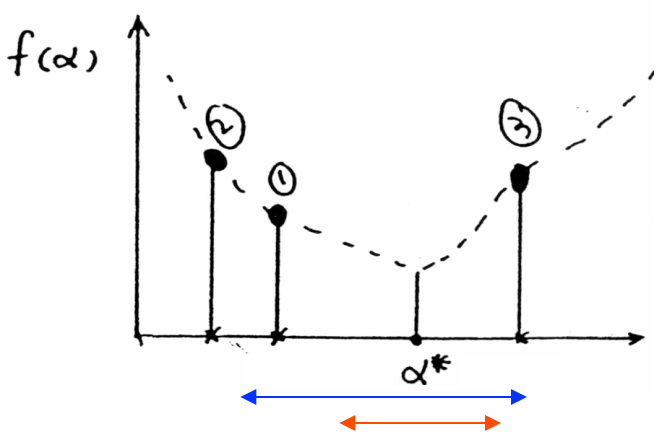
Introduction to Design Optimization: Search Methods

1-D Optimization

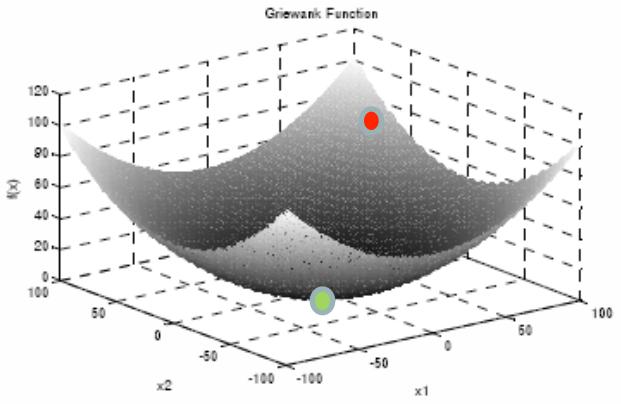


- The Search
  - We don't know the curve. Given α, we can calculate $f(\alpha)$.
  - By inspecting some points, we try to find the approximated shape of the curve, and to find the minimum, α^* numerically, using as few function evaluation as possible.
- The procedure can be divided into two parts:

  a) Finding the "range" or region " known " to contain α^* .

  b) Calculating the value of α^* as accurately as designed or as possible within the range – narrowing down the range.

N-Dimensional Search

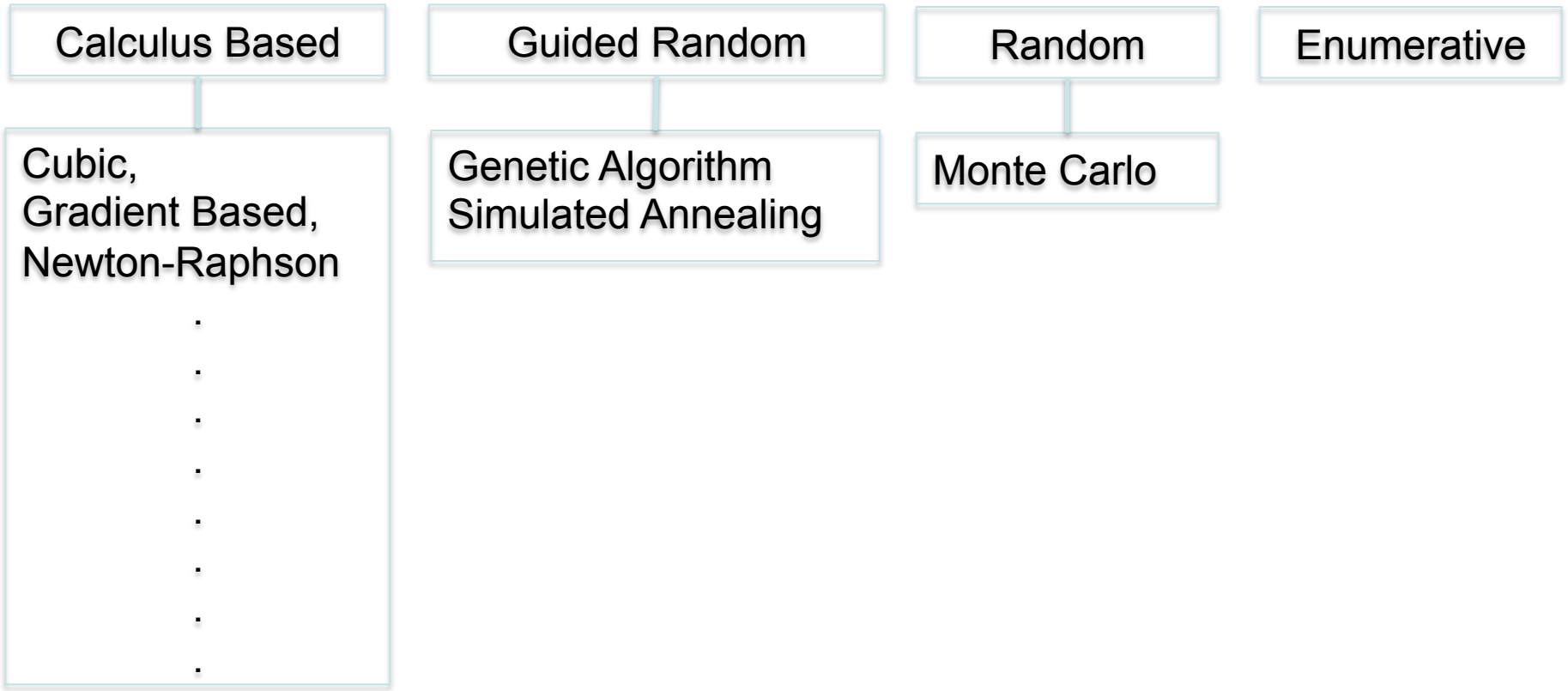


- The Problem now has N Design Variables.
- Solving the Multiple Design Variable Optimization (Minimization) Problem Using the 1-D Search Methods
- This is carried out by:
  - To choose a direction of search
    - To deal with one variable each time, in sequential order - easy, but take a long time (*e.g.* $x_1, x_2, \ldots, x_N$)
    - To introduce a new variable/direction that changes all variables simultaneously, more complex, but quicker (*e.g.* $\boldsymbol{S}$)
  - Then to decide how far to go in the search direction (small step $\varepsilon = \Delta x$, or large step determining α by 1D search)

Search Methods

- Typical approaches include:
  - Quadratic Interpolation (Interpolation Based)
  - Cubic Interpolation
  - Newton-Raphson Scheme (Derivative Based)
  - Fibonacei Search (Pattern Search Based)
  - Guided Random
  - Random
- Iterative Optimization Process:
  - Start point α_o → OPTIMIZATION → Estimated point α_k → New start point α_{k+1}
  - Repeat this process until the stopping rules are satisfied, then $\alpha^* = \alpha_n$.

N-D Search Methods



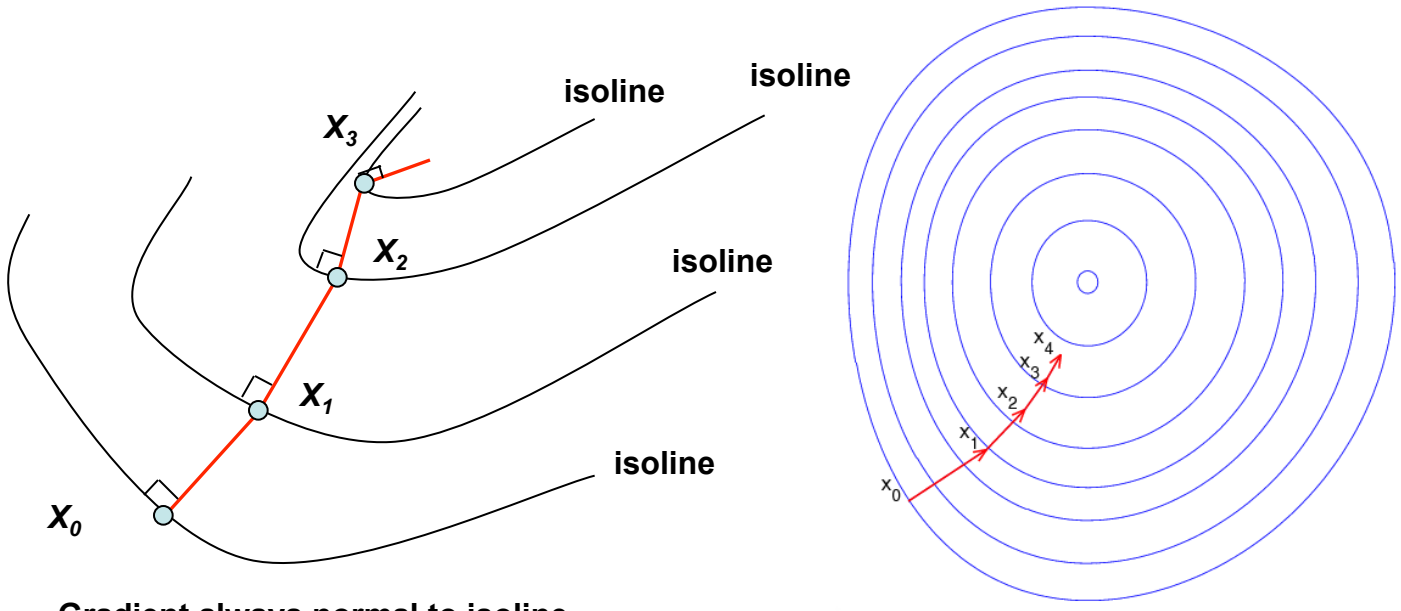
N-D Search Methods

- **Calculus Based**
  - Indirect method: knowing the objective function set the gradient to Zero. If we need to treat the function as a “black box” we cannot do this. We only know F(X) at the point we evaluate the function.
  - Direct Methods:
    - Steepest Descent method
    - Different flavors of Newton methods
- **Guided random search-combinatory techniques**
  - Genetic method
  - Simulated annealing
- **Random: Monte Carlo**
- **Enumerative method:** scan the whole domain. This is simple but time consuming

Steepest Descent or Gradient Descent

- The gradient of a scalar field is a vector field which points in the direction of the greatest rate of increase of the scalar field, and whose magnitude is the greatest rate of change. This means that if we move in its negative direction we should go downhill and find a minimum. This is the same path a river would follow. Given a point in the domain the next point is chosen as it follows:

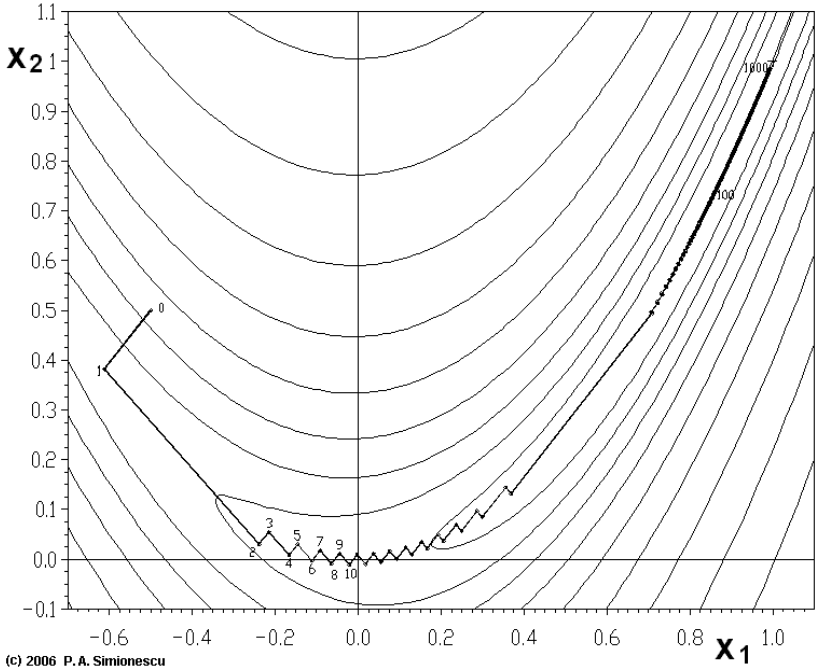
$$\mathbf{b} = \mathbf{a} - \gamma \nabla F(\mathbf{a})$$



Gradient always normal to isoline

Steepest descent and ill-conditioned, badly scaled functions

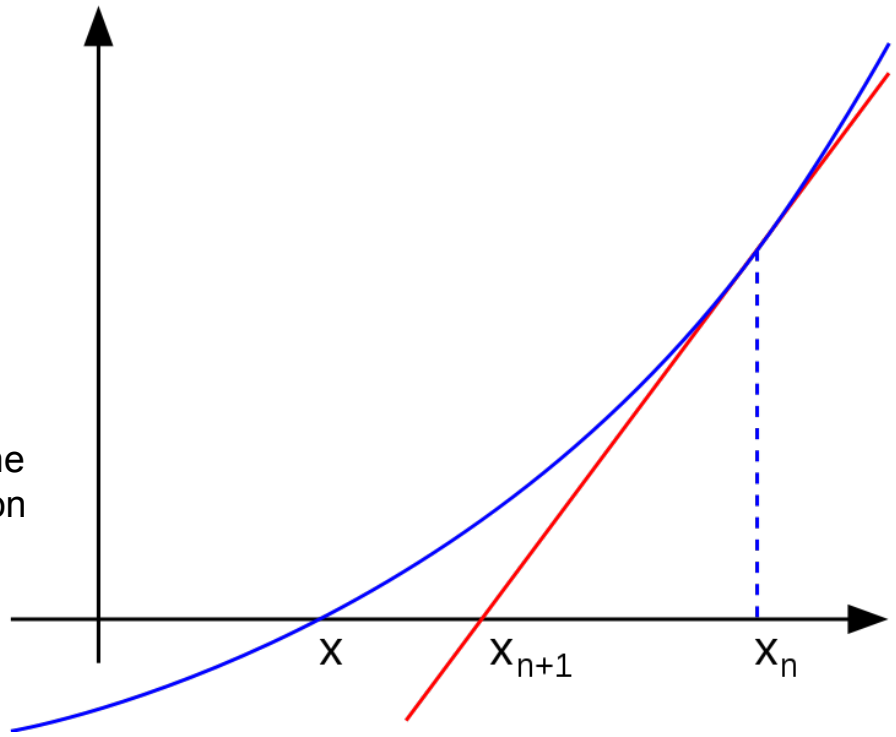
- Gradient descent has problems with ill-conditioned functions such as the Rosenbrock function shown here. The function has a narrow curved valley which contains the minimum. The bottom of the valley is very flat. Because of the curved flat valley the optimization is zig-zagging slowly with small stepsizes towards the minimum.



Newton-Raphson Method

- The Newton-Raphson method is defined by the recurring relation:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

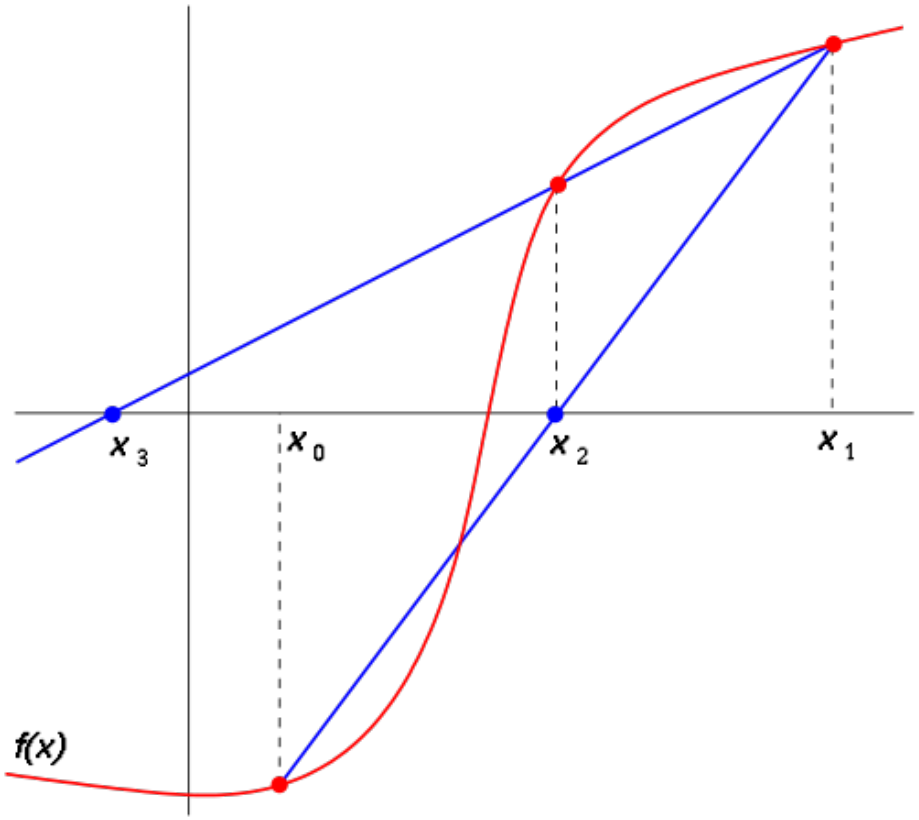


An illustration of one iteration of Newton's method (the function f is shown in blue and the tangent line is in red). We see that x_{n+1} is a better approximation than x_n for the root x of the function f.

Secant Method

- The Secant method is defined by the recurrence relation:

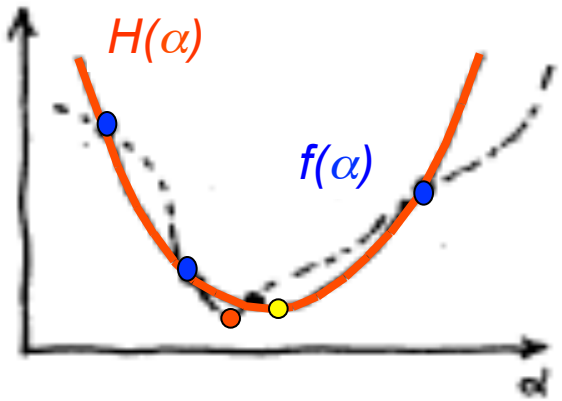
$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n)$$



The first two iterations of the secant method. The red curve shows the function f and the blue lines are the secants.

Quadratic Interpolation Method

$$f(\alpha) \Leftarrow H(\alpha) = a + b\alpha + c\alpha^2$$



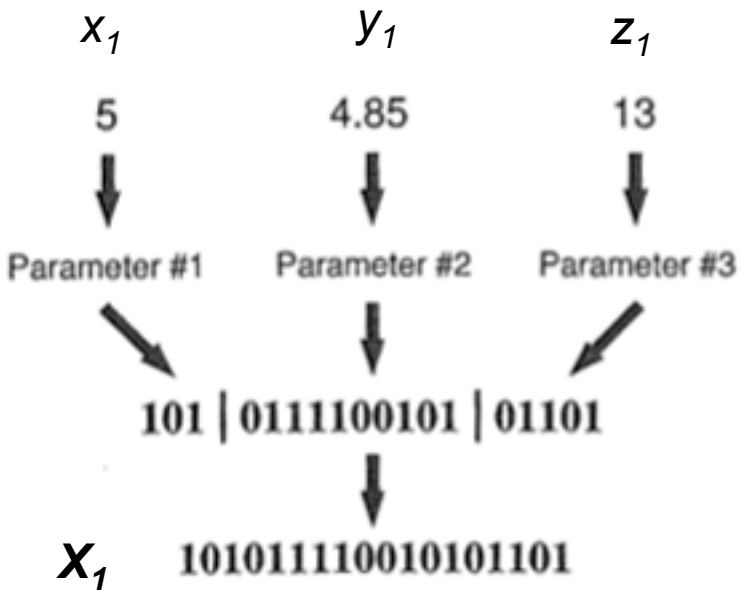
- Quadratic Interpolation uses a quadratic function, $H(\alpha)$, to approximate the "unknown" objective function, $f(\alpha)$.
- Parameters of the quadratic function are determined by several points of the objective function, $f(\alpha)$.
- The known optimum of the interpolation quadratic function is used to provide an estimated optimum of the objective function through an iterative process.
- The estimated optimum approaches the true optimum.
- The method requires proper range being found before started.
- It is relatively efficient, but sensitive to the shape of the objective

Combinatory Search: Genetic Algorithm

- Valid for discrete variables
- One of the best “all purposes” search method.
- Emulates the genetic evolution due to the “survival of the fittest”
- Each ***variable value*** >a ***GENE***, a binary string value in the variable range
- Vector variables ***X***> a ***CHROMOSOME***, a concatenation of a random combinations of ***Genes*** (strings) one per type (one value per variable). A ***Chromosome*** ($\boldsymbol{X_i}$) is a point in the ***X*** domain and is also defined as *genotype*.
- Objective Function ***F(X)***>*phenotype*. $\boldsymbol{F(X_i)}$ is a point in the Objective Function domain corresponding to $\boldsymbol{X_i}$.

Genetic Algorithm

- Construction of a chromosome $\boldsymbol{X_i}(x_i, y_i, z_i)$



Genetic Algorithm

1) Construct a population of genotypes (chromosomes) and evaluate the phenotypes (objective function).

2) Associate a fitness value between 0 and 1 to each phenotype with a fitness function. This function normalizes the phenotype (objective function) and assigns to its genotype an estimate (between 0 and 1) of its ability to survive.

3) Reproduction. The ability of a genotype to reproduce is a probabilistic law biased by the value given by the fitness function. Reproduction is done as it follows:

Given 2 candidate for reproduction, we have:

a) Cloning. The offspring is the same as the parents

b) Crossover. Chromosomes are split in two (head and tail) at a random point between genes and rejoined swapping the tails. When crossover is performed Mutation takes place. Each Gene is slightly changed to explore more possible outcomes.

4) Convergence. The algorithm stops when all genes in all individuals are at 95% percentile

<Genetic Algorithm>

```
BEGIN /*Genetic algorithm*/
    Generate initial population:
    Compute fitness of each individual

    WHILE NOT finished DO
    BEGIN /*Produce new generation*/

        FOR population_size/2 DO
        BEGIN /*Reproduction cycle*/
            Select two individuals from old generation for mating;
            /*Biased in favor of the fitter ones*/
            Recombine the two individual to give offspring;
            Compute fitness of the two offspring;
            Insert offspring in new generation;
        END

        IF population has converged THEN
            finished:=TRUE;

    END

END
```

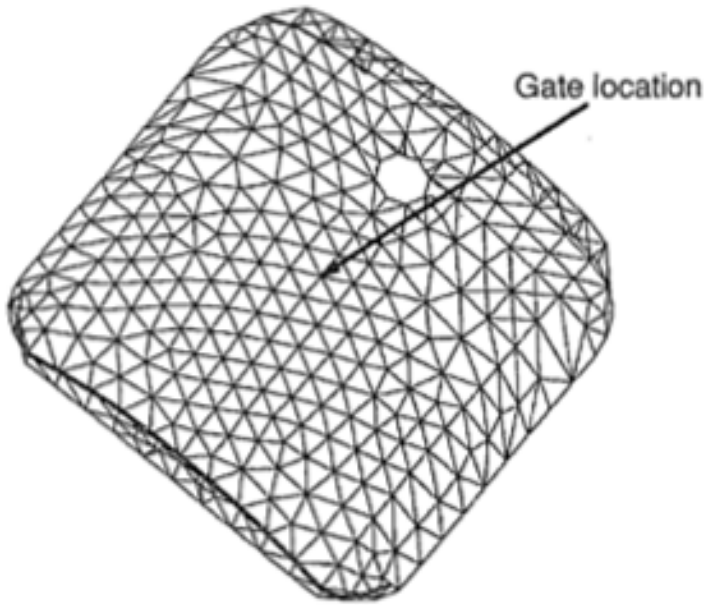
Genetic Algorithm. Example

9.5.2 Application

The genetic algorithm can be used to find the optimal values of the molding condition in the injection molding process of a plastic part. Let's consider an upper cover of a washing machine, as illustrated in Figure 9.19. We want to find the optimal mold temperature, melt temperature, and filling time for a maximum performance

Figure 9.19

Test model



Genetic Algorithm. Example

Figure 9.20

Range of molding conditions

	Minimum	Maximum	Values
Melt temperature	220	260	32
Mold temperature	50	70	32
Filling time	1	4	16

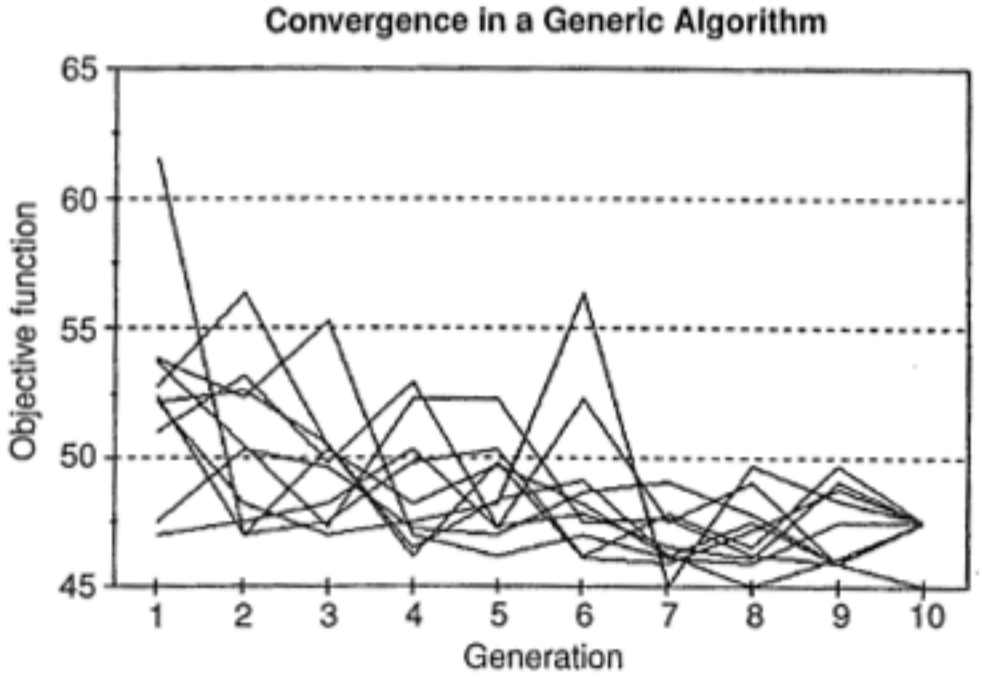
index that is related to the quality of the part to be fabricated. For simplicity, we assume that the performance index is calculated by simulation of the molding conditions. The lower and upper bounds of the optimization variables are given in Figure 9.20.

Now let's consider how to encode these variables. As shown in Figure 9.20, the melt temperature is assumed to have 32 discrete values and is thus represented by 5 digits. Similarly the mold temperature and the filling time are represented by 5 and 4 digits, respectively. Thus the chromosome in this example will be a binary string of length 14.

Genetic Algorithm. Example Results

Outcome:

	Melt temperature	Mold temperature	Filling time	Object function
Result	220.0	70.0	2.6	45.00



Simulated Annealing (wikipedia)

- The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat causes the atoms to become unstuck from their initial positions (a local minimum of the internal energy) and wander randomly through states of higher energy; the slow cooling gives them more chances of finding configurations with lower internal energy than the initial one.
- In the simulated annealing (SA) method, each point s of the search space is analogous to a state of some physical system, and the function E(s) to be minimized is analogous to the internal energy of the system in that state. The goal is to bring the system, from an arbitrary initial state, to a state with the minimum possible energy.

- By analogy with this physical process, each step of the SA algorithm attempts to replace the current solution by a random solution (chosen according to a candidate distribution, often constructed to sample from solutions near the current solution). The new solution may then be accepted with a probability that depends both on the difference between the corresponding function values and also on a global parameter T (called the temperature), that is gradually decreased during the process. The dependency is such that the choice between the previous and current solution is almost random when T is large, but increasingly selects the better or "downhill" solution (for a minimization problem) as T goes to zero. The allowance for "uphill" moves potentially saves the method from becoming stuck at local optima.

Monte Carlo Method

- Monte Carlo methods vary, but tend to follow a particular pattern:

- Define a domain of possible inputs.
- Generate inputs randomly from a probability distribution over the domain.
- Perform a deterministic computation on the inputs.
- Aggregate the results.

For example, consider a circle inscribed in a unit square. Given that the circle and the square have a ratio of areas that is π/4, the value of π can be approximated using a Monte Carlo method:

- Draw a square on the ground, then inscribe a circle within it.
- Uniformly scatter some objects of uniform size (grains of rice or sand) over the square.
- Count the number of objects inside the circle and the total number of objects.
- The ratio of the two counts is an estimate of the ratio of the two areas, which is π/4. Multiply the result by 4 to estimate π.
- In this procedure the domain of inputs is the square that circumscribes our circle. We generate random inputs by scattering grains over the square then perform a computation on each input (test whether it falls within the circle). Finally, we aggregate the results to obtain our final result, the approximation of π.

- To get an accurate approximation for π this procedure should have two other common properties of Monte Carlo methods. First, the inputs should truly be random. If grains are purposefully dropped into only the center of the circle, they will not be uniformly distributed, and so our approximation will be poor. Second, there should be a large number of inputs. The approximation will generally be poor if only a few grains are randomly dropped into the whole square. On average, the approximation improves as more grains are dropped.