DBA5103 Operations Research and Analytics

Integrality Gap Analysis of TSP Formulations

Name	Student ID
Tseng, Yung-Yi	A0318929M
Cho Eunsoo (Ashlyn)	A0246014R
Mark Hansen Nii Darku Dodoo	A0319136E
Zhao Qiya	A0331091R
Wang Ying	A0318271H

National University of Singapore

Code repository

Interactive results dashboard

Submission Date: November 23, 2025

Abstract

This project investigates the integrality gap in the Traveling Salesman Problem to quantify the cost of convexity associated with linear programming relaxations. We perform a two-dimensional analysis that examines the interaction between three mathematical formulations of TSP (Miller-Tucker-Zemlin, Gavish-Graves, and Dantzig-Fulkerson-Johnson), along with the Assignment Problem as a baseline formulation, and four distinct distance matrix structures. By solving 120 instances ranging from random to highly structured networks, we assess how formulation tightness and data characteristics jointly determine the quality of the lower bound. Our findings confirm that while the Dantzig-Fulkerson-Johnson formulation consistently provides near-optimal bounds, weaker models such as the Miller-Tucker-Zemlin formulations exhibit large integrality gaps that are highly sensitive to data structure, particularly in clustered environments. These results offer a diagnostic framework for selecting appropriate solution methods based on network topology.

1 Introduction

1.1 The Convexity Challenge in Discrete Optimization

A fundamental challenge in operations research lies in the structural distinction between convex and non-convex problems. Linear Programming (LP) problems are convex, allowing for efficient polynomial time solution methods. In contrast, Integer Programming (IP) problems such as the Traveling Salesman Problem (TSP) are inherently non-convex due to the discrete nature of their decision spaces. While convex problems typically enjoy strong duality, non-convex IPs exhibit a structural disconnect where the dual of the dual is equivalent not to the original problem but to the convexified version of the primal problem.

1.2 The Cost of Convexity

This convexification results in an integrality gap, which is the difference between the optimal integer solution (z_{IP}) and the optimal solution to the problem's LP relaxation (z_{LP}). This gap represents the "Cost of Convexity," quantifying the degree of optimality sacrificed or the estimation error incurred when strict integer constraints are relaxed to leverage the computational speed of LPs. Understanding this cost is critical because for large-scale combinatorial problems, solving for the true z_{IP} is often computationally intractable. If the convexified solution is a sufficiently close approximation, practitioners can rely on LP relaxations for efficient decision-making. Conversely, a large gap necessitates the use of expensive exact methods.

1.3 The Research Gap: A Two-Dimensional Problem

While existing literature has established that different IP formulations yield different LP relaxation qualities, and separately that distance matrix variance affects TSP difficulty, no prior

work has systematically examined how these two factors interact. This project addresses this gap through a comprehensive two-dimensional analysis. We posit that the "Cost of Convexity" is not static. Instead, we hypothesise that it is determined jointly by the formulation choice and the underlying distance structure. Specifically, we investigate whether weaker formulations amplify the integrality gap when applied to high-variance cost structures, while determining if tighter formulations provide robustness against such data variation.

1.4 Research Objective

This study aims to quantify the integrality gap across three distinct TSP formulations (MTZ, Gavish-Graves, and DFJ), along with the Assignment Problem as a baseline, when applied to four distinct distance structures (Grid, Random, Clustered, and Hub-and-Spoke). By isolating the interaction effects between model tightness and data structure, this research provides a diagnostic framework to guide practitioners in selecting the appropriate modeling approach based on their specific network characteristics.

2 Literature Review: Formulation Tightness and the Cost of Convexity

2.1 The Hierarchy of Formulations

This section establishes the theoretical framework for comparing formulation quality based on polyhedral theory. To rigorously assess the "cost of convexity" across different models, we must first define the criteria for evaluating formulation strength and introduce the four formulations under study.

The standard metric for comparing integer programming formulations is the tightness of their linear programming (LP) relaxations. This concept relies on polyhedral projection as established analytically by Padberg and Sung[1]. Consider two different formulations, F_1 and F_2, that model the same underlying integer problem. Let P_{F1} and P_{F2} denote the feasible regions (polyhedra) of their respective LP relaxations. Formulation F_1 is defined as stronger than F_2 if the polyhedron P_{F1} is strictly contained within P_{F2} when projected onto the space of the primary decision variables x_{ij}.

This geometric relationship has direct implications for the integrality gap. A tighter polyhedron means the feasible region of the relaxation is smaller and adheres more closely to the true integer solutions. Consequently, minimizing over a tighter set P_{F1} yields a solution value that is greater than or equal to the solution found minimizing over the looser set P_{F2}. In the context of a minimization problem like the TSP, a stronger formulation provides a higher and higher-quality lower bound (z_{LP}).

Based on this framework, this study analyzes four formulations that span the spectrum of relaxation quality. These are the Assignment Problem (AP) formulation, the Miller-Tucker-

[1] Padberg, M. W., & Sung, T. H. (1991). An analytical comparison of different formulations for the traveling salesman problem. Mathematical Programming, 52(1), 315-357.

Zemlin (MTZ) formulation, the Gavish-Graves (GG) formulation, and the Dantzig-Fulkerson-Johnson (DFJ) formulation. As categorized and empirically verified by Öncan, Altınel, and Laporte[2], the LP relaxations of these models follow a strict hierarchy of strength defined by the following central inequality:

$$z_{LP}^{Assignment} \leq z_{LP}^{MTZ} \leq z_{LP}^{GG} \leq z_{LP}^{DFJ} \leq z_{IP}$$

In this hierarchy, the Assignment Problem formulation serves as the baseline and represents the weakest relaxation. It effectively ignores connectivity requirements and provides the loosest lower bound. Conversely, the DFJ formulation represents the theoretical strongest bound. Its constraints define facets of the TSP polytope that approximate the convex hull of integer solutions more precisely than the compact formulations of MTZ and GG. The following sections detail the mathematical structure of these models to explain the mechanics driving this hierarchy.

2.2 The Assignment Problem (AP) Formulation

This section establishes the baseline formulation against which all subsequent TSP models are measured. The Assignment Problem (AP) serves as the most relaxed version of the TSP, enforcing only the condition that every city is entered and left exactly once, while ignoring the connectivity requirement for a single continuous tour. The mathematical formulation of AP can be found in Appendix A, and the solver implementation is provided in Appendix J.

2.2.1 Rationale of AP

We include the Assignment Problem formulation not as a valid method for solving the TSP, but as a fundamental benchmark for analyzing the cost of convexity. Mathematically, the AP is a relaxation of the TSP obtained by removing all subtour elimination constraints. Since any valid TSP tour must satisfy the degree constraints (entering and leaving each city exactly once), the set of feasible TSP solutions is a subset of the feasible AP solutions. Therefore, the optimal objective value of the Assignment Problem (z_{IP}^{AP}) provides an absolute lower bound for the optimal value of any valid TSP formulation (z_{IP}^{TSP}):

$$z_{IP}^{AP} \leq z_{IP}^{TSP}$$

Furthermore, because the constraint matrix of the Assignment Problem is totally unimodular, its linear programming relaxation naturally yields integer solutions. This means that for the AP, there is no integrality gap ($z_{LP}^{AP} = z_{IP}^{AP}$). Consequently, the value z_{LP}^{AP} serves as the weakest possible lower bound for the TSP's LP relaxation.

2.2.2 Weakness and the Integrality Gap of AP

The critical weakness of this formulation is that its feasible region allows for disconnected subtours. For example, in a problem with 6 cities, a valid assignment solution could consist of

[2] Öncan, T., Altınel, İ. K., & Laporte, G. (2009). A comparative analysis of several asymmetric traveling salesman problem formulations. Computers & Operations Research, 36(3), 637-654.

two disjoint cycles rather than a single tour visiting all 6 cities.

Because these subtour solutions are often significantly "cheaper" (having a lower total cost) than a connected tour, the optimal value of the AP relaxation (z_{LP}^{AP}) is frequently much smaller than the true TSP optimum (z_{IP}^{TSP}). This discrepancy creates a large integrality gap. In the context of our study, this formulation represents the "highest cost of convexity" scenario where the relaxed model fails to capture the essential structural property of the problem (connectivity), leading to a poor quality bound. As discussed by Balas and Toth[3], while the AP can be solved efficiently in $O(n^3)$ time, its utility as a bounding procedure for the TSP is limited by this inherent structural weakness.

2.3 The Miller-Tucker-Zemlin (MTZ) Formulation

This section examines the Miller-Tucker-Zemlin (MTZ) model, which is widely recognized as the standard "compact" formulation for the Traveling Salesman Problem. While it offers significant computational advantages in terms of size, it also introduces structural weaknesses that affect the quality of its linear programming relaxation. The mathematical formulation of MTZ can be found in Appendix B.

2.3.1 Rationale of MTZ

The primary advantage of the MTZ formulation is its compactness. Unlike the theoretical ideal (the Dantzig-Fulkerson-Johnson formulation), which requires an exponential number of constraints, the MTZ formulation requires only $O(n^2)$ constraints. Specifically, there are $(n-1)(n-2)$ subtour elimination constraints. This polynomial size means the entire model can be explicitly written and passed to a standard MIP solver without the need for complex algorithms like delayed column generation or cutting planes. For this reason, it serves as a practical, "implementation-friendly" benchmark.

2.3.2 Weakness and the Integrality Gap of MTZ

Despite its utility for integer programming, the MTZ formulation is notoriously weak when relaxed to a linear program. While the logic $u_i - u_j + nx_{ij} \leq n-1$ successfully prevents integer subtours (where $x_{ij} = 1$), it is very "loose" in continuous space.

When x_{ij} is allowed to take fractional values (e.g., $x_{ij} = 0.5$), the constraint becomes easy to satisfy without forcing a meaningful relationship between the potentials u_i and u_j. For instance, if $n = 10$ and $x_{ij} = 0.5$, the constraint becomes $u_i - u_j \leq 4$, which is a very weak restriction given that u values can range from 1 to 9. This looseness allows the LP relaxation to find solutions that are essentially equivalent to the Assignment Problem solution, often containing fractional "subtours" that the integer constraints would forbid.

[3] Balas, E., & Toth, P. (1985). Branch and bound methods. In The Traveling Salesman Problem (pp. 361-401). John Wiley & Sons.

Sherali and Driscoll[4] formally analyzed this weakness, demonstrating that the projection of the MTZ polytope onto the x-space is not significantly tighter than the Assignment polytope. Consequently, the lower bound provided by the LP relaxation of the MTZ formulation (z_{LP}^{MTZ}) is often identical, or very close, to the weak bound of the Assignment Problem (z_{LP}^{AP}), resulting in a similarly large integrality gap.

2.4 The Gavish-Graves (GG) Formulation

This section introduces the single-commodity flow formulation proposed by Gavish and Graves[5], which represents a significant structural improvement over the MTZ model. By utilizing a physical flow analogy to enforce connectivity, this formulation achieves a tighter linear programming relaxation while maintaining a polynomial number of constraints. The mathematical formulation of GG can be found in Appendix C.

2.4.1 Rationale of GG

We include the Gavish-Graves formulation because it occupies a strategic "middle ground" in the hierarchy of TSP formulations. Unlike the MTZ model, which relies on abstract node potentials that are loosely coupled to the routing decisions, the GG model uses a tangible flow analogy. The physical requirement that flow must originate at the source (node 0) and reach every other node forces connectivity more strictly. For example, a disconnected subtour cannot exist in the GG formulation because such a loop would have no connection to the source node 0, making it impossible to satisfy the flow conservation constraints.

Consequently, the GG formulation offers a "sweet spot" for optimization. As empirically demonstrated by Öncan et al., it consistently provides a significantly better lower bound than the MTZ formulation ($z_{LP}^{MTZ} < z_{LP}^{GG}$), reducing the integrality gap without requiring the exponential number of constraints found in the Dantzig-Fulkerson-Johnson (DFJ) formulation.

2.4.2 Weakness and the Integrality Gap of GG

The primary trade-off of the GG formulation is its increased size in terms of decision variables. While it retains a polynomial number of constraints ($O(n^2)$), it requires the addition of $O(n^2)$ continuous flow variables (f_{ij}). In contrast, the MTZ formulation only adds $O(n)$ continuous variables (u_i). This increased dimensionality can make the linear programming relaxation of GG computationally heavier to solve per iteration compared to MTZ, despite the benefit of a tighter bound.

---

[4] Sherali, H. D., & Driscoll, P. J. (2002). On tightening the relaxations of Miller-Tucker-Zemlin formulations for the traveling salesman problem. Operations Research, 50(4), 656-669.

[5] Gavish, B., & Graves, S. C. (1978). The Travelling Salesman Problem and Related Problems. Working Paper, MIT.

2.5 The Dantzig-Fulkerson-Johnson (DFJ) Formulation

This section describes the theoretical "Gold Standard" for TSP formulations: the Dantzig-Fulkerson-Johnson (DFJ) model. By directly defining the convex hull of valid tours with subtour elimination constraints, this formulation provides the tightest possible linear programming relaxation. The mathematical formulation of DFJ can be found in Appendix D.

2.5.1 Rationale of DFJ

The DFJ formulation is theoretically superior because its constraints define facets of the TSP polytope. This means they cut off fractional solutions more effectively than the polynomial-sized constraints of MTZ or GG. Consequently, the optimal value of the LP relaxation for the DFJ formulation (z_{LP}^{DFJ}) is significantly higher (closer to the true integer optimum) than that of any other known formulation.

In fact, the LP relaxation value z_{LP}^{DFJ} is equivalent to the famous Held-Karp Lower Bound[6]. For symmetric TSPs, this bound is notoriously tight, often falling within 1% of the true optimal integer solution (z_{IP}). In our hierarchy of formulations, this represents the strongest possible convex approximation:

$$z_{LP}^{GG} \leq z_{LP}^{DFJ} \leq z_{IP}$$

2.5.2 Weakness and the Integrality Gap of DFJ

The primary weakness of the DFJ formulation is its exponential complexity. The number of subsets S is $2^n - 2$, which grows explosively. For a small problem with $n = 20$ cities, there are over a million constraints. For $n = 50$, the number of constraints exceeds the capacity of any computer to store, let alone solve.

Therefore, the DFJ formulation cannot be solved directly by writing down all constraints at once. Instead, it requires advanced solution methods such as Row Generation or Cutting Plane algorithms. In these methods, we start with a relaxed problem (usually the Assignment Problem) and iteratively check for violated subtour constraints, adding them only as needed. This makes the DFJ formulation algorithmically complex to implement compared to the static, polynomial-sized models of MTZ or GG.

3 Literature Review: Cost Structures & Impact on TSP Relaxations

Research shows that the structure of the distance matrix not only strongly affects the difficulty of TSP, but also the tightness of its LP relaxation solution. Different spatial structures and statistical patterns can systematically enlarge or reduce the duality gap, even when the model form is fixed.

[6] Held, M., & Karp, R. M. (1970). The traveling-salesman problem and minimum spanning trees. Operations Research, 18(6), 1138-1162.

3.1 Cost Structure and Integrality Behavior in the TSP

For metric TSPs, classical results show that the integrality ratio is bounded between 4/3 and 3/2 for metric instances.[7] This highlights that different distance structures can lead to systematically different integrality gaps even with fixed formulation.

Therefore, the distance matrix structure is not only a data detail, but also a key factor in determining the cost of convexity.

3.2 Special Cost Structures Affecting Gap Size

A clear example comes from the {1,2}-TSP, where all distances take values only in {1,2}, illustrating how special cost structures can affect the gap size. It shows that the subtour LP behaves differently under this structured cost pattern, showing that there are significantly tighter bounds than other general metric settings. Therefore, restricting or structuring cost values can change the geometric structure of TSP polytope and narrow the duality gap.[8]

3.3 Geometric and Spatial Structures in TSP Instances

Different geometric patterns lead to predictable behavior in LP relaxations.

3.3.1 Random Euclidean Instances

As Held and Karp discussed[9], random Euclidean TSP instances (points sampled uniformly in a plane) are widely studied because they exhibit strong regularity. Empirical results show that the Held–Karp bound is typically within 1–2% of the optimal solution for Euclidean data. This indicates that low-variance geometric structures lead to well-behaved LP relaxations.

3.3.2 Clustered Instances

Prior work on clustered routing by Laporte and Semet[10] shows these structures promote fractional subtours and larger integrality gaps. It also demonstrates that this uneven distribution magnifies cost disparities, creates extremely cheap intra-cluster subtours and increases the likelihood of fractional subtours in weak formulations.

---

[7] Sahni, S., & Gonzalez, T. (1976). P-complete approximation problems. Journal of the ACM, 23(3), 555–565.

[8] Qian, J., Schalekamp, F., van Zuylen, A., & Williamson, D. P. (2015). On the integrality gap of the subtour LP for the {1,2}-TSP. Journal of Combinatorial Optimization, 29(1), 180–196.

[9] Held, M., & Karp, R. M. (1970). The traveling-salesman problem and minimum spanning trees. Operations Research, 18(6), 1138–1162.

[10] Laporte, G., & Semet, F. (2002). Classical heuristics for the clustered traveling salesman problem. Operations Research, 50(2), 251–265.

3.3.3 Hub-and-Spoke Structures

Transportation network study by Barnhart and Laporte[11] notes that hub-and-spoke graphs contain very cheap hub edges and expensive leaf-to-leaf edges, causing weak LP relaxations, especially under compact formulations like MTZ.

These structured cost asymmetries are known to create difficulties for routing algorithms and relaxations, representing high cost variance.

3.3.4 Grid or Regular Geometric Structures

Regularly spaced grid points exhibit low distance variability and behave similarly to Euclidean TSP, for which the LP relaxation is known to be tight.[12] Uniform spacing and low variance reduce opportunities for pathological fractional patterns. Hence, grid-like structures should yield smaller gaps and behave more like Euclidean TSP.

3.4 Literature Gap and Motivation for Our Study

Prior work typically examines mainly on formulation strength under a fixed cost structure, or cost structure effects under a fixed formulation. However, few studies perform a two-dimensional analysis of the combination of Formulation Choice and Cost Structure.

Therefore, even though theory strongly suggests that weaker formulations amplify the effects of pathological cost structures and stronger formulations are more robust across data types, this gap of two-dimensional analysis forms the motivation for our research design.

4 Methodology

4.1 Data Generation

We generated four types of TSP distance structures, Grid, Random Euclidean, Clustered, and Hub-and-Spoke, to represent increasing levels of cost variability (see Appendix E for sample visualizations). For each structure, 10 instances were created at problem sizes $n = 15, 18$ and 20, resulting in a total of 120 instances.

To illustrate the realised variability, the corresponding coefficients of variation (CV) were approximately: Grid: 0.42, Random Euclidean: 0.46, Clustered: 0.63, and Hub-and-Spoke: 0.65. A full CV validation plot for these generated instances is provided in Appendix F.

Although the intended CV targets were 0.2, 0.5, 0.65, and 1.0, the extreme values (0.2 and 1.0) were not attainable due to geometric and metric constraints, which means very low or very high CVs cannot be achieved without distorting the underlying spatial structure. Nevertheless,

[11] Barnhart, C., & Laporte, G. (2007). Transportation. Handbooks in Operations Research and Management Science, 14, 1–55.
[12] Reinelt, G. (1994). The traveling salesman: Computational solutions for TSP applications. Springer.

the realised CV range (0.42–0.65) still forms a meaningful gradient for analysing how different formulations react to increasing cost variability.

4.2 Two-Dimensional Analysis Formulation

To systematically evaluate relaxation tightness across both modelling and distance-structure dimensions, we construct an experiment involving four TSP formulations described in Section 2 — **Assignment, Miller-Tucker-Zemlin (MTZ), Dantzig-Fulkerson-Johnson (DFJ), and Gavish-Graves (GG)**, as well as four distance structures: **grid, Random Euclidean, clustered, and hub-and-spoke**, for a total of 16 test scenarios. For each TSP formulation and distance structure combination, we generate 10 random instances. Although instances were created at $n = 15, 18$, and 20, our main analysis focuses on $n = 15$ where all formulations could be solved to optimality without timeouts.

For every instance in each test scenario, we perform the following steps:

1. Solve the integer programming (IP) formulation to obtain the optimal value z
2. Solve the linear programming (LP) relaxation of the same formulation to obtain the corresponding lower bound w
3. Compute the integrality gap, defined as the ratio between the optimal value of a linear programming relaxation and the optimal value of the integer programming problem, or:

$$gap = (z - w)/z \times 100\%$$

After computing the integrality gaps for all ten instances within each scenario, we calculate the mean and standard deviation of the integrality gaps. These statistics reflect the expected relaxation tightness and the variability among the instance randomness.

4.3 Computational Limitations and Mitigation

During implementation, we observed that MTZ and Assignment Problem became computationally expensive for larger instances, with some runs exceeding ten hours for $n = 20$. To ensure feasibility, we imposed a 300-second timeout on these formulations. However, since timeouts return approximate solutions rather than proven optima, the resulting integrality gaps may be inaccurate. Therefore, the results presented in this study are based on $n = 15$ instances where all formulations could be solved to optimality.

In our implementation of the DFJ formulation, we employed lazy constraints to improve computational efficiency. Instead of including all subtour elimination constraints upfront, the solver starts with only the assignment constraints. The callback function detects subtours in integer-feasible solutions and adds the corresponding subtour-elimination constraints dynamically. This approach ensures that only violated subtour constraints are introduced and helps to accelerate the solving process.

5 Results

Table 1 summarizes the average integrality gap and the standard deviation observed in our experiments, where each instance contains 15 cities. Each cell reports the mean gap across 10 random instances for the corresponding TSP formulation and distance structure.

Table 1: Integrality gap among TSP formulations and distance structures (n=15)

Formulation	**Grid** (CV ≈ 0.41)	**Random** (CV ≈ 0.46)	**Clustered** (CV ≈ 0.62)	**Hub-Spoke** (CV ≈ 0.65)
AP	0.05% (±0.03)	21.07% (±5.99)	78.39% (±4.49)	16.17% (±1.42)
MTZ	0.05% (±0.02)	19.64% (±5.77)	77.83% (±4.62)	15.90% (±1.37)
DFJ	0.00% (±0.00)	0.09% (±0.22)	0.00% (±0.00)	0.00% (±0.00)
GG	0.03% (±0.02)	12.64% (±5.09)	45.27% (±8.64)	9.94% (±1.02)

AP = Assignment Problem. All numbers rounded to two decimal places; standard deviation in parentheses.

5.1 Formulation Effects

The result shows that the tighter formulations produce smaller integrality gaps. A consistent hierarchy is observed across all distance structures: the Assignment formulation exhibits the largest gaps, followed by Miller-Tucker-Zemlin (MTZ), then Gavish-Graves (GG), while Dantzig-Fulkerson-Johnson (DFJ) consistently gives the smallest integrality gaps.

Figure 1 illustrates the relationships between the LP relaxation and IP optimal values. Across all four distance structures, the DFJ formulation produces LP values that are nearly identical to their corresponding IP optima, forming a gap close to zero. The behavior is expected and aligns well with the established findings in the literature reviews, as DFJ formulation includes strong subtour-elimination constraints that make its LP relaxation particularly tight.

In contrast, the assignment formulation and MTZ consistently exhibit the largest integrality gaps in every distance structure. Our results are expected and reflect that the formulation enforces the inflow and outflow constraints, but lacks explicit sub-tour elimination constraints. As a result, the LP relaxation produces highly fractional solutions containing multiple small subtours, and yields a loose lower bound and thus a larger integrality gap.

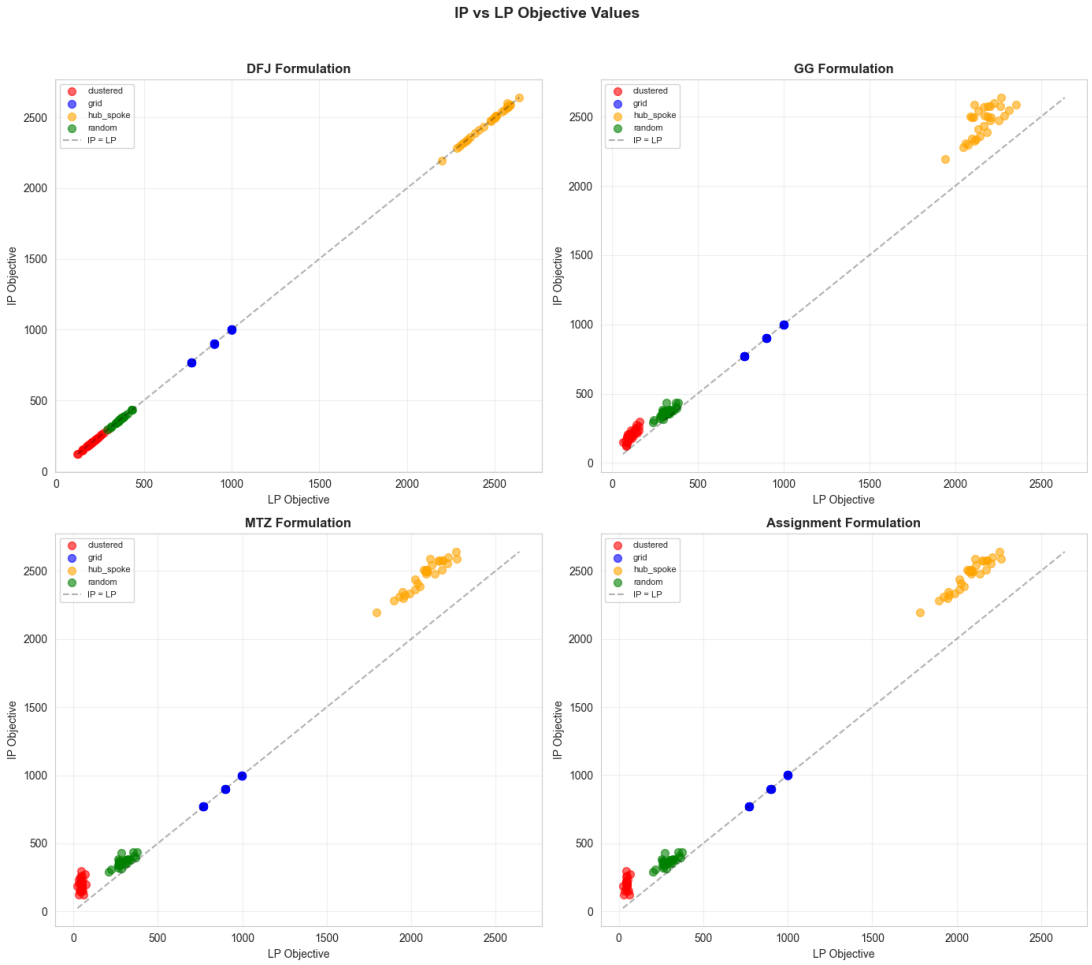


Figure 1: IP vs LP Objective Values for all four formulations. DFJ produces LP values nearly identical to IP optima (points on diagonal), while Assignment shows the largest deviations.

5.2 Cost Structure Effects

The second result shows that higher cost variation (CV) does not necessarily lead to larger integrality gaps across all TSP formulations.

Across the four cost structures, we first observe that the grid distance structure consistently yields an integrality gap of nearly 0%. This grid structure exhibits the lowest cost variation, and it lacks "systematic shortcuts," making it very difficult for the LP to find subtours that are significantly less expensive than the optimal full tour.

The Random Euclidean structure introduces moderate cost variation, and this distance structure separates the behavior of the weaker and stronger formulations. MTZ and GG exhibit noticeable increases in their integrality gaps, while DFJ remains extremely tight. In contrast, the clustered structure produces much larger gaps, especially for the loose TSP formulations.

However, hub-and-spoke contradicts our original hypothesis that a larger cost variation leads to a larger integrality gap among all four TSP formulations. Despite having the largest cost variation (CV), the outcome shows that the integrality gap is similar to Random Euclidean,

where the cost variation is relatively low, and the gap is far smaller than the clustered structure. This may be because the hub-and-spoke structure creates a natural tour pattern (visiting the hub between each spoke), which even weak formulations can approximate without forming cheap disconnected subtours.

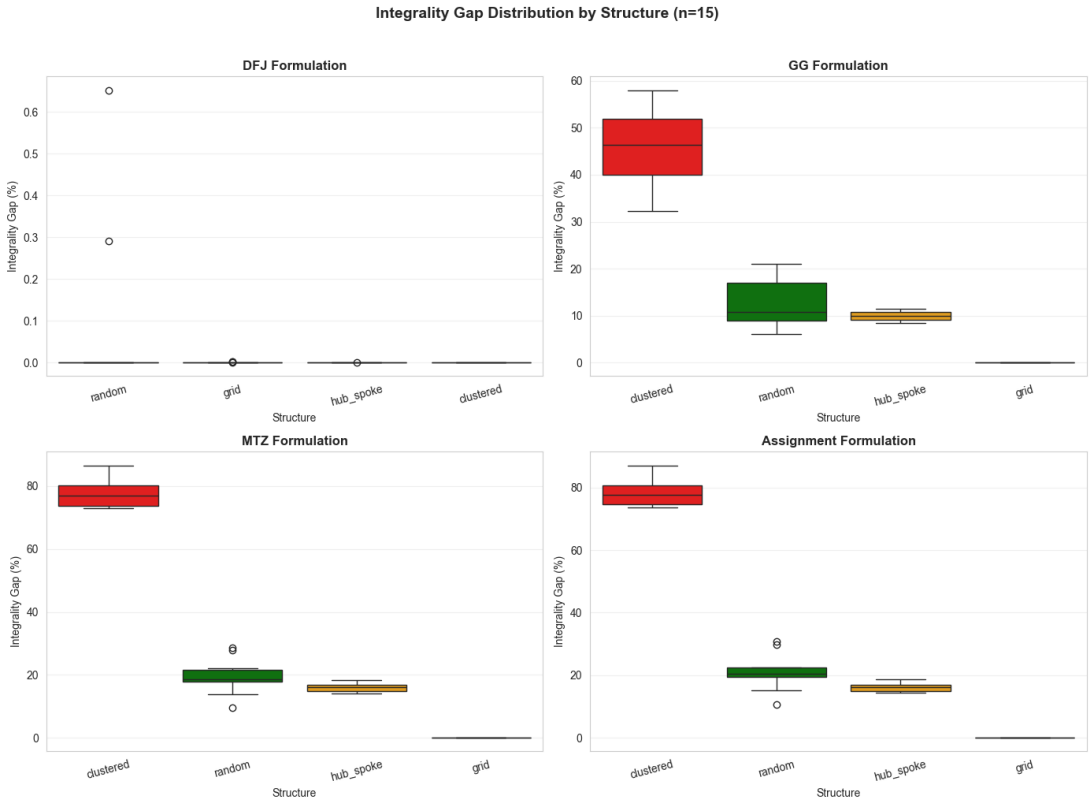


Figure 2: Integrality Gap Distribution by Distance Structure for each formulation. Clustered structure produces the largest gaps for weak formulations, while DFJ maintains near-zero gaps across all structures.

5.3 Statistical Analysis

To rigorously test whether the observed differences in integrality gaps are statistically significant, we performed three ANOVA tests examining the effects of structure, formulation, and their interaction.

5.3.1 One-Way ANOVA: Gap ∼ Structure

To understand whether the average integrality gap is the same across the four distance structures, we performed a one-way ANOVA for each TSP formulation.

The null hypothesis (H_0) is that there is no significant difference in the mean integrality gap across the four distance structures ($\mu_{Clustered} = \mu_{Grid} = \mu_{Hub\_Spoke} = \mu_{Random}$), while the alternative hypothesis (H_A) states that at least one of the distance structures has a significantly different mean integrality gap than the others.

Our statistical result shows that for GG, MTZ, and Assignment formulations, the p-values are all less than the decision boundary ($\alpha = 0.05$). Thus, we reject the null hypothesis for these formulations, inferring that the distance structure matters. However, for DFJ, the gaps do not differ significantly across structures ($p = 0.148$), which is expected since DFJ consistently achieves near-zero gaps regardless of structure.

Table 2 summarizes the p-value and F-statistic for each TSP formulation.

Table 2: One-Way ANOVA Results: Gap $\sim$ Structure ($n = 15$)

Formulation	**F-Stat**	**p-value**	**Significant**
DFJ	1.90	0.148	No
GG	151.78	1.75e-20	Yes
MTZ	820.52	3.50e-33	Yes
Assignment	803.94	5.03e-33	Yes

5.3.2 One-Way ANOVA: Gap $\sim$ Formulation

We also tested whether integrality gaps differ significantly across the four formulations.

- F-statistic: 13.7097
- p-value: 5.54×10^{-8}
- **Conclusion:** Gaps differ significantly across formulations

This confirms that the choice of LP formulation has a statistically significant impact on the integrality gap.

5.3.3 Two-Way ANOVA: Gap $\sim$ Structure $\times$ Formulation

Finally, we performed a two-way ANOVA to examine the main effects and interaction between structure and formulation. Table 3 presents the results.

Table 3: Two-Way ANOVA Results ($n = 15$)

Source	**SS**	**df**	**F**	**p-value**
Structure	57896.14	3	1427.49	<0.001
Formulation	21978.89	3	541.91	<0.001
Structure $\times$ Form.	23521.71	9	193.32	<0.001
Residual	1946.79	144	–	–

Interpretation:

- **Structure effect**: Cost structure significantly impacts integrality gap ($F = 1427.49$, $p < 0.001$)

- **Formulation effect**: LP formulation choice significantly impacts gap ($F = 541.91$, $p < 0.001$)
- **Interaction effect**: The effect of structure depends on formulation choice ($F = 193.32$, $p < 0.001$). This significant interaction means that weak formulations (Assignment, MTZ) are more sensitive to cost structure than strong formulations (DFJ).

6 Discussion and Conclusion

6.1 Weak Combinations (Large Integrality Gap)

A large integrality gap can be problematic since it indicates that the linear programming relaxation is a poor approximation of the true integer program. This means that the LP provides a loose lower bound, slows down the branch-and-bound, and makes the integer program difficult to solve even with the relaxations.

From the above integrality gap results, the weakest combinations are:

1. Assignment formulation with the clustered distance structure,
2. MTZ formulation with the clustered distance structure,
3. GG formulation with the clustered structure,

Each corresponding gap of approximately 78% for the former two combinations, and 45% for the last.

The clustered structure has very low local costs but high inter-cluster costs. The weak formulations, such as assignment, and MTZ exploit this weakness by forming small and "cheap" cycles on each sub-tour, and thus give an extremely low linear programming cost as well as avoiding the high cost of the required inter-cluster routes, and thus maximizing the integrality gap.

6.2 Practical Implications

Table 4 summarizes the recommended approach and the expected integrality gap among different data structures for varying costs. For any cost variation that requires a guaranteed tight bound, the DFJ formulation, which yields an almost zero integrality gap across all structures, will be suggested.

Table 4: Recommended TSP Approach for Distance Structures

CV	**Structure**	**Recommended**	**Expected Gap**
Low (0.42)	Grid	Any formulation	≈0%
Low (0.46)	Random Euclidean	GG, DFJ	≈13% (GG), ≈0% (DFJ)
Medium (0.63)	Clustered	DFJ (best), GG	≈0% (DFJ), ≈45% (GG)
Med–High (0.62–0.65)	Hub-and-Spoke	DFJ (best)	≈0% (DFJ), 10%–16% (others)

6.3 Conclusion

This project provided a comprehensive analysis of the integrality gap in the Traveling Salesman Problem (TSP) by exploring the interaction between mathematical formulations and cost structures. Through the use of four distinct data structures (Grid City, Random Euclidean, Clustered, and Hub-and-Spoke), we demonstrated how the coefficient of variation (CV) impacts the performance of different TSP formulations, including AP, DFJ, MTZ, and Gavish-Graves.

Key findings include:

- The integrality gap varies significantly across formulations and data structures, with certain formulations (e.g., DFJ) performing better in specific scenarios.
- While the DFJ formulation is theoretically exponential in the number of constraints, practical techniques like lazy constraints make it computationally feasible for moderate problem sizes.[13] This aligns with our empirical program results, where the solution process was significantly faster than anticipated.
- The MTZ formulation, while simpler, struggles with larger problem sizes due to its weaker LP relaxation.

Contributions:

- We introduced a two-dimensional framework that combines mathematical formulations and cost structures to analyze the integrality gap.
- The project provides a reproducible pipeline for generating test data, solving TSP instances, and analyzing results.

Limitations:

- Computational constraints limited the problem size to $n = 15$ for most benchmarks, especially for the assignment problem, for more extensive testing we may switch to gurobi cloud.
- The study focused on a small set of formulations and data structures, leaving room for broader exploration.

Future Work:

- Extend the analysis to larger problem sizes and additional formulations.
- Investigate solver-specific optimizations and parallelization techniques to improve scalability.
- Explore real-world cost structures to validate the findings in practical scenarios.

This study highlights the importance of understanding the interplay between formulations and data structures in TSP and provides a foundation for future research in this area.

[13] References: Applegate, D.L., Bixby, R.E., Chvátal, V., & Cook, W.J. (2006). *The Traveling Salesman Problem: A Computational Study*. Princeton University Press. Gurobi Optimization, LLC. (2025). *Gurobi Optimizer Reference Manual*. Lodi, A., & Tramontani, A. (2013). "Performance variability in mixed-integer programming." *Mathematical Programming Computation, 5(2), 127–162.*

A Mathematical Formulation of the Assignment Problem

Let $G = (V, A)$ be a complete directed graph where $V = \{1, \ldots, n\}$ is the set of cities and A is the set of arcs connecting them. For each arc $(i, j) \in A$, we define a binary decision variable:

Decision Variables:

$$x_{ij} \in \{0, 1\} : \text{A binary variable equal to 1 if the tour travels from city } i \text{ to city } j$$

Objective Function:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

Constraints:

$$\sum_{j=1, j \neq i}^{n} x_{ij} = 1, \quad \forall i \in V \tag{1}$$

$$\sum_{i=1, i \neq j}^{n} x_{ij} = 1, \quad \forall j \in V \tag{2}$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in A \tag{3}$$

B Mathematical Formulation of Miller-Tucker-Zemlin (MTZ)

The MTZ formulation builds upon the Assignment Problem by adding a set of variables and constraints designed to eliminate subtours.

Decision Variables:

$$x_{ij} \in \{0, 1\} : \text{A binary variable equal to 1 if the tour travels from city } i \text{ to city } j$$

$u_i \in \mathbb{R}$: A continuous variable representing the "potential" or visitation order of city i in the tour for $i = 2, \ldots, n$. The depot (node 1) is arbitrarily fixed as the starting point.

Objective Function:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

Constraints: In addition to the standard assignment constraints (degree constraints), the MTZ formulation imposes the following subtour elimination constraints:

$$u_i - u_j + n x_{ij} \leq n - 1, \quad \forall i, j \in V \setminus \{1\}, i \neq j \tag{4}$$

$$1 \leq u_i \leq n - 1, \quad \forall i \in V \setminus \{1\} \tag{5}$$

These constraints enforce connectivity by ensuring that if a path exists from i to j (i.e., $x_{ij} = 1$), then city j must be visited at a later step in the sequence than city i ($u_j \geq u_i + 1$). This logic prevents the formation of any cycle that does not include the depot, effectively eliminating subtours.

C Mathematical Formulation of Gavish-Graves (GG)

The Gavish-Graves (GG) formulation models the tour as a delivery route in a network. We imagine the salesman leaves the source node (arbitrarily defined as node 0) carrying a total of $n-1$ units of a generic commodity. As the salesman visits each subsequent city in the tour, exactly 1 unit of this commodity is consumed. This flow mechanism ensures that all visited nodes are connected to the source.

Sets and Parameters:

- $V = \{0, 1, \ldots, n-1\}$: Set of nodes (cities), where node 0 is the source.
- c_{ij}: Distance or cost from node i to node j.
- n: Number of nodes.

Decision Variables:

$$x_{ij} \in \{0, 1\} : \text{A binary variable equal to 1 if the tour travels from city } i \text{ to city } j$$

$f_{ij} \geq 0$: A continuous flow variable representing the amount of commodity flowing on edge (i, j)

Objective Function:

$$\min \sum_{i \in V} \sum_{j \in V, j \neq i} c_{ij} x_{ij}$$

Constraints: In addition to the standard degree constraints, the GG formulation imposes the following flow-based constraints to eliminate subtours.

Flow from source (node 0): The source sends out exactly $n-1$ units of flow.

$$\sum_{j \in V, j \neq 0} f_{0j} = n - 1$$

No flow into source: The source node does not receive any flow.

$$\sum_{i \in V, i \neq 0} f_{i0} = 0$$

Flow conservation: For every non-source node, the flow entering the node minus the flow leaving it must equal the demand of 1 unit.

$$\sum_{j \in V, j \neq i} f_{ji} - \sum_{j \in V, j \neq i} f_{ij} = 1, \quad \forall i \in V \setminus \{0\}$$

Flow capacity and linking: Flow is only permitted on an edge if that edge is selected in the tour ($x_{ij} = 1$). Furthermore, the flow on any single edge cannot exceed the total available commodity ($n - 1$).

$$f_{ij} \leq (n-1)x_{ij}, \quad \forall i, j \in V, i \neq j \tag{6}$$
$$f_{ij} \geq 0, \quad \forall i, j \in V, i \neq j \tag{7}$$

D Mathematical Formulation of Dantzig-Fulkerson-Johnson (DFJ)

The DFJ formulation[14] relies on the fundamental idea that any subset of cities S (where S is a proper subset of V containing at least 2 cities) must not contain a closed loop. Therefore, the number of edges connecting cities within S must be strictly less than the number of cities in S.

Constraints: In addition to the standard assignment (degree) constraints, the DFJ formulation imposes the following Subtour Elimination Constraints (SECs) for every proper subset of vertices $S \subset V$:

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1, \quad \forall S \subset V, 2 \leq |S| \leq |V| - 1$$

Alternatively, these can be expressed as "cutset" constraints, requiring that every proper subset S must have at least one outgoing edge to the rest of the graph:

$$\sum_{i \in S, j \notin S} x_{ij} \geq 1, \quad \forall S \subset V, \emptyset \neq S \neq V$$

[14] Dantzig, G., Fulkerson, R., & Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. Operations Research, 2(4), 393-410.

E Sample Distance Structures

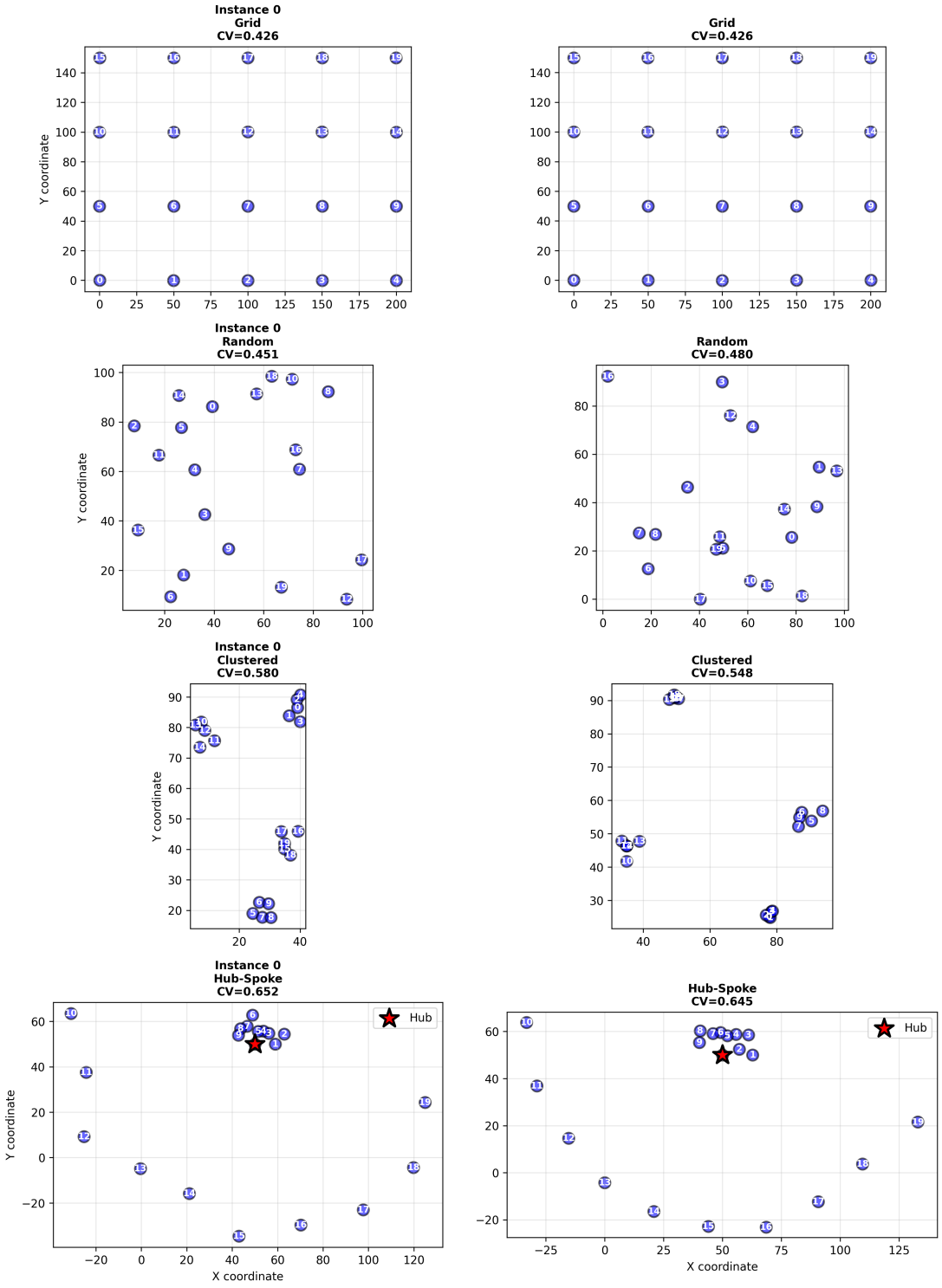
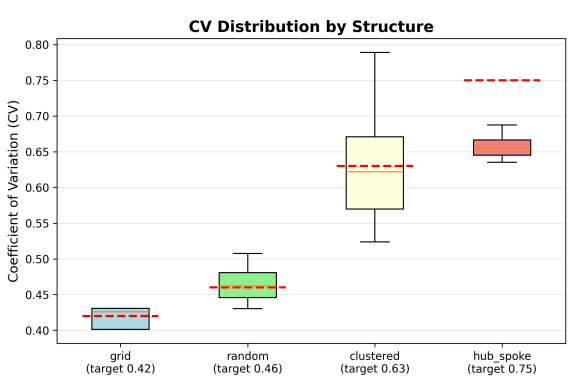


Figure 3: Sample instances illustrating the four distance structures used in this study: Grid (regular spacing), Random Euclidean (uniformly distributed points), Clustered (grouped points with high inter-cluster distances), and Hub-and-Spoke (central hub with peripheral nodes).

F CV Validation



CV Summary

Structure	Target	Mean	Std	Range
Grid	0.42	0.419	0.013	0.401-0.431
Random	0.46	0.463	0.022	0.430-0.508
Clustered	0.63	0.629	0.070	0.524-0.789
Hub-Spoke	0.75	0.658	0.015	0.635-0.687

Figure 4: Coefficient of Variation (CV) validation for each distance structure across all generated instances. Grid: CV ≈ 0.42, Random Euclidean: CV ≈ 0.46, Clustered: CV ≈ 0.63, Hub-and-Spoke: CV ≈ 0.65.

G Code: DFJ Solver

```
"""
Dantzig-Fulkerson-Johnson (DFJ) Formulation for TSP
Uses subtour elimination constraints with lazy callback

References:
- Held-Karp relaxation (Subtour Elimination Problem - SEP)
- arXiv:2507.07003 - The Integrality Gap of the TSP
- Standard formulation for studying DFJ integrality gap
"""

import gurobipy as gp
from gurobipy import GRB
import pandas as pd
import numpy as np
import sys
import time


def load_distance_matrix(filepath):
    """Load distance matrix from CSV file"""
    df = pd.read_csv(filepath, header=None)
    return df.values


def find_subtours(n, x_vals):
    """
    Find all subtours in current solution using DFS
    Args:
        n: number of nodes
```

```
        x_vals: dictionary of edge variables and their values {(i,j): value}
    Returns:
        List of subtours, where each subtour is a list of nodes
    """
    # Build adjacency list from edges with value close to 1, this is the graph algorithm in order to know the entire network is connected or not
    edges = [(i, j) for (i, j), val in x_vals.items() if val > 0.5]
    adj = {i: [] for i in range(n)}
    for i, j in edges:
        adj[i].append(j)

    visited = [False] * n
    subtours = []

    for start in range(n):
        if visited[start]:
            continue
        # Follow the tour from this node
        tour = []
        curr = start
        while not visited[curr]:
            visited[curr] = True
            tour.append(curr)
            # Find next node
            if adj[curr]:
                curr = adj[curr][0]
            else:
                break
        if len(tour) > 0:
            subtours.append(tour)

    return subtours


def build_dfj_model(dist_matrix, relaxation=False, subtour_constraints=None):
    """
    Build Dantzig-Fulkerson-Johnson TSP model

    Args:
        dist_matrix: nxn numpy array with distances
        relaxation: if True, solve LP relaxation; if False, solve IP
        subtour_constraints: List of subtour constraint sets to add (for LP)
    Returns:
        model: Gurobi model
        x: edge decision variables
```

```
        subtour_list: list to store subtour constraints (for IP only)
    """
    n = len(dist_matrix)
    model = gp.Model("TSP_DFJ")

    # Silent the solver output
    model.Params.OutputFlag = 0

    # Disable presolve reductions for lazy constraints
    if not relaxation:
        model.Params.LazyConstraints = 1

    # Decision variables: x[i,j] = 1 if edge (i,j) is in tour
    if relaxation:
        x = model.addVars(n, n, vtype=GRB.CONTINUOUS, lb=0, ub=1, name
            ="x")
    else:
        x = model.addVars(n, n, vtype=GRB.BINARY, name="x")

    # Objective: minimize total distance
    obj = gp.quicksum(dist_matrix[i, j] * x[i, j]
                      for i in range(n) for j in range(n) if i != j)
    model.setObjective(obj, GRB.MINIMIZE)

    # Constraint (1): Each node has exactly one outgoing edge
    for i in range(n):
        model.addConstr(
            gp.quicksum(x[i, j] for j in range(n) if j != i) == 1,
            name=f"out_{i}"
        )

    # Constraint (2): Each node has exactly one incoming edge
    for i in range(n):
        model.addConstr(
            gp.quicksum(x[j, i] for j in range(n) if j != i) == 1,
            name=f"in_{i}"
        )

    # For LP: Add subtour elimination constraints from IP solve
    if relaxation and subtour_constraints:
        for idx, subtour in enumerate(subtour_constraints):
            model.addConstr(
                gp.quicksum(x[i, j] for i in subtour for j in subtour
                    if i != j) <= len(subtour) - 1,
                name=f"subtour_lp_{idx}"
            )

    # For IP: Will use lazy callback (constraints added during solve)
    subtour_list = [] if not relaxation else None
```

```
    model.update()
    return model, x, subtour_list

def subtour_callback(model, where, x, n, subtour_list):
    """
    Gurobi callback function to add lazy subtour elimination
       constraints
    Args:
        model: Gurobi model
        where: callback location
        x: decision variables
        n: number of nodes
        subtour_list: list to store added subtour constraints
    """
    if where == GRB.Callback.MIPSOL:
        # Get current solution
        x_vals = model.cbGetSolution(x)

        # Find subtours in current solution
        subtours = find_subtours(n, x_vals)

        # If more than one subtour, add lazy constraints
        if len(subtours) > 1:
            for subtour in subtours:
                if len(subtour) < n:  # Don't add constraint for full
                   tour
                    # Add lazy constraint: sum of edges in subtour <=
                       |subtour| - 1
                    model.cbLazy(
                        gp.quicksum(x[i, j] for i in subtour for j in
                           subtour if i != j)
                        <= len(subtour) - 1
                    )
                    # Store the subtour for LP reuse
                    subtour_list.append(subtour)

def compute_cv(dist_matrix):
    """Compute coefficient of variation of distance matrix"""
    # Extract non-diagonal elements (actual distances)
    distances = dist_matrix[np.triu_indices_from(dist_matrix, k=1)]
    if len(distances) == 0 or np.mean(distances) == 0:
        return 0
    cv = np.std(distances) / np.mean(distances)
    return cv


def solve_instance(dist_matrix_path):
    # Load distance matrix
    dist_matrix = load_distance_matrix(dist_matrix_path)
```

```
    n = len(dist_matrix)
    results = {
        'instance': dist_matrix_path.split('/')[-1],
        'n': n
    }
    # Compute coefficient of variation
    results['cv'] = compute_cv(dist_matrix)

    # ========== Step 1: Solve IP with lazy constraints ==========
    # Time the IP solve (constraint generation phase)
    ip_start = time.time()
    model_ip, x_ip, subtour_list = build_dfj_model(dist_matrix,
       relaxation=False)

    # Set callback for lazy constraint generation
    model_ip.optimize(
        lambda model, where: subtour_callback(model, where, x_ip, n,
           subtour_list)
    )
    results['ip_solve_time'] = time.time() - ip_start

    if model_ip.status == GRB.OPTIMAL:
        results['IP_obj'] = model_ip.objVal
    else:
        results['IP_obj'] = None

    # ========== Step 2: Solve LP with same subtour constraints
       ==========
    # Time the LP relaxation (Held-Karp bound)
    lp_start = time.time()
    model_lp, x_lp, _ = build_dfj_model(dist_matrix, relaxation=True,
       subtour_constraints=subtour_list)
    model_lp.optimize()
    results['lp_solve_time'] = time.time() - lp_start

    if model_lp.status == GRB.OPTIMAL:
        results['LP_obj'] = model_lp.objVal
    else:
        results['LP_obj'] = None

    # Total time = constraint generation (IP) + LP solve
    results['total_solve_time'] = results['ip_solve_time'] + results['
       lp_solve_time']

    # ========== Step 3: Compute integrality gap ==========
    if results['IP_obj'] is not None and results['LP_obj'] is not None
       :
        results['gap_percent'] = ((results['IP_obj'] - results['LP_obj
           ']) / results['IP_obj']) * 100
```

```
        results['gap_absolute'] = results['IP_obj'] - results['LP_obj']

    return results

if __name__ == "__main__":
    if len(sys.argv) > 1:
        filepath = sys.argv[1]
        results = solve_instance(filepath)
```

H Code: GG Solver

```
"""
Gavish-Graves (GG) Formulation for TSP
Single-commodity flow formulation to prevent subtours
"""

import gurobipy as gp
from gurobipy import GRB
import pandas as pd
import numpy as np
import sys
import time

def load_distance_matrix(filepath):
    """Load distance matrix from CSV file"""
    df = pd.read_csv(filepath, header=None)
    return df.values


def build_gg_model(dist_matrix, relaxation=False):
    """
    Build Gavish-Graves TSP model
    Args:
        dist_matrix: nxn numpy array with distances
        relaxation: if True, solve LP relaxation; if False, solve IP
    Returns:
        model: Gurobi model
        x: edge decision variables
        f: flow variables
    """
    n = len(dist_matrix)
    model = gp.Model("TSP_GG")

    # Silent the solver output since it is noisy
    model.Params.OutputFlag = 0
```

```
35
36      # Decision variables
37      # x[i,j] = 1 if edge (i,j) is in tour
38      if relaxation:
39          x = model.addVars(n, n, vtype=GRB.CONTINUOUS, lb=0, ub=1, name
                ="x")
40      else:
41          x = model.addVars(n, n, vtype=GRB.BINARY, name="x")
42
43      # f[i,j] = flow on edge (i,j)
44      f = model.addVars(n, n, vtype=GRB.CONTINUOUS, lb=0, name="f")
45
46      # Objective: minimize total distance
47      obj = gp.quicksum(dist_matrix[i, j] * x[i, j]
48                        for i in range(n) for j in range(n) if i != j)
49      model.setObjective(obj, GRB.MINIMIZE)
50
51      # Constraint (1): Each node has exactly one outgoing edge
52      for i in range(n):
53          model.addConstr( gp.quicksum(x[i, j] for j in range(n) if j !=
                i) == 1, name=f"out_{i}")
54
55      # Constraint (2): Each node has exactly one incoming edge
56      for j in range(n):
57          model.addConstr(gp.quicksum(x[i, j] for i in range(n) if i !=
                j) == 1, name=f"in_{j}")
58
59      # Flow constraints (using node 0 as source)
60      source = 0
61
62      # Constraint (3): Source sends n-1 units of flow
63      model.addConstr(gp.quicksum(f[source, j] for j in range(n) if j !=
            source) == n - 1, name="flow_out_source")
64
65      # Constraint (4): Source receives no flow
66      model.addConstr(gp.quicksum(f[i, source] for i in range(n) if i !=
            source) == 0, name="flow_in_source")
67
68      # Constraint (5): Flow conservation for non-source nodes
69      # Each non-source node consumes exactly 1 unit of flow
70      for i in range(n):
71          if i != source:
72              model.addConstr( gp.quicksum(f[j, i] for j in range(n) if
                    j != i) - gp.quicksum(f[i, j] for j in range(n) if j !=
                     i) == 1, name=f"flow_balance_{i}")
73
74      # Constraint (6): Flow only on selected edges
75      for i in range(n):
76          for j in range(n):
```

```
            if i != j:
                model.addConstr(f[i, j] <= (n - 1) * x[i, j], name=f"
                    flow_capacity_{i}_{j}")
    model.update()
    return model, x, f


def solve_model(model):
    model.optimize()
    if model.status == GRB.OPTIMAL:
        return model.objVal
    else:
        return None

def compute_integrality_gap(ip_obj, lp_obj):
    if ip_obj == 0: return 0
    return ((ip_obj - lp_obj) / ip_obj) * 100


def solve_tsp_gg(dist_matrix, relaxation=False):
    model, x, f = build_gg_model(dist_matrix, relaxation=relaxation)
    obj = solve_model(model)
    return {'objective': obj}

def compute_cv(dist_matrix):
    """Compute coefficient of variation of distance matrix"""
    # Extract non-diagonal elements (actual distances)
    distances = dist_matrix[np.triu_indices_from(dist_matrix, k=1)]
    if len(distances) == 0 or np.mean(distances) == 0:
        return 0
    cv = np.std(distances) / np.mean(distances)
    return cv


def solve_instance(dist_matrix_path):
    # Load distance matrix
    dist_matrix = load_distance_matrix(dist_matrix_path)
    n = len(dist_matrix)

    results = {
        'instance': dist_matrix_path.split('/')[-1],
        'n': n
    }

    # Compute coefficient of variation
    results['cv'] = compute_cv(dist_matrix)

    ip_result = solve_tsp_gg(dist_matrix, relaxation=False)
    results['IP_obj'] = ip_result['objective']
```

```
125
126     # Time only the LP relaxation
127     lp_start = time.time()
128     lp_result = solve_tsp_gg(dist_matrix, relaxation=True)
129     results['lp_solve_time'] = time.time() - lp_start
130
131     results['LP_obj'] = lp_result['objective']
132
133     if results['IP_obj'] is not None and results['LP_obj'] is not None
            :
134         results['gap_percent'] = compute_integrality_gap(results['
                IP_obj'], results['LP_obj'])
135         results['gap_absolute'] = results['IP_obj'] - results['LP_obj'
                ]
136     return results
137
138
139 if __name__ == "__main__":
140     filepath = sys.argv[1]
141     results = solve_instance(filepath)
```

I Code: MTZ Solver

```
1  from gurobipy import Model, GRB, quicksum
2  import pandas as pd
3
4  def build_mtz_model(distance_matrix, is_ip=True):
5      N = distance_matrix.shape[0]
6      model = Model("MTZ_TSP")
7      model.Params.OutputFlag = 0
8      vtype = GRB.BINARY if is_ip else GRB.CONTINUOUS
9
10     x = model.addVars(N, N, lb=0, ub=1, vtype=vtype, name="x")
11     u = model.addVars(N, lb=0, ub=N-1, name="u")
12
13     model.addConstrs((quicksum(x[i, j] for j in range(N)) == 1 for i
           in range(N)))
14     model.addConstrs((quicksum(x[i, j] for i in range(N)) == 1 for j
           in range(N)))
15     model.addConstrs((x[i, i] == 0 for i in range(N)))
16     model.addConstr(u[0] == 0)
17
18     for i in range(1, N):
19         for j in range(1, N):
20             if i != j:
```

```
                model.addConstr(u[i] - u[j] + 1 <= (N-1)*(1 - x[i, j])
                    )

    return model, x

def solve_mtz(distance_matrix, is_ip=True, time_limit=None):
    model, x = build_mtz_model(distance_matrix, is_ip=is_ip)
    N = distance_matrix.shape[0]
    model.setObjective(quicksum(distance_matrix[i, j] * x[i, j] for i
        in range(N) for j in range(N)), GRB.MINIMIZE)
    if time_limit:
        model.Params.TimeLimit = time_limit
    if model.ModelName == "MTZ_TSP":
        model.Params.MIPFocus = 1
    model.optimize()
    if model.Status == GRB.OPTIMAL:
        return model.objVal, False
    elif model.Status == GRB.TIME_LIMIT:
        if model.SolCount > 0:
            return model.objVal, True
        else:
            return None, True
    else:
        return None, False
```

J Code: AP Solver

```

import gurobipy as gp
from gurobipy import GRB, quicksum
import numpy as np

def build_ap_model(distance_matrix):
    """
    Builds the Assignment Problem (AP) model.
    AP is a relaxation of TSP where subtour elimination constraints
        are removed.
    """
    n = distance_matrix.shape[0]
    model = gp.Model("Assignment_Problem")
    model.Params.OutputFlag = 0

    # Decision variables: x[i,j]
    # AP has the integrality property, so solving as LP (Continuous)
    # yields integer solutions (0 or 1) automatically.
    x = model.addVars(n, n, vtype=GRB.CONTINUOUS, lb=0, name="x")
```

```
19
20      # Objective: minimize total distance
21      obj = quicksum(distance_matrix[i, j] * x[i, j]
22                     for i in range(n) for j in range(n) if i != j)
23      model.setObjective(obj, GRB.MINIMIZE)
24
25      # Constraint (1): Each node has exactly one outgoing edge
26      model.addConstrs((quicksum(x[i, j] for j in range(n) if j != i) ==
            1
27                       for i in range(n)), name="out")
28
29      # Constraint (2): Each node has exactly one incoming edge
30      model.addConstrs((quicksum(x[i, j] for i in range(n) if j != i) ==
            1
31                       for j in range(n)), name="in")
32
33      # Prevent self-loops
34      for i in range(n):
35          x[i, i].ub = 0
36
37      return model, x
38
39  def solve_ap(distance_matrix):
40      """
41      Solves the Assignment Problem (AP) relaxation.
42      Returns the optimal objective value.
43      """
44      model, x = build_ap_model(distance_matrix)
45      model.optimize()
46
47      if model.status == GRB.OPTIMAL:
48          return model.objVal
49      else:
50          return None
```