CHAPTER-1

INTRODUCTION TO NP-COMPLETE AND NP-HARD PROBLEMS

"Premature optimization is the root of all evil" – Donald Knuth

To find the best solution of a problem among all feasible solutions is optimization. All computation problems cannot be solved by algorithms, irrespective of time.

1.1 Time Complexity of Algorithms

The complexity of algorithms is a measure of the number of computations (mainly comparisons) and time spent to solve a problem [1]. On the basis of the complexity, algorithms are divided into two groups. Group-1 consists of those problems whose solutions are bound by a polynomial of small degree. This group contains the problems like sorting of *n* numbers, searching an item in a list etc. Group-2 consists of problems whose best known solution/algorithm is non-polynomial. This group contains the problems like Travelling Salesman Problem, Graph Coloring Problem, N-Queen Problem, Multiple Knapsack Problem and many more.

Further, on the basis of complexity, algorithms are divided into classes namely Polynomial Time P, Nondeterministic Polynomial Time NP, NP-Hard and NP-Complete [1].

P: It is the class of problems, which can be solved by a deterministic polynomial time algorithm.

NP: It is the class of problems, which can be solved by a non-deterministic polynomial time algorithm.

NP-Hard: It is the class of problems to which every NP-Complete problem reduces. These are the problems which are at least as hard as any problem in NP.

NP-Complete: It is the class of problems which are NP-Hard and belong to NP. All NP-Complete problems are NP-Hard but all NP-Hard problems are not NP-Complete.

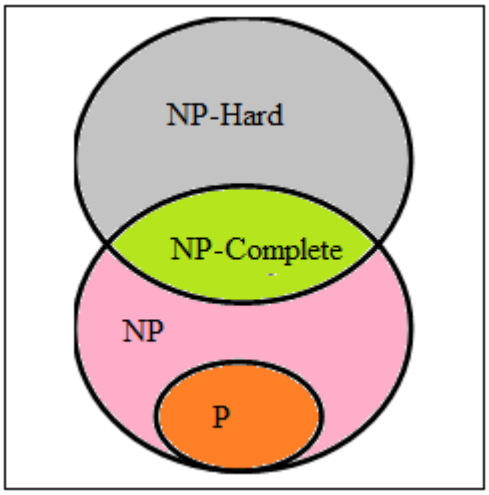

Figure 1.1: Relationship between P, NP, NP-Hard and NP-Complete

Many mathematicians and computer science researchers work on finding the solution of NP-Hard and NP-Complete problems. There are mainly two assumptions, P is equal to NP and P is not equal to NP. Figure 1.1 is showing the relationship between P, NP, NPH and NPC problems [1] assuming that P is not equal to NP.

Cook–Levin theorem – Stephen Cook and Leonid Levin gave a theorem related to complexities of the algorithms. The theorem is also known as Cook's theorem which states

"The Boolean satisfiability problem is NP-complete. That is, any problem in NP can be reduced in polynomial time by a deterministic Turing machine to the problem of determining whether a Boolean formula is satisfiable."

The Cook theorem is based on Boolean satisfiability problem. If there exists a deterministic polynomial time algorithm for solving Boolean satisfiability, then all the

problem in the class NP problem can be solved by a deterministic polynomial time algorithm. But the question is that whether or not such an algorithm exist. The most important unsolved problem named as P versus NP problem, which specifies, if the solution can be quickly and easily verified then also be solved quickly and easily.

1.2 Some NP-Hard and NP-Complete Problems

NP-Hard are the problems which are at least as hard as any problem in NP. To prove that a given problem is NP-Complete, we have to first prove that it belongs to class NP [1] then it is NP-Complete if it is further reducible to any other NP-Complete problem. NP-Complete and NP-Hard sets contain a large number of problems. Some well-known NP-Complete and NP-Hard problems are Boolean Satisfiability Problem (SAT) [116], Hamiltonian Path Problem [115], Subgraph Isomorphism Problem, Vertex Cover Problem [116], Dominating Set Problem, Clique Problem [116], Knapsack Problem [71,72], Travelling Salesman Problem [31,32], Closest String Problem, Graph Bandwidth Problem , K-Minimum Spanning Tree Problem, Job Scheduling Problem, N-Queen Problem [88,89] etc. Travelling Salesman Problem (TSP), N-Queen Problem and Multiple Knapsack problem (MKP) are three NP-Hard (NPH) and NP-Complete (NPC) problems which are solved in this work. The detail of these three problems is given below.

1.3 Travelling Salesman Problem

In Travelling Salesman Problem [2], a salesman has to visit N number of cities with the imposed constraints that salesman have to visit all the cities and every city is visited only once. Always, the salesman starts and stops the journey from same city. The objective is to find a solution such that the distance travelled by the salesman should be minimal.

The TSP problem can be simulated by a weighted graph $G = (V; E)$ where every city is represented by a vertex of the graph and edges of the graph represent the route from one city to another. The weight on the edge represents the distance between the two

cities. The requirement is to find a route in the graph which visits all the vertices exactly once. This route is called the Hamiltonian Cycle. The problem was raised by Hamilton in 1857.

TSP is NP-Hard - In TSP, we have to find all the possible routes that visited every city at least once and then we find the cycle that cost minimum i.e. route with minimum distance. So the TSP problem belongs to the NP-Hard problems and thus TSP is also NP-Hard problem.

TSP is NP-Complete - All the decision problems answers [1] the solution in the form of Yes or No. In TSP, a version is called decision problem if salesman has given a list of cities with distances to each other and a total distance D, then it is asked that whether salesman can visit all the cities by travelling a distance which is less than or equal to D. Such type of problem is called as a decision problem because its answer is either Yes or No. This version of TSP problem is NP-Complete.

Table 1.1 Relationship between Number of Vertices and Hamiltonian Cycles

Number of Vertices	**Number of Hamiltonian Cycles**
3	1
4	3
5	12
6	60
7	360
8	2520
9	20160
10	181440
20	$6.08 * 10^{16}$
30	$4.42 * 10^{30}$
40	$1.02 * 10^{46}$

As the value of the number of vertices in a graph *G* increases the number of Hamiltonian Cycles possible in the graph also increases. If we have n number of vertices in a graph, then Table 1.1 shows how many Hamilton Cycles are possible in that graph [115]. We can see that, as the n increases the number of cycles also increases. For a given value of n, there exist $\frac{(n-1)!}{2}$ numbers of Hamilton Cycles in the graph. The value of $\frac{(n-1)!}{2}$ calculated through given values of *n* are shown in Table 1.1. The TSP problem can be solved in many ways by finding the exact solution of the problem and also by finding the approximate solutions of the problem.

Techniques for finding exact solution of TSP problem are categorized as

1. Brute Force Method
2. Branch and Bound Method

and techniques for finding an approximate solution of TSP problem are categorized as

1. Nearest Neighbor Approach
2. Greedy Approach

1.3.1 Brute force method

This method is the simplest solution of the TSP problem to understand. But it is the most exhaustive solution of the problem because brute force technique performs many computations. In brute force method we first find all the possible tours with their path length. Then we select the path that visits all the vertices in minimum path length. The steps taken by brute force method are as follows:

Step 1. Find total number of tours possible
Step 2. Find a list of all the possible tours
Step 3. Find the distance traveled in all the tours in the list
Step 4. Select the tour with minimum path length is the solution.

1.3.2 Branch and bound method

In branch and bound method, the problem is divided into a number of sub problems. The sub problems may have multiple solutions and the solution selected may affect the overall result. The solutions of these sub problems are combined to find the overall solution. The steps taken in branch and bound method are as follows:

Step 1. Select a starting node.

Step 2. Set the value of bound to a large value to ∞.

Step 3. Find and select the cheapest arc between the current node and unvisited node and the distance between the current node and unvisited node. Repeat it while the current distance is less than the bound.

Step 4. Add the distance and the bound to find the new current distance.

Step 5. Repeat step 4 until all the arcs are not covered.

1.3.3 Nearest neighbor approach

In nearest neighbor approach, first select a starting node. Then find a node which is at the minimum distance from the current node and select this node as a new next node. Further, find a node which is not yet visited and also at a minimum distance from the new current node. Repeat this procedure until all the cities are not visited. At the last, come back to the starting node. The complexity of nearest neighbor method is $O(n^2)$. The working of nearest neighbor approach is as follows:

Step 1. Select a city as the starting city, set it as current city.

Step 2. Find the nearest city (from the unvisited cities) to current city, and set it as current city.

Step 3. If any unvisited city left, then go to step-2.

Step 4. Return to the starting city.

1.3.4 Greedy algorithm

Greedy algorithm starts the tour from a starting node. Further, calculates the distance from that node to all other remaining $n - 1$ nodes and selects the next node which is closest to the current node. Repeats the process until all the nodes are not visited at least once. After visiting all the nodes, the algorithm selects the starting node as the last node. The complexity of greedy approach is $O(n^2\ log_2\ (n))$.

1.4 N-Queen Problem

N-Queen problem in mathematics as well as in computer science cannot be solved using traditional algorithms. In N-Queen problem, *N* number of queens has to be placed on a chess board of *N* rows and *N* columns. The queens must be placed such that no two queens should attack each other. So, in every row and every column the challenge is to place only one queen. In N-Queen problem certain constraints needs to be satisfied, hence, N-Queen problem is also known as constraint satisfaction problem. The problem consists of variables, values assigned to these variables and some constraints, required to be satisfied. Figure 1.2, shows a sample solution of 8 Queen problem. Q1, Q2, Q3 . . . . Q8 are queens placed on the chess board.

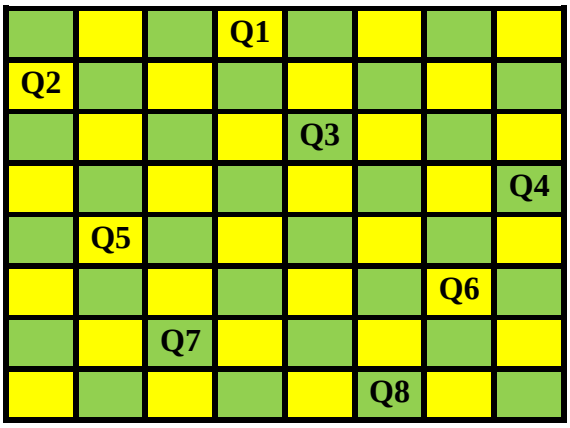

Figure 1.2: A Sample Solution of 8-Queen Problem

The N Queen problem has many applications in Science and Engineering, and can be used to solve problems in real time computer systems, error correction and detection,

designing of communication systems, designing of VLSI circuits, resource management in computer systems, testing of VLSI circuits, scheduling of tasks in operating system, solving routing problems in computer networks, balancing of load on different microprocessors in computers, parallel processing in optics, controlling traffic, data and image compression, parallel storage of memory, prevention of deadlocks in operating system, assignment of tasks and many more. In recent years, several researchers solved the N-Queen problem using different techniques.

Table 1.2 Number of Available Solutions for Queens with N= 4 To 25

Number of Queens	**Possible Solutions**
4	2
5	10
6	4
7	40
8	92
9	352
10	724
11	2,680
12	14,200
13	73,712
14	365,596
15	2,279,184
16	14,722,512
17	95,815,104
18	666,090,624
19	4,968,057,848
20	39,029,188,884
21	314,666,222,712
22	2,691,008,701,644

23	24,233,937,684,440
24	227,514,171,973,736
25	2,207,893,435,808,350

This problem is well known as “EIGHT QUEENS PUZZLE” with a lot of hits over internet. Important to notice that the 92 solutions are a subset of all 8! i.e. 40320.

Table 1.2 is showing number of solutions possible for N-Queen problem for number of queens (N) ranges from 4 to 25 [117].

Table 1.2 shows that as the number of queens increases, the number of possible ways to place these queens increases exponentially. It is still hard for people to find solution for larger values of number of queens (N).

1.5 Knapsack Problem

The knapsack problem is a well-known NP-Hard problem. The knapsack problem is an optimization problem in which the overall profit is to be optimized and should be maximized while satisfying all the given constraints [15]. The Knapsack problem is of many types such as 0/1 knapsack problem, multidimensional knapsack problem, multiple knapsack problem etc.

The *0/1* multiple knapsack problem is a generalization of *0/1* knapsack problem.

In *0/1* multiple knapsack problem (MKP) a set of n items is given, and set of m knapsacks is given. Each item j has a profit P_j and weight W_j. Each knapsack i has a capacity C_i. The objective is to assign items in knapsacks such that one item may be placed at most in one knapsack. Assign the items into knapsacks such that none of the capacity constraints of any of the knapsacks is violated. The total profit of assigning items to knapsacks should also be maximized.

Mathematically the multiple knapsack problem can be formulated as shown below. Here $i = 1$ to *m* is the knapsack number, *j=1* to *n* is the number of items. Max is the value of the profit to be maximized. P_j and W_j are the profit and weight of the j^{th} item.

$$Max = \sum_{i=1}^{m} \sum_{j=1}^{n} P_j W_j \quad (1)$$

such that

$$\sum_{j=0}^{n} W_i * X_{ij} \leq C_i \quad where \ (i = 1,2 \ldots \ldots . . m)$$

$$\sum_{j=0}^{m} X_{ij} \quad where \ (i = 1,2 \ldots \ldots . . m \text{ and } j = 1,2 \ \ldots \ldots m)$$

$Where\ X_{ij}$ is either 0 or 1 for (*i*=1,2,3 …… *m*, *j*=1,2,3 …….. *n*), X_{ij} = 1 if item *j* is assigned to knapsack *i* otherwise X_{ij} = 0.

1.5.1 Types of knapsack problems

The Knapsack problem has many variations or types. Different types of knapsack problems have different applications in Science and Engineering. These types differ in the value of parameters in knapsack problem. The parameter includes number of items, number of knapsacks, number of constraints, objectives etc.

The types of knapsack problems are:

1. Multi-objective knapsack problem
2. Multi-dimensional knapsack problem
3. Multiple knapsack problem

Multi-objective knapsack problem

To get maximum profit is the main objective of the knapsack problem. This variant may have some other objectives also. These objectives may include social goals, environmental goals, other economic goals, transportation of items etc.

Multi-dimensional knapsack problem

In this variant the weight of the knapsack items are defined by a D-Dimensional vector, every knapsack has a D-Dimensional capacity vector. The main objective is to maximize the profit from all the items from the knapsack such that the sum of weights of knapsack in each dimension does not exceed. It is more difficult problem as compared to standard knapsack problem.

Multiple knapsack problem

Multiple Knapsack Problem (MKP) is a variant of knapsack problem in which there are two or more knapsacks available to fill the items. Each of the knapsacks is having its own capacity. Items are to be filled in the knapsack such that the total profit earned from the items is maximum and items are filled in the knapsacks within the capacity of the knapsacks. This variant is more complex as compared to standard knapsack problem. It has many application areas such as transportation of goods in trucks and in loading and scheduling goods in operation research problems and many more.

The above discussion gives an idea of several problems needs to be solved and Traditional methods are not proper to be utilized for solving such problems.

1.6 Gaps in Research

For solving NPH and NPC problems following gaps in research are found:

(1) There is no efficient algorithm available to solve the NPC and NPH problems. The related literature shows that not much effort has been implied to solve all kinds of NP-problems. However, these problems are usually solved with the help of approximation or optimization techniques such as GA, ACO, PSO [5,6,7,46] etc.

(2) The performance of GA depends on various factors and convergence affects when the problem size increases [2,3]. Hence, the requirement is to enhance the performance

of GA so that the GA converges in reasonable time and can handle the large size NPH and NPC problems.

(3) Although hybridization is used for improving the performance of GA [2,3,4] still it is observed that exhaustive research work has not been done to increase the performance of GA in the case of NPH and NPC problems.

Thus the present research work is intended to fill the above mentioned research gaps and to critically examine the hybridization of GA by applying the greedy approach.

1.7 Research Objectives

The objectives of this research are to enhance the performance of genetic algorithm for solving NP-Hard and NP-Complete problems. The detailed objectives of this research are as follows:

1. To enhance the genetic algorithm for solving NP-Hard and NP-Complete problems by applying hybrid genetic operators such as mutation and cross over.
2. To establish the effectiveness of greedy approach in hybridization of various genetic operators.
3. To analyze the performance of hybridized genetic operators in solving various NP-Complete and NP-Hard problems such as TSP, Knapsack, N-Queen etc.

1.8 Problem Formulation

NP-Hard (NPH) and NP-Complete (NPC) are well known problems in computer science, which cannot be solved using traditional algorithms. These problems can be solved using special techniques like Ant Colony Optimization (ACO) [7,19,22], Particle Swarm Optimization (PSO) [6,10,47] and Genetic Algorithms (GA) [24,25,27,28]. While reviewing the literature, it has been observed that GA can solve NPH and NPC problems better than the other techniques like ACO and PSO [6,7]. Further, the performance of the genetic algorithm to solve any problem depends upon the performance of its genetic operations such as selection, mutation and reproduction.

In the present research the focus is to solve the identified problems stated in gaps in research and following are the points of action.

1. GA has been applied to solve different kinds of NPC and NPH problems.
2. In order to improve the optimization performance of GA, existing operators such as cross over and mutation are efficiently hybridized. For hybridization, greedy approach is applied over cross over operator and mutation operator.
3. The proposed hybridized GA operators are applied to solve numerous instances of TSP, N-Queen and Knapsack problems which belong to NPC and NPH classes.
4. The performance of proposed hybridized GA operators is analyzed. The results of existing contemporary techniques to solve NPH and NPC problems are compared with proposed hybridized GA.

1.9 Organization of Thesis

This thesis is organized into six chapters consisting of the details of present research work. Chapter 1 introduced the NP-Complete and NP-Hard problems and explored the concept of time complexity of algorithm, different classes of problems such as P, NP, NPH and NPC. Travelling Salesman Problem, N-Queen problem and Knapsack problem were illustrated. The chapter discussed the gaps in research, research objectives, and problem formulation.

Chapter 2 is about the detail literature survey of the research work carried out in this direction. Chapter 2 focuses on the main research carried out by different researchers to solve NP-Hard and NP-Complete problems. It mainly discusses how useful is to solve NP-Hard and NP-Complete problems using soft computing techniques. .

Chapter 3 is about the details of Genetic Algorithm and all its operators. It explains the basic steps of the genetic algorithm and all of its operations such as initialization, selection, reproduction, mutation and termination in detail.

In chapter 4, the greedy approach is proposed to solve NPH and NPC problems using genetic algorithm. The details of Greedy Genetic Algorithm (GGA) to solve Travelling Salesman Problem, N-Queen problem and Knapsack problem are presented.

In chapter 5, the implementation of proposed GGA algorithm is presented. Further, the result of these implementations to solve Travelling Salesman Problem, N-Queen problem and Knapsack problem using proposed greedy approach are analyzed.

Chapter 6 is concluding about usefulness of presented research in solving Travelling Salesman Problem, N-Queen problem and Knapsack problem using Greedy Genetic Algorithm. This chapter also gives an insight into the future scope on which further research can be done. At last references used in this thesis are written.

The next chapter discusses the previous work done so far in this area by many researchers, and critically analyzing the previous related work.