Facultat de Matemàtiques i Estadística (FME)
Màster en Estadística i Investigació Operativa

Models i Mètodes de la Investigació Operativa

Final Project
Solving the Traveling Salesman Problem (TSP)

Authors: Leonel Fernando Nabaza Ruibal

Professor: Dr. Maria Paz Linares Herreros

Barcelona, January 11, 2026

Resum

Aquest projecte aborda la resolució del Problema del Viatjan de Comerç (TSP) per a una instància generada aleatòriament de 15 ciutats, utilitzant el llenguatge de modelització AMPL i el solver CPLEX. L'estudi es divideix en dues fases principals: l'establiment d'una fita superior i la determinació d'una fita inferior ajustada. En primer lloc, es va implementar una heurística constructiva (Veí Més Proper) seguida d'un algorisme de cerca local (2-Opt), aconseguint una solució factible amb un cost de 305. Posteriorment, es va analitzar la relaxació lineal del problema afegint progressivament desigualtats vàlides, incloent-hi restriccions automàtiques d'eliminació de subtours, talls de connectivitat i talls de Gomory manuals. Aquest procediment de plans de tall va elevar la fita inferior d'un valor inicial de 261 a 295.5, deixant un marge inferior a l'1% respecte a la solució òptima entera (sense el·liminació de subtours) de 297 trobada mitjançant Branch and Bound.

Abstract

This project addresses the resolution of the Traveling Salesman Problem (TSP) for a randomly generated instance of 15 cities using the AMPL modeling language and the CPLEX solver. The study is divided into two main phases: the establishment of an upper bound and the determination of a tight lower bound. First, a constructive heuristic (Nearest Neighbor) followed by a local search algorithm (2-Opt) were implemented, achieving a feasible solution with a cost of 305. Subsequently, the linear relaxation of the problem was analyzed by progressively adding valid inequalities, including automatic subtour elimination constraints, connectivity cuts, and manual Gomory cuts. This cutting-plane procedure raised the lower bound from an initial value of 261 to 295.5, leaving a gap of less than 1% relative to the optimal integer solution (without subtour elimination constraints) of 297 found via Branch and Bound.

Contents

Chapter 1

Introduction

We are considering the solution of the symmetric version of the TSP over a complete graph $G = (V, E)$ with $n = |V|$ nodes. The solution approach will be through complementary techniques applied to different formulations. On one hand, we will obtain upper bounds through developing a heuristic method. On the other hand, we will obtain lower bounds associated to problem relaxations.

To achieve these objectives, the work is structured into specific methodological stages. First, to establish a feasible Upper Bound, we implement a constructive heuristic (Nearest Neighbor) followed by a local search improvement algorithm (2-Opt). Subsequently, to determine a tight Lower Bound, we analyze the Linear Programming relaxation of the model. We progressively strengthen this relaxation by iteratively adding valid inequalities, ranging from automatic Subtour Elimination Constraints (SECs) to specific Connectivity and Gomory cuts. Finally, the quality of these bounds is evaluated by comparing them against the exact optimal solution obtained via the Branch and Bound algorithm.

Chapter 2

Data Generation

For this problem a symmetric matrix of positive costs was generated by the means of a Python script, named `generate_data.py`. In my problem I have decided to use 15 nodes.

Specifically, the script assigns random (x, y) coordinates to each node within a 100×100 grid. The costs represent the Euclidean distance between these points, rounded to the nearest integer.

In my execution, it generated the following file `tsp.dat`:

```
data;

set NODES := 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15;

param dist : 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 :=
1   0   9   88  10  19  98  102 31  59  55  89  16  33  82  39
2   9   0   96  5   29  107 108 40  59  55  97  10  42  88  48
3   88  96  0   94  71  13  37  57  84  85  7   97  64  44  61
4   10  5   94  0   29  104 105 39  54  50  95  6   43  84  49
5   19  29  71  29  0   82  89  14  62  60  73  34  14  73  20
6   98  107 13  104 82  0   28  68  87  89  9   107 76  42  73
7   102 108 37  105 89  28  0   75  74  76  30  106 88  24  86
8   31  40  57  39  14  68  75  0   58  56  59  43  16  60  19
9   59  59  84  54  62  87  74  58  0   4   81  51  73  50  77
10  55  55  85  50  60  89  76  56  4   0   82  47  71  52  75
11  89  97  7   95  73  9   30  59  81  82  0   98  67  39  64
12  16  10  97  6   34  107 106 43  51  47  98  0   48  84  54
13  33  42  64  43  14  76  88  16  73  71  67  48  0   75  6
14  82  88  44  84  73  42  24  60  50  52  39  84  75  0   75
```

```
15  39  48  61  49  20  73  86  19  77  75  64  54  6   75  0
```

As for the vocabulary of the problem, I have referred both the nodes as nodes or cities (such as in the original framework of the problem) and the edges to the cost related to traveling among the cities.

Chapter 3

Upper Bound

Since the Traveling Salesman Problem is NP-hard, obtaining a high-quality initial feasible solution is important for two reasons: it provides a valid tour for the salesman immediately, and it establishes an upper bound for the optimal cost. This limits the search space for the application of other algorithms.

For this project, we implemented a two-stage heuristic procedure consisting of a **Constructive Phase** followed by an **Improvement Phase**.

3.1 Phase 1: Constructive Heuristic (Nearest Neighbor)

3.1.1 Explanation

The first step employs the **Nearest Neighbor (NN)** algorithm. This is a 'greedy' strategy that builds the tour step-by-step. The logic is straightforward:

1. Start at an arbitrary node (in our implementation, Node 1).
2. From the current city, identify the closest city that has not yet been visited.
3. Travel to that city and mark it as visited.
4. Repeat the process until all N cities have been visited.
5. Finally, return to the starting city to complete the cycle.

Mathematically, given a current node i and a set of unvisited nodes U, the algorithm selects the next node j such that:

$$j = \text{argmin}_{k \in U}\{c_{ik}\}$$

3.1.2 Comments on Behavior

The Nearest Neighbor algorithm is computationally efficient (very fast). However, its greedy nature is 'short-sighted'. It makes optimal local decisions early in the process but often paints itself into a corner. As the list of unvisited nodes shrinks, the algorithm is forced to connect the current node to whatever is left, regardless of the distance. This typically results in the final few edges of the tour being disproportionately long, often traversing the entire map to close the loop.

3.1.3 AMPL Implementation

The following AMPL code implements this logic using a 'visited' array to track progress:

```
# --- 1. Constructive Phase: Nearest Neighbor ---
let tour[1] := 1;
let visited[1] := 1;
let current_node := 1;

for {step in 2..n} {
    let min_dist := Infinity;
    let next_node := 0;
    # Scan for the closest unvisited neighbor
    for {j in NODES} {
        if visited[j] = 0 and dist[current_node, j] < min_dist then {
            let min_dist := dist[current_node, j];
            let next_node := j;
        }
    }
    let tour[step] := next_node;
    let visited[next_node] := 1;
    let current_node := next_node;
}
# Close the loop
let tour[n+1] := 1;
```

3.1.4 Obtained Results (Phase 1)

Upon executing the constructive phase, we obtained the following initial feasible solution:

Constructive Heuristic (NN) Cost: 328
Path sequence: $1 \to 2 \to 4 \to 12 \to 5 \to 8 \to 13 \to 15 \to 3 \to 11 \to 6 \to 7 \to 14 \to 9 \to 10 \to 1$

The cost of **328** serves as our initial Upper Bound.

3.2 Phase 2: Improvement Heuristic (2-Opt Local Search)

3.2.1 Explanation

To mitigate the suboptimal decisions made by the greedy algorithm, we applied a 2-Opt Local Search. This is an iterative improvement algorithm that takes an existing tour and tries to improve it by untangling 'crossings'.

The algorithm examines every pair of non-adjacent edges in the tour, say $(i, i+1)$ and $(j, j+1)$. It checks if the total distance would decrease by removing these two edges and reconnecting the path via (i, j) and $(i+1, j+1)$. This operation effectively reverses the segment of the tour between nodes $i+1$ and j.

The swap condition is checked as follows:

$$\text{dist}(i, j) + \text{dist}(i+1, j+1) < \text{dist}(i, i+1) + \text{dist}(j, j+1)$$

If the inequality holds, the swap is performed, and the improvement flag is set to true. The process repeats until no further pairwise swaps can reduce the cost (reaching a 2-optimal local minimum).

3.2.2 Comments on Behavior

The 2-Opt algorithm is highly effective at eliminating the 'long edges' created at the end of the Nearest Neighbor phase. Geometrically, on a Euclidean plane, the triangle inequality implies that paths should not cross. 2-Opt systematically identifies and removes these crossings. While it is slower than the constructive phase (as it requires multiple passes over the edges), the improvement in solution quality is usually significant.

3.2.3 AMPL Implementation

The implementation uses a nested loop to check all pairs and performs the reversal if a better configuration is found:

```
# --- 2. Improvement Phase: 2-Opt Local Search ---
let improved := 1;
# Repeat until no more improvements can be found
repeat while improved > 0 {
    let improved := 0;
    for {i in 1..n-2} {
        for {j in i+2..n} {
             # Check swap condition (Triangle Inequality logic)
             if dist[tour[i], tour[j]] + dist[tour[i+1], tour[j+1]] <
                dist[tour[i], tour[i+1]] + dist[tour[j], tour[j+1]] then {

                # Perform the swap (reverse sub-segment)
                for {k in 0..floor((j - (i+1))/2)} {
                    let tmp := tour[i+1+k];
                    let tour[i+1+k] := tour[j-k];
                    let tour[j-k] := tmp;
                }
                let improved := 1;
                break;
            }
        }
        if improved = 1 then break;
    }
}
```

3.2.4 Obtained Results (Phase 2)

After applying the local search, the tour was optimized, resulting in a significant cost reduction.

> **Improved Heuristic (2-Opt) Cost: 305**
> Path sequence: $1 \to 5 \to 8 \to 13 \to 15 \to 3 \to 11 \to 6 \to 7 \to 14 \to 9 \to 10 \to 12 \to 4 \to 2 \to 1$

The procedure successfully lowered the Upper Bound from **328** to **305**, an improvement of approximately **7**%. This tighter upper bound provides a much stronger starting point for the subsequent relaxation and Branch and Bound steps.

Chapter 4

Lower Bound

To further narrow down the solution space of our problem, it is not only necessary to establish an upper bound, but lower bound as well. In this part, this is achieved by solving the Linear Programming (LP) relaxation (LR) of the TSP model (relaxing the integrality constraints $x_{ij} \in \{0, 1\}$ to $0 \leq x_{ij} \leq 1$) and progressively adding valid inequalities to cut off fractional solutions.

4.1 Evolution of Inequalities and Lower Bounds

The process was carried out in four distinct stages. Below is a detailed explanation of the inequalities obtained and the methods used to identify them.

4.1.1 Initial Relaxation and Automatic Subtour Elimination (Sections a & b)

I started by solving the degree-constrained assignment problem (only degree constraints, no subtour elimination). Since the number of possible subtours is exponential (2^n), we cannot add all SECs at once. Instead, I used an Automatic Separation Algorithm. This algorithm solves the relaxed problem, builds a graph using edges with $x_{ij} > 0$, and runs a connected components search (Flood-fill). If the graph is disconnected, the algorithm identifies the connected component S and adds the violation cut:

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1$$

Basically, the implementation works by inspecting the solution provided by the solver to see which edges are currently active (those with a value greater than a small tolerance like 0.001).

To detect the subtours, the script uses a loop that picks a starting node and tries to find all the other nodes connected to it, grouping them into a component. If the size of this group is smaller than the total number of cities (N), it means we have found an isolated cycle. Once this is detected, the code automatically creates a new constraint for this specific group of nodes and adds it to the model, forcing the solver to connect this isolated component with the rest of the graph in the next iteration.

Results:

- **Initial Relaxation:** The solver produced a solution with a cost of **261.0**.
- **Iteration 1:** The separation algorithm detected a large subtour consisting of 11 nodes. It automatically added the constraint:

$$\sum_{i,j \in S_{11}} x_{ij} \leq 10$$

- **Iteration 2:** After re-solving, the graph became connected. The Lower Bound improved significantly to **273.5**.

The solution proposed by the solver at this stage (after subtour elimination) is:

```
(1, 2)=0.50,  (1, 4)=0.50,  (1, 5)=1.00,  (2, 4)=0.50,
(2, 12)=1.00, (3, 6)=1.00,  (3, 11)=1.00, (4, 12)=1.00,
(5, 13)=1.00, (6, 7)=0.50,  (6, 11)=0.50, (7, 11)=0.50,
(7, 14)=1.00, (8, 10)=1.00, (8, 15)=1.00, (9, 10)=1.00,
(9, 14)=1.00, (13, 15)=1.00
```

4.1.2 Connectivity Inequalities (Section c)

Although the graph was technically connected after Section (b), a visual inspection of the active edges revealed a structural weakness. The cluster of nodes $S = \{2, 4, 12\}$ formed a 'bottleneck'. The sum of flow on the edges connecting this cluster to the rest of the graph was exactly 1.0 ($x_{1,2} = 0.5$ and $x_{1,4} = 0.5$). For a TSP tour to enter and leave a set of nodes, the cut capacity must be at least 2. I applied a Cut-Set Inequality (also known as a Connectivity Constraint), which is stronger than a standard SEC in this context as it enforces interaction with the outside:

$$\sum_{i \in S, j \notin S} x_{ij} \geq 2$$

Results: I added the constraint $\sum_{i \in \{2,4,12\}, j \notin \{2,4,12\}} x_{ij} \geq 2$. Re-solving the LP raised the Lower Bound to **279.0**.

The solution proposed by the solver after adding this connectivity cut is:

```
(1, 2)=1.00,  (1, 5)=1.00,  (2, 4)=0.50,  (2, 12)=0.50,
(3, 6)=1.00,  (3, 11)=1.00, (4, 5)=0.50,  (4, 12)=1.00,
(5, 13)=0.50, (6, 7)=0.50,  (6, 11)=0.50, (7, 11)=0.50,
(7, 14)=1.00, (8, 10)=0.50, (8, 13)=0.50, (8, 15)=1.00,
(9, 10)=1.00, (9, 14)=1.00, (10, 12)=0.50, (13, 15)=1.00
```

4.1.3 Gomory Cuts (Section d)

With the graph connected and standard structural constraints satisfied, I looked for fractional solutions that violated integer logic using the Gomory cuts. I inspected the output for paths where the sum of variables was fractional (e.g., $k.5$). Since the sum of binary variables must be an integer, any sum $\leq k.5$ implies a valid cut $\leq k$. If we want to phrase it such as the formulation we used in the course for Gomory cuts, what we are doing here is adding a new constraint, say for instance $x_1 + x_2 \leq k.5$ (although we know it holds as an equality in the solutions) and then apply the fact that (this comes from the notes):

$$\sum_{j=1}^{n} \lfloor ua_j \rfloor x_j \leq \lfloor ub \rfloor$$

with $u = (0 \ldots 01)$ and $b = k.5$.

Results:

- **Cut 1:** I targeted the fractional path involving nodes $\{2, 4, 5\}$ and $\{2, 12\}$ which summed to 1.5. I added $x_{2,4} + x_{2,12} + x_{4,5} \leq 1$. Interestingly, the objective value remained at **279.0**, indicating alternative fractional solutions existed at the same cost.

  The solution proposed after this first cut (still at cost 279.0) is:

  ```
  (1, 2)=1.00,  (1, 5)=1.00,  (2, 4)=1.00,  (3, 6)=1.00,
  (3, 11)=1.00, (4, 12)=1.00, (5, 12)=0.50, (5, 13)=0.50,
  (6, 7)=0.50,  (6, 11)=0.50, (7, 11)=0.50, (7, 14)=1.00,
  (8, 10)=0.50, (8, 13)=0.50, (8, 15)=1.00, (9, 10)=1.00,
  (9, 14)=1.00, (10, 12)=0.50, (13, 15)=1.00
  ```

- **Cut 2:** I identified a second violation in the triangle $\{6, 7, 11\}$ and its neighbors. The sum of the edges was fractional, allowing us to impose:

  $$x_{6,7} + x_{6,11} + x_{7,11} \leq 1$$

This cut was decisive. It forced the solver to abandon the cheap fractional structure, raising the Lower Bound dramatically to **295.5**.

The solution proposed after this second cut is:

```
(1, 2)=1.00,  (1, 5)=1.00,  (2, 4)=1.00,  (3, 6)=1.00,
(3, 11)=1.00, (4, 12)=1.00, (5, 12)=0.50, (5, 13)=0.50,
(6, 7)=0.50,  (6, 11)=0.50, (7, 9)=0.50,  (7, 14)=1.00,
(8, 10)=0.50, (8, 13)=0.50, (8, 15)=1.00, (9, 10)=1.00,
(9, 14)=0.50, (10, 12)=0.50, (11, 14)=0.50, (13, 15)=1.00
```

4.2 Summary of Linear Relaxation Solutions

The following table summarizes the sequence of Lower Bounds obtained throughout the procedure.

Step	Constraint Type	LB Value	Improvement	% comparison B&B
(a)	Degree Constraints Only	261.0	-	87.8%
(b)	Subtour Elimination (Auto)	273.5	+12.5	92.1%
(c)	Cut-Set (Connectivity)	279.0	+5.5	93.9%
(d.1)	Gomory Cut 1	279.0	+0.0	93.9%
(d.2)	Gomory Cut 2	**295.5**	+16.5	**99.5%**

Table 4.1: Sequence of Lower Bounds obtained via progressive cuts. Last column refers to the ratio between the current bound and the Branch and Bound procedure

4.3 Evaluation of Quality and Optimality Interval

Finally, I executed the **Branch and Bound** algorithm (Section e) to restore integrality. The solver determined the Optimal (without explicit SEC) Solution to be 297.

```
Edge (i,j)   | Cost
------------------
   ( 1,  2)  |    9
   ( 1,  5)  |   19
   ( 2,  4)  |    5
   ( 3,  6)  |   13
```

```
   ( 3, 11)  |    7
   ( 4, 12)  |    6
   ( 5, 12)  |   34
   ( 6, 11)  |    9
   ( 7, 10)  |   76
   ( 7, 14)  |   24
   ( 8, 13)  |   16
   ( 8, 15)  |   19
   ( 9, 10)  |    4
   ( 9, 14)  |   50
   (13, 15)  |    6
------------------
TOTAL COST:      297
```

4.3.1 Quality of Lower Bounds

The applied procedure was highly effective.

1. The initial relaxation (261) had a gap of **12.1%** from the optimal.
2. After applying our specific cuts (SECs, Connectivity, and Gomory), the Lower Bound reached **295.5**.
3. The final gap prior to branching was merely $297 - 295.5 = \mathbf{1.5}$ units (or **0.5%**).

This indicates that the combination of manual and automatic cuts was strong enough to eliminate almost all invalid fractional solutions, bringing the lower bound very close to the upper bound. The Branch and Bound algorithm had very little work to do (only bridging the last 1.5 units of cost), proving that the cutting plane method is a powerful tool for this instance.

4.3.2 Optimality Interval

Based on our Heuristic Upper Bound (from the previous chapter) and our best Linear Relaxation Lower Bound, we established the following optimality interval inside which an exact solution would lie:

$$\text{LB} \leq z^* \leq \text{UB}$$

$$297 \leq z^* \leq 305$$

Chapter 5

Conclusions

The development of this project has allowed for a comprehensive analysis of the Traveling Salesman Problem, illustrating the duality between heuristic approximation and exact optimization methods. The initial application of the Nearest Neighbor heuristic, while computationally efficient, highlighted the limitations of greedy strategies, producing a tour that was significantly improved by the subsequent 2-Opt local search. This improvement phase was critical not only for obtaining a better valid solution for the salesman but also for establishing a tighter upper bound that restricts the search space for exact algorithms.

Regarding the lower bound analysis, the progressive addition of valid inequalities proved to be the most decisive factor in proving optimality. The evolution from the initial degree-constrained relaxation to the final tight bound demonstrated that standard subtour elimination constraints, while necessary, are often insufficient on their own to close the optimality gap. The manual identification and application of connectivity and Gomory cuts allowed us to refine the feasible region of the relaxation to an extent where it almost perfectly matched the integer solution of the Branch and Bound without SEC. Specifically, the jump in the lower bound observed after applying the Gomory cuts highlights the importance of integer rounding principles in tightening relaxations. Ultimately, the fact that the Branch and Bound algorithm only needed to bridge a negligible gap of 0.5% validates the robustness of the cutting-plane procedure applied, confirming that a strong formulation is the key to solving combinatorial problems efficiently.

Chapter 6

Appendix

The scripts and updated content can be found at https://github.com/LeonelFNR/MMIO_Traveling_Salesman_Problem_Project.