Multiobjective Traveling Salesman Problem

Swetha Murali Deepak Nair Glen Menezes

Section I : Introduction

Travelling salesman problem (henceforth referred to as TSP) is an extensively studied problem which attracts a considerable amount of research efforts [1]. Intuitively, the TSP is the problem of a salesman who wants to find a shortest possible trip through a given set of customer cities and back to his home town while visiting each city exactly once. The best known deterministic methods of solving this problem take an amount of time exponential in the number of cities. The problem is known to be NP-hard, thus it is well suited for metaheuristics [2]. Like in many real world applications, there exist more than one objective that should be taken into account while evaluating the TSP. For example, some cities may have high priority customers while other cities might be expendable if time is running short. Another example is when the salesman does not want to be on the move constantly and could have some designated rest times at some preferred cities.

There has been a lot of recent research interest in solving multi-objective TSP using stochastic algorithms[3][4]. These works were the inspiration for our project.

The goal of this project is to find the optimal path for a salesman without visiting any location twice under constraints such as time required to traverse the path (denoted by path weight) and maximum time within which a city needs to be visited. The objectives of optimization are to Minimize Total Distance Travelled, Minimize total time taken, and maximize customer satisfaction score (represents the number of high priority customer cities visited in a timely manner). We have implemented and compared four stochastic algorithms - Tabu Search, Genetic Algorithm, NSGA-2, SPEA-2.

The organization of the report is distributed into separate sections each focussing on a main topic with other sub-topics in it. Section 2 gives information about the Algorithms used for solving the problem and other related work. Section 3 lists the assumptions that were made during the implementation of the project. Section 4 details our implementation of the model that was used to solve the problem and get optimized solutions. Section 5 describes working of the project. Section 6 is used to display the results of our implementation of the project. Section 7 details future work that can be extended from our project and its scope and viability in the society. And Section 8 concludes our project.

Section 2: Algorithms Used and Related Work

In this section we provide background information and prior academic work related to our project.

Algorithms Used

Tabu Search

Tabu search is a metaheuristic that guides a local or neighborhood heuristic search procedure to explore the solution space beyond local optimality. Local search methods have a tendency of becoming stuck in regions of local maxima (sub-optimal regions). Tabu search enhances the performance of local search by relaxing its basic rule of looking for better solutions only per step. Instead, at each step worsening moves can be accepted if no improving move is available (like when the search is stuck at a strict local maximum). In addition, tabu search avoids visiting solutions that have already been discovered in recent iterations by maintaining a list of neighborhood generation moves that it considers forbidden. Once a move enters this list it stays there for a fixed number of iterations. Thus, the search space changes continuously. Tabu search terminates when certain terminating conditions, related to maximum number of iterations or execution time or quality of solution are met [5].

Genetic Algorithm (GA)

Genetic Algorithms is part of the family of Evolutionary Algorithms which are metaheuristics based on natural selection of solutions from a population of candidate solutions based on the fitness of the solutions. They are stochastic in nature and uses the functions of mutation, crossover and selection to generate and find the optimal solution. Genetic Algorithms are most commonly used for optimizing properties of the solution or for searching the best solution. There are multiple types of Genetic Algorithms, the types we have implemented are - Regular genetic Algorithm, Non-dominated Sorting Genetic Algorithm - II and Strength Pareto Evolutionary Algorithm [9].

Non-dominated Sorting Genetic Algorithm (NSGA2)

Non-dominated Sorting Genetic Algorithm, like Genetic Algorithm is a stochastic, multi objective optimization algorithm which also is part of the Evolutionary Algorithm family. The NSGA has been updated to a new version NSGA-2 which is implemented in the project. NSGA2 as the name suggests works by sorting the candidate solutions by the dominance of the solutions on other solutions based on the objectives. The solutions are evolved using selection, crossover and mutation as in Genetic Algorithm. The evolved set of solutions and the original solutions are mixed and the Pareto front for this set is found and is to be used as the base population for the next iteration of evolution. The algorithms thus advances the sets of solutions based on the Pareto front for each iteration until there is no significant improvement in the evolved solutions from the original set of solutions [10].

Strength Pareto Evolutionary Algorithm (SPEA2)

SPEA2 also is a stochastic algorithm which is an Evolutionary Algorithm and can be used to optimize multiple objectives. SPEA too has been extended to SPEA2 which is used in this project. SPEA2 also uses Pareto optimality to find the solutions that are non dominated among the population of candidate solutions. The SPEA2 uses elitism to determine the set of solutions that are to be used to generate the next set of solutions using mutation, crossover and selection. The selection process of the optimal solution uses a combination of the degree of

dominance of the solution over other candidate solutions and the spread of the current pareto frontier [11].

Related Work

Tabu Search

Tabu search is one of the most widely used optimization algorithm for the TSP, there are well over 60 papers which use Tabu search for TSP [6]. We can summarize the related work based on various parameters as follows:-

Problem Size: The problem size is usually limited to 400 nodes due to computational complexity limitations.

Initial Population: For the generation of initial population Randomized Insertion and Nearest Neighbor methods are the most popular. Randomized insertion starts with a subset of the total nodes (partial tour) and then inserts nodes randomly without allowing formation of sub tours when adding. It stops when all nodes have been added. Nearest neighbor method also starts with a partial number of nodes and then looks for the nearest node not in this partial tour and adds it to the tour. If after this process there are some nodes that remain unconnected to the TSP graph, It uses Randomized Insertion to add them.

Iterations: For the generation of new iterations, the popular methods are 2-opt move, vertex insertion and vertex exchange. In 2-opt move, two non-adjacent edges are removed from an existing tour and two new edges are created by connecting the heads and tails of the removed edges. In vertex insertion, a random node is removed and placed at a new location thereby generating new neighborhood tours. In vertex exchange, the position of two nodes is exchanged to create a new tour.

Tabu Tenure: Tabu tenure specifies the duration (in terms of number of iterations) for which the algorithm cannot jump to a solution present in the tabu list. Tabu tenure can be static or dynamic. In most recent papers the dynamic approach is chosen over static approach, wherein the tabu tenure changes dynamically based on solutions appearing in previous iterations depending on the problem.

Aspiration Criteria: This specifies certain conditions under which an iteration can jump to a forbidden solution. Most commonly such a jump is allowed only when a forbidden solution leads to a solution better than the best found so far. Most papers employ such an aspiration criteria.

Tabu Search uses these parameters to generate new solutions based on the population of solutions and selects or rejects the solution based on its presence in the Tabu list and moves on to the next solution in hope to find the global best solution without getting stuck at a local minima of a solution.

Genetic Algorithm (GA)
Genetic Algorithms have been used to analyse the Traveling Salesman Problem for some time. N. Gambhava et. al. studied the use of Genetic Algorithms and the effect of single point crossover and mutation with increasing population sizes[12]. While M. Dorigo et. al. proved that for single objective optimization, Ant Colony optimization is better than Simulated Annealing and Evolutionary Computation[13]. These implementations of various flavours of GA has been made to single objective travelling salesman problems which are supported by these algorithms. However they fail to provide the most optimal solution when the problem contains multiple objectives which needs to be optimized to find the right solution. This is where NSGA2 and SPEA2 come in.

Non-dominated Sorting Genetic Algorithm 2 (NSGA2)
NSGA-2 has been used for solving two objective travelling salesman problem while minimizing the cost of travel as well as the distance of travel by Wei Peng et. al.[7]. They had compared NSGA-2 to Multi Objective Evolutionary Algorithm (MOEA/D) with and without local search for the same instance of traveling salesman problem. NSGA-2 has also been used to solve Multiple Traveling Salesman Problem using binary domination for two objectives[8]. This implements the NSGA algorithm with elitism enforced using binary domination of the two objectives, distance and time of travel with no constraints.

Strength Pareto Evolutionary Algorithm 2 (SPEA2)
SPEA has not been formally used to solve TSP and hence there is no performance comparison of the algorithm against NSGA-2 as it has not been previously recorded for travelling salesman problem.

Section 3: Assumptions of our project

The implementation of our solution to the problem takes into consideration certain assumptions which are vital for the problem and the approach towards solving the problem.

We are assuming that the agent or the salesman is continuously moving through the cities and the delivery takes place any time of the day. These assumptions are implemented by running the program clock only when the agent takes an action updating the clock accordingly. Also the time of arrival of the agent at a city is directly checked against the expected time of delivery of the city without filtering work hours for the city.

We are assuming that all the cities can be directly reached from other cities and a direct path exists between each pair of cities. We are also assigning certain weights to all the paths connecting the various cities. These weights can be open to interpretation, however we are assuming these weights to be a form of drag on a path between the two nodes which will affect the time of travel for the agent between the two cities, for instance speed limits on a road.

We are also assuming that the agent will visit each city even though he could be or is already later than the expected time of delivery for that city and end up not getting any satisfaction points from that city. This assumption can be altered to fit a different real world scenario where the agent will refuse to visit a city unless he gets the satisfaction points from that city and due to this decision he can save some time and visit other cities earlier and not lose out on the satisfaction points of those cities depending on delivery criticality or profit criticality of the product.

Section 4: Implementation of the model, and the optimizer

We implemented the model of Traveling Salesman Problem using concepts of Object oriented and domain specific language. We implemented our program entirely in Python and used relevant Python libraries to help us. We implemented classes for various entities and used the object of these classes to refer to the functionalities, entities and objectives while solving the problem. The *Problem* class is the main and the central class for the project which imports all the other classes and calls them as and when required. This class has properties relevant to the main problem we are solving - TSP. This class has details of the list of cities and the paths between them along with the respective weights. These values are randomly generated in this class itself depending on the number of cities which is given as an input. These randomly generated values are used to generate weight and distance matrices which stores the weight and distance values for every path respectively.

The Problem class creates Node object for each city randomly which stores the coordinates of the city on the map. A Route class object is created for each pair of cities which stores the details like distance and weight of the route along with endpoints of the route. A vehicle class object defines the agent traversing the map. An object of class Solution is used to store a candidate solution which contains values like total distance, total time and total satisfaction the solution provides.

Each algorithm has its own class which includes Tabu, GA and NSGA to SPEA class. The algorithms consist of the values that are required for the implementation of that algorithm which may include population, problem, number of generations or iterations and other details. We have implemented Continuous Domination as well as Binary Domination for all the algorithms. The binary domination works on the simple principle of the the objectives being at least equal or better for all the options and strictly better for at least one option. Whereas the continuous domination works on the principle of loss function for the solution to traverse between one solution to another. The solution with the lesser loss function to reach the other solution, dominates the other solution. These domination functions take in two solutions and returns a boolean value depending on whether the first solution dominates the second one.

We started with a naive implementation of Tabu search to use it as an initial exploratory algorithm to determine the spread. The initial population was randomly generated. Tabu search

was run with increasing number of iterations from ten to ninety. We observed that higher number of iterations did not always produce better results. Also the Tabu tenure was kept static and was varied from ten to ninety. We observed no specific trends in terms of quality of results with variations in tabu tenure. Drastic improvements in performance in consecutive iterations might not have been achieved because we did not implement any aspiration criteria. However, the naive implementation did prove to us that there was considerable scope for generation of optimal outputs based on the wide spread of the solutions.

The GA class creates a random population of solution to start with and uses single point crossover and mutation to generate child solutions based on two random solutions and finally uses selection based on binary or continuous domination to select the fittest of the population to be carried forward into the next generation. Repeating this till the child solutions do not show considerable improvement over parent solutions or till a fixed number of generations are the two options we have used.

The NSGA class uses similar approach as that of GA save using two-point crossover rather than single point crossover and polynomial bounded mutation. The crossover and mutation function also takes as input the crowding factor which influences the nature of the child solution that is generated. Higher crowding factor gives child solutions which resembles more with the parent solutions and vice versa. The selection is also based on the crowding of the solutions and fitness of the solutions. Both binary and continuous domination are implemented into NSGA2 algorithm by us. The NSGA2 uses the pareto from the population for generating the next generation of solutions. Finally the NSGA algorithm returns the final population and pareto frontier for the problem among the population.

The SPEA algorithm is implemented using the Deap package in python. The objectives are defined as required by Deap package and the options for crossover, mutation and selection are defined initially which are used when the algorithm runs. SPEA2 also works on the concept of using the Pareto frontier to generate next generation of solutions. It uses the concept of Partially Matched crossover provided by the Deap package and the shuffle indices mutation method. We have also implemented binary and continuous domination functions to determine the pareto frontier.

The problem class also contains the implementation of the functions used to calculate the spread and the inverted generational distance (IGD) values of the population of the solutions generated by each of the algorithms.

Section 5: Methodology of the project

We used the Problem class to generate a random problem with the number of cities given as input, those nodes are randomly generated on a location on the map and each node is connected to every other node. The path distances are calculated using the formula for

Eucledian distance and stored in the distance matrix or an adjacency matrix. The paths are randomly given a weight which becomes a factor by which the time to traverse the path is multiplied to give the effect of drag on the path and these values are stored in another matrix called speed_matrix. Once the details of the map are calculated and all the information is stored, a random problem is created. The problem is passed onto each of the algorithms. The number of generations that the algorithm generates are individually customisable and type of dominance used to check for the dominance of each solution is also individually customisable. The algorithm objects returns a response which is recorded in the form of final population and the pareto frontier of the final generation of each algorithms. The pareto frontiers together are used to find the ideal pareto frontier which is used to compare against the individual pareto frontiers to calculate the IGD values of the populations of the algorithms. The spread is calculated by normalizing the pareto frontier solutions and then finding the difference between the extremes and the average distances between the solutions. The pareto frontier values and the population values can also be plotted on a 3-D graph to show the pareto points given by the algorithm. The code also provides the running time for each algorithm which can be noted to compare them for each algorithm.

Section 6: Findings

We are showing our results in the form of comparison of the Runtime, Spread and IGD values of each algorithms for a common problem. The variance in each of the values shows which algorithm gives the best results in the least amount of time. For making the comparison evident, we have generated graphs of the values of Runtime, Spread and IGD for each algorithm based on the Binary Domination. The algorithms fare relatively similarly when the binary domination is substituted with continuous domination. We also have generated the pareto frontier for each algorithm where we have plotted the solutions that are part of the population in yellow and the pareto solutions in blue so that they are clearly visible.

Runtime Analysis:

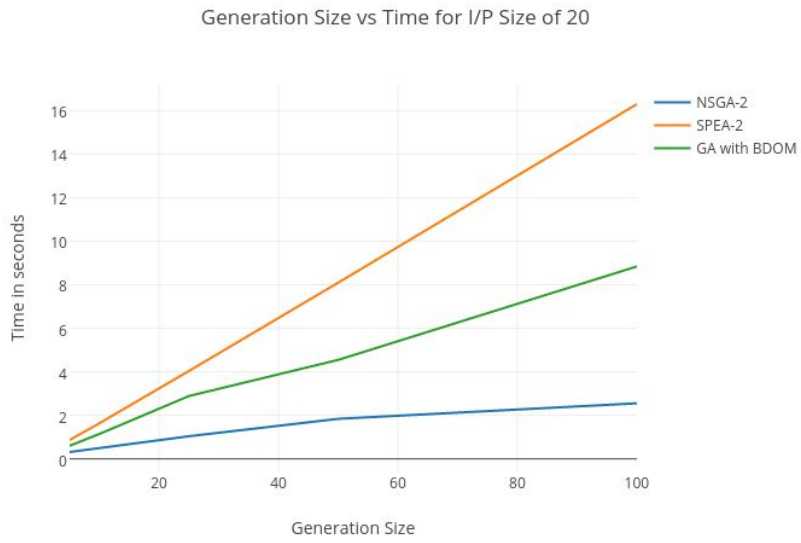

Fig. 1: Runtime for all the algorithms with fixed input size of 20 cities and varying number of generations.

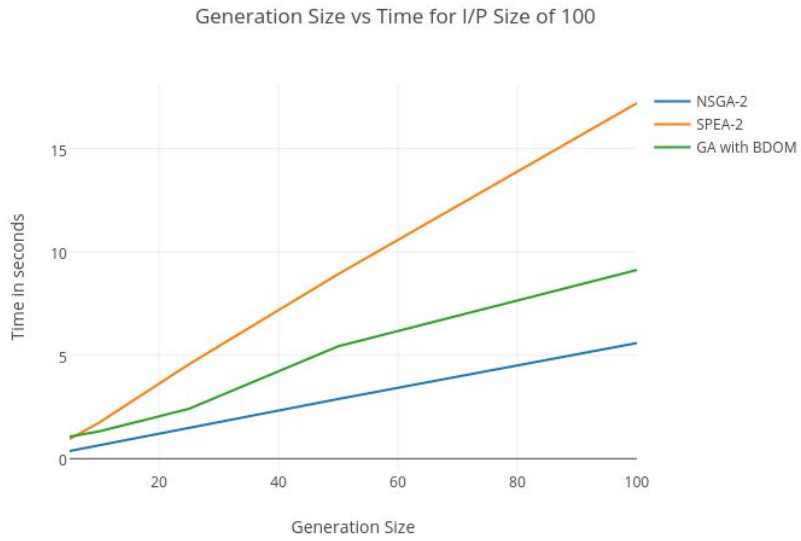

Fig. 2: Runtime for all the algorithms with fixed input size of 100 cities and varying number of generations.

The runtime goes up as the number of generations for the algorithm increases. The response improves as the number of generations increase but it is a trade-off between quality of pareto solution and time taken to execute the algorithm.

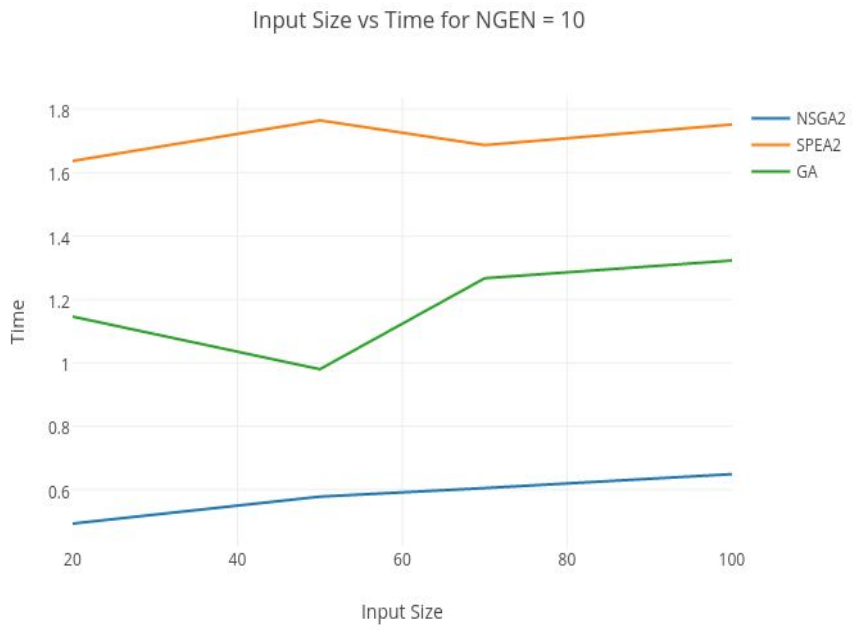

Fig. 3: Runtime analysis of the algorithms with fixed number of generations of 10 and varying input size of cities

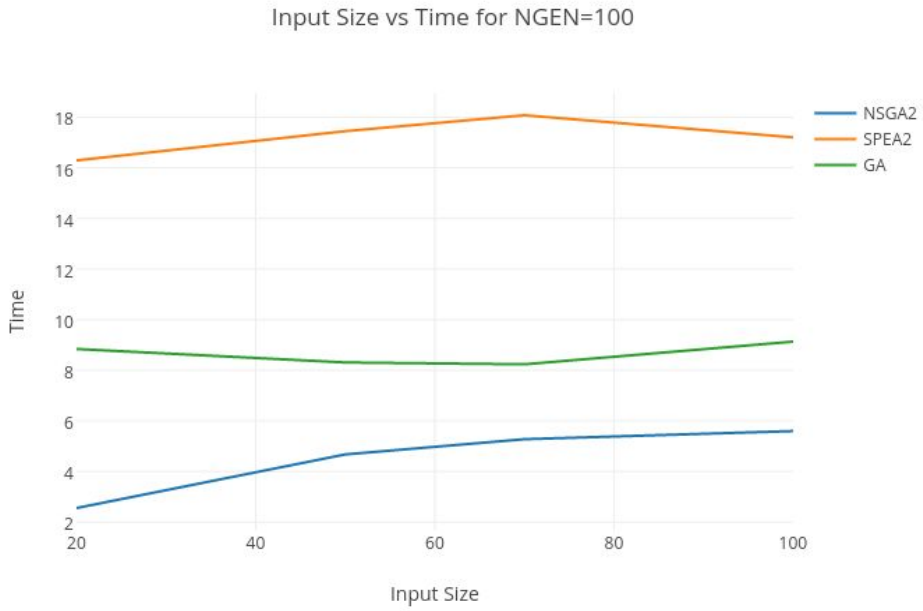

Fig. 4: Runtime analysis of all algorithms with fixed number of generations of 100 and varying input size of cities

This comparison shows us that the running time of a code depends mostly on the number of generations that needs to be generated and it remains more or less the same with slight change in the number of cities passed on as the input.

Spread Values:

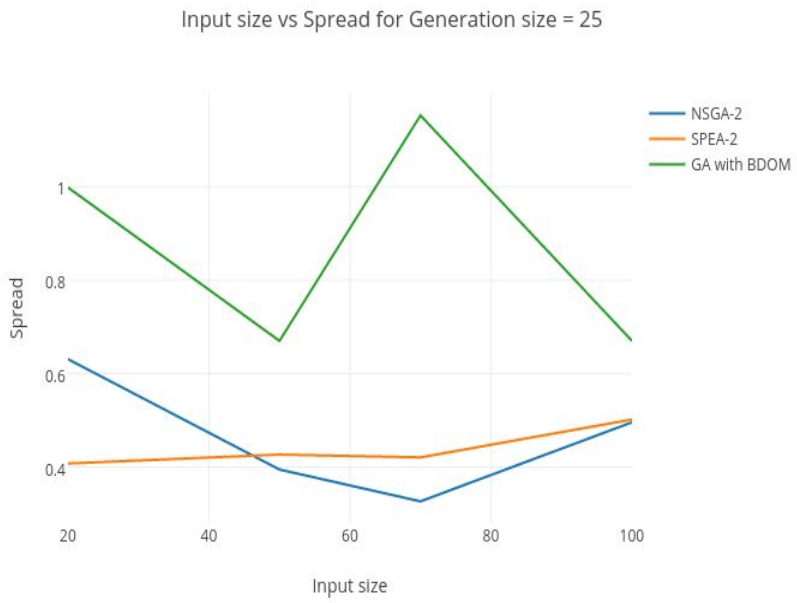

Fig. 5: Spread Values of all the algorithms with fixed number of generations of 25 and varying number of cities as input.

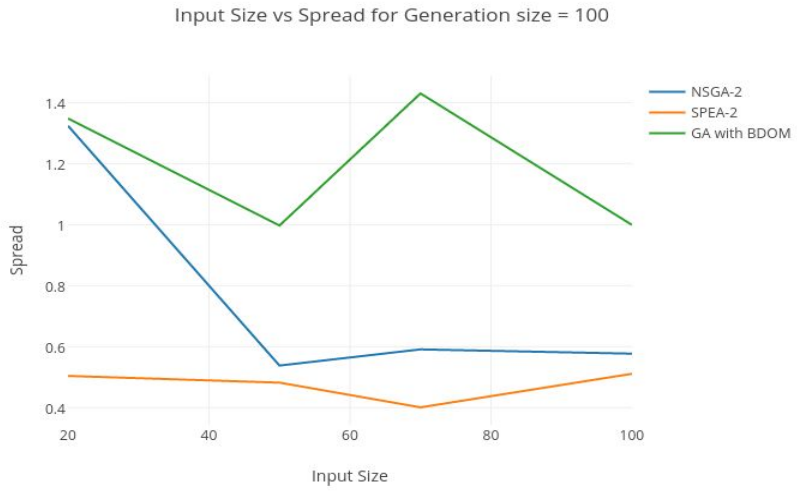

Fig. 6: Spread Values of all the algorithms with fixed number of generations of 100 and varying number of cities as input.

Genetic Algorithm gives a spread of solutions which varies as the number of cities in the input varies. However, Genetic Algorithms does give the best spread overall compared to NSGA2 and SPEA2.

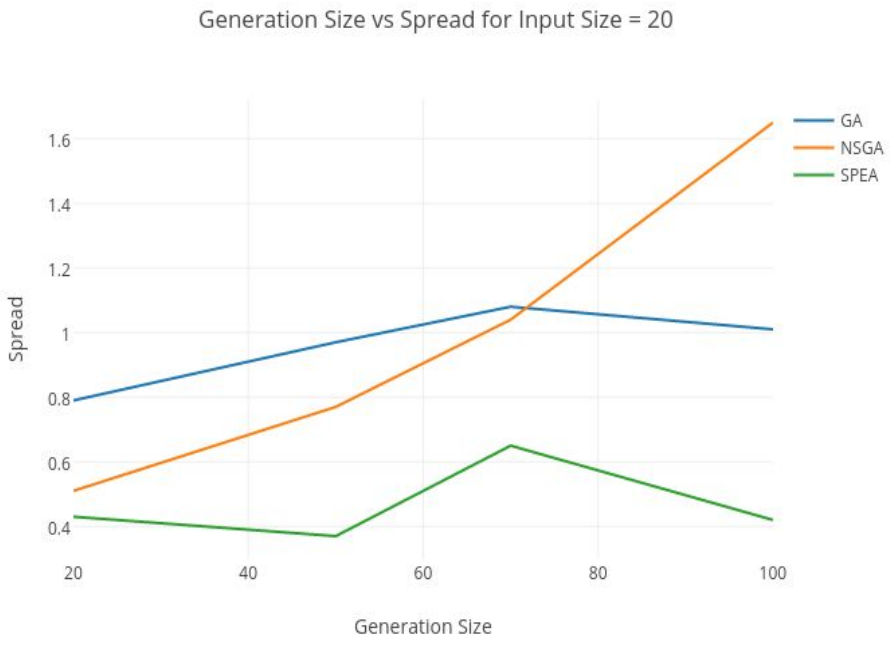

Fig. 7: Spread values of all the algorithms with fixed number of input size as 20 and varying number of generations.

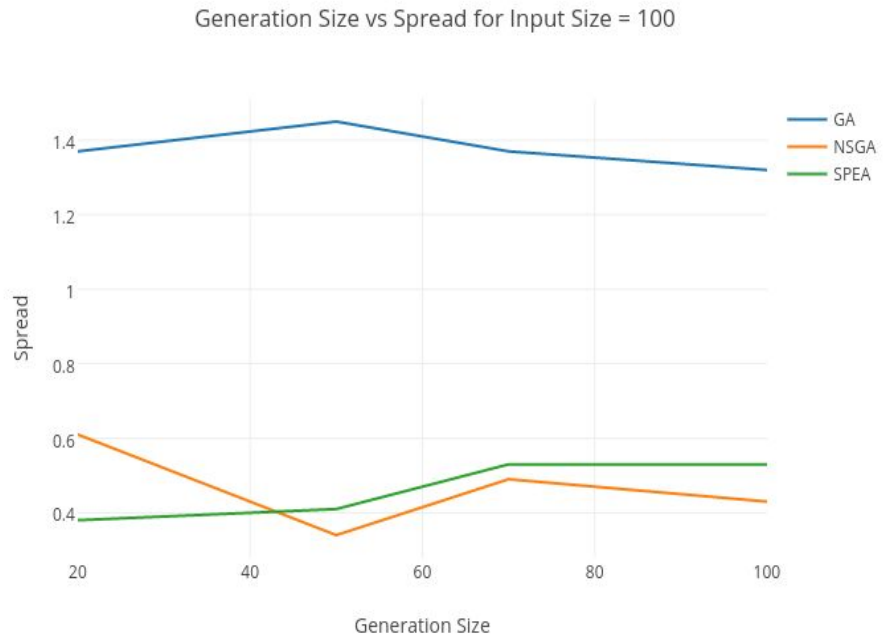

Fig. 8: Spread values of all the algorithms with fixed number of input size as 100 and varying number of generations.

IGD Values:

Fig. 9: IGD values with a fixed input size of 100 cities and varying number of generations

Fig. 10: IGD values with a fixed input size of 20 cities and varying number of generations

The IGD values of Genetic Algorithms are extremely high and that skews the differentiation between the IGD values of NSGA2 and SPEA2 to a very small scale. To check the actual

difference between the values, we have generated graphs without the IGD values of Genetic Algorithms.

Fig. 11: IGD values of NSGA2 and SPEA2 with constant input size of 20 cities and varying number of generations.

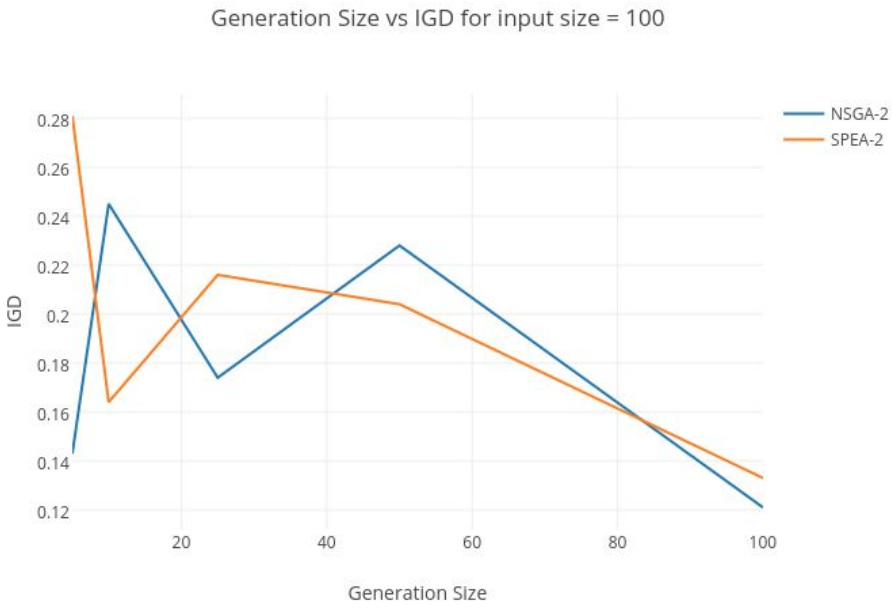

Fig. 12: IGD values of NSGA2 and SPEA2 with constant input size of 100 cities and varying number of generations.

This Comparison shows that the IGD values are more or less similar for both NSGA2 and SPEA2 however the SPEA2 IGD value tends to increase more than that of NSGA2 as the generation size and the input size of number of cities increases.

This drives us a conclusion that NSGA2 gives the best set of solutions for the least amount of time spent in trying to find the optimal solution.
Pareto Frontier:

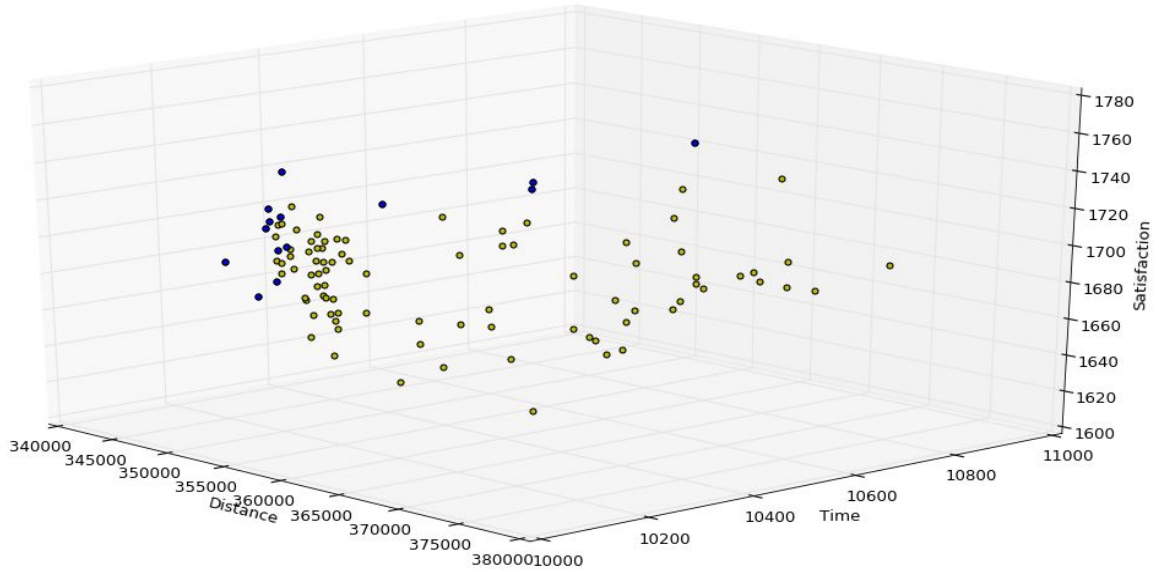

Fig. 13: Pareto Frontier of Genetic Algorithm with Binary Domination

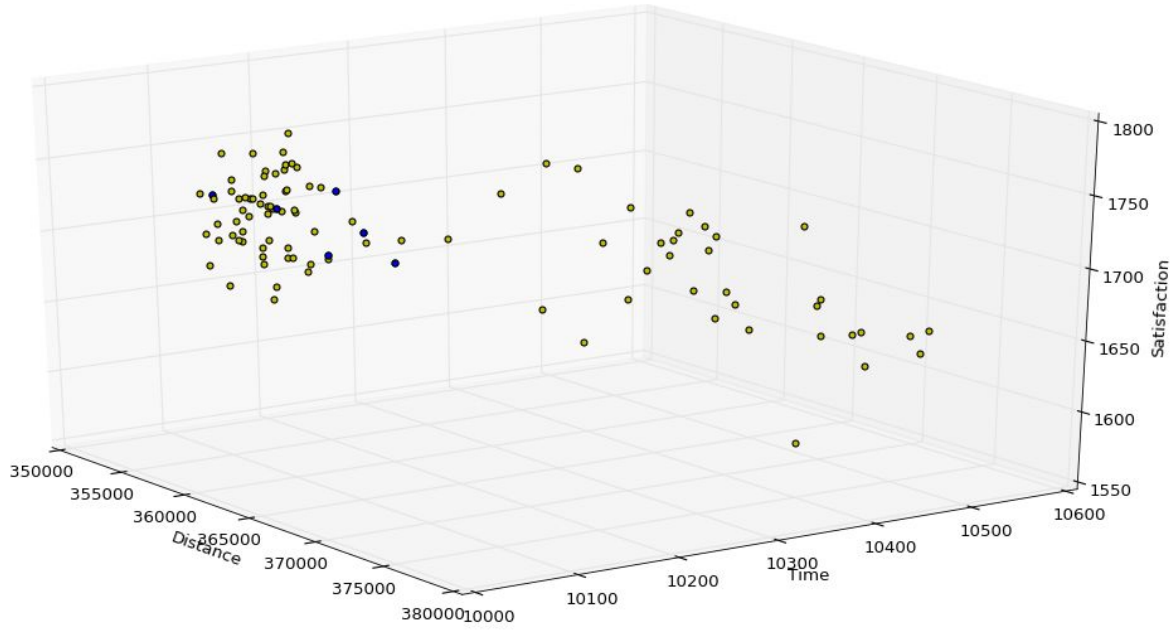

Fig. 14: Pareto Frontier of Genetic Algorithm with Continuous Domination

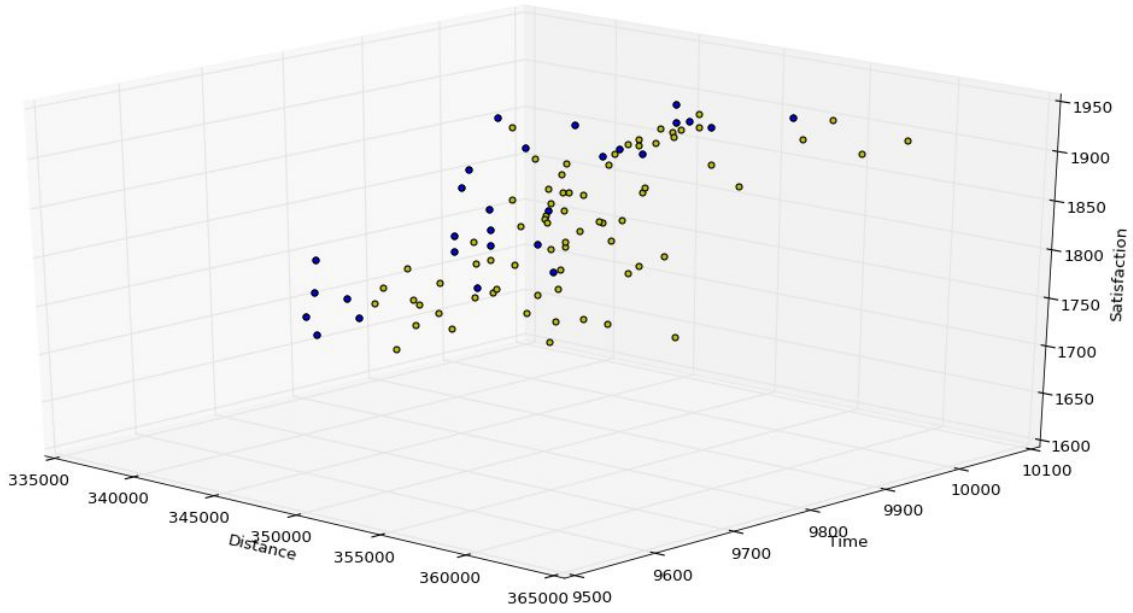

Fig. 15: Pareto Frontier of Non-dominated Sorting Genetic Algorithm with Binary Domination

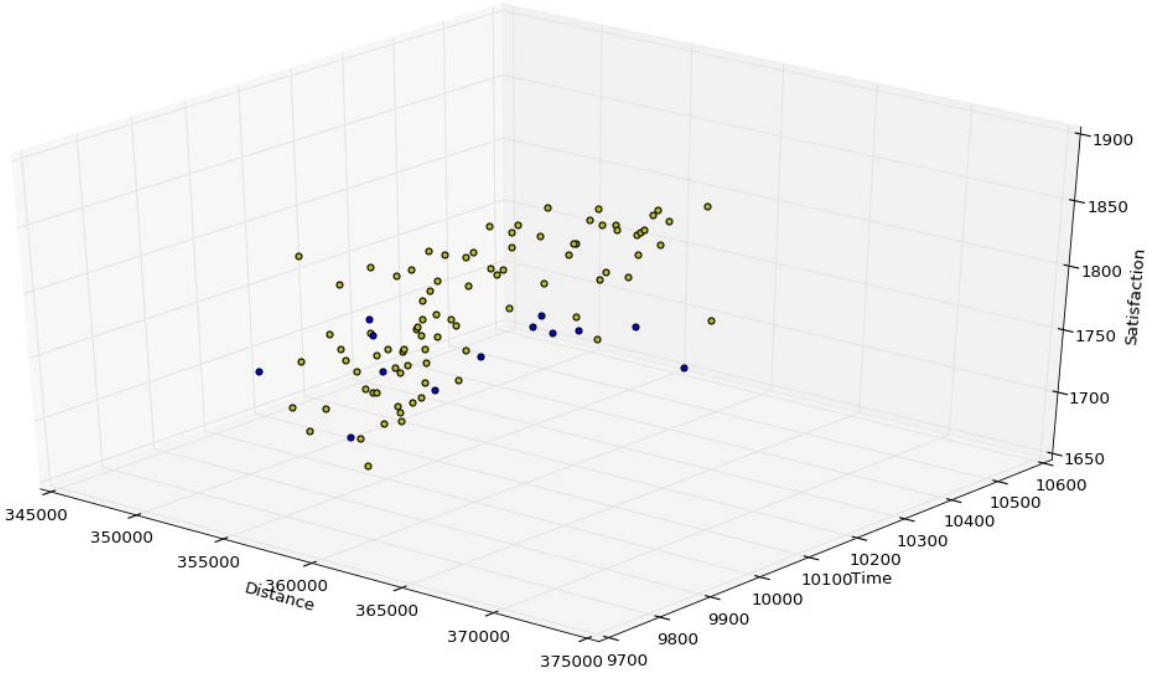

Fig. 16: Pareto Frontier of Non-dominated Sorting Genetic Algorithm with Continuous Domination

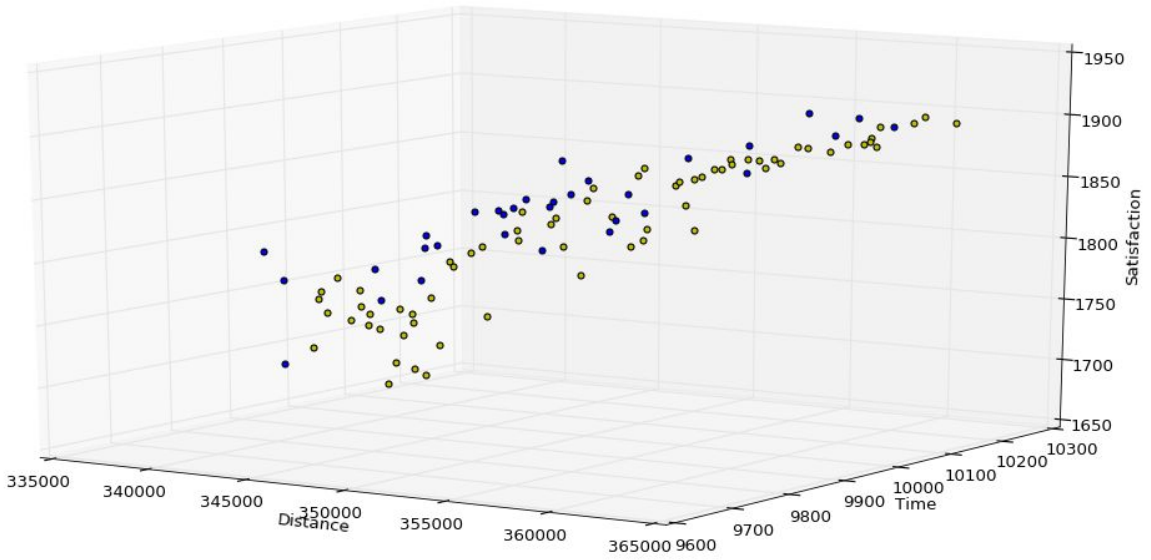

Fig. 17: Pareto Frontier of Strength Pareto Evolutionary Algorithm with Binary Domination

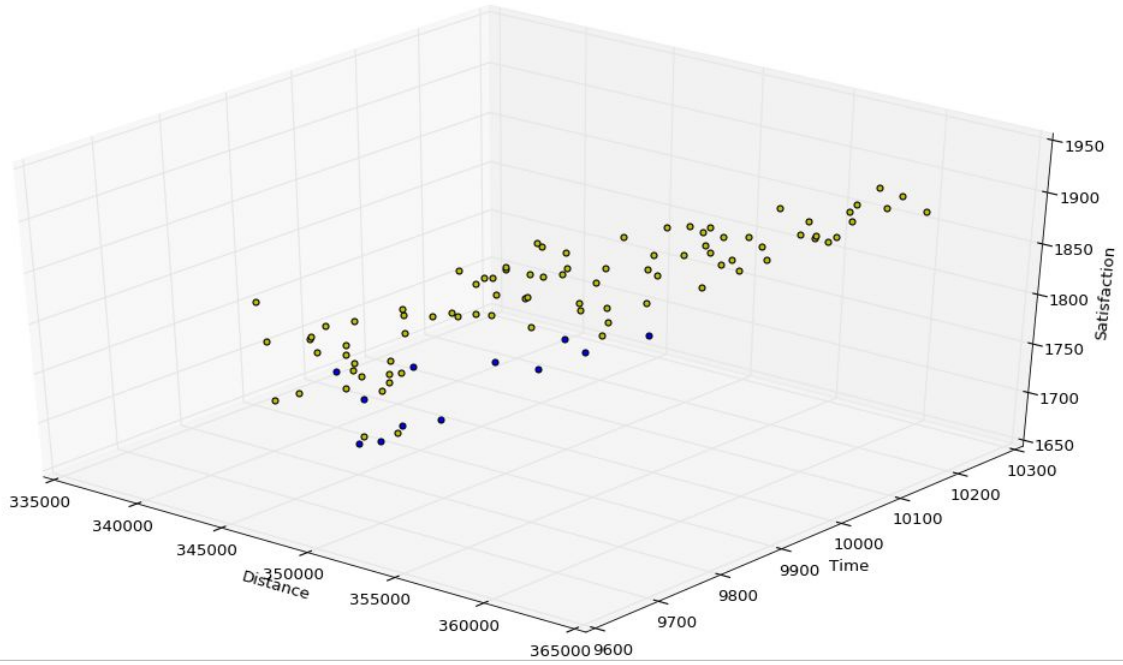

Fig. 18: Pareto Frontier of Strength Pareto Evolutionary Algorithm with Continuous Domination

The Pareto frontiers show the wide range of solutions that are provided by NSGA and the variations of the solution on changing the type of domination from binary to continuous. The Genetic Algorithm provides us with solutions that are more clustered than NSGA2, while SPEA2 as well gives a nice spread of optimal solutions to choose from.

Section 7: Future directions of the project

This project has a lot of real world applications and scope for extensibility. Travelling Salesman Problem being one of the problems currently in the Industry, newer requirements of solving the problem and optimizing the solution for various objectives and under various constraints will keep cropping up. With the onset of automation and automated vehicles, the Travelling Salesman can be broken in smaller pieces and applied into different requirements ranging from automated Taxi service to automated Delivery service.

This solution can also be extended to plan trips for user when he has requirements of visiting a list of locations and has preferences of events in those locations which will be at specific times reaching after which will be worthless. The user will be the traveling salesman with satisfaction rating denoting the attendance of maximum events.

The solution can also be brought close to reality by allowing the traveling salesman some rest each day and denoting hours of office when the customers can accept the delivery of the order and maybe even giving windows of times when a route is the fastest.

Also extending this Traveling Salesman Problem is Vehicle Routing Problem where the salesmen are multiple in number for a list of locations needing delivery. Optimizing the utilization of resources such that each of the location is visited and minimal expenditure is utilized forms a vital problem to be solved by taxi and delivery companies.

Section 8: Conclusion

This project has given us a good insight into the application of Multi Objective Evolutionary Algorithms to optimize the solutions of classical NP-Hard problems like TSP. We implemented four such algorithms and compared their performance based on several parameters. We also studied how these algorithms were being applied in research to solve problems like the TSP. Our modular implementation has scope for being extended and enhanced to include more objectives, decisions and algorithms. It provides a platform for the comparison of stochastic algorithms for solving the TSP. We did find that NSGA2 gave the best results among the three algorithms in the least amount of time spent trying to optimize the solutions. It also ranks second fastest among the algorithms that we implemented behind Genetic Algorithm. We noticed that different algorithms performed well under different conditions and parameters of comparison.

Section 9: Acknowledgement

We wish to thank the course instructor, Dr. Tim Menzies, and teaching assistant George Matthew for giving us valuable advice in implementing the project.

Section 10: References

[1] Daniel J. Rosenkrantz, Richard E. Stearns, Philip M. LewisII, "An analysis of several heuristics for the traveling salesman problem", SIAMJ.Computing,Vol.6,No.3,Sept.1977, pp.563–581.
[2] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum, New York, 1972.
[3]Thibaut Lust, Jacques Teghem, "The Multiobjective Traveling Salesman Problem: A Survey and a New Approach", Advances in Multi-Objective Nature Inspired Computing, Volume 272 of the series Studies in Computational Intelligence pp. 119-141
[4] Juanjuan He, "Solving the Multiobjective Multiple Traveling Salesmen Problem Using Membrane Algorithm", 9th International Conference, BIC-TA 2014, Wuhan, China, October 16-19, 2014. Proceedings, pp. 171-175.
[5] Glover, F., 1989. Tabu Search, Part I, ORSA Journal on Computing, vol. 1, no. 3, 190-206.
[6] Sumanta Basu, "Tabu Search Implementation on Traveling Salesman Problem and Its Variations: A Literature Survey", American Journal of Operations Research, 2012, 2, 163-173
[7] Wei Peng, Qingfu Zhang, Hui Li, "Comparison between MOEA/D and NSGA-II on the Multi-Objective Travelling Salesman Problem", Multi-Objective Memetic Algorithms, Vol 171, pp. 309-324.
[8]R. I. Bolanos, M. G. Echeverry, J. W. Escobar, "A multiobjective non-dominated sorting genetic algorithm for the Multiple Traveling Salesman Problem", Decision Science Letters 4, 2015, pp. 559-568.
[9] K. F. Man; K. S. Tang; S. Kwong, "Genetic algorithms: Concepts and Applications", IEEE Transactions on Industrial Electronics, Year: 1996, Volume: 43, Issue: 5, Pages: 519 - 534.
[10] K. Deb; A. Pratap; S. Agarwal; T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II", IEEE Transactions on Evolutionary Computation, Year: 2002, Volume: 6, Issue: 2, Pages: 182 - 197.
[11] Strength Pareto Evolutionary Algorithm (SPGA) (Zitzler and Thiele, 1999)
[12] N. Gambhava, G. Sanghani, "Traveling Salesman Problem using Genetic Algorithm", (2003)
[13] M. Dorigo, L.M.Gambardella, "Ant Colony System : A cooperative Learning Approach to Travelling Salesman Problem", in *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53-66, Apr 1997.