Heuristic Programming

Heuristics are designed to find good approximate solutions to difficult combinatorial problems that otherwise cannot be solved by available optimization algorithms. A heuristic is a direct search technique that uses favorable rules of thumb to locate improved solutions.

- The advantage of heuristics is that they usually find (good) Solutions quickly.
- The disadvantage is that the quality of the solution (relative to the optimum) is generally unknown.

Early generations of heuristics are based on the greedy search rule that mandate making improvement in the value of the objective function with each search move. The search ends at a local optimum where no further improvements are possible. In the 1980s, a new generation of metaheuristic sought to improve the quality of the heuristic solutions by allowing the search to escape entrapment at local optima. The realized advantage comes at the expense of increased computations.

Metaheuristic Algorithms

A metaheuristic is an advanced technique or heuristic designed to locate, create, or select a heuristic that may provide adequately superior result to an optimization problem, especially with partial or imperfect information. A metaheuristic sample a set of solutions which is too large to be completely sampled. Metaheuristic may make few assumptions about the optimization problem being solved, and so they may be usable for a variety of problems. A heuristic is an approach to problem solving, learning, or discovery that employs a practical method not guaranteed to be optimal or perfect, but sufficient for the immediate goals. Where finding an optimal solution is impossible or impractical, heuristic methods can be used to speed up the process of finding a satisfactory solution.

Metaheuristic Algorithms

Genetic Algorithm (GA) This algorithm differ primarily in the manner in which the search escapes a local optimum. It is efficient with wide search space and the ability to arrive at a global optimum solution. It can be applied to engineering design problems.

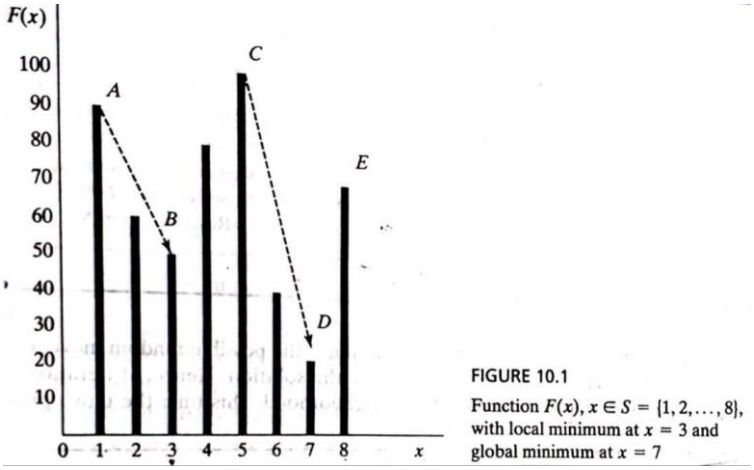


FIGURE 10.1
Function $F(x)$, $x \in S = \{1, 2, \ldots, 8\}$, with local minimum at $x = 3$ and global minimum at $x = 7$

- The GA mimics the biological evolution process of "survival of the fittest."
- Each feasible solution of a problem is regarded as a chromosome encoded by a set of genes.
- The most common gene codes are binary (0, 1) and numeric (0, 1, 2, ...).

For example, the chromosomes of a single variable whose feasible values are 0, 1, ..., and 8 can be represented by the binary codes (0000, 1000, 0100, 1100, 0010, 1010, 0110, 1110, and 0001). The chromosomes for a two-variable problem ($x1$, $x2$) with x1 = {0, 1} and x2 = {0, 1, 2, 3} can be represented by the numeric codes (0, 0), (0, 1), (0, 2), (0, 3), (1, 0), (1, 1), (1, 2), and (1, 3).
There are other coding schemes, including node code for network models A set of N feasible solutions is referred to as a population with N chromosomes. The fitness of a chromosome is measured in terms of an appropriate objective function. A fit chromosome yields a better value of the objective function.

- The overall idea of GA is to select two parents from a population. The genes of the two parents are then crossed over and (possibly) mutated to produce two children. The offspring replace the two weakest (least fit) chromosomes in the population, and the process of selecting new parents is repeated.

Example (Minimization of a Single-Variable Function) **from p 391 10.3**
The GA is applied to the single variable discrete problem with the feasible domain x = {1, 2, 3, 4, 5, 6, 7, 8}. We will arbitrarily specify a population of size N = 4 parents whose chromosomes are determined from X using uniform random sampling. The random number R is applied to the uniform distribution to generate the four members (N = 4) of the initial population and their fitness, the solution for i = 4 is a repeat of the solution for i = 3 ($x3 = x4$), hence the solution for i = 4 is discarded. The initial population is x0 = {8, 3, 5, 1}, and the associated best solution is $x* = 3$ with $F(x*) = 50$. To parents can be selected from the initial population $x0$ = {8, 3, 5, 1} in a number of ways:

1) Select the two fittest members.
2) Select the fittest member and then a random one from the remaining members.
3) Select two parents randomly from $x0$.

In this presentation, we use the third option. Specifically, the two random numbers $R1 = .2869\ and\ R2 = .0281\ yield\ x = 3\ with\ F(3) = 50\ and\ x = 8\ with\ F(8) = 70$
The two children are created from the two selected parents by using gene crossover. There are several methods for implementing crossover.
1. **Uniform crossover**. In this rule, parents' common genes apply to both children. The remaining genes for one child are determined randomly, with the other child getting the complement gene.

2. **One-point crossover.** The genes of parents $P1$ and $P2$ are split randomly at the same point and then swapped; that is, $P1 = (P11, P12)$ and $P2 = (P21, P22)$ yield the children chromosomes as $c1 = \{P11, P22\}$ and $c2 = \{P21, P12\}$.
3. **Multipoint crossover.** This rule extends the one-point crossover to multiple random points. For example, in a 2-point crossover, $P1 = (P11, P12, P13)\ and\ P2 = (P21, P22, P23)\ yield\ c1 = (P11, P22, P13)\ and\ c2 = (P21, P12, P23)$.

This example uses the uniform crossover rule. The one-point crossover rule will be used in Example 2.6. For the two parents (3, 8) generated in Table 1, we have $P1$ = (1 100) $P2$ = (0001)
Table 1: Generation of the Starting Population with $N = 4$.

i	R_i	x_i	Binary coded x_i	$f(x_i)$
1	0.9842	8	0001	70
2	0.3025	3	1100	50
3	0.5839	5	1010	100
4	0.5712	5	Discard	
5	0.0926	1	1000	90

In uniform crossover, the common (underlined) third gene in P1 and P2 carries over to both children. The remaining three genes are determined randomly as follows: For child 1, the gene is 1 $if\ 0 \le R < 0.5$ and 0 if $0.5 \le R \le 1$. The corresponding genes for child 2 are the complements of those assigned to child 1. For example, the three random numbers 0.2307, 0.7346, and 0.6220 show that genes 1, 2, and 4 for child 1 are 1, 0, and 0, respectively, which

automatically assigns the complement genes 0, 1, and 1 to child 2. Thus $c1$ = (1000) ($or\ x$ = 1) $c2$ = (0101) ($or\ x$ = 10) Child 2 corresponds to an infeasible solution (recall that the feasible range is x = 1, 2, . . . , 8). However, before discarding a child infeasible solution, we first apply random mutation (replacing one gene with another) and then check the mutated offspring for feasibility. If infeasibility persists, totally new offspring must be created (from the same parents). The process can be repeated as necessary until feasibility is achieved.
The probability of mutation is usually about.1, meaning a gene is mutated if $0 \leq R < 0.1$. For child 1, the random number sequence 0.6901, 0.7698, 0.0871, 0.9534 shows that the third gene only is mutated from 0 to 1, yielding $c1$ = (1010) ($or\ x$ = 5 $with\ F(5)$ = 100). For Child 2, the sequence 0.5954, 0.2632 0.6731, 0.0983 mutates gene 4 and yields $c2$= (010 0) ($or\ x$ = 2 $with\ F(2)$ = 60). Both child chromosomes are feasible, but neither yields a better solution. Hence, the solution $x*$ = 3 of the initial population continues to be the best so far. The least-fit parents in $x0$ (= 5 $and\ x$ = 1) are now replaced with the two offspring solutions (x = 5 $and\ x$ = 2). This, in effect, says that the next population is $x1$ = (8, 3, 5, 2). We now use x, to start a new iteration.

Summary of Genetic Algorithm:
Step 0:
(a) Generate a random population X of N feasible chromosomes.
(b) For each chromosome s in the selected population, evaluate its associated fitness. Record $s*$ as the best solution so far available.
(c) Encode each chromosome using binary or numeric representation.
Step 1:
(a) Select two parent chromosomes from population x.
(b) Crossover the parents genes to create two children.
(c) Mutate the children genes randomly.

infeasibility persists, totally new offspring must be created (from the same parents). The process can be repeated as necessary until feasibility is achieved.
The probability of mutation is usually about.1, meaning a gene is mutated if $0 \leq R < 0.1$. For child 1, the random number sequence 0.6901, 0.7698, 0.0871, 0.9534 shows that the third gene only is mutated from 0 to 1, yielding $c1$ = (1010) ($or\ x$ = 5 $with\ F(5)$ = 100). For Child 2, the sequence 0.5954, 0.2632 0.6731, 0.0983 mutates gene 4 and yields $c2$= (010 0) ($or\ x$ = 2 $with\ F(2)$ = 60). Both child chromosomes are feasible, but neither yields a better solution. Hence, the solution $x*$ = 3 of the initial population continues to be the best so far. The least-fit parents in $x0$ (= 5 $and\ x$ = 1) are now replaced with the two offspring solutions (x = 5 $and\ x$ = 2). This, in effect, says that the next population is $x1$ = (8, 3, 5, 2). We now use x, to start a new iteration.
Summary of Genetic Algorithm:
Step 0:

(a) Generate a random population X of N feasible chromosomes.
(b) For each chromosome s in the selected population, evaluate its associated fitness. Record $s*$ as the best solution so far available.
(c) Encode each chromosome using binary or numeric representation.

Step 1:

(a) Select two parent chromosomes from population x.
(b) Crossover the parents genes to create two children.
(c) Mutate the children genes randomly.

The Travel Salesman Problem (TSP) is a famous problem introduced in. The Traveling Salesman Problem is one of the most intensively studied problems in computational mathematics. These pages are devoted to the history, applications, and current research of this challenge of finding the shortest route visiting each member of a collection of locations, and returning to your starting point. In this chapter, we will apply the GA to this problem and provide an example. **3.2 TSP Mathematical Model** A TSP model is defined by the number of cities n and the distance matrix $\|dij\|$. The definition of a tour disallows linking a city to itself by a very high penalty to the diagonal elements of the distance matrix. A TSP is symmetric if $dij = dij$ for all i and j. Otherwise, the TSP is asymmetric.

$$x_{ij} = \begin{cases} 1, if\ city\ j\ is\ reached\ from\ city\ i \\ 0, otherwise \end{cases}$$

The TSP model is given as

$$\text{Minimize } z = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij}\, x_{ij}\ \ d_{ij} = \infty\ for\ all\ i = j$$

Subject to: Type equation here.

$$\sum_{j=1}^{n} x_{ij} = 1,\ \ i = 1,2,\dots n$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1,2,\dots n$$

$$x_{ij} = (0,1)$$

TSP Genetic Algorithm In the genetic metaheuristic introduced in Chapter 2, two parents are selected from a population to create two children. The children then become parents themselves replacing the two least-fit (in terms of tour

length) parents in the population. The process of creating children and of retiring parents is repeated until a termination condition is reached. The following is a description of the main elements of the genetic metaheuristic as it applies to the TSP. 1. **Gene coding.** The coding can be binary or numeric. The literature presents heuristics based on both types of coding. This presentation adopts the direct numeric tour code (e.g., 1-2-5-4-3-1). 2. **Initial population.** The first step is to identify the sets of outgoing nodes from each in the network that can be reached by a finite tour leg. Starting from a specific (home) node, a tour is constructed by adding in the rightmost position a unique non redundant node selected from among all the outgoing nodes of the last-added node. If a point is reached where no unique outgoing node exists, the entire process is repeated until a finite-length tour is found. The requirement stipulating that outgoing nodes be reached by finite links guarantees that the constructed tour is feasible (has a finite length). Unlike tabu and simulated annealing where a new search move can be infeasible, infeasible parent tours may never lead to the creation of feasible child tours. This result is particularly true when the distance matrix is sparse.

Table 3.3: Steps for Creating Children C1 and 2 from Parents P1 and P2 Using Order Crossover

Action	Example (assume n=7)
Select PI and P2 from population.	P1=1-2-**5**-**4**-**3**-7-6 (link back to node)
	P2=5-4-**2**-**6**-**3**-1-7(link back to node)
Randomly select two crossover points c_1 and c_2 with $c_1<c_2$.	R= 4425 yields C_1 =int(7*3425)
Swap positions (c_1. c_1 +1,... c_2) in P_1 and P_2 to partially form C_2 and C_1.respectively. Create list LI(L2) by rearranging the elements of P1(P2) in the clockwise order	R= 7123 yields C_2 =int(7*7123)
	C1=?-?-**2**-**6**-**3**-?-?
c_2 + 1. c_2 +2....n, 1.2..... . c_2 .	C2=?-?-**5**-**4**-3-?-?
From LI (L2).create LI' (L2') by deleting the nodes already assigned to C1(2) in step 2 while preserving the order in Li and L2.	L1=(7,6,1,2,5,4,3)
	L1=(1,7,5,4,2,6,3)
Assign the elements of L1' (L2) to the missing elements in C1(C2) in the order	L1`=L1-(2,6,3)=(7,1,5,4)
	L2`=L2-(5,4,3)=(1,7,2,6)
c_2 + 1. c_1 + 2,...,n. 1.2 …c_1 -1.	C1=5-4-2-63-7-1(link back to node5)
	C1=2-6-5-4-3-1-7(link back to node5)

3. **Child creation**. The process starts by selecting two parents Pl and P2, whose genes are swapped to create two children, C1 and C2. We will assume that Pl represents the best parent (in terms of tour length) and P2 the next best. There are numerous ways for gene swapping [see Larrañaga et al. (1999) for a list of 25 such procedures]. In this presentation, we will use the ordered crossover procedure, whose steps are explained in Table 3.3. The proposed procedure for creating children may lead to infeasible tours (with missing legs). If this happens, the procedure should be repeated as necessary until or spring feasibility is realized.

4. **Mutation**. Mutation in child genes takes place with a small probability of interchanging the nodes of two randomly selected positions in the tour (excluding those of the home node). Random selection may be repeated to secure two distinct positions.

Example 3.2: The matrix below summarizes the distances in miles in a 5- cities TSP.

$$\|d_{ij}\| = \begin{pmatrix} \infty & 120 & 220 & 150 & 210 \\ 120 & \infty & 100 & 110 & 130 \\ 220 & 80 & \infty & 160 & 185 \\ 150 & 40 & 160 & \infty & 190 \\ 210 & 130 & 185 & 0 & \infty \end{pmatrix}$$

The list of outgoing nodes can be determined from the distance matrix as:
Table 3.4: provides the details of iterations 1, 2, and 11. Iteration 11 provides the best (which also happens to be optimum). The intervening iterations were omitted to conserve space.

Node i	outgoing nodes
1	[2,3,4,5]
2	[1,3,4,5]
3	[1,2,4,5]
4	[1,3,5]
5	[1,2,3]

We demonstrate the determination of initial population (6 parents) in iteration 1 by considering parent 1. Starting with home node 1, node 4 is selected randomly from the outgoing node set [2, 3, 4, 5]. Next, the outgoing nodes from node 4 are [1, 3, 5] – [1] because [1] is already in the partial tour. Selecting node 5 randomly yields the partial tour 1-4-5. The process is repeated until all tour 1-4-5-2-3-1 is constructed. Keep in mind that if the construction of the tour is dead. Ended (no new nodes can be added), then the entire process must be repeated anew. For simple, tour construction cannot continue past partial tour 1-2-3-5 because there is no link from node 5 to (the only remaining) outgoing node 4.

iteration	Member	Tour	Crossovers	Length
1	1	1-4-5-2-3-1		790
	2	3-2-4-5-1-3		810
	3	1-2-4-5-3-1		825
	(Parent 2) 4	2-5-**3-4-1**-2	3 and 5	745
	5	3-4-5-1-2-3		780
	(Parent 1) 6	1-5-**3-2-4**-1		735
	Child 1	5-2-**3-4-1**-5		750
	Child 2	5-1-**3-2-4**-5		810
2	1	1-4-5-2-3-1		790
	2	5-1-3-2-4-5		810
	3	5-2-3-4-1-5		750
	(Parent 2) 4	2-5-3-**4-1**-2	4 and 5	745
	5	3-4-5-1-2-3		780
	(Parent 1) 6	1-5-3-**2-4**-1		735
	Child1	5-3-2-4-1-5		735
	Child 2	5-3-1-2-4-5		825
…	…	…	…	…
11	(Parent 2) 1	1-5-**3-2**-4-1		735
	2	5-3-2-4-1-5		735
	3	5-3-2-4-1-5		735
	4	5-3-2-4-1-5	3 and 4	735
	(Parent 1) 5	4-5-**3-2**-1-4		725
	6	4-5-3-2-4-1		735
	Child1	1-5-3-2-4-1		725
	Child 2	5-3-1-2-4-5		835