The application of the lazy constraint to solving the symmetric TSP problem

LI Zikang 20824907

Abstract

This project presents an overview of the lazy constraint and its application to the TSP problem. At first, I will give the description of the symmetric TSP and its most commonly formulation. After introducing the lazy constraints, I will use the standard TSP instances to test the effectiveness of lazy formulations. Then I will bring another formulation, MTZ formulation, in the symmetric TSP and explore its efficiency. Finally, I will design experiments to compare

1 Introduction

TSP problem was first mentioned by Hamilton, where the problem is to find the shortest Hamitonian cycle. Since then, solving all kinds of TSP problems has become a heated topic of research and with the increase of calculation power and the development of algorithms, some large TSP problems once considered intractable now can be solved almost optimally.[3]

Under this background, I am going to explore how and why the lazy constraint will be helpful in solving TSPs. Furthermore, I will use some TSP standard datasets to test the methods with Gurobi.

I will compare it with another TSP formulations using u_i variables.

This project is organized as follows. In section 2, I will give the definition of the symmetric TSP problem, which have received a lot of attention in past decades. In section 3, I will discuss the important role of the lazy constraint in many fields of operation research probelms. And how this method can be implemented with user cut in the symmetric TSPs. In Section 5, I will give the corresponding results with the implementation of the lazy constraint and user cut. Finally, in Section 4, I will present to use another formulation as the subtour-elimination constraints and give the results.

2 Problem Description

TSP problem can be defined as follows: for a given list of cities and the distances between each pair of them, one would like to find the shortest possible route that goes to each city once and returns to the origin city. In this project, I only consider the symmetric TSPs which assume that the distance of going from city i to city j is the same as going from city j to city i. Besides, I use two-dimentions Euclidean distances. But one thing to mention is that the TSP model formulation is independent of the way in which the distances are determined.

2.1 Symmetric TSP Formulation

There are multiple ways to formulate symmetric TSP as an integer programming problem, I will adopt the most frequently used. At first, there is a set of locations N. Take d_{ij} as the distance from location $i \in N$ to location $j \in N$. We also need to define the decision variables $x_{ij} \in \{0, 1\}$, this variable is equal to 1 if we decide to connect location i with location j. Otherwise, the decision variable is equal to 0. Then we can have the following integer programming program:

$$\min \sum_{i \in N} \sum_{\substack{j \in N \\ j<i}} d_{ij} x_{ij} \tag{1}$$

$$\text{s.t.} \sum_{j \in N} x_{ij} = 2, \forall i \in N \tag{2}$$

$$x_{ij} = x_{ji}, \forall i \in N, \forall j \in N \tag{3}$$

$$x_{ii} = 0, \forall i \in N \tag{4}$$

$$\sum_{i \in S} \sum_{\substack{j \in S \\ j<i}} x_{ij} \leq |S| - 1, \forall S \subset N, S \neq \emptyset \tag{5}$$

$$x_{ij} \in \{0, 1\}, \forall i \in N, \forall j \in N \tag{6}$$

The objective function (1) is to minimise the total distance of a route. A route is a sequence of locations where the person visits each one only once and returns to the starting location. Constraints (2) ensure that every node is connected to exactly two nodes, and constraints (3) make the variables symmetric, ensuring that the nodes i and j are connected. Constraints (4) restrict nodes connecting to themselves, and constraints (5) are to prevent disconnected sub-tours from appearing in the solution, which are also known as the subtour elimination constraints.

In general, if the number of cities of the TSP is n, then the possible number of routes is $n!$. Since there are an exponential number of constraints 2^{n-1} to eliminate sub-tours, we use lazy constraints to dynamically eliminate those tours.

3 Lazy Constraint Fashion

Note that constraints 5 contain exponential constraints, thus a more efficient way to solve the TSP is to start without constraints 5, and each time a new integer solution is found, inspect it for sub-tours. If a sub-tour is found, add the relevant constraint to the starting problem, and continue to search another solution.

As we know, the lazy constraint is used to add the constraints when we got an integer solution, and user-cut is used to cut the LP solution.

This method is very similar to row generation, the key point is how to find the core constructure of the problem and put the tedious part into the lazy constraint pool.

4 Miller-Tucker-Zemlin Constraints

The presented formulation is obviously not the only way of solving the TSP, there are other formulations that do not have an exponential number of constraints. For example, the following constraints (11) are time-based, which means that z_i indicates the ordering of node i. In fact, the set of constraints is a variation of MTZ constraints, due to the authors who first proposed them.[2]

However, this formulation requires the addition of auxiliary variables to enforce the sequential connection of the tour. And

$$\min \sum_{i \in N} \sum_{j \in N} d_{ij} x_{ij} \tag{7}$$

$$\text{s.t.} \sum_{j \in N} x_{ij} = 1, \forall i \in N \tag{8}$$

$$\sum_{j \in N} x_{ji} = 1, \forall i \in N \tag{9}$$

$$x_{ii} = 0, \forall i \in N \tag{10}$$

$$z_j \geq z_i + d_{ij} - M\left(1 - x_{ij}\right), \forall i \in N, \forall j \in N \backslash \{0\} \tag{11}$$

$$x_{ij} \in \{0, 1\}, z_i \geq 0, \forall i \in N, \forall j \in N \tag{12}$$

5 Result

In this section, I present the result of applying lazy constraints fashion to the symmetric TSP. I conduct experiments using a Windows 10 PC with an Intel Core i9-10900 running at 2.80GHz and 64 RAM. All the experiments are implemented with Python3.9 and the solver Gurobi9.1

For the initial experiments, I designed a method to read a benchmark file from `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/`.

Then I use the standard datasets to test the correctness of the implementation of the lazy constraint. Just use two sets of data as examples.

For berlin52.tsp, we have the optimal value 7585, the optimal route is [0, 48, 31, 44, 18, 40, 7, 8, 9, 42, 32, 50, 10, 12, 13, 46, 25, 26, 27, 11, 24, 3, 5, 14, 4, 23, 47, 37, 36, 39, 38, 35, 34, 33, 43, 45, 15, 28, 49, 19, 22, 29, 51, 1, 6, 41, 20, 16, 2, 17, 30, 21, 0]. Although the result is different from the optimal solution presented on the website, I use MTZ formulation to get the same results as the lazy formulation. And I use the given optimal tour to calculate the minimal cost which is larger than my optimal value. Maybe the original data, berlin52.tsp, has to be tranformed in some way.

But for att48.tsp, we have the same result under these two formulations as the optimal solution.

Besides, the two formulations will always give the same result. Thus, I have reasons to believe the lazy formulations will give the optimal solution under the simplest assumption.(Two-dimentions Euclidean Distance)

Then, I develop random 10 TSP instances, each with 30,40,50 nodes, respectively. The points are uniformly selected from the square area $([0, 100], [0, 100])$ and the distances between locations are euclidean. Set the time threshold to be 3600s and the times in the following tables are the average of ten instances.

The time consumed by two formulations is showed below.

Formulation-node number	Time (s)	Max Time(s)	Min Time (s)
Lazy-30	0.073	0.1	0.02
MTZ-30	3.84	17.96	0.31
Lazy-40	0.09	0.19	0.04
MTZ-40	23.02	121.52	1.26
Lazy-50	0.22	0.39	0.12
MTZ-50	958.96	3600	2.55
Lazy-100	2.25	5.24	0.92

One thing to mention is that though the number of node is the same, the time spent by MTZ is highly dependent on the given initial node position. For example, for the case of MTZ-

40, the largest time can be 121.52 and the shortest time is 1.26. While the lazy constraint is more stable.

The third experiment is that I set some rooted and regional TSP instances to see if the cluster points distribution will affect the complexity of TSP. The center points are randomly generated in the square area $([10, 90], [10, 90])$, then for each center points (x, y), generate several nodes in the region $(x \pm 10, y \pm 10)$.

Formulation-(center node number,total node number)	Time (s)	Max Time(s)	Min Time (s)
Lazy-(5,30)	0.068	0.13	0.03
MTZ-(5,30)	3.84	17.96	0.31
Lazy-(5,40)	0.21	0.53	0.05
MTZ-(5,40)		3600	112.91
Lazy-(5,50)	0.52	1.12	0.22
Lazy-(5,100)	2.70	3.65	1.97

The results show that the cluster distribution significantly increases the complexity of TSP. This phenomenon can be explained by the fact that the cluster points are easy to form a cycle, which will increase the possibility of forming sub-tours.

Through the experiments, we can see that the complexity of MTZ is still high despite the polynomial constraints in (11). Could we embed the similar idea of lazy constraints in the MTZ formulation? This reference gives us the answer and the corresponding result. In this way, the MTZ formulation will be transformed into a Benders decomposition. Although the Benders formulation will shorten the calculation time, the efficiency is still low comparing with the lazy formulation. The following results can be found here [3].

Formulation	Time (s)	Nodes	Lazy Constraints
Lazy	0.46	1135	107
MTZ	142.17	145209	–
Benders	115.23	42690	2554

Hence, I think that is why Gurobi will take the Lazy constraint as an example to solve the TSP.[1] This method is efficient indeed.

5.1 Summary

1. efficiency between lazy and MTZ clearly MTZ has too many constraint because it is a symmetric problem add too much unrelevant variables.

2. cluster points, One intuition is that once cluster points will give more chance to forming a sub-tour.

References

[1] Gurobi documents. `https://www.gurobi.com/documentation/`.

[2] Clair E Miller, Albert W Tucker, and Richard A Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960.

[3] Robin Harris Pearce. Towards a general formulation of lazy constraints. 2019.