IE 517 FINAL PROJECT

Alican Yılmaz(2016402093) , Cemre Çadır(2016401126)

JUNE 2021

1. Problem Introduction

Travelling Salesperson Problem with Profits (TSPP) is a well-known combinatorial optimization problem in the domain of TSP problems. In this variant, the salesperson gets a profit for each city visited, and a travel cost is incurred depending on the distance between the cities. Unlike normal TSP, the traveler does not have to visit all the n cities defined. In our problem, the salesperson always starts and ends its tour at the depot location. The objective of the TSPP is to find a tour that maximizes the total profit minus total travelling cost. In the literature, several metaheuristics are used to solve TSPP and its variants, and some have satisfactory results on the benchmark instances. In the problem, general-VNS (i.e., VNS with VND) is preferred and some conclusions as well as comments have been made about the results obtained. It is important to mention that the algorithm and operators used in this problem are highly inspired from [1] by Dasari et.al.

In Section 2, the algorithm (VNS with VND) is explained. In Section 3, the solution representation used in the problem is explained. In Section 4, the operators used in the local search/shaking phase as well as the greedy construction heuristic used in obtaining the initial solution are explained. Finally, in Section 5 and Section 6, numerical results, conclusions on the results and implementation of the algorithm with possible further-improvement suggestions are commented.

2. VNS with VND Algorithm

Variable neighborhood search is implemented with two neighborhoods (i.e. kmax = 2). Multi-removal and reinsertion operators together with multi-removal and addition operators are used in the shaking step and variable neighborhood descent (VND) is used as the local search step. In the VND phase, shaking step is omitted and the local search neighborhoods change with varying k. Three neighborhood structures are used for the local search, which are namely addition, removal, and 2-opt. The VND is terminated when the current solution is the local optimum solution with respect to all three operators. The stopping condition for the VNS is the total CPU time spent. The maximum time set to 3 minutes (180 seconds) for this problem as default value. Adding to that, if the incumbent objective does not improve for a certain amount of time called patience (40 seconds as default), the algorithm terminates.

The high-level pseudocode of the VNS algorithm implemented is as follows:

1. *Initialization:* Neighborhood structure $N_k\{k=1,2\}$ is defined. Initial solution is obtained with construction heuristic. Stopping condition is set as max CPU-time. (180 seconds in this problem setting) and patience time is set. (40 seconds)
2. **Repeat** the following **until** *stopping condition* is met:
3. Set k<-1, **Until** $k=k_{max}$ **repeat 4-6**:
4. *Shaking*: Generate x' at random from the k'th neighborhood of x.
5. *Local Search:* Apply local search(VND) x' to obtain local optimum x''.
6. *Move or not:* If this local optimum is better than the incumbent, move there (x<- x'') and continue the search with k<-1; otherwise set k<-k+1
7. **If** the incumbent solution does not improve for patience time, **stop**.

3. Solution Representation

Since the problem includes an ordering of the cities visited, *"permutation representation"*, where each element refers to the cities visited by the salesperson, is preferred in this problem. Since the salesperson always starts with "depot 0", the starting and ending depots are not included in the solution encoding. For instance, [4,13,17,12,50,1] means that the salesperson first leaves the depot 0 and then travels the cities in the order of 4->13->17->12->50->1 and finally returns to the depot 0 again. Thus, in this example, the salesperson travels 6 customers in total (not including depot).

4. VNS with VND Operators

4.1. Greedy Initial Construction

Initial solution is obtained with a construction heuristic described in [1]. First, the total number of customers(k) that are to be visited is determined randomly. (2<=k<=n). Then those cities are inserted one by one, at the best position of the tour, i.e., the position which yields the highest objective value. This process is repeated till we have k cities in the tour.

4.2. Addition

This operator adds an unvisited city at its best possible position in the tour if it leads to the largest improvement in the objective function value. To find the largest improvement, all untraveled locations tried on all possible locations to be added. If there is no improvement in the objective function value, the solution does not change. The time complexity of this operator is at worst O(n^2) as explained in [1].

For instance, [1,5,13,7]->[1,28,5,13,7]. Here, the unvisited city is inserted to the second location and tour length is increased by one.

4.3. Removal

This operator removes a city that is visited before if it leads to the largest increase in the objection function value. To find the largest improvement, all travelled cities are tried. Similar to addition operator, if there is no improvement in the objective function value, the solution does not change. This operator performs as a destructive operator as opposed to addition operator and its time complexity is at worst O(n) as explained in [1].

For instance: [1,5,13,7]->[5,13,7]. Here, the visited city 1 is removed from the tour and tour length is decreased by one.

4.4. 2-opt

This operator performs a classical 2-opt process to the cities visited. It, basically, removes two non-adjacent tours and connects two edges in place of them to obtain a valid tour. 2-opt operator, also checks all possible 2-opt moves in a given tour and performs the operation on the best two edges if it leads to improvement in the objective function value. Note that, this operator does not change the tour length, instead it changes the tour order.

For instance, [3,7,11,59,16,23,45]->[3,7,23,16,59,11,45]. The order between location 7 and 45 is reversed after the operation in that example.

4.5. Multi-removal and Reinsertion

Multi-removal and reinsertion operator as used in [1] removes the customers in the tour with a probability of 0.5, and then reinserts removed customers in a random order by inserting each removed customer to the best possible place in the solution. The resulting solution is a permutation of the initial solution (the solution before the operator is applied), and the number of visited cities remains the same but the objective is increased (or stayed the same). The complexity of the operator is O(n^2).

For instance: [1,5,13,7]->[7,1,5,13]. Here, 5 and 7 are removed and 7 is reinserted to the first location. Then, 5 is reinserted to the third location.

4.6. Multi-removal and Addition

Multi-removal and addition operator is defined in [1[and it removes the customers in the tour with a probability of 0.4. Then, the removed cities and the unvisited cities are added to the best possible location in the solution in random order if the addition increases the objective value. The resulting solution might have a different number of customers compared with the initial solution. This operator represents the furthermost neighborhood in our algorithm.

For instance: [1,5,13,7]->[4,7,3,1,13,9,]. Here, 5 and 7 are removed and 7 is reinserted to the first location. Then, 4, 9, and 3 are reinserted to the best possible location in the current solution.

5. Numerical Results

This section describes the computational experiments carried out on a machine with an Intel Core i7-7700HQ processor and 16 GB of RAM. Algorithm is coded in Python. Total of 6 experiments were conducted on 51,76 and 101 customers, each having “low” and “high” profit . Results are summarized in “Summary Table” as given in the project description.

Table-1: Summary of the Experiment Results

Instance	Best Objective	No. of cust	Sequence of customers visited	CPU Time (s)
eil51-LP	50,74	22	0,31,50,45,4,37,8,15,49,33,29,9,32,44,14,43,16,3,17,13,5,47	52,38
eil51-HP	171,29	43	0,21,19,34,35,2,27,30,7,25,6,22,47,5,26,50,45,11,46,16,3,17,13,24,12,40,18,41,43,14,44,32,9,29,33,49,15,8,48,4,37,10,31	71,78
eil76-LP	67,83	49	0,62,15,2,43,31,8,38,71,57,9,37,64,10,52,13,18,34,7,45,33,51,26,12,56,14,36,19,69,59,70,68,35,46,20,73,29,1,67,74,75,66,25,11,39,16,50,32,72	59,64
eil76-HP	415	67	0,42,41,40,55,22,48,23,2,43,31,49,17,54,24,8,38,71,57,9,37,64,65,10,58,13,18,34,7,45,33,51,26,12,53,56,14,36,19,69,59,70,68,35,46,20,60,21,61,27,73,29,1,5,67,3,74,75,25,11,39,16,50,15,62,32,72	131,91
eil101-LP	257,35	74	0,68,30,87,6,81,47,46,18,10,61,9,89,31,19,50,8,70,34,33,77,80,32,78,2,76,67,79,28,23,54,24,38,22,55,74,71,73,21,40,56,14,42,41,86,96,91,97,36,99,84,90,43,85,15,60,83,4,59,82,17,88,5,95,94,93,12,57,39,52,100,27,75,49	97,12
eil101-HP	262,4	73	0,68,30,9,61,10,18,46,47,81,6,17,82,59,4,83,60,15,85,43,90,84,99,36,97,91,58,98,95,5,88,100,52,39,57,12,93,94,96,86,41,42,14,56,1,72,20,71,73,21,74,55,22,38,24,54,23,28,79,67,75,76,2,78,77,33,34,70,8,50,80,32,49	176,72
eil101-HP*	266,55	70	0,68,30,9,61,10,18,46,47,81,6,17,82,59,4,83,60,15,85,43,90,84,99,97,36,91,58,98,95,5,93,12,94,96,86,41,42,14,56,40,21,22,38,55,74,73,71,72,20,39,57,52,100,27,75,76,2,67,79,23,28,77,33,34,70,8,50,80,32,49	297,01

In the eil101-HP instance, the CPU time for the best objective is close to 180 seconds, which is the maximum time limit. Thus, the algorithm is run for this instance by changing the maximum time limit to 540 seconds, and the patience to 120 seconds and the results are given in the table as the eil101-HP* instance. The best objective value is increased by only 1.58% and hence, the time limit and the patience are not increased further. The number of customers in the optimal solution for the eil101-HP is lower than that of the eil101-LP, which suggests that the solution found for the eil101-HP may not be the optimal solution.

The CPU time increases as the total number of customers to be visited increases. Also, in all of the cases, CPU time increases as the customers profit levels change from low to high. Similarly, the total number of

customers visited increases in the high profit case compared to low profit case in almost all of the instances, which can be expected.

6. Conclusion & Further Remarks

In all six benchmark instances the algorithm has found profitable results. A reliable comment can only be made after the global optimum of the instances are found via commercial mathematical optimization solvers or after approximation algorithms are utilized to get bound on the optimal value. However, it is certain that the model proposed could be improved further. As explained in [1], different operators such as swap and exchange could be added for more diversification.(i.e., k_max could be increased.) Similarly, 3-opt could also be used as a local search operator to search in diverse neighborhoods. The algorithm proposed for this problem utilized different neighborhoods in the shaking phase to better escape from local optima. Still, we believe, the efficiency of the algorithm could be improved by incorporating machine learning methods to make wiser search in the neighborhoods. Overall, the process of working on this problem was quite informative and interesting for both of us.

7. References

[1]*Dasari, K. V., Pandiri, V., & Singh, A. (2021). Multi-start heuristics for the profitable tour problem. Swarm and Evolutionary Computation, 64, 100897. https://doi.org/10.1016/j.swevo.2021.100897*