Multi-objective optimization of the Multiple Traveling Salesmen Problem Using a Non-dominated Sorting Genetic Algorithm (NSGA-II)

Rahul Balamurugan

*Abstract*— **The multi-objective multiple traveling salesmen problem (MOmTSP) is a rather popular multi-objective combinatorial optimization problem (MOCOP) with many real-world application. The non-dominated sorting genetic algorithm NSGA-II is a fast, elitist genetic algorithm, and has been used to solve the problem and other similar MOCOPs in literature. This project report details the implementation of the NSGA-II algorithm to solve two different formulations of the MOmTSP- one, to minimize the difference between the longest and shortest individual tour lengths, and the other, to minimize the sum of the difference between the average tour time and time taken for each individual tour, both in addition to minimizing the total traveled distance. Four crossover operators, a small range of mutation probabilities, and three population sizes were tested to find the best combination of values among them for both cases. Additionally, the implementation showed better or comparable performance when compared against the benchmark and results in literature. The obtained experimental results are detailed in this report.**

1. INTRODUCTION

Solutions to multi-objective optimization problems (MOOPs) often find applications in the real world [], as many real world problems often have multiple conflicting objectives. A few of the domains that benefit from research into such solutions are resource management [], manufacturing [], network coverage [], and various other domains []. A popular example of such problems is the multi-objective multiple traveling salesmen problem (MOmTSP), first studied by Frederickson, Hecht, and Kim [] in 1978.

The MOmTSP, which is NP-Hard, can be briefly formulated as minimizing the total distance travelled and balancing the workload for *m* salesmen visiting *n* cities exactly once, with each salesman originally departing and finally returning to a common origin city or ‘depot’. A particular modern application of the problem that comes to mind would be in the deployment of multiple drones to achieve certain goals (e.g. delivery of packages) at certain locations and return. Due to limitations on battery life, there is a need to ensure all deployed drones are active for similar times so the coverage of locations can be maximized and all drones allowed to return before the end of their battery life. In addition, many other multi-objective combinatorial optimization problems (MOCOPs) like the vehicle routing problem [] and job shop scheduling problem [] can be transformed into the MOmTSP and solved, which allows the MOmTSP to act as an adequate test problem for MOCOP solvers.

Evolutionary algorithms have been at the forefront of heuristic algorithms for solving MOCOPs and MOOPs for over a decade [], with the non-dominated sorting genetic algorithm NSGA-II, proposed by Deb [] in 2002, being the most cited and used algorithm in literature for solving problems in this class []. Some salient points which make this algorithm so popular are that it incorporates an explicit diversity preservation mechanism, as well as implicit elitism [] to retain the best solutions across generations. Due to the mentioned advantages, proven effectiveness [] as well as the open availability of the NSGA-II [], the algorithm has been used as a sort of gold standard for comparing other (novel) algorithms [], especially since similarly (or more) efficient tools used in industry are inaccessible.

However, the NSGA-II when used with conventional crossover operators devised for permutation-based problems like the Partially Mapped Crossover (PMX) has been noted to converge prematurely []. Although there is a lack of studies as to why this is the case, it may be intuitively attributed to two reasons: first, a lack of evolvability when using the mentioned operators, as the ‘good’ subsequences or arrangement of genes are not preserved during inheritance; and secondly, because in later generations, the population of solutions may lack in diversity [].

On the other hand, studies [] have noted that modifications to the NSGA-II, in terms of specialized genetic operators, replacing or supplementing the diversity preservation mechanism, or hybridizing with other algorithms [] can allow the resultant algorithm to be just as good if not better than the benchmark algorithms for the considered problem domain.

In this project, the main objective is to implement the NSGA-II algorithm to solve the MOmTSP and test a number of crossover operators (PMX, CX, OX, HGAX), and effects of varying population size and mutation probabilities on instances from TSPLIB.

Running with the drone application mentioned earlier, two different objective functions were considered separately in addition to minimizing the total distance traveled— *first*, minimizing the difference between the longest and shortest individual tours, which would be representative of the ideal case where all salesmen (or drones) deployed travel at the same speed and without any delays between cities; and the *second*, minimizing the difference of the total travel times of each salesman with the average travel time, assuming that each traversal between two cities *i* and *j* is done at a random speed between a given range. The former formulation has benchmarks for certain instances from TSPLIB in literature obtained using the brute force CPLEX [], while the latter does not. Either ways, the effectiveness of the implementation can be observed by comparing the final range of total traveled distances obtained with the benchmarks as well as from [] and [], both of which have tested either of the two cases.

The main deliverable for the project is the software written for the implementation in Python, submitted alongside the report and also made available in a public access GitHub repository. The repository contains functions to implement the NSGA-II as well as the various crossover operators mentioned, four mutation operators, and a selection operator; and also the main code to solve the MOmTSP (both variations) for both random instances and select instances (eil51, berlin52, eil76, and rat99) from TSPLIB, which are also included as text files.

The project report beyond this introduction is structured as follows: first, the relevant background information needed for the project, such as terms related to multi-objective optimization and the NSGA-II algorithms are introduced along with the mathematical formulation(s) of the MOmTSP; the next section briefly mentions related work on NSGA-II as well as its (and other GA) application to variations of the mTSP with specially mentioned proposed solutions to the above discussed issues with EAs; thirdly, the implementation of the algorithm and genetic operators is detailed along with the design of the experiments to test the implementation; and lastly, the obtained results are discussed, with some critical analysis connecting the results to the broader issues in EAs and their usage in solving MOCOPs.

2. BACKGROUND

***2.1. Pareto-Optimality and Dominance***

In multi-objective problems, it is required to find an optima for every one of the objectives, and thus, for conflicting objectives, it becomes impossible to find a singular optimal solution. Instead, the goal of solving the problem is to determine an optimal pareto front, which is the set of optimal solutions to the problem (Pareto-Optimal Solutions or POS), from which appropriate trade-offs can be considered to choose a suboptimal solution.

The pareto-optimal front is composed of solutions that “dominate” all other solutions but not each other. Here, a solution, say X, is said to dominate another, say Y, if X evaluates to better than Y for at least one objective function, and equal to Y for the rest of the objective functions. Graphically, this can be seen in Figure 1, where the round solutions dominate the square ones for two objectives (obj1 and obj 2). Thus, the round solutions constitute the optimal pareto front.

More generally, a pareto front (alternatively pareto frontier) consists of solutions that do not dominate each other. This is the concept of non-dominance which is exploited in the NSGA and NSGA-II.

***2.2. Multiple Traveling Salesmen Problem***

As described earlier, the mTSP involves *m* salesmen visiting *n* cities exactly once. While there are multiple variations that arise from this point, such as multiple origin cities, different origin and final destination cities, and so on, the problem considered is of the conventional closed circuit with all salesmen originating and finally returning to the same depot, called the SD-MTSP.

Considering the problem in graph form, it can be represented by the undirected graph with *V* being the set of vertices representing cities and *A* being the set of arcs between the cities. There is an associated distance matrix C of dimensions , with each element being the distance between *i* and *j*. Each salesman , travels from city *i* to city *j* and is an indicator of whether *k* travelled the arc between *i* and *j* (if yes,=1; else,=0).The set of constraints on the tours can then be mathematically described as:

|  |  |
| --- | --- |
|  | (1) |
|  | (2) |
|  | (3) |
|  | (4) |
|  | (5) |

where ‘0’ is taken to be the depot city.

Constraints (1) and (2) limit the traversal of each arc to just one time across all salesmen; constraints (3) and (4) ensure all salesmen start and end their tours at the depot; and lastly, constraint (5) is to ensure that no sub-tours excluding the depot are created.

The first combination of objectives is to minimize the total distance travelled by all salesmen and the difference between the longest and shortest tours, which can be represented by the equations (6) and (7) respectively:

|  |  |
| --- | --- |
|  | (6) |

and,

|  |  |
| --- | --- |
|  | (7) |

where,

The second combination of objectives is (6) and also minimizing the differences of the travel times of each salesman with the average, represented by (8). The

|  |  |
| --- | --- |
|  | (8) |

where,

and,

***2.3. NSGA-II***

The Non-dominated Sorting Genetic Algorithm was proposed by Deb and Agarwal [] in 2002 as an improvement to the older NSGA, with the addition of elitism. The basic principle behind the titular fast sorting mechanism is the concept of ‘non-dominance,’ or simply that the solutions in a population be ranked based on their level of non-dominance.

In the algorithm, the solutions which are not dominated by any other solutions are ranked first, with solutions that are dominated by a single other solution being ranked second. Naturally, solutions having the same rank constitute a pareto frontier, and thus, each solution only needs to be compared once with all other solutions in the population, which is the mechanism exploited by the algorithm to speedily sort the solution population and determine elite solutions.

There are two main methods in the NSGA-II: fast non-dominated sorting, and crowding distance assignment, of which the former is described above. The crowding distance of a solution, as described by the authors, is the sum of the differences between each objective function evaluation of the nearest neighbors of the solution. The nearest neighbors are solutions such that they ‘box’ in the solution. Thus for a bi-objective problem, the crowding distance of a solution would be the perimeter of the cuboid formed if the function values of its nearest neighbors were plotted on diagonally opposite vertices.

According to the algorithm, the less crowded solutions, i.e., ones that are farther away from any other would be relatively unique and thus worth preserving more than more crowded solutions in the same non-dominance level. Thus the algorithm discriminates between the solutions first on the basis of non-dominance level, and second (if the solutions are on same rank), on the basis of crowding distance. This is the diversity preservation mechanism in the algorithm.

While the crowding distance operator acts as a comparison function for the binary tournament selection of parents, it also acts as a way to sort the solutions of the same non-dominance level. In the algorithm, the offspring population is combined with the parent population after recombination and mutation, and the next generation (same size as parent population) is chosen from the best of this combined population, first sorting by non-dominance level, and if there is a front that will have leftover solutions after cutting off the rest, enough solutions in that front are chosen starting from the least crowded.

The steps of the algorithm are: *1)* Initial population formation (size=*N*); *2)* Selection of parents for next generation by binary crowded tournament; *3)* Recombination and mutation to form an offspring population of size *N*; *4)* Parents and offspring are clumped together in a population of size *2N* from which the top *N* solutions are selected via non-dominated sorting and comparison of crowding distance (elitism); and, *5)* The steps *2*-*5* are repeated until a satisfactory number of iterations or until there is no to little change in the pareto-front (convergence).

3. RELATED WORK

Both the mTSP and NSGA-II (and other GAs), separately and in combination, have been extensively studied in literature. In this section, *first*, an overview of EAs and other solvers for MOOPs and MOCOPs is provided; *secondly*, some significant papers on the topic and other recent novel methods of solving mTSP are reported; and *thirdly*, papers that describe the modification, improvement and/or direct implementation of the NSGA-II, both generally and in solving variations of the mTSP are described.

Verma, Pant, and Snasel. Kumar, Gopal and Kumar. Von Lucken, Baran, and Brizuela. De Buck et al. Toffolo and Benini. Hassanat et al.

Wang, Fand, Li and Jin. Yousefikhoshbakht, Didehvar, and Rahmati. Pang, Li, Dai, and Yu.

Shuang, Yunfeng and Kai. Alves and Lopes. Bolanos, Echeverry, and Escobar. Yokoyama and Sato. D’Souza, Sekaran and Kandasamy. Wang et al. (water).

4. METHODOLOGY

***4.1. Design***

*4.1.1. Chromosome Representation*

.

*4.1.2. Crossover Operators*

.

*4.1.3. Mutation Operators*

.

*4.1.4. Overall Algorithm*

.

***4.2. Implementation***

The program to run the algorithm and solve the MOmTSP was written in Python 3 and run on an Intel Core i7-7700HQ CPU (Windows 10 OS). The code was written from scratch, with one code snippet referred from [wrlwansky] (marked in appended code).

***4.3.Experiment Design***

.

*4.2.1. min (F1,F2)*

.

*4.2.2. min (F1,F3)*

.

5. EXPERIMENTAL RESULTS

The results from running the experiments detailed in the previous section are shown and inferenced from in this section, which as such is split into two parts— the first detailing the results of the parametric experiments, and the second presenting how well the implementation performed against the benchmark and/or results from literature.

***5.1. Parametric tests***

The parameters considered for this half of the experiments are the type of crossover operator, mutation probability, and population size. Note that as the obtained pareto fronts are irregular and very different from one another, it is not meaningful to have a metric quantifying how much better one parametric combination is over another. Rather, the results are discussed more subjectively and in the relative terms “slightly”, “moderately”, and “significantly”. More than the level of difference is the observation of which objective function axis the pareto front(s) is oriented towards, and the degree to which that is the case. Such observations provide insight into which objective was emphasized by the combination, which could then be used to hybridize or tune a GA for future problems.

*5.1.1. min (F1,F2)*

.

*5.1.2. min (F1,F3)*

.

***5.2. Benchmark and literature comparison***

*5.2.1. min (F1,F2)*

.

*5.2.2. min (F1,F3)*

.

***5.3. Discussion on the experiment and results***

.

6. CONCLUSION

.

REFERENCES

[1].

[2].

[3].

[4].

[5].

[6].

[7].

[8].

[9].

[10].

[11].

[12].