CHAPTER-4

SOLUTION OF NP-COMPLETE AND NP-HARD PROBLEMS BY HYBRID GENETIC ALGORITHM

This chapter illustrates the Greedy Genetic Algorithm to solve TSP, MKP and N-Queen problem. Algorithms for greedy initial population, greedy cross over and greedy mutation are proposed. These greedy algorithms are applied on TSP, MKP and N-Queen problems.

4.1 Greedy Approach

Greedy approach is technique to solve problems in Mathematics, Operation Research, Optimization, Computer Science and Engineering. In this work the heuristic approach is used which select the best local solution at each stage to find the optimal global solution. Greedy algorithm is used to solve many problems such as minimum spanning tree, knapsack problem, activity selection problem etc. In the presented work, genetic algorithm operators are designed using greedy approach and applied to solve NP-Hard and NP-Complete problems like Travelling Salesman Problem, N-Queen Problem, and 0/1 Multiple Knapsack Problem.

4.2 Proposed Greedy Algorithm to Solve TSP

This section is based on our publications, *"Solving Travelling Salesman Problem Using Greedy Genetic Algorithm GGA",* published in International Journal of Engineering and Technology (IJET) Vol 9 No 2 Apr-May 2017. Greedy Genetic Algorithm GGA to solve TSP problem is presented.

To solve TSP problem, the proposed Greedy Genetic Algorithm (GGA) uses greedy approach on two operations of the genetic algorithm:

1. Greedy initial population generation
2. Greedy cross over

4.2.1 Greedy initial population generation

In this step of genetic algorithm, an initial population of chromosomes is generated. This population is called as seed population. Random initial population generation is most common technique for it and thus applied. In this proposed GGA algorithm the initial population is hybrid i.e. some chromosomes of the initial population are generated randomly and some chromosomes are generated using greedy algorithm (nearest neighbor first). Because greedy algorithm provides either best solution or near to best solution that's why the fitness value of greedy chromosomes is very high. Algorithm 4.1 is showing steps for generating chromosomes using random approach for solving TSP.

Algorithm 4.1 Algorithm for random generation of chromosomes for TSP

Input: An instance of TSP problem

Output: A chromosome of given TSP instance

1. *Initialise chromosome CHR as a list of cities and add STARTING_CITY in CHR*
2. *Add all the cities other than the starting city in NOT_VISITED list*
3. *Repeat steps 4 to 6 until NOT_VISITED is not empty*
4. *Select a city CURRENT_CITY from NOT_VISITED*
5. *Add CURRENT_CITY in CHR*
6. *Remove CURRENT_CITY from NOT_VISITED*
7. *Return CHR*

Algorithm 4.2 is showing steps for generating chromosomes using greedy approach.

Algorithm 4.2 Algorithm for generating chromosomes using greedy approach

Input: An instance of TSP problem

Output: A chromosome of given TSP instance

1. *Initialise chromosome CHR as a list of cities and add STARTING_CITY in CHR*
2. *Set CURRENT_CITY = STARTING_CITY*
3. *Add all the cities other than the STARTING_CITY in NOT_VISITED list*

4.	*Repeat steps 5 to 8 until NOT_VISITED list is not empty*
5.	*Select NEXT_CITY from NOT_VISITED which is at the minimum distance from CURRENT_CITY.*
6.	*Remove NEXT_CITY from NOT_VISITED list*
7.	*Add NEXT_CITY in CHR*
8.	*Set CURRENT_CITY = NEXT_CITY*
9.	*Return CHR*

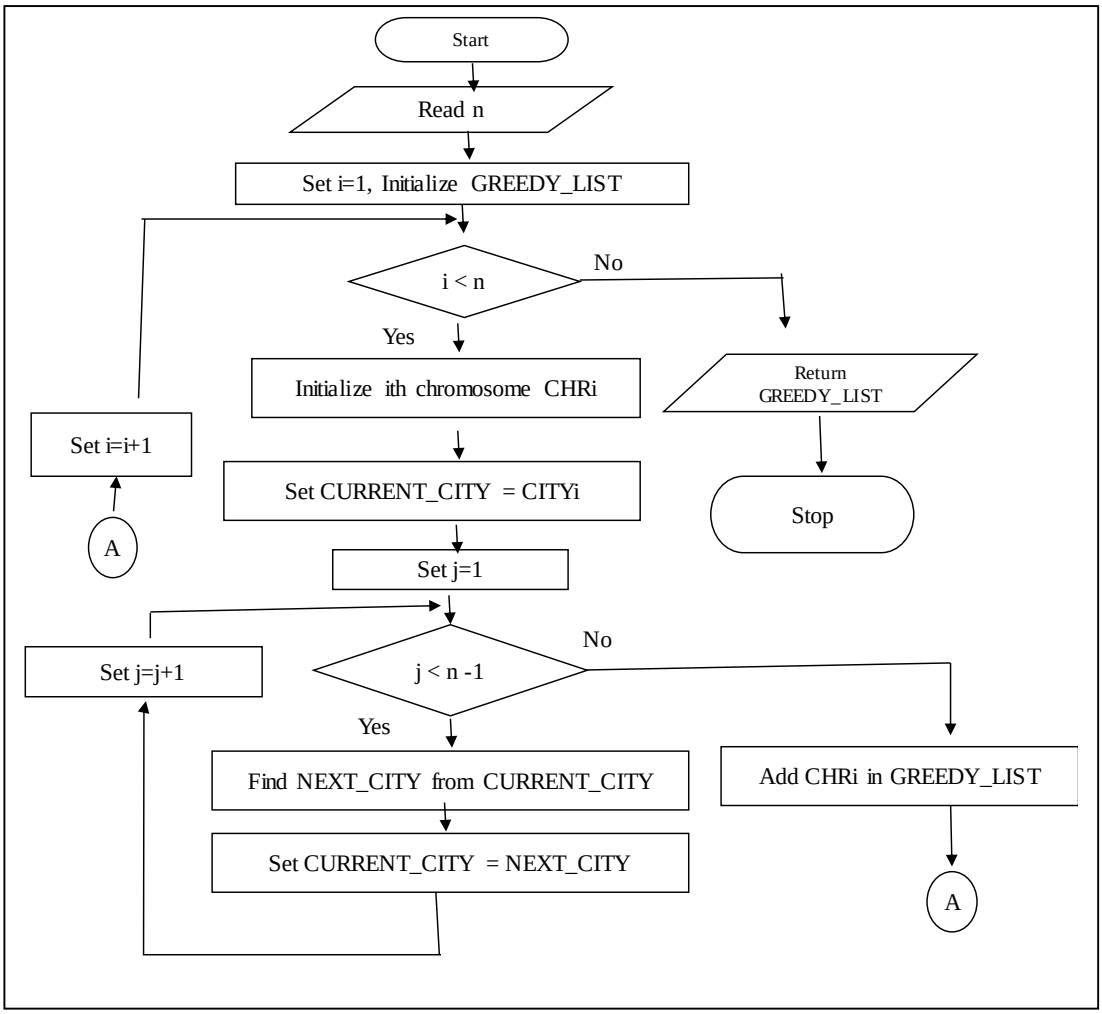

Figure 4.1: Flowchart to Generate a List of Greedy Chromosomes for TSP Problem

Figure 4.1 is showing a flowchart to generate a list of greedy chromosomes. This list of greedy chromosomes is added to the list of chromosomes generated randomly. So the initial population a combination of chromosomes generated using random approach and greedy approach. This hybrid initial population improves the overall convergence of greedy genetic algorithm, which is discussed and analyzed in here.

4.2.2 Greedy cross over

While going through the literature it has been observed that if the cross over operator uses some greedy technique, the overall performance improves, thus the cross over operator is optimized using greedy approach. The nearest neighbor first greedy approach is used in cross over operation. One-point cross over is used here to perform the cross over. The child is copied from one parent from starting point to cross over point. After cross over point cities are selected using nearest neighbor approach. Algorithm 4.3 is illustrating the steps of greedy cross over operation.

Algorithm 4.3 Algorithm for greedy cross over operation
Input: Two parent chromosomes PARENT1 and PARENT2
Output: Two children chromosome generated using cross over

1. *Initialise chromosome CHR_CHILD1 and CHR_CHILD2*
2. *From starting point to cross over point, copy cities from PARENT1 into CHR_CHILD1*
3. *From cross over point to end point, copy cities from PARENT1 into CHR_CHILD1 using nearest neighbour approach.*
4. *From cross over point to end point, copy cities from PARENT2 into CHR_CHILD2*
5. *From starting point to cross over point, copy cities from PARENT2 into CHR_CHILD2 using nearest neighbour approach.*
6. *Return CHR_CHILD1 and CHR_CHILD2*

4.2.3 Greedy genetic algorithm to solve TSP

The proposed greedy genetic algorithm to solve TSP uses Algorithm 4.1, Algorithm 4.2 and Algorithm 4.3. The steps of proposed greedy genetic algorithm are illustrated in Algorithm 4.4. It uses Algorithm 4.1 and Algorithm 4.2 to generate greedy initial population. Algorithm 4.3 is used to perform greedy cross over.

Algorithm 4.4 Proposed Greedy Genetic Algorithm GGA for solving TSP
Input: A given TSP problem
Output: Solution for the given TSP problem

1. *Generate hybrid INITIAL_POPULATION of chromosomes using Algorithm-1 and Algorithm-2*
2. *Calculate fitness of every chromosome in INITIAL_POPULATION*
3. *Repeat steps 4 to 11*
4. *Perform selection to select chromosomes which will participate in cross over.*
5. *Perform greedy cross over (using Algorithm 4.3) on parent chromosomes and generate CHILDREN_CHROMOSOMES_LIST of children chromosomes*
6. *Calculate fitness of every chromosome in CHILDREN_CHROMOSOMES_LIST*
7. *Add all chromosomes in CHILDREN_CHROMOSOMES_LIST into INITIAL_POPULATION*
8. *Perform mutation on chromosomes of INITIAL_POPULATION*
9. *Calculate fitness of all chromosomes of INITIAL_POPULATION*
10. *Generate NEXT_POPULATION from INITIAL_POPULATION by discarding least fit chromosomes*
11. *If terminating criteria satisfied, then go to step 12*
12. *Return chromosome with highest fitness value from INITIAL_POPULATION as solution*

Algorithm 4.4 is illustrating all the steps of Greedy Genetic Algorithm to solve any instance of TSP problem. Various types of standard instances of TSP problem such as

EIL51, ATT48, EIL76 etc. are solved using algorithm 4.4. The results are explained in chapter 5.

4.3 Proposed Greedy Algorithm to Solve N-Queen Problem

N-Queen problem is already explained in chapter-1. In this work, the N-Queen problem is solved using greedy genetic algorithm. The proposed algorithm first encodes the N-Queen problem into genetic form using permutation encoding. In GA, a solution is represented in the form of chromosome. For a given instance of N-Queen problem the chromosome is a list of numbers showing placements of different queens in different columns on a chess board of size *n***n*. Let 4 queens are to be placed on a 4*4 chess board. An instance of chromosome for placing 4 queens is shown in Figure 4.2.

Figure 4.2: Chromosome of 4-Queen Problem

In the given chromosome the first queen is placed in column 3 of row 1. The second queen is placed in column 1 of row 2. The third queen is placed in column 4 of row 3 and fourth queen is placed in column 2 of row 4. The placements of queens of given chromosome is shown in Figure 4.3, which will be further checked whether any the queens attacking each other.

4.3.1 Initial population generation for N-Queen problem

Initial population contains the seed chromosomes to start GA. For N-Queen problem a set of chromosomes (Equal to population size) is generated randomly using permutation encoding scheme. Initially the population is generated randomly and the chromosomes may have any number of attacks(clashes) between the queens. This step will generate a random INITIAL_POPULATION of chromosomes.

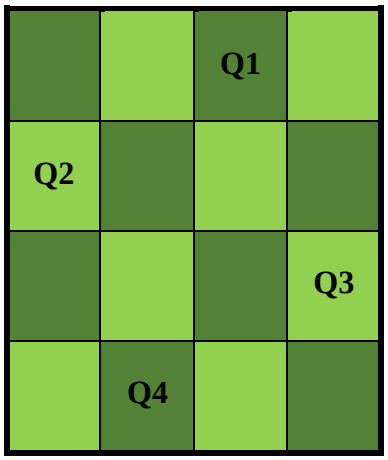

Figure 4.3: Placements of Queens on a 4*4 Chess Board

4.3.2 Fitness function for N-Queen problem

Fitness function performs a very important role in genetic algorithm. The fitness function determines the quality of a given chromosome to solve the problem. To solve N-Queen using GA, the fitness of chromosomes in population is to be maximized. The fitness of a chromosome is calculated by calculating number of clashes between the Queens in a given chromosome. Smaller is the number of clashes, higher is the fitness of a chromosome.

In N-Queen problem, the fitness of a chromosome 'x' is calculated as follows:

$$Fitness\ (x)\ =\ \frac{1}{Clash\ Count}$$

Where clash count is the total number of clashes (sum of row clashes, column clashes and diagonal clashes) between the queens.

In the chromosome shown in figure 4.3 for 4-Queen problem, the clash count is zero. So it is the required solution of the problem. The fitness function calculates the fitness of every chromosome of the population.

4.3.3 Greedy mutation operator for N-Queen problem

In this work, greedy approach is used to perform mutation operation. The greedy approach is used to improve the fitness of the chromosomes. The child chromosome is mutated in such a way that the fitness value of chromosome become better. To ensure it while performing the mutation some changes in the chromosomes are made which reduced the number of clashes between the queens. We know, less the clashes better is the fitness of the chromosome. To perform greedy mutation a queen is selected which is producing clashes. That queen position is changed and shifted to any other location randomly. This greedy approach reduced the number of clashes while performing the mutation.

3	6	1	8	0	5	4	2	5	9

Figure 4.4: A Chromosome of 10-Queen Problem

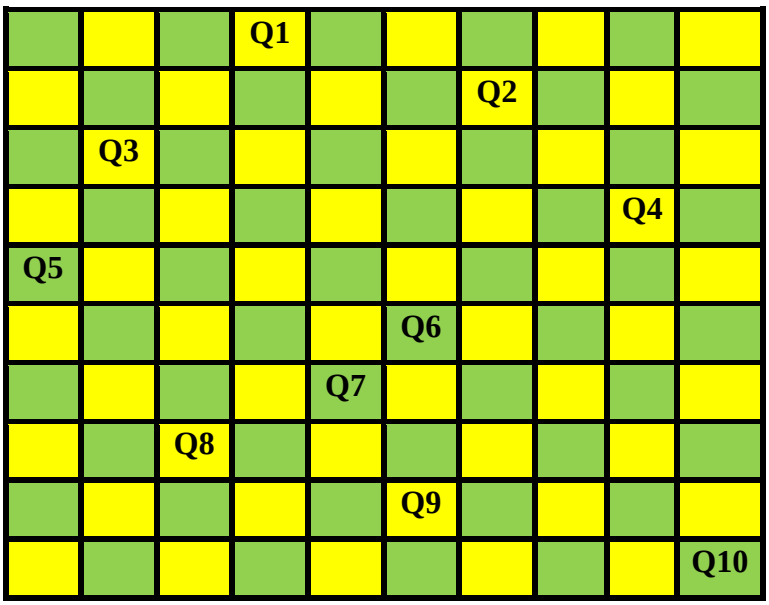

Figure 4.5: Placement of Queens of 10-Queen Problem on a 10*10 Chessboard

Figure 4.4 is showing an instance of 10-Queen problem. Q1, Q2, … , Q9 are the queens which are placed in different position on a 10*10 chess board. The integer value is

specifying the placement of 10 queens in different columns. Figure 4.5 is showing the given chromosome by placing the queens at different places specified in chromosome.

The number of clashes in the given chromosome is 10. Five queens are attacking to each other. The attacking queens are Q6 with Q7, Q6 and Q10, Q1 with Q3 and Q2 with Q4. So the total number of clashes is 10 for the chromosome shown in Figure 4.5.

In greedy mutation a queen is selected which is clashing with other queen(s) and the position of this queen is changed randomly. Let queen Q6 is selected and using greedy mutation the position of this queen is shifted from column 6 to column 8. Figure 4.5 is showing the chromosome before mutation and Figure 4.7 is showing the chromosome after mutation. Figure 4.6 is showing the placements of queens after mutation.

3	6	1	8	0	7	4	2	5	9

Figure 4.6: Mutated Chromosome of 10-Queen Problem

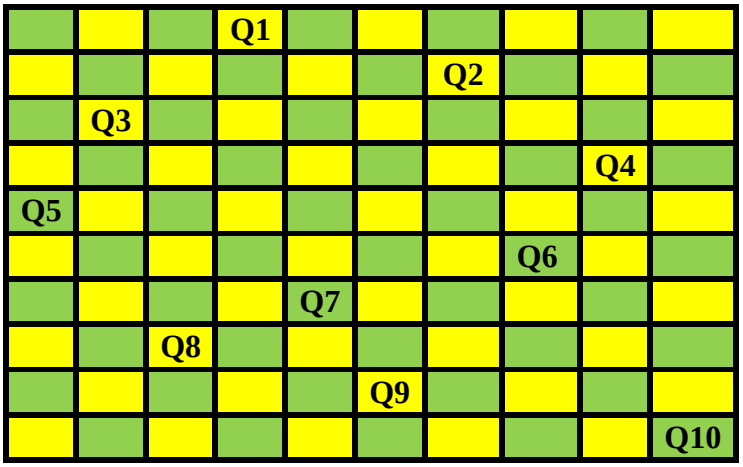

Figure 4.7: Placement of Queens of Mutated Chromosome on a 10*10 Chessboard

From the Figure 4.7 it is clear that the mutated chromosome is having only 4 clashes because now Q6 is not clashing with any other queen. The four clashes are because of clashing Q1 with Q3 and Q2 with Q4. Algorithm 4.5 is showing the steps of greedy mutation. Algorithm 4.6 is showing the steps of greedy genetic algorithm to solve N-Queen problem.

Algorithm 4.5 Algorithm for greedy mutation for N-Queen problem

Input: A chromosome for mutation

Output: Mutated chromosome using greedy mutation

1. *Find position POS1 of a queen which is making clashes with other queens*
2. *Select a column position POS2 randomly*
3. *Shift the queen POS1 at new position POS2*
4. *Return the mutated chromosome.*

Algorithm 4.6 Greedy Genetic Algorithm to solve N-Queen problem

Input: An instance of N-Queen problem

Output: Solution of N-Queen problem

1. *Encode the given N-Queen problem into genetic form*
2. *Generate INITIAL_POPULATION of chromosomes*
3. *Calculate fitness of every chromosome in INITIAL_POPULATION*
4. *Repeat steps 5 to 8*
5. *Perform selection to select parent for cross over*
6. *Perform cross over (one-point cross over) and generate new children*
7. *Perform greedy mutation*
8. *Calculate fitness of every chromosome in the population*
9. *If terminating criteria reached, then go to step 10*
10. *Generate population for next generation and go to step 5*
11. *Return chromosome with best fitness as solution of the N-Queen problem*

4.4 Proposed Greedy Approach to Solve Knapsack Problem

Knapsack problem is an optimization problem. The proposed algorithm is a greedy genetic algorithm to solve 0/1 Multiple Knapsack Problem. The algorithm is a greedy algorithm because it generates greedy initial population. In greedy initial population the algorithm tries to fill those items first which have high value of profit/weight ratio.

Filling items with high value of profit/weight ration generates chromosome with high fitness value. Algorithm 4.7 is illustrating the steps to generate greedy chromosomes.

Algorithm 4.7 Generation of greedy chromosome of 0/1 MKP problem

Input: An instance of 0/1 MKP problem

Output: Greedy chromosome for input 0/1 MKP problem

1. *Arrange all items of given instance in decreasing order on profit/weight ratio*
2. *Select an item from the LIST_OF_ITEMS which is not yet selected*
3. *If (all knapsacks are full) then return solution otherwise select a knapsack randomly in which item can be filled and add the item in selected knapsack*
4. *If all items are selected or All knapsacks are full then return chromosome otherwise go to step 2.*

Algorithm 4.8 Greedy Genetic Algorithm to solve MKP problem

Input: An instance of MKP problem

Output: Solution of MKP problem

1. *Encode the given MKP problem into genetic form*
2. *Generate INITIAL_POPULATION of chromosomes using greedy approach by Algorithm -4.7*
3. *Calculate fitness of every chromosome in INITIAL_POPULATION*
4. *Repeat steps 5 to 8 until termination criteria is not reached*
5. *Perform selection to select parent for cross over*
6. *Perform cross over (one-point cross over) and generate new children*
7. *Perform mutation*
8. *Calculate fitness of every chromosome in the population and generate population for next generation*

Algorithm 4.8 illustrates the steps of Greedy Genetic Algorithm to solve MKP problem. Figure 4.8 is showing a flowchart of the proposed greedy genetic algorithm to solve 0/1 MKP problem.

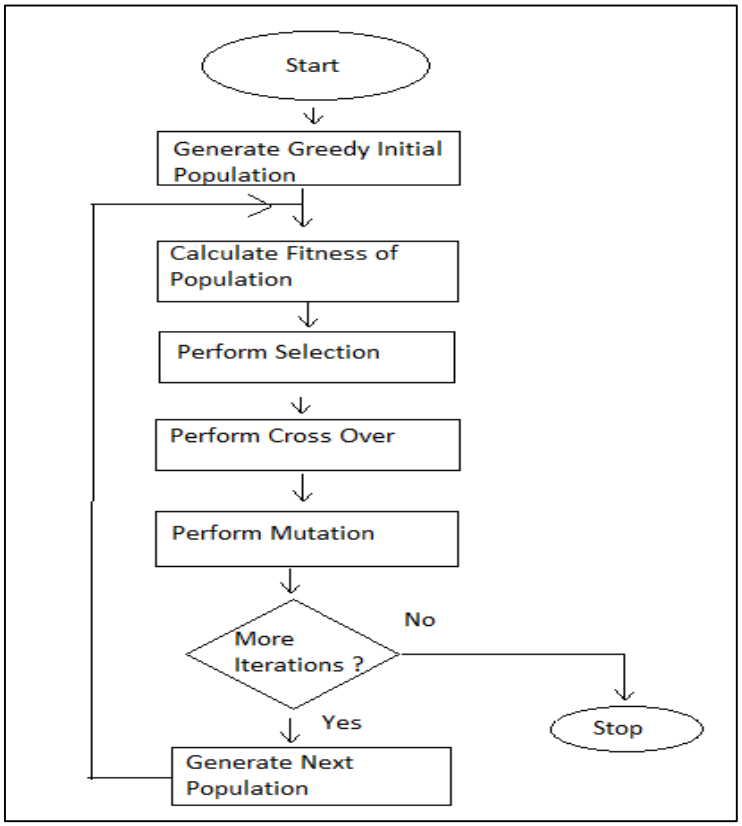

Figure 4.8 Flowchart of Proposed Greedy Genetic Algorithm to Solve MKP Problem

In Figure 4.8 the very first step is generating greedy initial population of chromosomes. The greedy chromosomes are generated using Algorithm 4.7. Then the fitness of every chromosome is calculated. Then selection operator selects some chromosomes to perform cross over. Then cross over operation is performed and it generates new children. Then mutation operation is applied. If terminating criteria is reached, then best solution in the current population is returned. Otherwise next population of chromosomes is generated and steps of genetic algorithm (fitness calculation, selection, cross over and mutation) are repeated.

Appendix-1 is illustrating the steps of all the algorithms explained in this chapter in detail.

In this work, the Greedy Genetic Algorithm is applied over Travelling Salesman Problem, N-Queen Problem and 0/1 Multiple Knapsack Problem. To analyze the performance and capabilities of proposed Greedy Genetic Algorithm, it is implemented to solve TSP, MKP and N-Queen problems. Chapter 5 discusses the results found from these implementations and analysis of these results.