we maximize the profit or minimize the cost as the case may be.
3. Multimodal problems are mathematical or engineering problems that have several peaks and valleys.

Now if we look at unimodal problems, the conjugate gradient or the steepest ascent/descent method will be very efficient. In fig. 8.2, we see that the efficiency is almost close to 1. But if we try to apply the same to a multimodal function, it may not converge or may converge very slowly. That means we have to restart with different initial points. So the efficiency is very low. Even when the exhaustive search or the random walk algorithm is used for a combinatorial problem, the efficiency is very low. This is because for a combinatorial type of problem, we may try to exhaustively search for the optima.

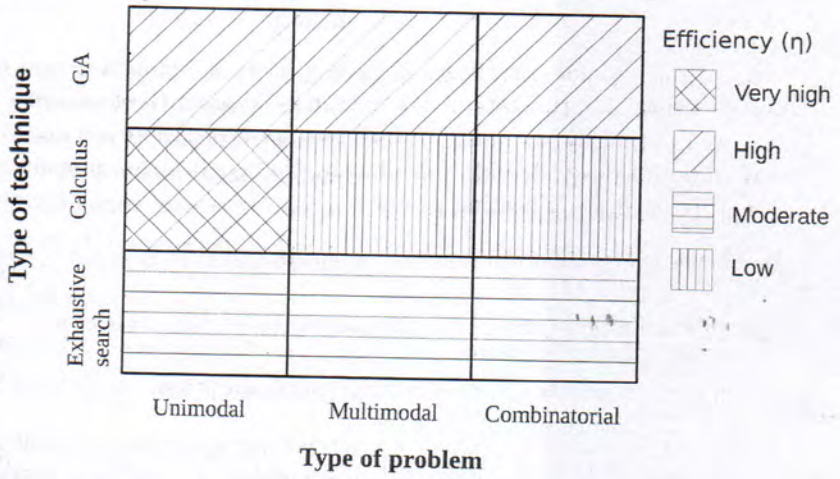


Figure 8.2: Efficiency of various classes of optimization algorithms for three kinds of problems

Now, if we look at GA, its efficiency is more or less the same for all classes of problems and looks, as shown here. So the GA has an efficiency that is far greater compared to what a traditional optimization technique or exhaustive search method has for multimodal and combinatorial problems. However, for a specific unimodal problem, the efficiency of GA will be lower than a sophisticated traditional technique developed for it.

While the exhaustive search or the random walk works with a uniform or systematically low efficiency, for a class of problems, GA works with a reasonably high efficiency for a wide range of problems. This is known as robustness. We cannot claim that for all the problems, the efficiency of GA is the highest. That is far from the truth!

8.2.9 Philosophy of Optimization

Now we have to look at the philosophy of optimization. What is the goal of optimization? We keep improving the performance to reach a or some optimal points. We see that as y improves, we keep going in that direction.

The implications are

- We seek improvement to approach some optimal point. There is always an optimal point and the goal is to reach that. That is the belief!
- What Prof. Goldberg alleges is that this is a highly calculus oriented view of optimization. We have learned so much of calculus that we feel that dy/dx must be equal to 0 such that there is a peak and on both sides y should drop sharply. He says it never happens in nature or any engineering problem.
- He alleges that it is not a natural emphasis.

According to human perception:

- Goodness is invariably judged relative to competition.
- The best scientist, tennis player or cricketer, poet, we cannot define a maximum y in these cases! When somebody is outstanding it means that compared to others, he or she is very good and is far better compared to the others. But after 20 years, someone else

may come who is better than him/her. We cannot simply specify criterion and say he/she satisfies all this, he/she is the best.

- Convergence to the best is besides the point as no one knows the definition of best or what the criteria for it are.
- So the definition of best in most of the situations is far better compared to others, but there is no objectivity here.
- Can we say the best human being has arrived? How do we define such a person? Someone who has the maximum money or has solved a 200 year problem or propose a radically new theory?
- Doing better relative to the others is the key. This also justifies, in a lighter vein, why many universities in the world have a relative grading in place.

Therefore we have to look at whether the optimization algorithm goes in the right direction. After it has reached some particular level, we just stop working on the problem. We do not try to get one optimum solution. So from the perspective of a GA analyst, the priorities of optimization have to be relooked. The most important goal of optimization is improvement. We look at how the objective function changes. The key point now is that instead of optimizing, can we reach a satisfying level of performance quickly? For example, with the design variable like temperature, pressure and so on, can we quickly reach a good level of efficiency for the power plant? That is the goal. So the attainment of optimum is not a critical requirement for complex systems.

A parallel that can be drawn here is the Kaizen philosophy used extensively in Japan. In Japanese, Kaizen means " continuous improvement" is that we do not have a separate set of supervisors. Usually, someone makes a product and someone else checks it. They got rid of it and said that the person who makes it also checks it. He/she reports the efficiency and there is a self correction. We cannot see rapid progress in Kaizen but there is continuous improvement. But because there is an incremental progress over a period of time, we get substantial progress. The Japanese car maker Toyota uses this, and this automobile is known for its quality consciousness.

8.2.10 Key differences between GA and traditional optimization methods

- **GA employs the coding of the parameter set** and not the parameters themselves. Generally, the variables x_1 to x_n are

replaced by their binary equivalent such that we convert them into 0s and 1s. While it is possible to write a genetic algorithm code without this binary representation also, the most popular of them all is the binary representation.

- **The GA searches for a population of points and not a single point**. Instead of taking a single value of x_1 and x_2 and taking y, we see how y changes with different sets of x_1 and x_2. If we have a 2 variable problem, we take a, b, c and d which represent different (x_1, x_2) values. a,b,c,d are known as candidates. For a, b, c and d, we now calculate y(a), y(b), y(c) and y(d). We then find which among them is maximum and rank them. We convert x_1 and x_2 into 0 and 1. We mix and match the bits from better parents. The better parents are those for which y is higher. Now produce new values of a, b, c and d. The children are now produced, which then become parents. The strategy involved is that among all these parents, only the fittest can survive and reproduce. The number of members in a population, (the sample size) is kept fixed. So if we have 4 or 5 solutions, we look at how these solutions evolve generation after generation. After a certain stage, we can take the average of these points and say that that is the optimum. We will reach a stage where there is little difference between members of the species. That is the way it should be. If there is continuous improvement, all members should be equally strong. That is what GA strives to achieve.

- **GA uses the objective function information** and does not use information of the derivative or second derivative.

- **GA uses stochastic transition rules** and not deterministic rules.

8.2.11 The Basic GA

- A population of 2n-4n trial solutions is used, where n is the number of variables. So for a 2 variable problem, we start with 4 to 8 solutions.

- Each solution is represented by a string of binary variables corresponding to chromosomes in genetics. So each solution is represented as 0s and 1s. This is not the only way of coding, but is the most popular one.

- The string length can be made long enough to accomplish any desired accuracy of the fitness and thus, any desired accuracy is attainable.
- The numerical value of y mimics the concept of "fitness" in genetics. The fittest members of the species will survive and procreate. This comes from the Darwinian theory. Fitness in genetics is analogous to the objective function value in optimization problems.
- After the candidate solutions are selected, a new generation is produced by selecting, using stochastic principles, the fittest parents to produce children from among the trial solutions. There is no distinction of father or mother in so far as the selection of parents is concerned in GA.
- In each child, we mix and match the bits from 2 parents and produce new children. Then we again convert the 0s and 1s back into y. We now calculate y for the new values of the variables.
- We rank the solution in the ascending or descending order of fitness. For a maximization problem, we choose those values of design parameters that give the highest values of y.
- The random alteration of binary digits in a string may introduce beneficial or harmful effects. Sometimes what happens is some solution is already having a high fitness, and because we always want 2 parents to mate and produce children, half the chromosomes will be cut from this fit parent and half the chromosomes from some other parent will be joined. It may happen sometimes that this child may have a poor fitness. So this constant mixing and matching may result in a drop in quality and we randomly do what is called **mutation**. That is, one in 20 bits, we change 1 to 0 and 0 to 1. This is randomly or stochastically done so that the undesirable effects of this crossover are neutralized.

8.2.12 Elitist strategy

Genetic algorithms have some other techniques also to overcome this problem of losing strings of high quality. There is another strategy called the **elitist strategy**. What we do in this is that out of the n solutions, the best solution is left untouched and automatically goes to the next generation. So all the crossover and permutations-combinations are done for the (n-1) solutions while the "king" remains as it is. But "he" cannot remain king forever. When we do mixing and matching and the (n-1)

parents recombine to produce new (n-1) children, the king is added to the list and ranked along with the others. If he is no longer king, i.e. he does not have the highest fitness, then he also joins the process of crossover and mutation. The king in a particular generation is not touched. This is the elitist strategy.

This is used in what is called **particle swarm optimization**. This is also very similar to the genetic algorithm and is an evolutionary technique. For example, when a group of birds are moving and searching for food, they will adjust their orientation such that, each of the birds is at the smallest possible distance from the leader and the leader is the one who is closest to the food. After 15 minutes, the position of the leader may change. It is equivalent to the elitist strategy.

8.2.13 Some doubts about GA?

(a) How do we represent a set of variables as a string?
Let us answer this through an example, $X = (x_1, x_2, x_3, ...x_n)$. If we have 4 variables, $x_1 = 15$, $x_2 = 4$, $x_3 = 21$, $x_4 = 7.$, we know that in the binary system, 15 can be represented as 01111, 4 as 00100 and so on. So $X = (01111, 00100, 10101, 00111)$. Now we remove the comma separating the values of the variables from X. In our computer program, we know that the last 5 bits represent x_4, the next 5 bits represent x_3 and so on. So each X is a design vector, which is a combination of the design variables that are represented as combinations of 0 and 1.

(b) How do we handle decimals?
Suppose we want to represent 4.87, we multiply this by 100 which gives 487. We need 9 bits to represent 487 because $2^8 = 256$ and 111111111 = 511. So 487 being smaller than 511, 9 bits are required to represent it. But we have to remember that when we interpret the final value of these 9 bits, we have to divide it by 100. So if the accuracy we seek keeps increasing, then the number of bits required will also increase dramatically. So the values of the variables alone do not decide the string length, the accuracy also influences it. So if we know for a fact that a variable x lies between 1 and 5, and we need 2 digit accuracy, then 9 bits are required or are sufficient for its representation.

8.2.14 The Main features of GA

GA has evolved a lot since it first made its appearance in 1975.

) children, the king is added to the
If he is no longer king, i.e. he does
ılso joins the process of crossover
ır generation is not touched. This

:le swarm optimization. This
lgorithm and is an evolutionary
of birds are moving and searching
ıtion such that, each of the birds
m the leader and the leader is the
: 15 minutes, the position of the
the elitist strategy.

ʌ?

variables as a string?
ɔle, $X = (x_1, x_2, x_3, ...x_n)$. If we
$= 21$, $x_4 = 7$., we know that in
ıted as 01111, 4 as 00100 and so
11). Now we remove the comma
rom X. In our computer program,
t x_4, the next 5 bits represent x_3
or, which is a combination of the
ıs combinations of 0 and 1.

multiply this by 100 which gives
because $2^8 = 256$ and 111111111
., 9 bits are required to represent
vhen we interpret the final value
by 100. So if the accuracy we
of bits required will also increase
bles alone do not decide the string
. So if we know for a fact that a
need 2 digit accuracy, then 9 bits
ɔresentation.

ʒA

le its appearance in 1975.

1. GA is conceptually easy to understand.
2. GA works very well in exceedingly complex problems and in problems, where getting derivatives of the objective function, is very cumbersome.
3. Lots of approaches and possibilities exist and so considerable subjectivity exists in implementation.
4. It is easy to parallelize. That is, when we want the value of y for 4 different cases, each can be sent to a different processor on a server and this speeds up the process. However, GA is not massively parallel as cross over, reproduction, mutation and so on, cannot be parallelized.
5. It is not easy to make GA work very efficiently though it is easy to make it work.

In summary, GA is a contemporary powerful tool available to us to solve complex optimization problems. However, not all the features of GA can be established rigourously by mathematics. Even so, GA is certainly not a helter-skelter technique that has no philosophy or logic behind it.

Example 8.1: *The cost of engines plus fuel for a cargo ship (in lakhs of rupees per year for 100 tons of cargo carried) varies with speed and is given by $0.2x^2$ where x is the speed of the ship in m/s. The fixed cost of hull and crew (again in the same units) is given by $450/x$. Using genetic algorithms, perform 2 iterations to determine the optimal operating speed of the ship. The original interval of uncertainty can be taken as $0.5 \leq x \leq 25.5 m/s$ and 1.*

Solution:
The largest value of x is given as 25.5m/s and one decimal accuracy is required. So the number we are looking at is 255. The largest number with 8 bits is 255 and so, 8 bits are enough to represent x here. $y = 0.2x^2 + 450/x$; $0.5 \leq x \leq 25.5$ m/s We will work with 4n solutions, where n is the number of variables. In this case, n=1. Number of design solutions = 4n = 4.

We randomly take 4 values of x that are uniformly spread over the interval $0.5 \leq x \leq 25.5$. Here, when we are using 8 bits and the maximum value is 255, we must remember after all the operations are done and we are reconverting from binary to decimal, we have to divide by 10 to get the value of x.

Now we convert these values to binary and check for bias. We have 16 1s and 16 0s, which is very good. We can also generate the initial population by using a random number table. A typical random number is given at the end of the book. In this table, we proceed from the top to the bottom and once a column is over, we proceed from left to right. We now use the first random number. If this is less than 0.5, we generate '0' in the binary form of the first candidate solution. Else it is 1. We then go to the next random number and apply the same rule and proceed till we generate 32 bits.

Table 8.1: Initial population and mating pool for example 8.1

String No	Initial population, x	Initial population (binary)	y_i	$y_i/\sum y_i$	Count	Mating pool
1	4.2	00101010	110.67	0.286	1	01100101
2	10.1	01100101	64.95	0.168	2	01100101
3	16.4	10100100	81.23	0.21	1	00101010
4	23.5	11101011	129.59	0.335	0	10100100
$\sum$			386.44			

Next we calculate the value of y by substituting the values of x in the equation $y = 0.2x^2 + 450/x$. We determine $\Sigma y = 386.44$ and average fitness $\bar{y} = 98.61$. Please note that the original GA is for maximization while we are trying to do minimization.

For a minimization problem, the fittest is string 2 whose y value is 64.95 and relative fitness is 0.168. Now we want to know the count so that we can decide on the mating pool. We must keep total count as 4n so that we have a constant number of members in the mating pool. Now we do the single point crossover though we can do a uniform crossover bit by bit. We now generate the new population by deciding on (i) the mate for a particular parent and (ii) the exact point of crossover. We need to do both of these randomly (or stochastically).

Table 8.2: First iteration for example 8.1

Mating pool	Mate	crossover site	New population	New population (decimal)	y_i	$y_i/\sum y_i$	count
0110 | 0101	4	4	01100100	10	65	0.2	2
01100 | 101	3	5	01100010	98	65.12	0.206	1
00101 | 010	2	5	00101101	4.5	104.05	0.329	0
1010 | 0100	1	4	10100101	16.5	81.72	0.259	1
$\sum$					315.89		

ıry and check for bias. We have
We can also generate the initial
r table. A typical random number
; table, we proceed from the top to
we proceed from left to right. We
this is less than 0.5, we generate
ıdidate solution. Else it is 1. We
ıd apply the same rule and proceed

mating pool for example 8.1

y_i	$y_i/\sum y_i$	Count	Mating pool
110.67	0.286	1	01100101
64.95	0.168	2	01100101
81.23	0.21	1	00101010
129.59	0.335	0	10100100
386.44			

ubstituting the values of x in the
ermine $\Sigma y = 386.44$ and average
e original GA is for maximization
n.

t is string 2 whose y value is 64.95
vant to know the count so that we
ust keep total count as 4n so that
rs in the mating pool. Now we do
:an do a uniform crossover bit by
ation by deciding on (i) the mate
ıct point of crossover. We need to
tically).

ɔn for example 8.1

ew population (decimal)	y_i	$y_i/\sum y_i$	count
10	65	0.2	2
98	65.12	0.206	1
4.5	104.05	0.329	0
16.5	81.72	0.259	1
	315.89		

After these operations with the mates and crossover sites are done, as shown in the table, we generate the new population and evaluate the fitness (y_i) of all the candidates. We circle the one with the lowest y_i, meaning the one with the lowest cost.

Table 8.3: Second iteration for example 8.1

Mating pool	Mate	crossover site	New population	New population (decimal)	y_i	$y_i/\sum y_i$	count
011001 | 00	3	6	01100110	10.2	65	0.24	1
0110 | 0100	4	4	01100101	10.1	64.95	0.23	2
011000 | 10	1	6	01100000	9.6	65.31	0.24	1
1010 | 0101	2	4	10100100	16.4	81.23	0.29	0
$\sum$					276.5		

The beauty of the GA is that in just one iteration, Σy has come down from 386.44 to 315.89 and the average fitness has come down from 98.6 to 78.97. This means the cost has come down dramatically. This may sound controversial but we want the average fitness to decrease as we are looking at a minimization problem. The most important thing in Genetic Algorithm is that the variance in y or the difference between the minimum and maximum values of y will dramatically come down, with generations. Initially this variation was 64.64 and now it is 39.05. It will come down even further and we have to do one more iteration according to the question.

We can have a double crossover also in this stage if we want and in 3 or 4 iterations, we will get an average fitness that is very close to the correct answer. We may again say that we can do exhaustive search, but for multi modal problems, it will not work. However, GA searches in the whole solution space and even if a solution is very weak, it is not discarded. How does this happen? This can come in the form of mutations or some patterns of strings can be picked up and so on. In the tournament selection , we get a good mating pool compared to just ranking the trial solution where we compare fitness of two candidates at a time, just as we do in cricket or tennis tournament in ascending or descending order. This way, some diversity in the population is preserved and we can avoid premature convergence.

8.2.15 A research example

Suppose we have assorted electronic equipment in the cabinet of a desktop, each of which is generating some heat, we want to look at the optimum position of the heat source such that maximum cooling is

achieved. When is maximum cooling achieved? When the temperature is the lowest and the Nusselt number is the highest. But as we can see from Fig.8.3, there are infinite possibilities for these 4 devices to be kept in this space. Let us see how this problem is tackled using GA.

There is air flow here and it is a very simplified configuration of actual electronic equipment. There are 4 heat generating devices and we want to minimize the maximum temperature. We can solve this problem using a CFD software or develop our own code. Upon doing this, we can get the temperature distribution around each of these chips or heat sources.

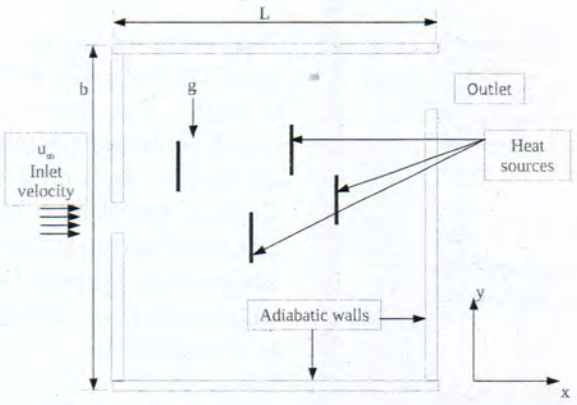


Figure 8.3: Schematic of electronic equipment in a desktop cabinet

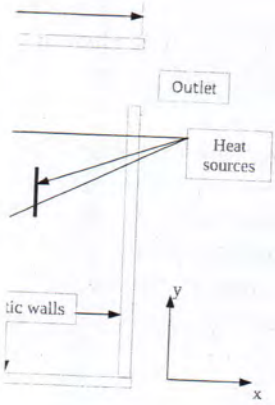
What we then do is basically take some arbitrary positions of the devices, get the CFD solution and obtain the maximum temperatures. Then we use the GA and generate 4 new positions. The new configurations are again solved for the maximum temperatures using CFD. We use GA again and get 4 new positions. We kept repeating this till convergence is obtained.

8.3 Simulated Annealing (SA)

We now look at another non traditional optimization technique called "Simulated Annealing". It is non traditional in the sense that, we do not use calculus and neither do we use the method of dividing into intervals

achieved? When the temperature
is the highest. But as we can see
ities for these 4 devices to be kept
)lem is tackled using GA.

simplified configuration of actual
ıt generating devices and we want
e. We can solve this problem using
ode. Upon doing this, we can get
ach of these chips or heat sources.



equipment in a desktop cabinet

arbitrary positions of the devices,
naximum temperatures. Then we
ions. The new configurations are
eratures using CFD. We use GA
pt repeating this till convergence

(SA)

ıal optimization technique called
tional in the sense that, we do not
method of dividing into intervals

and eliminating portions of it repeatedly. However it is also a search technique and we use some probabilistic laws and hence it is a *stochastic optimization technique*. Apart from that, we draw upon from certain principles used in metallurgy, eg: the process of annealing- slow cooling of an object in air, in solving the problem and hence the name "Simulated Annealing".
Some features of simulated annealing:

- It is similar to genetic algorithms in one sense that it is also based on probabilistic rules.
- It was developed by Kirpatrick, Gelatt and Vecchi in 1983 and was published in the Journal Science.
- It is an offshoot of the Metropolis algorithm which is a very powerful sampling technique in statistics.
- Simulated annealing is a global optimization procedure like genetic algorithms which ensures that there is no premature convergence.
- It is very robust but not very efficient in the sense that it does not converge quickly.
- It works very well for discrete optimization problems as well as for exceedingly complex problems.

8.3.1 Basic Philosophy

Consider the cooling process of molten metals through annealing. At high temperatures, the atoms in the molten state can move freely with respect to one another. However as the temperature reduces, the movement gets restricted. So analogously, during the initial iterations, the samples are free to move anywhere in the domain. For a single variable problem in x, x can move anywhere in the domain. Just like during the starting process of annealing, the atoms have the probability of being in any state.

But as the energy becomes low, the probability of attaining a higher energy state also becomes low. If the energy is high, the system has equal probability of attaining any of the states. In short, it means that the freedom gets reduced as the energy level goes down. Similarly in the initial iterations, the freedom is very high. If we have variables x_1 and x_2, they can move here and there initially. But as the iterations proceed, the conditions for accepting a particular sample, i.e. when we are proceeding from $(x_1, x_2)_i$ to $(x_1, x_2)_j$, the conditions for accepting

the latter $(x_1, x_2)_{i+1}$ become stricter and stricter.

At the end of annealing, the atoms begin to get ordered and finally form crystals with the minimum potential energy. If the cooling takes place very fast, it may not reach the final state of minimum potential energy. So the crucial parameter here is the cooling rate. The cooling rate is the one that decides if eventually we reach the state of minimum potential energy. So the cooling rate has to be tweaked, fine tuned or controlled in such a way that we get the optimum end product in metallurgy. Similarly, the convergence rate of the algorithm is controlled in such a way that we reach global convergence.

If the cooling rate is not properly controlled in metallurgy, instead of reaching the crystalline state, finally one may reach a poly crystalline state that may be at a higher energy state than the crystalline state. Analogously, for the optimization problem, we may get a solution which has converged prematurely. It is an optimum but unfortunately it is a local optimum. There is no guarantee that this is the global optimum.

So to achieve the absolute minimum state, the temperature must be reduced slowly. This slow cooling is known as annealing in metallurgy and simulated annealing mimics this process. Achieving the global optimum is akin to reaching the minimum energy state in the end.

8.3.2 Key point in SA

The key point is that the cooling is controlled by a parameter called the temperature that is closely related to the concept of Boltzmann probability distribution. According to the Boltzmann distribution, a system in thermal equilibrium at a temperature T has its energy distributed probabilistically according to

$$P(E) = e^{-\frac{E}{kT}} \tag{8.3}$$

In equation 8.3, k is *Boltzmann constant*, $1.38 \times 10^{-23} J/K$

Equation 8.3 suggests that as T increases, the system has uniform probability of being in any energy state. If T is very high, it means that the expression reduces to the exponent raised to the power of minus of a very low quantity, which makes it close to 1, i.e. $P(E) = e^{-very\ small\ quantity} \approx 1$, for all E.

While equation 8.3 suggests that when T increases, the system has

nd stricter.

begin to get ordered and finally
ıtial energy. If the cooling takes
final state of minimum potential
is the cooling rate. The cooling
y we reach the state of minimum
e has to be tweaked, fine tuned
get the optimum end product in
rate of the algorithm is controlled
vergence.

ıtrolled in metallurgy, instead of
ne may reach a poly crystalline
state than the crystalline state.
lem, we may get a solution which
ɔtimum but unfortunately it is a
that this is the global optimum.

state, the temperature must be
nown as annealing in metallurgy
process. Achieving the global
ıum energy state in the end.

:ontrolled by a parameter called
d to the concept of Boltzmann
to the Boltzmann distribution,
ı temperature T has its energy
:o

$$-\frac{E}{kT} \qquad (8.3)$$

ınt, $1.38 \times 10^{-23} J/K$

·eases, the system has uniform
te. If T is very high, it means
ɛponent raised to the power of
ıakes it close to 1, i.e. $P(E) =$

en T increases, the system has

uniform probability of being at any energy state, it also tells us that when T decreases, the value of the expression $P(E) = e^{-\frac{E}{kT}}$ takes on a very small value and hence the system has a small probability of being at a higher energy state. How does it work? The probability is exponential to the power of a negative quantity. So for small values of (-E/kT), P(E) will be $e^{-0.1}$, $e^{-0.2}$, $e^{-0.05}$ and so, there is a chance of getting different values of P(E). But once we have reached e^{-4}, e^{-5} and so on, P(E) is almost 0. Therefore by controlling T and assuming that the search process follows equation 8.3 (the Boltzmann probability distribution), the convergence of the algorithm can be controlled. So we use this Boltzmann distribution like condition, to decide if the next sample will be accepted or not.

What is it which is different in what we are now saying? If we are looking for a search algorithm, conventional thinking says that when Y(x) has to be minimized, when we go from X^0 to X^1 and from Y^0 to Y^1, we accept X^1 only if $Y^1 < Y^0$. In simulated annealing too, there is no problem with this. X is the design vector with variables x_1 to x_n. So what we are doing in simulated annealing is that if we are seeking a minimum and Y^1 decreases compared to Y^0, there can be no doubt in our mind that X^1 has to be accepted.

But the beauty is that if X^1 is such that $Y^1 > Y^0$, we do not reject X^1 right away. Instead we reject it with a probability. We get this probability from the Boltzmann distribution. We recast equation 8.3 as $P(E) = e^{-\frac{\Delta E}{kT}}$, where ΔE is the change in energy in the metallurgical context and is the change in objective function $Y^1 - Y^0$, in the context of optimization. Furthermore, the Boltzmann constant k can be made 1 for our algorithm and there are several ways to represent the "temperature" in the Boltzmann distribution. This temperature could be the average of Y for say 4 values of X.

How the system proceeds is like this. Initially if we want to start, what we do is, if we have only one X, we take 4 arbitrary values of X and obtain the 4 values of Y. We take the average. If we recall, GA also proceeded the same way. Now we assume that $\bar{Y} = T$ and generate a new sample X^1 from X^0. There are several ways of doing this. Either we use the random number table or a Gaussian distribution. Now we compare Y^1 and Y^0. If Y^1 decreases, we accept X^1. But if Y^1 increases, we apply the probability criterion and we will get a number between 0 and 1. We generate another random number r between 0 and 1. We compare r with P. If r is less than or equal to P, we accept the new sample.

The major departure in SA is that even if the objective function becomes worse, i.e. if for a maximization problem, Y decreases or for a minimization problem, Y increases, initially we allow such an aberration. But as we proceed, what happens is that, this T will come down as generations proceed because T represents the average Y. When T comes down, it has a smaller probability of being at a higher state. Therefore compared to the random number r that we are generating, this probability P which is also varying from 0 to 1, will be such that if the function decreases for a maximization or if the function increases for a minimization, as the iterations proceed, when the temperature T comes down, it will become more and more difficult for us to accept the sample, if ΔY increases for a minimization problem. The algorithm will proceed along the route of conventional thinking only after the solution space has been initially thoroughly searched.

In 1955, Metropolis suggested a way to implement the Boltzmann probability distribution in "simulated" thermodynamic systems. This can also be used in optimization problems too. The Metropolis Hastings algorithm is basically a sampling algorithm which helps us get samples of the design vector $X(x_1, x_2, ...x_n)$.

Let us say the current point is $X^{(t)}$ and the value of the function at that point is $E(t) = f(X^{(t)})$. The probability of the next point being at $X^{(t+1)}$ depends on $\delta E = E(t+1) - E(t)$ = change in objective function. For a minimization problem, applying the Boltzmann probability distribution, we get $P[E(t+1)] = min[1, e^{-\frac{\Delta E}{kT}}]$. If $\Delta E \leq 0$, Y^1 is less than Y^0, the probability is 1 and hence $X^{(t+1)}$ always gets accepted for the minimization problem. We are not questioning conventional thinking. While the acceptance is straight forward, the rejection is not so immediate. We reject only with a probability and this probability is such that the rejection becomes stricter and stricter as iterations proceed.

The random number r will always be between 0 and 1. But the P will be such that it becomes very close to 0 as the iterations keep progressing. The criterion is the same. We generate a random number r. If $r \leq P$, then accept it, else reject it and stick to the criterion. P will also vary between 0 and 1. So now if we consider the condition as stated before that $r \leq P$ for the sample to be accepted, in the first few iterations, when P is close to 1, the chance of X^1 being accepted is higher as the criterion $r \leq P$ may be easily satisfied. But as P reduces and becomes almost 0 as the iterations proceed, while r remains a random value between 0 an 1, the probability that X^1 will be accepted when $Y^1 \geq Y^0$ for a

even if the objective function
ːation problem, Y decreases or
ıses, initially we allow such an
appens is that, this T will come
represents the average Y. When
ɔility of being at a higher state.
ımber r that we are generating,
ʒ from 0 to 1, will be such that
ation or if the function increases
·oceed, when the temperature T
ıore difficult for us to accept the
ion problem. The algorithm will
thinking only after the solution
·ched.

to implement the Boltzmann
thermodynamic systems. This
ns too. The Metropolis Hastings
thm which helps us get samples

and the value of the function
e probability of the next point
$E(t+1) - E(t)$ = change in
ıroblem, applying the Boltzmann
$\cdot 1)] = min[1, e^{-\frac{\Delta E}{kT}}]$. If $\Delta E \leq 0$,
is 1 and hence $X^{(t+1)}$ always
ɔblem. We are not questioning
ːptance is straight forward, the
ɛct only with a probability and
on becomes stricter and stricter

tween 0 and 1. But the P will be
the iterations keep progressing.
· a random number r. If $r \leq P$,
o the criterion. P will also vary
r the condition as stated before
, in the first few iterations, when
ccepted is higher as the criterion
P reduces and becomes almost
mains a random value between
· accepted when $Y^1 \geq Y^0$ for a

minimization problem becomes lesser and lesser.

8.3.3 Steps involved in SA

So unlike GA, SA is generally a point by point method [SA for multiple points is also available].

- We start with a point and continue to the next point. The normal SA used is therefore a point by point method.
- We begin with an initial point and a high temperature T.
- A second point is created in the vicinity of the initial point by using a sampling technique and ΔE is calculated.
- If ΔE is negative, the new point is accepted right away.
- If ΔE is positive (for a minimization problem), the point is accepted with a probability of $e^{-\frac{E}{kT}}$, k=1. This completes one iteration.
- In the next generation, again a new point is chosen, but now the temperature T is reduced. This is where the cooling rate is controlled. Usually $T_{new} = \frac{T_{old}}{2}$.
- At every temperature, a number of points (4 or 6) is evaluated before reducing the temperature.
- The stopping criterion is either $\Delta E \leq \epsilon_1$ *or* $T \leq \epsilon_2$ *or* $|X^{i+1} - X^i| \leq \epsilon_3$.

The progress of iterations in a typical 2 variable minimization problem in x_1 and x_2 is shown in fig. 8.4. We can see that the initial search covers almost every possible sub-region of the solution space, so that premature convergence is avoided. Consider the cargo ship problem, we discussed while learning GA.

$$y = 0.2x^2 + \frac{450}{x};\ 0.5 \leq x \leq 25.5m/s.$$

Now let us take the first sample as $x_0 = \frac{(0.5+25.5)}{2} = 13m/s$. This is the mean such that we do not want samples to exceed 25.5 m/s and we do not want them to fall below 0.5 m/s. 99% of the time we can do this, if we follow a Gaussian or a normal distribution whose mean = 13m/s and whose $3\sigma = \frac{(25.5-0.5)}{2} = 12.5m/s$ (see fig.8.5). So we start with $x_0 = \mu = 13m/s$. How do we generate x_1 now? The normal

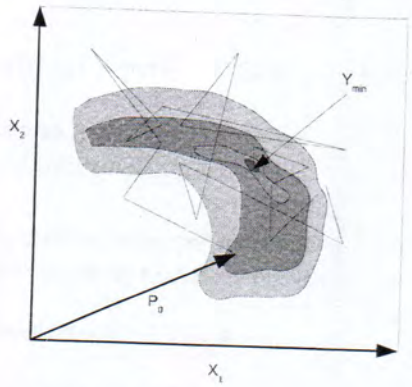


Figure 8.4: Illustration of the iteration process in a 2 variable minimization problem with SA

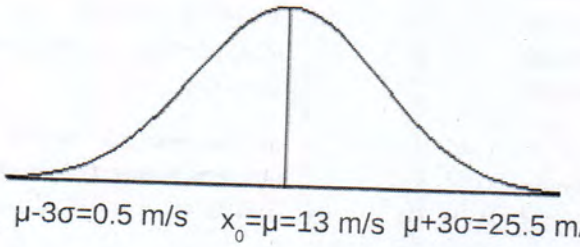
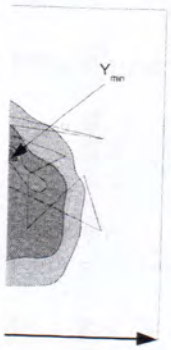


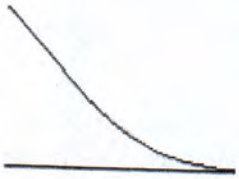
Figure 8.5: Typical Gaussian distribution for μ=13 and σ=4.17

distribution function can be written as $f = \frac{1}{\sqrt{2\pi}\times\sigma} \times e^{\frac{-(x-\mu)^2}{2\sigma^2}}$. What is the ordinate of this distribution? When $x = \mu$, $f = \frac{1}{\sqrt{2\pi}\times\sigma}$. This is the maximum probability. So what we do is first generate a random number . We will take it as f and since we know μ and σ, we will determine the new x. The problem with this procedure is sometimes it may give some meaningless values and since f here is between 0 and $\frac{1}{\sqrt{2\pi}\times\sigma}$ while we actually want f to be between 0 and 1. We need to use the normalized standard distribution.

We have a sample which is the mean. We want the next sample either to the left or right of this. So what we do here is generate the random number, divide it by $\sqrt{2\pi}\times\sigma$ and then equate to the probability distribution function 'f'. The new problem is that since it is $(x-\mu)^2$, when we generate the random value, it will always go to one side of μ. So what we do to overcome this is that we generate another random number



ation process in a 2 variable



m/s μ+3σ=25.5 m/s

bution for μ=13 and σ=4.17

$f = \frac{1}{\sqrt{2\pi}\times\sigma} \times e^{\frac{-(x-\mu)^2}{2\sigma^2}}$. What is
n $x = \mu$, $f = \frac{1}{\sqrt{2\pi}\times\sigma}$. This is the
s first generate a random number
v μ and σ, we will determine the
re is sometimes it may give some
between 0 and $\frac{1}{\sqrt{2\pi}\times\sigma}$ while we
We need to use the normalized

ın. We want the next sample
what we do here is generate the
nd then equate to the probability
blem is that since it is $(x-\mu)^2$,
will always go to one side of μ. So
generate another random number

between 0 and 1, say k. So if $k > 0.5$, we use $+\Delta x$ and if $k < 0.5$, we use -Δx.

So much of tweaking is required. But this is the variety or diversity we are introducing in the population. So if there is a treacherous function that goes up and down, the SA will not leave it, it will be able to catch it. The Golden section search, for example, will work only for unimodal functions. SA and GA methods can work for any function. They may be slow but they will get the global optimum. In any case, these are infinitely superior to exhaustive search.

So we have to use 3 sets of random numbers for simulated annealing. The first set of random numbers is for sampling, we can stick to rows 1 and 2 of the random numbers given in Table 8.4. Row 3 of the Table can be used for generating r. Row 4 or row 5 can be used for generating the value k, which decides whether we take $+\Delta x$ or $-\Delta x$.

What is the cooling schedule mentioned in step 6? $T^{i+1} = \frac{T^i}{2}$ is the cooling schedule. If T is high, convergence is slow, while if T is low, it may lead to premature convergence. Hence T and n govern convergence. To calculate the initial T, we draw a few random values of x and calculate the average of f(x). Now we set T=average f(x).

Example 8.2: *Consider the cargo ship problem. We would like to solve it using SA. Perform 4 iterations of the SA with an initial interval of uncertainty* $0.5 \leq x \leq 25.5$. *Use the random number table provided to you.*

Solution:

$$y = 0.2x^2 + \frac{450}{x};\ 0.5 \leq x \leq 25.5m/s.$$

Iteration 1

Calculate the initial cooling rate T. For this we need to use 4 values of x and the corresponding y. We already did this for GA (refer to example 8.1), we will use the same 4 values. We got y = 98.61 ; Hence T = 98.61.

$$x_0 = \mu = \frac{0.5+25.5}{2} = 13m/s$$

$$3\sigma = \frac{25.5-0.5}{2}; \sigma = 4.17m/s$$

We now draw a Gaussian around $x_0 = \mu = 13m/s$. The objective of writing the Gaussian distribution is that we have to solve this equation to obtain x_1.

$$\frac{0.001213}{\sqrt{2\pi}\times 4.16} = 1.16 \times 10^{-4}$$

$$1.16 \times 10^{-4} = \frac{1}{\sqrt{2\pi}\times 4.16} \times e^{-\frac{(x_1-13)^2}{2(4.16)^2}}$$

$$x_1 = 28.24 m/s$$

Using the random number table, we consider the first random number and use it to calculate x_1. But we get x_1 to be 28.24 m/s which is out of the range given for this problem for x. Hence we discard this and repeat the calculations with the next random number. We can do another thing in such a case. First we got 28.24 by adding to the positive side of 13. But we are allowed to do this with the negative side also. This is because whether Δx is positive or negative, we are taking its square in the calculations. We decide this by another random number from column 5 we said. But of course we will not do this here, but proceed with the second random number in column 1.

$$y(x_0) = 68.41 lakhs$$

$$f = \frac{0.499629}{\sqrt{2\pi}\times 4.16}$$

$$\Delta x = \pm 4.99 m/s;\ x_1 = 13 - 4.99 = 8.01 m/s;\ y(x_1) = 68.68 lakhs$$

Table 8.4: A Random Number Table

0.001213	0.898980	0.578800	0.676216	0.050106
0.499629	0.282693	0.730594	0.701195	0.182840
0.108501	0.386183	0.769105	0.683348	0.551702
0.557434	0.799824	0.456790	0.216310	0.876167
0.092645	0.589628	0.332164	0.031858	0.611683
0.762627	0.696237	0.170288	0.054759	0.915126
0.032722	0.299315	0.308614	0.833586	0.517813
0.352862	0.574100	0.265936	0.859031	0.433081
0.941875	0.240002	0.655595	0.385079	0.908297
0.199044	0.936553	0.888098	0.817720	0.369820
0.339548	0.543258	0.624006	0.091330	0.416789
0.155062	0.582447	0.858532	0.887525	0.337294
0.751033	0.239493	0.535597	0.333813	0.493837
0.634536	0.199621	0.650020	0.745795	0.791130
0.227241	0.191479	0.406443	0.081288	0.734352
0.721023	0.222878	0.072814	0.641837	0.442675
0.789616	0.052303	0.106994	0.558774	0.141519
0.760869	0.120791	0.277380	0.657266	0.792691
0.805480	0.826543	0.294530	0.208524	0.429894
0.585186	0.986111	0.344882	0.343580	0.115375

$\times 10^{-4}$

$\times e^{-\frac{(x_1-13)^2}{2(4.16)^2}}$

n/s

nsider the first random number
:t x_1 to be 28.24 m/s which is
n for x. Hence we discard this
:xt random number. We can do
:t 28.24 by adding to the positive
with the negative side also. This
gative, we are taking its square
another random number from
ll not do this here, but proceed
mn 1.

lakhs

$\frac{29}{.16}$

$.01m/s$; $y(x_1) = 68.68lakhs$

Number Table

0.676216	0.050106
0.701195	0.182840
0.683348	0.551702
0.216310	0.876167
0.031858	0.611683
0.054759	0.915126
0.833586	0.517813
0.859031	0.433081
0.385079	0.908297
0.817720	0.369820
0.091330	0.416789
0.887525	0.337294
0.333813	0.493837
0.745795	0.791130
0.081288	0.734352
0.641837	0.442675
0.558774	0.141519
0.657266	0.792691
0.208524	0.429894
0.343580	0.115375

We find that $y(x_1) > y(x_o)$. Though we are seeking a minimum, we do not reject it right away. We reject it based on the Boltzmann distribution. So we now generate random number r (we use the third column for this). The first value here is 0.5788. $r = 0.5788$, $P = e^{-\frac{\Delta y}{T}} = 0.997$. $r < P$ and therefore we accept this sample even though $y(x_1) > y(x_o)$. It may look counter intuitive but it works in the long run. We can see that it can be eminently programmed.

Iteration 2

$x_1 = 8.01m/s = \mu_1$, $T_1 = \frac{T_0}{2} = 49.31lakhs$. Here, we are going to keep 3σ the same as before. But there is a possibility that this value may go out of the defined range. But this is basic SA that we are doing. In the actual Metropolis, the 3σ also changes with every iteration depending on the current mean and it is normally taken as 5% of the current mean. If we keep σ the same and it so happens, that we generate a random number, such that the new estimate exceeds 25.5, we just discard this value and take the next random number and proceed.

$$0.108501 = e^{-\frac{(x_2-8.01)^2}{2(4.16)^2}}$$

$$x_2 = 16.78m/s;\ y(x_2) = 83.13lakhs$$

Let us apply the Metropolis algorithm once more. $y(x_2) > y(x_1)$. But we do not reject it outright. We generate r = 0.73. $P = e^{-\frac{\Delta y}{T}} = 0.74$. Since $r < P$, we have to accept it. But we can see that it is getting stricter. The denominator of the power of e is going down. We present a condensed version of iterations 3 and 4 below.

Iteration 3

x_2=16.78 m/s, σ=4.16 m/s.

$$0.557434 = e^{-\frac{(x_3-16.78)^2}{2(4.16)^2}}$$

x_3=21.28 m/s, $y(x_3)$= 111.71, $y(x_3) > y(x_2)$

We generate r=0.77, $P = e^{-\frac{\Delta y}{T}} = 0.313$, Since $r > P$ we have to reject x_3. Now select the next random number r=0.092645

$$0.092645 = e^{-\frac{(x_3-16.78)^2}{2(4.16)^2}}$$

$$x_3\text{=}7.71 \text{ m/s, } y(x_3)\text{= } 70.25, \; y(x_3)<y(x_2)$$

we generate the next random number r=0.46, $P = e^{-\frac{\Delta y}{T}} = 1.68$, so $r < P$ we accept the x_3 value. Now x_3 becomes 7.71 m/s.

Iteration 4

$$x_3\text{=}7.71 \text{ m/s, } \sigma\text{=}4.16 \text{ m/s}$$

$$0.762627\text{=}e^{-\frac{(x_4-7.71)^2}{2(4.16)^2}}$$

$$x_4 = 10.77 m/s; \; y(x_4) = 64.98 \; lakhs, \; y(x_4)<y(x_3)$$

we generate a random number r=0.17, $P = e^{-\frac{64.98-70.25}{24.65/2}} = 1.53$, $r < P$, we accept the value, $\therefore$ x_4=10.77 m/s.

So this is how simulated annealing works and after some time, by the zigzag path, it will cover the whole solution space well, so that the global optimum is not missed. If the objective function is computationally expensive, for example, if a CFD solution or something as complex is required for getting each value, we can develop a neural network. We can run it for certain combinations of x, validate it and train it. After we get the optimum, we can substitute the values of x back into our original forward model or governing equations and check if the values predicted by the neural network are the same, as got by substituting in the equations. This completes the loop.

- We are suggesting that the cooling rate algorithm for this problem can be $\frac{T}{2}$ at every stage. We can also have different rates. Ultimately the reduction in cooling rate has to be decided based on our problem. We do not want to reduce it by 4 times or 8 times or 10 times because the rejection will becomes very strict and though it may accelerate our convergence, the convergence could be premature.
- Only while starting, we got the average of y at 4 points and used it as T, as we did not know what this T was then.
- Also the new value of x becomes the mean of the distribution for that iteration. That is the way all sampling algorithms work. When we start with 13, the mean is around 13. When x reduces to 8, the mean is around 8. Next if x becomes 8.6, the mean is around 8.6. The new value of x becomes μ automatically.

Problems

8.1 Consider the problem of maximization of the volume of a rectangular box to be made of sheet metal of a total area of $2m^2$. The three dimensions of the box are length-x, breadth-y and height-z (all in m).

(a) Convert this to a two variable unconstrained optimization problem in say, x and y.

(b) We would like to use GA to solve the above problem, as a two variable maximization problem in x and y.

Perform three iterations of the GA with an initial population size of 4.

You may assume that $0.1 \leq x$ or $y \leq 2.55m$. Two decimal accuracy is desired on the dimensions. Decide appropriate strategies for crossover. No mutation is required. Report maximum, minimum and average fitness values for the initial population (which may be randomly chosen) and the populations at the end of the first, second and third iterations. Make maximum use of the random number tables.

8.2 Consider the problem of minimization of convective heat loss from a cylindrical storage heater that makes use of the solar energy collected by a suitable system. The volume of the tank is 5 m^3 and is fixed. The radius of the tank is "r" and the height is "h".

(a) Convert this to a single variable unconstrained optimization problem in radius "r".

(b) We would like to use Simulated Annealing (SA) to solve the above problem, as a single variable minimization problem in "r".

Perform four iterations of the SA with a starting value of r=2m. You may assume that $0.1 \leq r \leq 4$m. Use random number tables. The "initial temperature T" (used in the algorithm that does not correspond to the physical temperature in this problem) may be taken to be the average of four objective function values for appropriately chosen values of the radius.