CHAPTER-5

ANALYSIS OF RESULTS

This chapter presents the results of implementation of Greedy Genetic Algorithm GGA, illustrated in chapter 4, while solving multiple instances of TSP, N-Queen and Multiple Knapsack problems.

5.1 Analysis of TSP Solution

The proposed algorithm presented in chapter 4, to solve Travelling Salesman Problem is implemented using NetBeans8.0.2 IDE and JAVA (JDK1.7). Some standard instances of TSP available in TSPLIB [87] are solved using proposed greedy genetic algorithm. Table 5.1 is showing the standard TSP instances on which the proposed algorithm is implemented.

Table 5.1 Standard TSP Instances Solved Using GGA Algorithm

Sr. No	TSP Problems	Number of Nodes (n)
1	EIL51	51
2	EIL 76	76
3	A280	280
4	ATT48	48

Table 5.2 shows the path length of the best path of the TSP instances. While solving *EIL51* TSP instance, SWAP_GA algorithm finds the solution with path length 439, OX_SIM finds a solution with path length 493, and MOC_SIM finds the solution with path length 444 and Gohar Vahdati et.al [3] finds the solution with path length 429. The proposed greedy genetic algorithm finds the solution with path length 415. The result of the proposed algorithm is 6% better than the SWAP_GA algorithm, 19% better than the OX_SIM, 7% better than the MOC_SIM and 3% better than algorithms proposed in [3]. For solving *Eil76* problem the proposed GGA algorithm performs

better than the other algorithms. TSP instances of *A280* and *Att48* were not calculated by [3] whereas we have considered these TSP instances and obtained the path length as shown in Table 5.2.

Table 5.2 Comparison of Results with SWAP_GATSP, OX_SIM, MOC_SIM and Proposed GGA Algorithm

Standard TSP Problem	**SWAP_ GATSP**	**OX_SIM**	**MOC_SIM**	**Gohar Vahdati et al. [3]**	**ECACR-M [123]**	**Proposed GGA Algorithm**
Eil51	439	493	444	429	*	415
Eil76	548	597	562	549	*	541
A280	*	*	*	*	*	2773
Att48	*	*	*	*	33453	32139
*Note - * indicates that as reported in research work of [3]and [123], A280 and Att48 problems not considered hence best path length not exists.*						

Table 5.3: Comparison of Results of Proposed GGA Algorithm with PSO, ACO and HPSACO Algorithms

TSP Problem	**PSO**		**ACO**		**HPSACO**		**Proposed GGA Algorithm**	
	Best	**Average**	**Best**	**Average**	**Best**	**Average**	**Best**	**Average**
Att48	33734	33982	33649	33731	33524	33667	32139	32403

Table 5.3 is showing the results for solving *Att48* TSP instance using Particle Swarm Optimization PSO algorithm, Ant Colony Optimization ACO algorithm and Hybrid Particle Swarm and Ant Colony Optimization HPSACO algorithm. Table 5.3 shows two values of the *Att48* TSP instance for every algorithm, one is the best (minimum) path length of the chromosome in the population and second is the average path length of all the chromosomes in the population (after executing 2000 iterations). PSO solve

it with the best path length of 33734, ACO solves it with a path length of 33649 and HPSACO solve it with a path length of 33524 [8].

The proposed greedy genetic algorithm solves *Att48* TSP instance with a path length of 32139. The solution of the proposed algorithm is 4.96% better than the PSO Algorithm, 4.70% better than the ACO Algorithm, and 4.31% better than the HPSACO Algorithm.

Figure 5.1 is showing a snapshot of solving *Att48* TSP instance using proposed GGA algorithm. Figure 5.1 depicts that a solution with distance 32139 is found after executing 2001 iterations of GGA algorithm (Algorithm 4.4). In Figure 5.1, the final solution of *Att48* problem (with path length 32139) is circled by a thick circle.

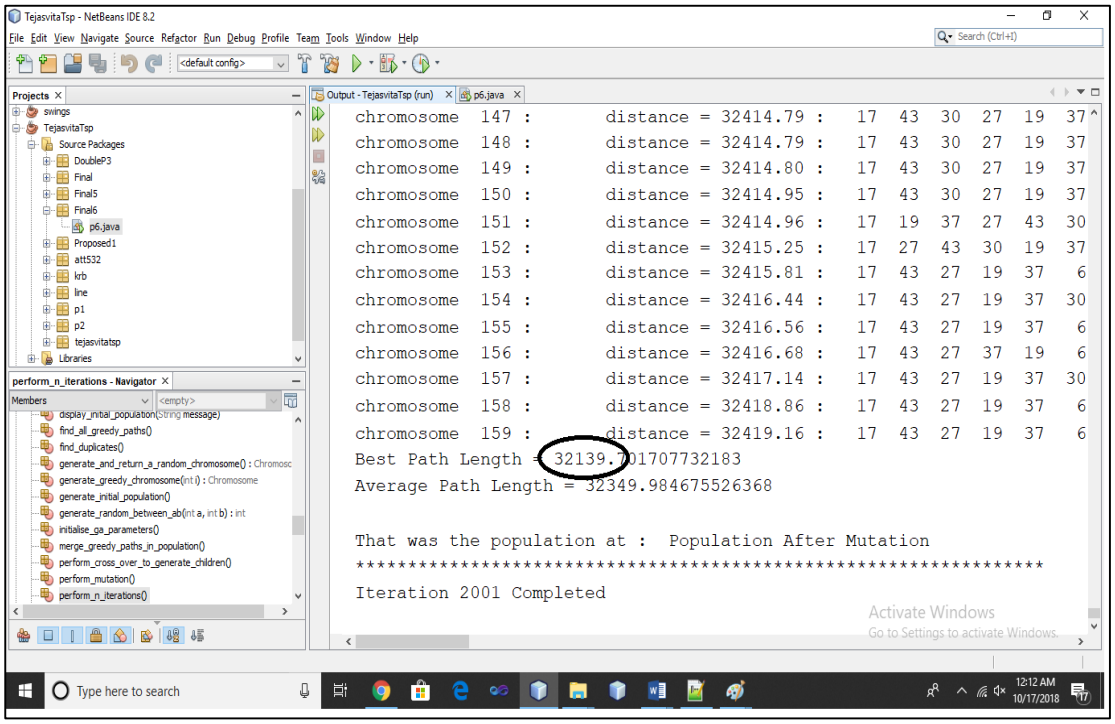

Figure 5.1: Snapshot of Solving TSP Problem

Table 5.4: Order of Cities in The Best Path Found for TSP Problems *ATT48*, *EIL51* and *EIL76*

Best Known Path using GGA algorithm		
ATT48	***EIL51***	***EIL76***
17, 43, 27, 19 ,37 6, 30, 28, 36, 7, 18, 44, 31, 38, 9, 8, 1, 16, 22, 3, 23, 11, 12, 40, 15, 46, 33, 20, 47, 21, 13, 25, 14, 34, 41, 29, 5, 48, 39, 32, 24, 10, 42, 2, 26, 4, 35, 45, -1 EOF	40, 19, 41, 13, 25, 14 , 6 , 48, 27 , 51, 46, 12 , 47 , 18 , 4 , 17, 37 , 42 , 44 , 15 , 45 , 33 , 39 , 10, 30 , 34 , 21 , 29, 2 , 16, 50 , 9 , 49, 5 , 38, 11 , 32 , 1 , 22 , 20 , 35, 36 , 3 , 28, 31 , 8 , 26, 7 , 23, 24 , 43, -1 EOF	21, 47, 48, 29, 45, 27, 52, 34, 46, 8, 35, 7, 53, 14, 19, 54, 13, 57, 15, 5, 37, 20, 70, 60, 71, 36, 69, 61, 62, 73, 1, 33, 63 , 16 , 51, 6 , 68, 75 , 76 , 67 , 4 , 30 , 2 , 74, 28, 22, 64, 42, 43, 41, 56, 23, 49, 24, 18, 50, 32, 44, 3, 40, 17, 26, 12, 58, 72, 39, 9, 25, 55, 31, 10, 38, 65, 66, 11, 59, -1 EOF

Table 5.4 is showing the order of cities in the best solution found for the TSP instances *EIL51*, *ATT48* and *EIL76* problems. The order of cities in which cities are visited by salesman is shown for TSP instances *Att48*, *EIL51* and *EIL76* are shown in Table 5.4.

Figure 5.2 is showing graphical representation of solution of *Att48* problem. Figure 5.3 is showing graphical representation of solution of *EIL51* problem. Figure 5.4 is showing graphical representation of solution of *EIL76* problem. The position of cities for TSP instances are available in TSPLIB data set [87].

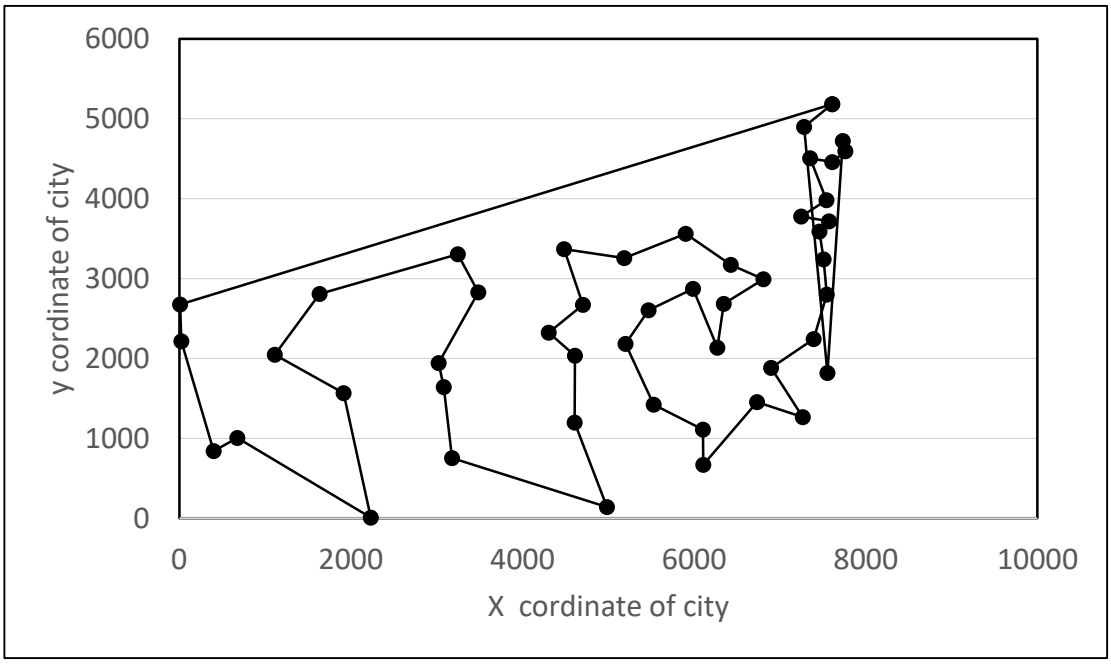

Figure 5.2: Solution of *Att48* TSP Instance Using GGA Algorithm

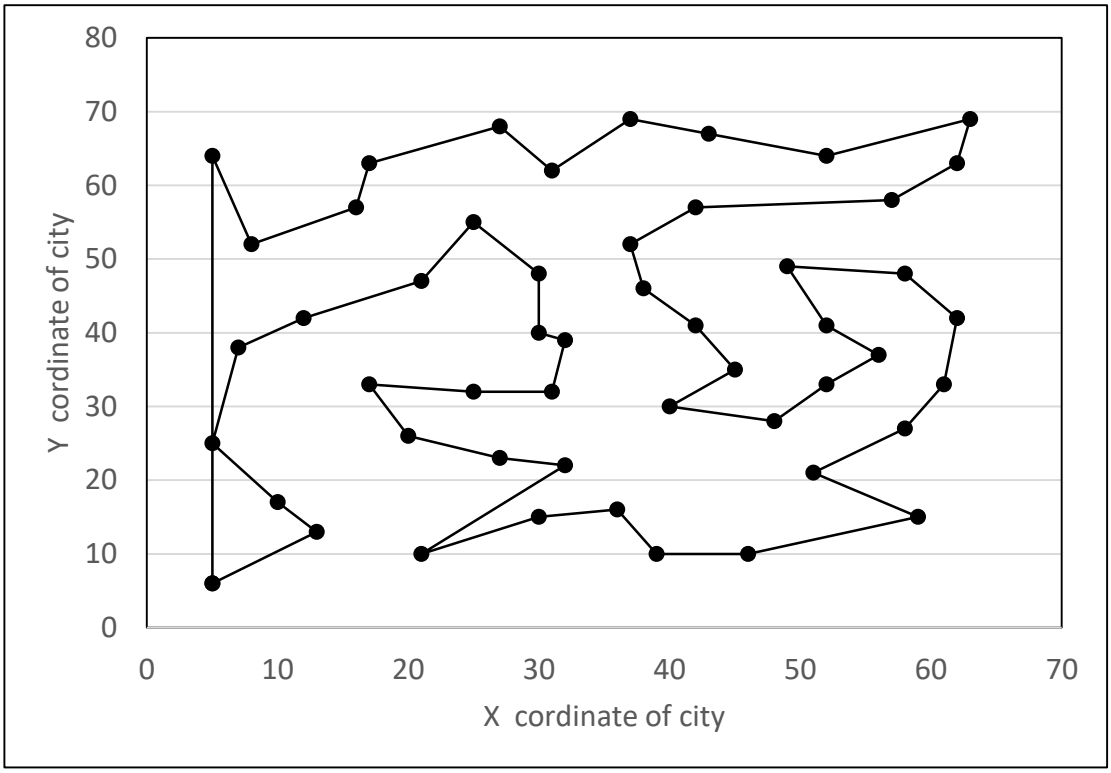

Figure 5.3: Solution of *Eil51* TSP Instance Using GGA Algorithm

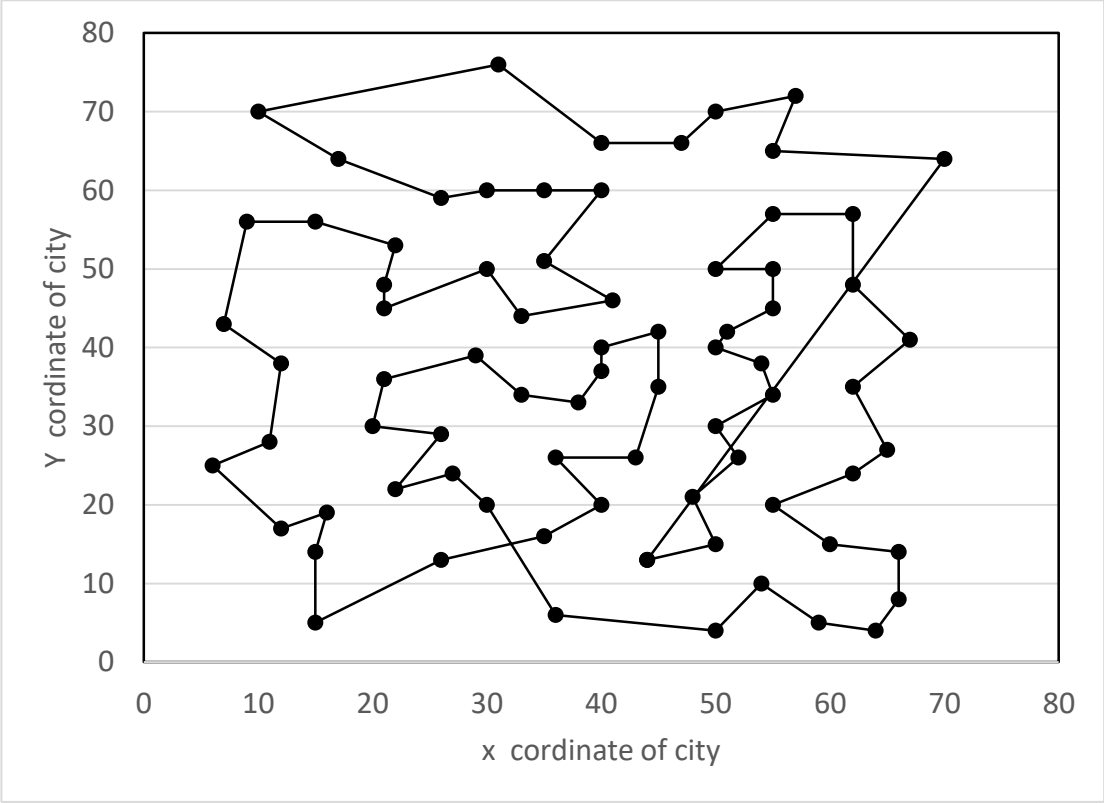

Figure 5.4: Solution of *Eil76* TSP Instance Using GGA Algorithm

5.2 Analysis of N-Queen Solution

The proposed Greedy Genetic Algorithm GGA for solving N-Queen problem is implemented in JAVA (using JDK1.7). Results are calculated in terms of execution time (in seconds). The algorithm 4.6 is applied on N-Queen problem having 8, 16, 30, 40, 50 and 100 queens.

Table 5.5 is illustrating the performance of different algorithms discussed in [10], on the basis of execution time taken by the algorithm to find the solution. For 30 Queens problem SRPSO is taking 6.59 seconds, Per-PSO is taking 10.32 seconds, Old-GA is taking 17.29 seconds and proposed GGA algorithm is taking 0.3420 seconds. For 40

Queen problem SRPSO is taking 23.73 seconds, Per-PSO is taking 34.30 seconds, Old-GA is taking 35.66 seconds and proposed GGA algorithm is taking 0.7885 seconds. For 50 Queen problem SRPSO is taking 40.12 seconds, Per-PSO is taking 53.25 seconds, Old-GA is taking 54.43 seconds and proposed GGA algorithm is taking 1.0443 seconds. Proposed GGA algorithm is taking 0.0307 seconds to solve 8 Queen and 14.3688 seconds for 100 Queen problem. It can be analyzed that the performance of proposed GGA algorithm is better than the all the other three algorithms because the GGA algorithm find the solution in very less time as compared to the other three algorithms. The reason behind this performance is the greedy approach which enhances the convergence speed of GGA algorithm.

Figure 5.5 is showing a snapshot of solving N-Queen problem using proposed GGA algorithm. Snapshot display the results of solving 8-Queen, 16-Queen, 30-Queen, 40-Queen, 50-Queen, and 100 Queen problems. Figure 5.5 displays the solution with time taken in execution to find solution. In Figure 5.5 the solution of 8-Queen problem and its execution time is circled.

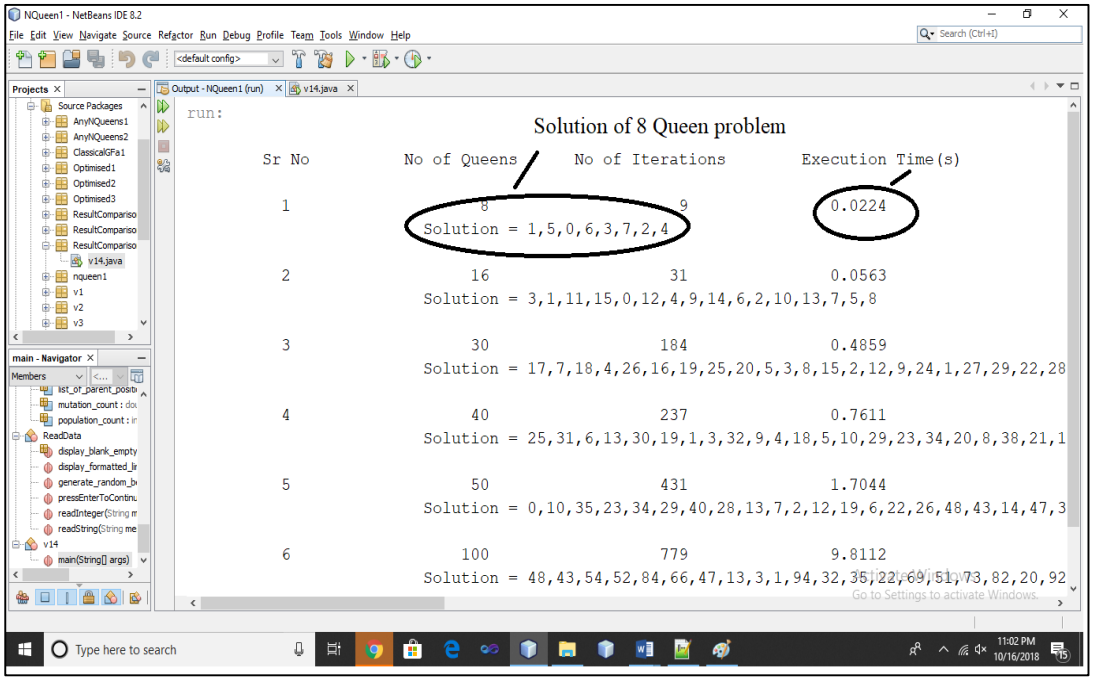

Figure 5.5: Snapshot of Solving N-Queen Problem

Figure 5.6 is showing execution time of different existing algorithm and proposed greedy genetic algorithm to solve N-Queen problem for 30,40 and 50 queens. Graph is a bar chart which is showing the results for 30-Queen, 40-Queen, 50-Queens instances because, for 8-Queen and 100-Queen instances, results have not reported in literature [10]. It is clear from the Figure 5.6 that the proposed algorithm takes very less time as compared to SR-PSO, Per-PSO and Old-GA algorithms.

Table 5.5 Comparison of Results for Solving Six Instances of N-Queen Problem

Algorithm	**Exécution Time in Second**					
	8 *Queens*	**16 *Queens***	**30 *Queens***	**40 *Queens***	**50 *Queens***	**100 *Queens***
SRPSO	*	*	6.59 sec	23.73 sec	40.12 sec	*
Per-PSO	*	*	10.32 sec	34.30 sec	53.25 sec	*
Old-GA	*	*	17.29 sec	35.66 sec	54.43 sec	*
Proposed GGA Algorithm	**0.0307 sec**	**0.0972 sec**	**0.342 sec**	**0.7885 sec**	**1.0443 sec**	**14.3688 sec**
Note - * indicates N queen instances not considered in [10]						

Table 5.6 is showing results of solving 13-Queen and 14-Queen problem using proposed GGA algorithm and the results of algorithm proposed by Lijo V. P. and Jasmin T. Jose [11]. The proposed algorithm is taking only 49.2*msec* to solve 13 Queen problem and 52.7*msec* to solve 14 Queen problem whereas algorithm proposed by Lijo V. P. and Jasmin T. Jose is taking 44.14 seconds for 13 Queen problem and 87.56 seconds for 14 Queens problem.

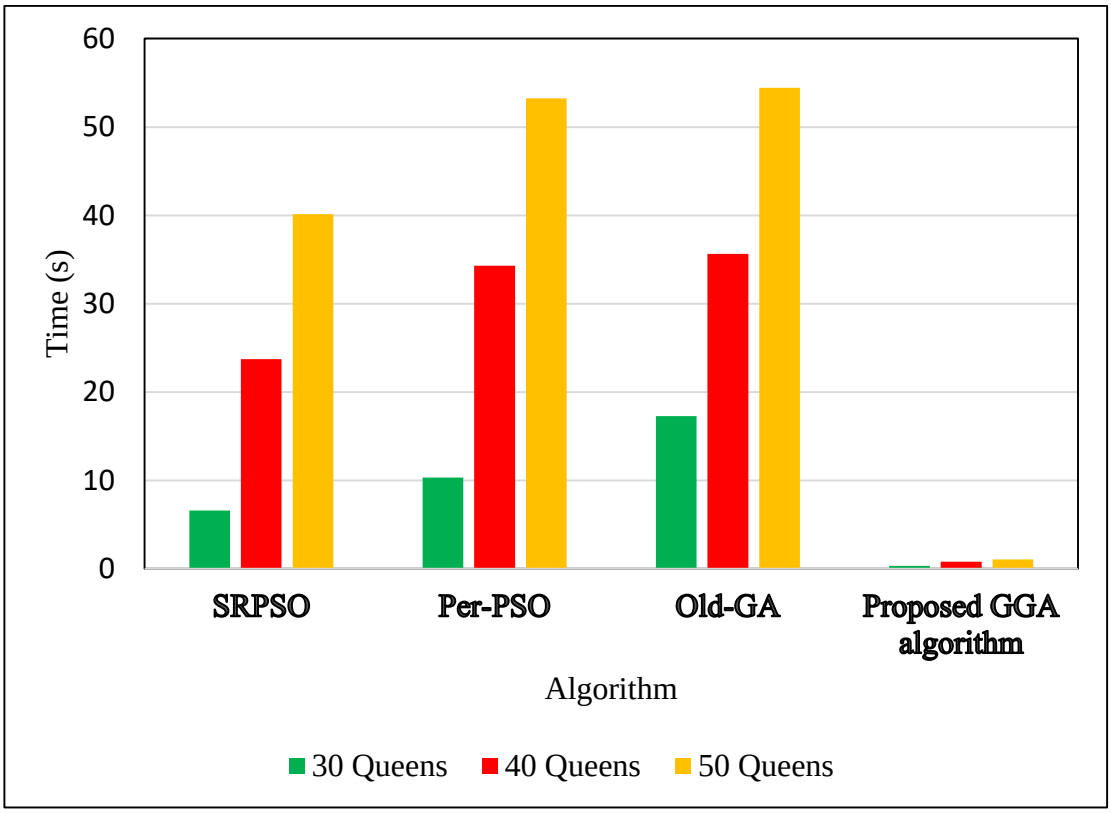

Figure 5.6: Comparison of Results for Solving Three Instances of N-Queen Problem (30 Queen, 40 Queen, and 50 Queen)

Table 5.6: Comparison of Results for Solving 13-Queen and 14-Queen Problem

Algorithm	**13-Queens**		**14-Queens**	
	Execution Time	**Iterations**	**Execution Time**	**Iterations**
N-Queen problem by Prediction approach[11]	44.14 (sec)	84,034,432	87.56 (sec)	543,672,172
Proposed GGA Algorithm	49.2 (ms)	18	52.7 (ms)	34

The proposed algorithm is taking only 18 iterations to solve 13 Queen problem and 34 iterations to solve 14 Queen problem whereas algorithm proposed by Lijo V. P. and Jasmin T. Jose [11] is taking 84,034,432 iterations for 13 Queen problem and 543,672,172 iterations for 14 Queens problem. The reason for such a good performance is the use of greedy approach in GGA algorithm. The greedy approach enhances the convergence of GGA algorithm and the algorithm converges in very less iterations (18 and 34 iteration) and finds the solution in very less time (49.2*msec* and 52.7*msec*) while solving 13 Queen and 14 Queen problems.

5.3 Analysis of Multiple Knapsack Problem Solution

The proposed algorithm is applied to *p01* instance of a data set [9] for multiple knapsack problem. Table 5.7 and Table 5.8 are showing the results of instance p01 using standard genetic algorithm and proposed greedy genetic algorithm. In the data set [9], the best known solution for the *p01* instance is 333 (profit earned in 0/1 MKP problem). The columns in the Table 5.7 shows generation number, best known solution in data set, best solution by standard GA, average solution in that generation and percentage of less profit earn. Percentage of less profit earn is calculated by the following formula:

$$Less\ profit\ earn\ \% = \frac{(Best\ Solution\ Known - Best\ Solution\ By\ GA)}{Best\ Solution\ By\ GA} * 100$$

From the Table 5.7, it is observed that the standard genetic algorithm converges to the solution after iteration number 15 and there is no change in the best solution and average solution after 15^{th} iteration. The percentage of less profit earn is 18.09% using standard genetic algorithm.

Table 5.8 is showing the results of applying GGA for solving *p01* MKP instance. The columns of Table 5.8 are same as of Table 5.7. Here the GGA algorithm find a solution with profit value 309 for *p01* MKP instance in first generation and a profit value 326

in 20th generation. The profit value 326 remains same even after executing 50 iterations.

Table 5.7 *P01* MKP Solution Using Standard Genetic Algorithm

Multiple Knapsack Instance - p01				
Standard Genetic Algorithm				
Generation No	**Best Solution Known**	**Best Solution By GA**	**Average Solution Generation Wise**	**Less Profit Earn %**
1	333	241	192.15	38.17
5	333	241	206.7	38.17
10	333	244	213.35	36.48
15	333	282	230.8	18.09
20	333	282	230.8	18.09
25	333	282	230.8	18.09
30	333	282	230.8	18.09
35	333	282	230.8	18.09
40	333	282	230.8	18.09
45	333	282	230.8	18.09
50	333	282	230.8	18.09

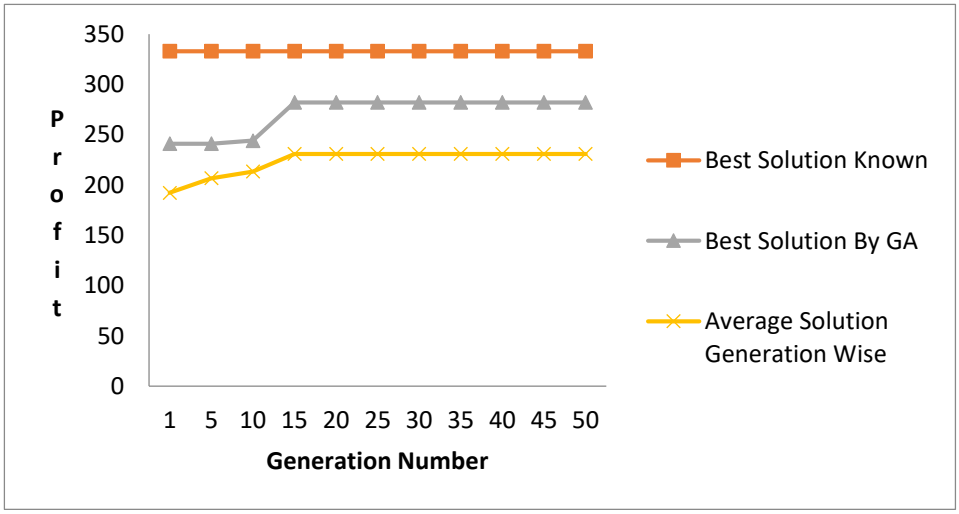

Figure 5.7: Generation Wise Performance of Standard GA for P01 Knapsack

Table 5.8 *P01* MKP Solution Using Greedy Genetic Algorithm (GGA)

Multiple Knapsack Instance - *p01*				
Greedy Genetic Algorithm GGA				
Generation No	**Best Solution Known**	**Best Solution by GGA**	**Average Solution Generation wise**	**Less Profit Earn %**
1	333	285	231.85	16.84
5	333	285	244	16.84
10	333	285	253.7	16.84
15	333	285	263.95	16.84
20	333	326	266.1	2.15
25	333	326	268.85	2.15
30	333	326	268.85	2.15
35	333	326	268.85	2.15
40	333	326	268.85	2.15
45	333	326	268.85	2.15
50	333	326	268.85	2.15

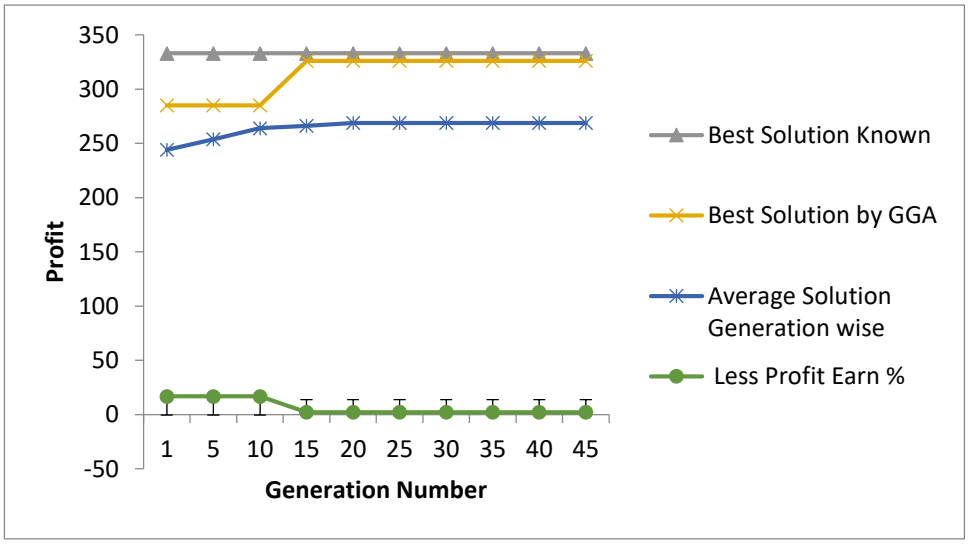

Figure 5.8: Generation Wise Performance of Greedy GA for P01 Knapsack

Table 5.9: Results Comparison of Solving MKP Problem Using Standard Genetic Algorithm and Greedy Genetic Algorithm

MKP Instance	**Optimal Solution by Data Set [9]**	**Best Solution Found Using Standard GA**	**Best Solution Found Using Proposed GGA Algorithm**
p01	333	282	326

Table 5.9 is illustrating a comparison of results found using the standard genetic algorithm and Greedy Genetic Algorithm GGA. The optimal result (available in data set) for *p01* MKP instance is with profit 333. After executing 50 iterations of standard genetic algorithm, a solution with profit value 282 is found. After executing 50 iterations of proposed Greedy Genetic Algorithm a solution with profit value 326 is found. So the proposed Greedy Genetic Algorithm performs 15.94% better (18.09%-2.15%) as compared to standard genetic algorithm.

Figure 5.7 is showing a graph which is analyzing generation wise performance of standard GA for solving *P01* multiple knapsack problem. The graph is showing the performance of standard GA in terms of best solution known, best solution of the whole population and average solution of the population with respect to generation number. Figure 5.8 is showing a graph which is analyzing generation wise performance of Greedy Genetic Algorithm (GGA) for solving *P01* multiple knapsack problem. It is also the graph drawn between best solution known, best solution of the whole population and average solution of the population with respect to generation number. After comparing graphs shown in figure 5.7 and 5.8, it is analyzed that the Greedy Genetic Algorithm is performing better than the standard Genetic Algorithm.

Figure 5.9 is showing a snapshot of results found in solving *p01* instance of MKP problem using greedy genetic algorithm. The snapshot depicts that a solution with profit 285 is found in first generation. A solution of profit 326 is found in 20^{th} generation and it will remain same even after executing 50 iterations and GGA algorithm converges at this solution.

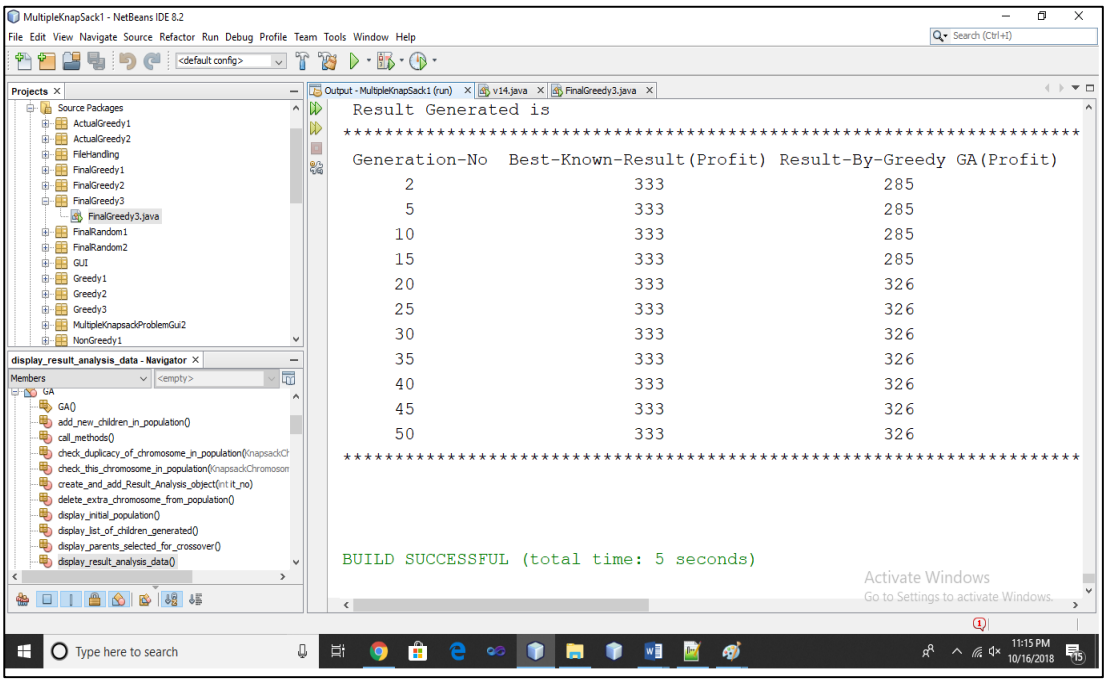

Figure 5.9: Snapshot to Solve MKP Problem Using Proposed GGA Algorithm

In this presented work the Greedy Genetic Algorithm (GGA) is applied on Travelling Salesman Problem, N-Queen Problem and 0/1 Multiple Knapsack Problem. The performance of Greedy Genetic Algorithm is better as compared to existing algorithms to solve these problems. All these three problems belong to the set of NP-Hard/NP-Complete. So the results indicate that the proposed Greedy Genetic Algorithm will also be useful to solve other problems in the set of NP-Hard/NP-Complete such as set cover problem, graph coloring problem etc. The next chapter illustrates the conclusion and future scope of this work.