U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Master's in Data Science and Engineering

Analytical Decision Support Systems Report

Aircraft Landings Scheduling

Group B3
Bruno Fernandes - up202108871
Hugo Abelheira - up202409899
Tiago Coelho - up202105004

January 16, 2025

Contents

List of Figures

Abstract

This report addresses the Aircraft Landing Scheduling Problem, focusing on optimizing landing schedules for single and multiple runway scenarios. We formulated the problem using Mixed-Integer Programming (MIP), Constraint Programming (CP), and a hybrid MIP_CP approach to evaluate their performance in terms of efficiency and solution quality. Key performance metrics, including execution time, variable and constraint counts, and solution quality, were analyzed. Results highlight that MIP excels in simpler scenarios, while CP is more effective in handling complex constraints. The hybrid approach combines the strengths of both. By leveraging advanced strategies like branching and hints, we demonstrated significant improvements in solver performance, offering insights into the trade-offs between computational efficiency and adaptability in real-world airport operations.

1 Introduction

Efficient aircraft landing scheduling is a critical challenge in modern airport operations, driven by increasing air traffic and limited runway capacity. This problem entails determining optimal landing times for incoming flights while adhering to constraints such as time windows, mandatory separation times, and runway availability. The objective is to ensure safety, minimize delays, and optimize resource utilization. With rising global air travel demand, addressing these challenges has become vital for maintaining efficient and reliable airport operations.

In this project, we address the static case of the Aircraft Landing Scheduling Problem (ALSP), where all flight details are known in advance. Our study extends the problem to both single and multiple runways, reflecting the practical complexities of real-world airport environments. We employ two analytical methodologies: Mixed-Integer Programming (MIP), Constraint Programming (CP) and MIP with a CP solver. MIP focuses on optimizing an objective function under linear constraints, while CP offers flexibility in modeling and solving problems with complex logical conditions. Together, these methods provide a robust framework for developing and evaluating solutions.

This report presents a comprehensive exploration of the ALSP within the scope of the Analytical Decision Support Systems course. Beginning with mathematical modeling using MIP, we expand to a CP-based approach to better handle intricate constraints and extend the analysis to multiple runways. Through computational experiments, we assess the effectiveness of these methodologies and analyze their results to draw actionable insights. This work shows the importance of advanced analytical techniques in addressing the operational challenges of aircraft landing scheduling.

2 Project Structure

The project is organized into a well-defined directory structure to separate different components based on their functionality and purpose. Below is an overview of the structure:

- **ALS/:** Contains the core Python scripts used for solving the Aircraft Landing Scheduling Problem (ALSP). This directory includes:
    1. **CP.py:** Implements functions for solving the problem using Constraint Programming (CP).
    2. **MIP.py:** Contains functions for solving the problem with Mixed-Integer Programming (MIP).
    3. **utils.py:** Includes utility functions, such as data parsing and preprocessing.
    4. **performanceCP.py:** Focuses on performance evaluation for CP-based solutions.
    5. **performanceMIP.py:** Evaluates the performance of MIP-based solutions.
    6. **visualization.py:** Provides utilities for visualizing the results and outputs of the solutions.
- **data/:** Holds various datasets used in the analysis.
- **notebook.ipynb:** The main notebook that compiles the analysis, including the comparison of different solution methods and the presentation of results.

3 Single and Multiple Runways Scenarios

In this project, we tackle the Aircraft Landing Scheduling Problem (ALSP) for both single and multiple runway scenarios. While single runway scheduling is simpler, the introduction of multiple runways adds complexity to the problem, as it requires managing aircraft allocations to different runways while respecting separation times and operational constraints. By solving both configurations, we are able to compare the effectiveness and efficiency of the MIP and CP approaches in handling both simpler and more complex settings.

This comparison highlights the strengths and limitations of each method in different operational environments.

4 MIP

The Aircraft Landing Scheduling Problem (ALSP) involves determining the optimal landing times for a set of aircrafts while satisfying operational constraints. This section presents the Mixed-Integer Programming (MIP) formulation for the problem, extending it to multiple runways. The objective is to minimize total deviations from the target landing times while ensuring safety and operational feasibility.

4.1 Notation

Sets and Indices

- i, j: Indices for aircraft ($i, j = 1, \ldots, P$).
- r: Index for runways ($r = 1, \ldots, R$).
- U: the set of pairs (i, j) of planes for which we are uncertain whether plane i lands before plane j.
- V: the set of pairs (i, j) of planes for which i definitely lands before j (but for which the separation constraint is not automatically satisfied).
- W: the set of pairs (i, j) of planes for which i definitely lands before j (and for which the separation constraint is automatically satisfied).

Parameters

- P: Number of planes.
- R: Number of runways.
- E_i: Earliest allowable landing time for aircraft i.
- L_i: Latest allowable landing time for aircraft i.
- T_i: Target (preferred) landing time for aircraft i.
- g_i: Penalty cost per unit of time for landing before T_i.
- h_i: Penalty cost per unit of time for landing after T_i.
- S_{ij}: Minimum separation time between the landings of aircraft i and j (if i lands before j).

Decision Variables

- x_i: Actual landing time of aircraft i.
- α_i: Deviation from T_i if aircraft i lands earlier ($\alpha_i \geq 0$).
- β_i: Deviation from T_i if aircraft i lands later ($\beta_i \geq 0$).
- δ_{ij}: Binary variable, $\delta_{ij} = 1$ if aircraft i lands before j, 0 otherwise.
- z_{ij}: Binary variable, $z_{ij} = 1$ if aircraft i and j land on the same runway, 0 otherwise.
- y_{ir}: Binary variable, $y_{ir} = 1$ if aircraft i is assigned to runway r, 0 otherwise.

4.2 Mathematical Formulation

Objective Function

$$\text{Minimize} \quad \sum_{i=1}^{P} (g_i \alpha_i + h_i \beta_i)$$

Sets

- $U = [\,(i,j) \mid i = 1, ..., P;\ j = 1, ..., P;\ i \neq j;\ E_j \leq E_i \leq L_j$ or $E_j \leq L_i \leq L_j$ or $E_i \leq E_j \leq L_i$ or $E_i \leq L_j \leq L_i\,]$
- $V = [\,(i,j) \mid L_i < E_j$ and $L_i + S_{ij} > E_j\ i = 1, \ldots, P;\ j = 1, \ldots, P;\ i \neq j]$.
- $W = [\,(i,j) \mid L_i < E_j$ and $L_i + S_{ij} \leq E_j\ i = 1, \ldots, P;\ j = 1, \ldots, P;\ i \neq j]$.

Constraints

1. **Time Window Constraints** Each aircraft must land within its allowable time window:

$$E_i \leq x_i \leq L_i, \quad \forall i$$

2. **Deviation Definition** Define earliness and lateness relative to the target landing time:

$$x_i = T_i - \alpha_i + \beta_i, \quad \forall i$$

$$\alpha_i \geq T_i - x_i, \quad 0 \leq \alpha_i \leq T_i - E_i, \quad \beta_i \geq x_i - T_i, \quad 0 \leq \beta_i \leq L_i - T_i, \quad \forall i$$

3. **Landing Order** For any pair of aircraft i and j, only one order is possible:

$$\delta_{ij} + \delta_{ji} = 1, \quad \forall (i,j), \quad j > i$$

4. **Separation Constraints** Aircraft i and j must maintain the required separation time depending on their landing order:

$$\delta_{ij} = 1, \quad \forall (i,j) \in W \cup V$$

$$x_j \geq x_i + S_{ij}, \quad \forall (i,j) \in V$$

$$x_j \geq x_i + S_{ij}\delta_{ij} - (L_i - E_i)\delta_{ij}, \quad \forall (i,j) \in U \quad *$$

5. **Variable Domains** Permissible values for all decision variables:

$$\alpha_i, \beta_i \geq 0, \quad x_i \geq 0, \quad z_{ij} \in \{0,1\}, \quad y_{ir} \in \{0,1\}, \quad \forall (i,j,r)$$

4.2.1 Extension for Multiple Runways

To extend the model for multiple runways, we introduce additional variables and constraints that ensure aircrafts are assigned to runways appropriately and maintain necessary separations only when sharing the same runway.

6. **Runway Assignment** Each aircraft must be assigned to exactly one runway:

$$y_{ir} = \begin{cases} 1, & \text{if plane } i \text{ lands on runway } r, \quad \forall (i, r) \\ 0, & \text{otherwise.} \end{cases}$$

$$\sum_{r=1}^{R} y_{ir} = 1, \quad \forall i.$$

7. **Runway Consistency** Aircrafts assigned to the same runway have consistent separation constraints applied:

$$z_{ij} = \begin{cases} 1, & \text{if planes } i \text{ and } j \text{ land on the same runway}, \quad \forall (i, j), \quad i \neq j \\ 0, & \text{otherwise.} \end{cases}$$

$$y_{ir} + y_{jr} \leq 1 + z_{ij} + z_{ji}, \quad \forall (i, j),\, i \neq j,\, \forall r.$$

Also, the third equation in the Separation Constraints* must be replaced by the following constraint:

$$x_j \geq x_i + S_{ij}\delta_{ij} - (L_i - E_i)\delta_{ij}, \quad \forall (i, j) \in U$$

4.3 Model Features

This MIP formulation emphasizes precision and scalability by incorporating large constants strategically to linearize complex constraints. The use of binary decision variables allows explicit modeling of separation requirements and landing sequences. Additionally, the formulation benefits from the inherent optimization capabilities of MIP solvers, ensuring robust solutions for large-scale instances. While computationally more intensive than CP, this approach is well-suited for problems requiring exact optimization and guarantees on solution quality.

5 Constraint Programming Model

In this section, we present the Constraint Programming (CP) model for the Aircraft Landing Scheduling Problem (ALSP). This formulation leverages the permutation-based approach to avoid large constants typically required in Mixed-Integer Programming (MIP) models. The model ensures that aircraft landing times respect operational constraints while minimizing total deviation costs.

5.1 Notation

Sets and Indices

- i, j: Indices for aircraft ($i, j = 1, \ldots, P$).

Parameters

- P: Number of planes.
- R: Number of runways.
- E_i: Earliest allowable landing time for aircraft i.
- L_i: Latest allowable landing time for aircraft i.

- T_i: Target (preferred) landing time for aircraft i.
- g_i: Penalty cost per unit of time for landing before T_i.
- h_i: Penalty cost per unit of time for landing after T_i.
- S_{ij}: Minimum separation time between the landings of aircraft i and j (if i lands before j).

Decision Variables

- position_i: The position of aircraft i in the landing order ($\text{position}_i \in \{1, \ldots, P\}$).
- x_i: Actual landing time of aircraft i.
- e_i: Earliness of aircraft i ($e_i \geq 0$).
- l_i: Lateness of aircraft i ($l_i \geq 0$).
- before_{ij}: Binary variable indicating if aircraft i lands before j ($\text{before}_{ij} \in \{0, 1\}$).
- runway_i: Runway assigned to aircraft i ($\text{runway}_i \in \{1, \ldots, R\}$).
- same_runway_{ij}: Binary variable indicating if aircraft i and j share the same runway ($\text{same\_runway}_{ij} \in \{0, 1\}$).

5.2 Mathematical Formulation

Objective Function

$$\text{Minimize} \quad \sum_{i=1}^{P} (g_i e_i + h_i l_i)$$

Constraints

1. **Time Window Constraints** Each aircraft must land within its allowable time window:

$$E_i \leq x_i \leq L_i, \quad \forall i.$$

2. **Deviation Definition** Define earliness and lateness relative to the target landing time:

$$e_i \geq T_i - x_i, \quad l_i \geq x_i - T_i, \quad \forall i.$$

$$e_i, l_i \geq 0, \quad \forall i.$$

3. **Landing Order (Permutation Constraints)** The positions of aircraft in the landing order must form a valid permutation:

$$\text{position}_i \neq \text{position}_j, \quad \forall i \neq j.$$

4. **Separation Constraints** Aircraft i and j must maintain the required separation time depending on their landing order:

$$\text{if position}_i < \text{position}_j, \quad x_j \geq x_i + S_{ij}.$$

$$\text{if position}_j < \text{position}_i, \quad x_i \geq x_j + S_{ji}.$$

To model this in CP, we introduce a binary variable before_{ij}:

$$\text{before}_{ij} = 1 \implies x_j \geq x_i + S_{ij}.$$

$$\text{before}_{ij} = 0 \implies x_i \geq x_j + S_{ji}.$$

5.2.1 Extension for Multiple Runways

To extend the model for multiple runways, we introduce additional variables and constraints that ensure aircrafts are assigned to runways appropriately and maintain necessary separations only when sharing the same runway.

5. **Runway Assignment** Each aircraft must be assigned to exactly one runway:

$$\text{runway}_i \in \{1, \ldots, R\}, \quad \forall i.$$

6. **Same Runway Indicator** For each pair of aircraft i and j, define whether they share the same runway:

$$\text{same\_runway}_{ij} = \begin{cases} 1 & \text{if runway}_i = \text{runway}_j, \\ 0 & \text{otherwise.} \end{cases}$$

This can be enforced in CP by:

$$\text{same\_runway}_{ij} = 1 \iff \text{runway}_i = \text{runway}_j, \quad \forall i < j.$$

7. **Conditional Separation Based on Runway** Apply separation constraints **only** if aircraft i and j are assigned to the same runway:

$$x_j \geq x_i + S_{ij} \quad \text{if before}_{ij} = 1 \text{ and same\_runway}_{ij} = 1.$$

$$x_i \geq x_j + S_{ji} \quad \text{if before}_{ij} = 0 \text{ and same\_runway}_{ij} = 1.$$

In CP terms, this can be modeled as:

$$x_j \geq x_i + S_{ij} \quad \text{only if } (\text{before}_{ij} = 1) \text{ and } (\text{same\_runway}_{ij} = 1).$$

$$x_i \geq x_j + S_{ji} \quad \text{only if } (\text{before}_{ij} = 0) \text{ and } (\text{same\_runway}_{ij} = 1).$$

These constraints ensure that separation times are enforced **only** when aircrafts share the same runway, allowing parallel landings on different runways without unnecessary delays.

5.3 Model Features

This CP formulation ensures computational efficiency by avoiding large constants (as used in MIP) and leveraging a permutation-based approach. The use of before_{ij} variables allows flexibility in enforcing separation constraints dynamically based on landing order. This approach is particularly suited for real-time or near-real-time applications due to its ability to handle complex constraints effectively.

6 Variable, Branching and Value Strategies

In this project, we explore the use of different strategies to optimize the performance of both the MIP and CP models. These strategies are essential for improving the search process, enabling us to experiment with different approaches, and ultimately comparing the quality and efficiency of the solutions obtained. The strategies are passed as parameters to the models during execution, and they influence the decision-making process during the search for feasible solutions.

6.1 Branching Strategies

Branching strategies dictate how the search tree is explored. Different strategies can prioritize certain branches of the solution space, affecting both the speed of the solver and the quality of the final solution.

- **Portfolio_Search:** This strategy uses a combination of pre-defined heuristics to dynamically adapt the search process, aiming to balance exploration and exploitation.
- **Lp_Search:** This strategy leverages linear programming relaxations to guide the search by prioritizing branches that are more likely to lead to optimal solutions.
- **Pseudo_Cost_Search:** This strategy estimates the impact of branching on each variable based on pseudo-costs, favoring branches expected to reduce the objective value the most effectively.

6.2 Variable Selection Strategies

Variable selection strategies help determine which variables should be assigned values first during the search process. The choice of variable selection can impact both the speed and accuracy of finding a feasible solution.

- **Choose_Min_Domain_Size:** Selects the variable with the smallest domain size, prioritizing variables that are closer to being fully assigned to reduce the search space faster.
- **Choose_Max_Domain_Size:** Selects the variable with the largest domain size, aiming to focus on the most uncertain variables earlier in the search.

6.3 Value Selection Strategies

Value selection strategies decide which values should be assigned to a selected variable. These strategies guide the search process by influencing the order in which values are explored.

- **Select_Lower_Half:** Chooses values from the lower half of the variable's domain, guiding the solver towards smaller values first.
- **Select_Upper_Half:** Chooses values from the upper half of the variable's domain, guiding the solver towards larger values first.

6.4 Using Hints for Optimizing Search

In addition to experimenting with various decision, branching, and value strategies, we also use hints to provide guidance to the solver, potentially improving the efficiency and effectiveness of the search process. A hint in Constraint Programming can be used to fix a variable to a specific value or guide the solver's exploration.

For our problem, we utilize the hint to suggest that the landing times of aircraft should be close to, or equal to, their target landing times. This guidance helps the solver focus on solutions that align with operational preferences, thereby reducing the search space and improving efficiency. By hinting at these preferred landing times, we provide a starting point that reflects realistic scheduling scenarios and can potentially accelerate the solution process.

The use of hints is another form of hyperparameter tuning, where we fine-tune the behavior of the solver to achieve faster or more accurate results, depending on the scenario. This approach complements the decision strategies and provides a more controlled environment for solving the problem.

7 Solution Visualization

To better analyze and interpret the solutions for the Aircraft Landing Scheduling Optimization Problem, a custom function was developed to generate detailed visualizations. These plots consolidate all the critical information necessary to understand the landing schedules, constraints, and deviations.

Figure 1 illustrates an example of a solution visualization. The key elements included in the plot are as follows:

- **Available Time Range:** Represented as horizontal lines for each plane, indicating the earliest and latest permissible landing times.
- **Earliest and Latest Times:** Marked with black triangles (E_i and L_i), providing the bounds for the feasible landing window.
- **Target Landing Time:** Shown as blue dots (T_i), representing the preferred landing times for each plane.
- **Optimal Landing Time:** Indicated by green crosses (x_i), which are the scheduled landing times determined by the optimization model.
- **Deviation Indicators:** Highlighted in red (early deviations, α_i) and orange (late deviations, β_i), quantifying the penalties associated with deviations from the target times.
- **Optimal Landing Achieved:** Depicted by purple stars ($x_i = T_i$), indicating planes that landed precisely at their target times.

Additionally, penalty values for early or late landings are displayed below the respective deviations to provide a clear understanding of the associated costs.

This visualization effectively combines the scheduling constraints, optimization results, and penalty analysis into a single, comprehensive figure. It serves as a valuable tool for evaluating the performance of the scheduling model and identifying areas for improvement.

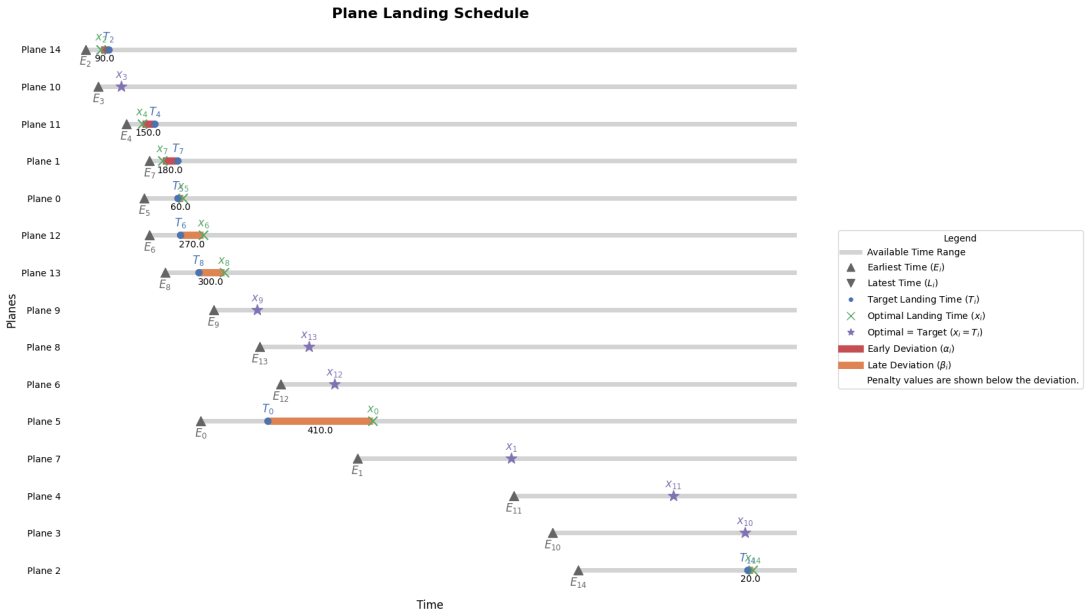

Figure 1: Solution with the MIP in the single runway scenario

8 Performance Metrics

Performance metrics are essential tools for evaluating the efficiency and quality of solutions generated by optimization algorithms. These metrics provide insight into the computational resources required, such as time and memory, as well as the solver's ability to handle constraints and explore the solution space effectively. By analyzing these metrics, researchers and practitioners can identify bottlenecks, optimize models, and select the most suitable algorithm for a given problem.

8.1 MIP

In the context of Mixed Integer Programming (MIP), the performance metrics focus on aspects specific to mathematical optimization models. These include:

- **Execution Time:** This metric measures the time taken by the solver to find an optimal solution or determine that no feasible solution exists. It is a critical factor for comparing solver efficiency, particularly in large-scale problems.
- **Number of Variables:** This represents the total number of decision variables in the model. A higher number of variables generally increases the computational complexity of the problem.
- **Number of Constraints:** The total number of constraints defines the problem's complexity by indicating the rules that the solution must satisfy. Complex models with many constraints require more computational effort.
- **Total Penalty:** This metric reflects the objective function value of the solution, representing the cost, profit, or other key performance indicators the model seeks to optimize. It provides insight into the quality of the solution.

These metrics provide a comprehensive view of the solver's performance in handling MIP problems, enabling better decision-making in model design and solver selection.

8.2 CP

Constraint Programming (CP) focuses on solving combinatorial problems, and its performance metrics reflect the characteristics of this approach. The main metrics include:

- **Execution Time:** Similar to MIP, this measures the time taken to solve the problem, often influenced by the complexity of constraints and the branching strategies used.
- **Solution Status:** This indicates whether the solver found an optimal solution, proved infeasibility, or terminated due to time or resource limits. Understanding the status helps assess the solver's effectiveness.
- **Number of Conflicts:** This metric tracks the number of conflicts encountered during the search process. A conflict occurs when a partial solution violates one or more constraints, requiring the solver to backtrack and explore alternative solutions.
- **Number of Branches:** This represents the total number of branches explored in the search tree. A high branching factor may indicate increased computational effort, while a low number of branches often reflects efficient pruning.
- **Best Objective Bound:** For optimization problems, this metric represents the best bound achieved during the search, providing an estimate of how close the solution is to optimality (note that all the solutions we reached were optimal, so this value, actually, is the optimal value).

By analyzing these metrics, the performance of CP solvers can be evaluated in terms of both efficiency and solution quality. They also offer valuable insights into how well the solver handles specific constraints and search strategies.

9 Result Analysis

The results presented in this study were obtained by systematically evaluating various combinations of decision strategies and branching strategies using Constraint Programming (CP) solvers. For each scenario, a set of input files containing air traffic data was processed. Two runway scenarios, single and multiple, were considered to simulate different operational conditions.

For each configuration, a range of decision strategy combinations was tested, varying the branching strategies, variable selection strategies, and value selection strategies. Each combination was evaluated multiple times to ensure consistency and robustness. During each run, the solver was executed, and key performance metrics, such as solution time and memory usage, were collected.

The metrics were averaged across successful runs for each configuration. Runs that failed due to solver errors or infeasible scenarios were logged and analyzed separately. This approach ensured a comprehensive exploration of the solution space while providing insights into the performance of each strategy under different conditions.

To facilitate the analysis, several graphs were developed to compare the performance metrics across different combinations of variable, branching, and value strategies. These visualizations provided a clear understanding of how each strategy impacted the solver's efficiency and solution quality.

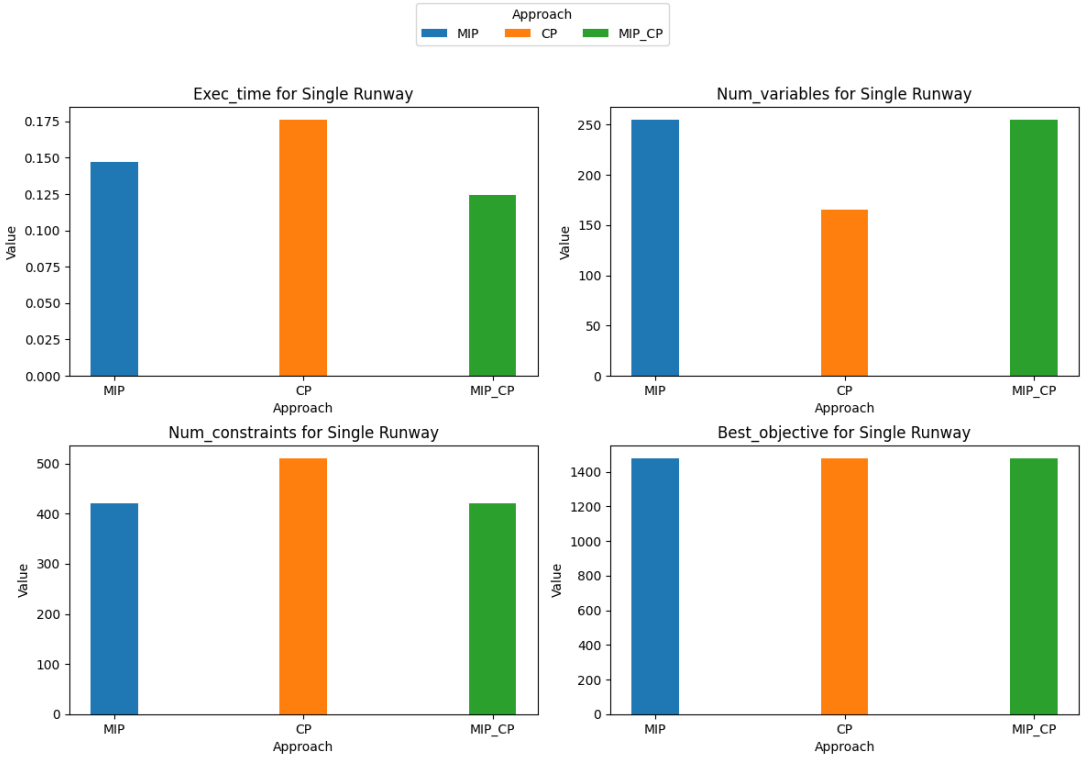

Figure 2: Performance comparison for different approaches in the single runway scenario

The results for the single-runway scenario highlight key differences across the MIP, CP, and MIP_CP approaches. In terms of execution time, MIP is the fastest, while CP is the slowest, reflecting the computational overhead of handling a denser constraint space. Despite using fewer variables, CP introduces more constraints, which likely contributes to its slower performance.

Interestingly, all approaches achieve similar best objective values, indicating that while their computational efficiency varies, the solution quality remains consistent. The hybrid MIP_CP shows a balanced trade-off, with moderate execution time and comparable solution quality.

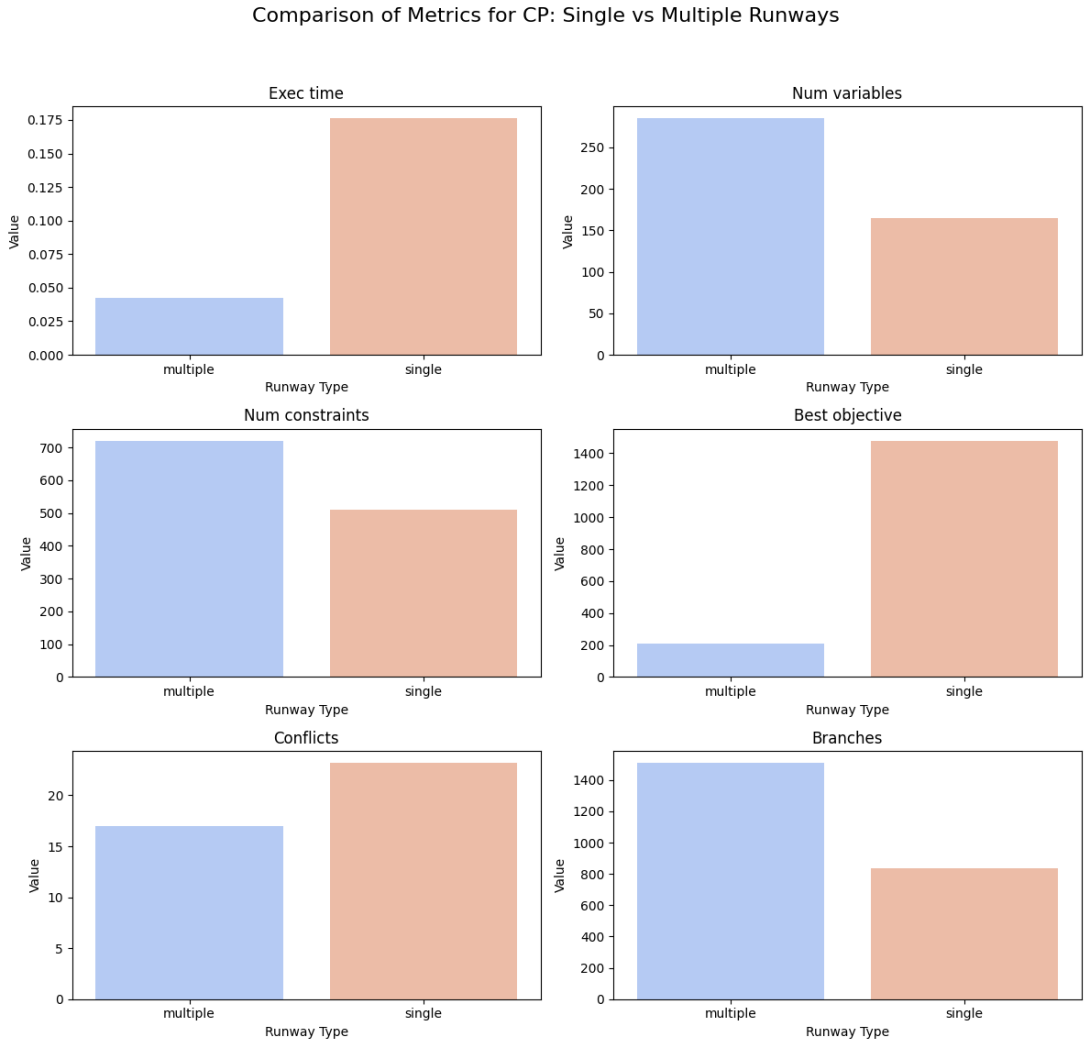

Figure 3: Performance comparison for CP in the single and the multiple runways scenarios

For CP, the single-runway scenario shows significantly longer execution times despite having fewer variables and constraints than the multiple-runway case. This is due to stricter constraints in the single-runway scenario, leading to more conflicts and backtracking.

The multiple-runway scenario involves higher complexity, with more variables and constraints, but benefits from better flexibility, reducing conflicts and achieving faster execution. The best objective value is higher for the single-runway case, reflecting the greater difficulty of optimizing under tighter constraints. These results underscore how increased flexibility in multiple-runway scenarios can enhance solver efficiency despite greater complexity.

10 Conclusion

This study provides key insights into solving the Aircraft Landing Scheduling Problem using MIP, CP, and a hybrid MIP_CP approach. For single-runway scenarios, MIP delivered faster execution times by handling simpler formulations efficiently, while CP proved effective in managing complex constraints, albeit at a higher computational cost. The hybrid MIP_CP approach struck a balance, combining MIP's speed with CP's flexibility.

In multiple-runway scenarios, CP excelled at managing increased complexity, benefiting from the added flexibility of multiple runways, which reduced conflicts and execution times despite higher variable and constraint counts. These results underscore the importance of selecting methods suited to the problem structure, as greater flexibility can offset computational challenges.

The study also demonstrated that strategies like branching and hints significantly impact solver efficiency. By narrowing the search space—for instance, hinting landing times close to target times—solutions were achieved more quickly and effectively.

In conclusion, the choice of method must align with the problem's complexity and operational needs, with CP suited for intricate scenarios, MIP for simpler problems, and hybrid methods offering a versatile compromise. This analysis highlights the trade-offs between solution quality, efficiency, and adaptability in real-world scheduling problems.

References

[1] J. E. Beasley, *Scheduling Aircraft Landings — The Static Case*, Journal of the Operational Research Society, 1990. Available at: `https://bura.brunel.ac.uk/bitstream/2438/4769/1/Fulltext.pdf` (Accessed: 2025-01-16).

[2] Google OR-Tools Team, *OR-Tools Documentation: Linear Solver (Pywraplp)*, Available at: `https://ortools.github.io/docs/pdoc/ortools/linear_solver/pywraplp.html` (Accessed: 2025-01-16).

[3] Google OR-Tools Team, *OR-Tools Documentation: CP-SAT Model Reference*, Available at: `https://developers.google.com/optimization/reference/python/sat/python/cp_model` (Accessed: 2025-01-16).

A Additional Visualizations for the Result Analysis

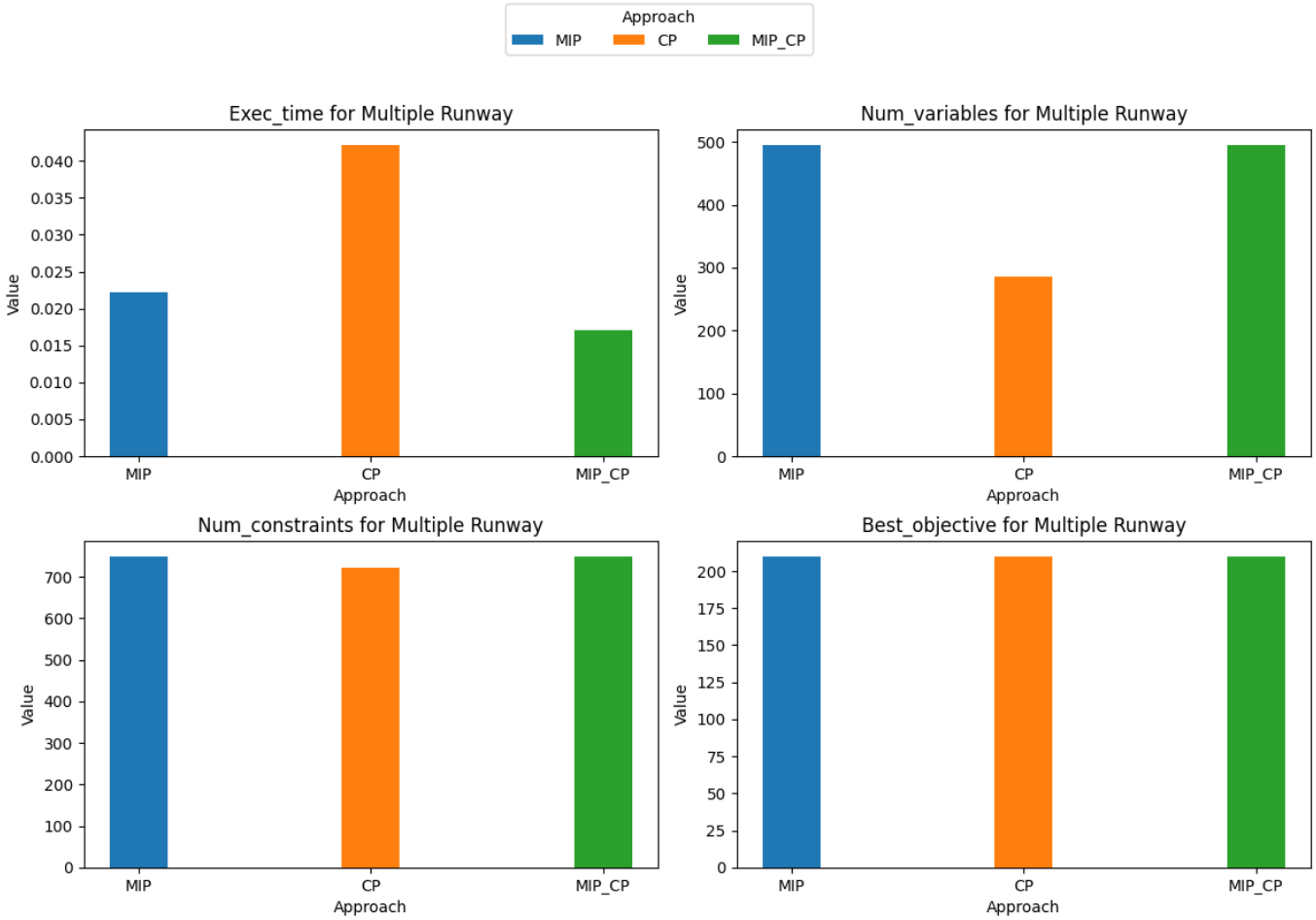

Figure 4: Comparison between different approaches

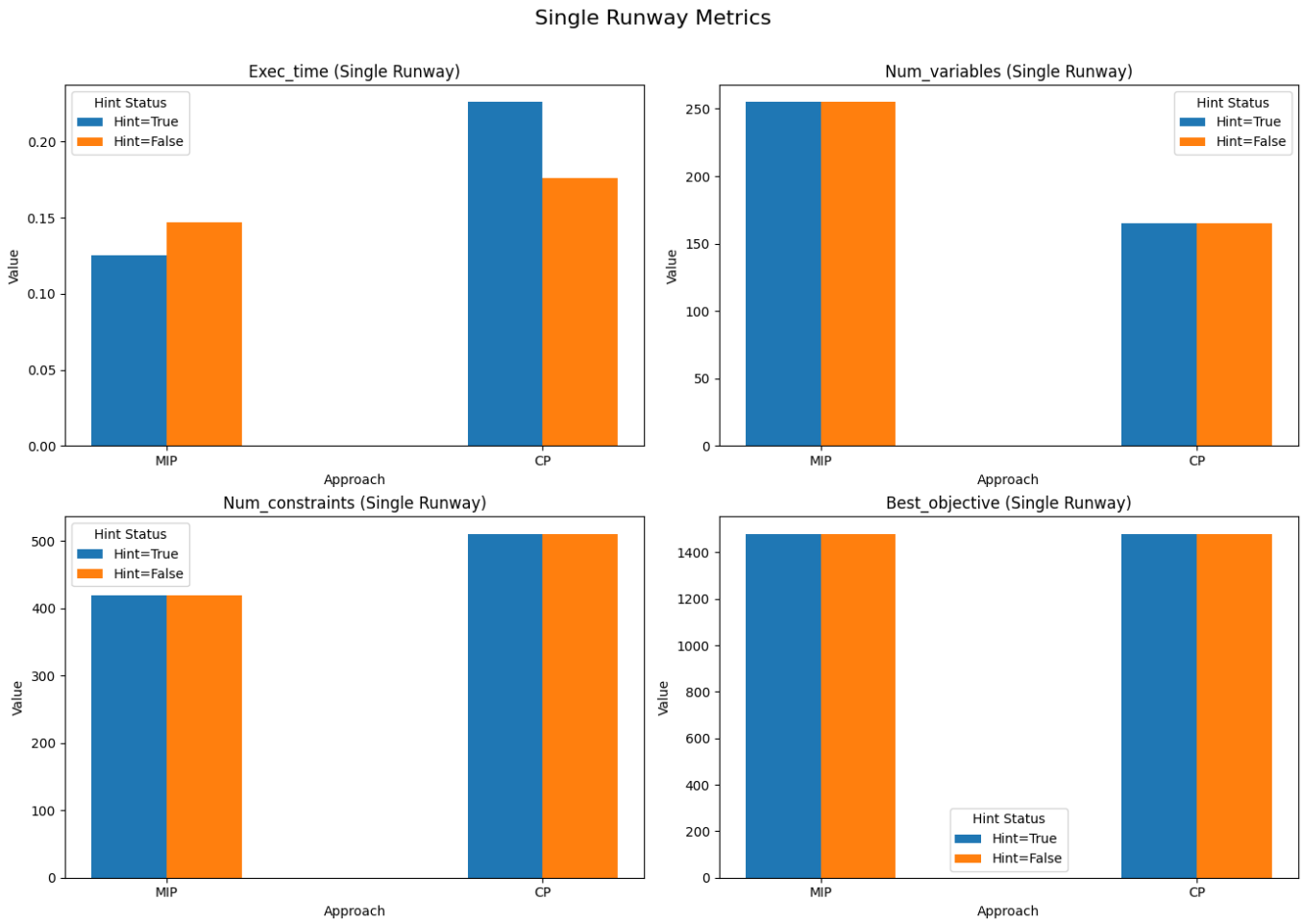

Figure 5: Comparison between MIP and CP metrics for the single runway scenario

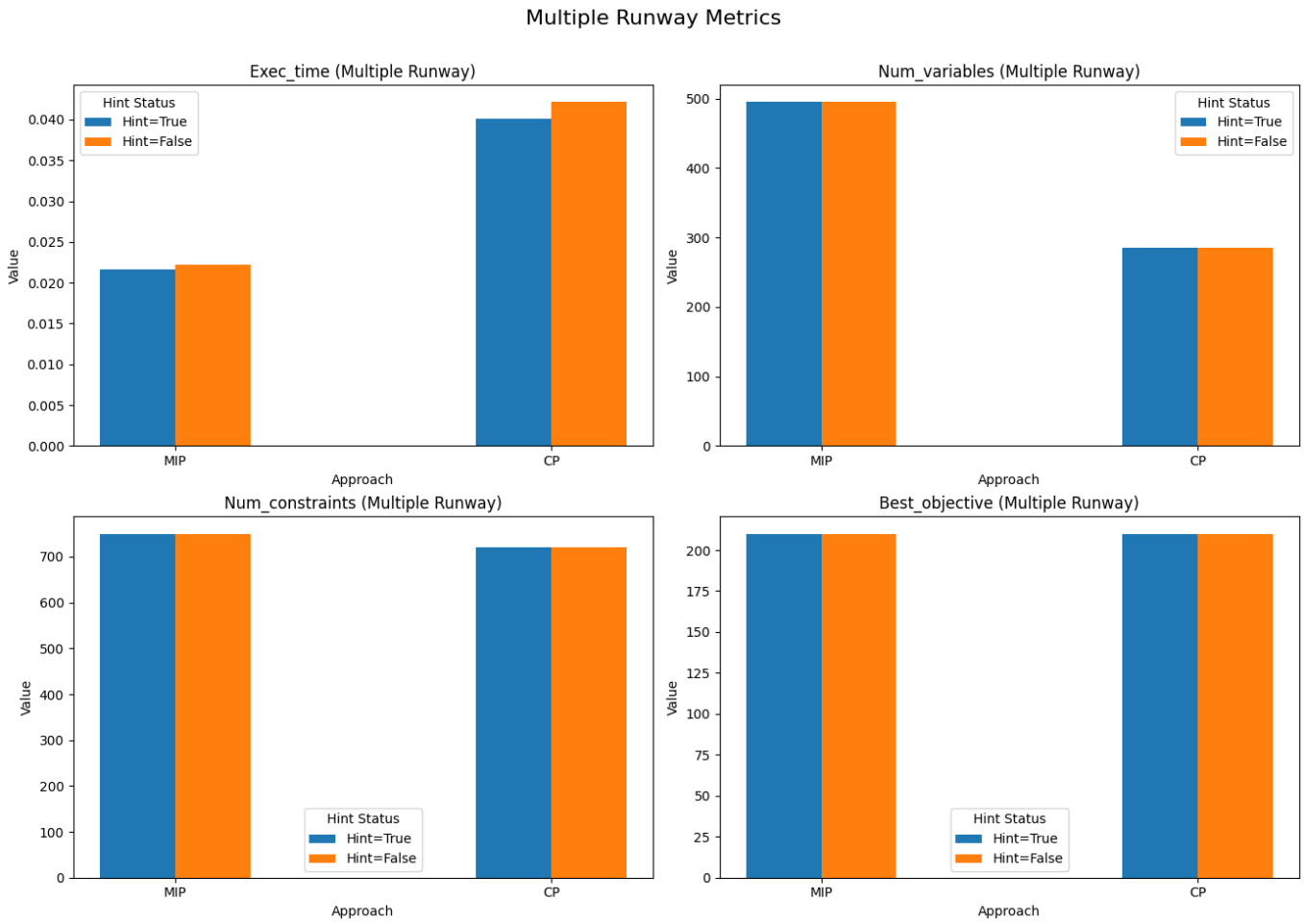

Figure 6: Comparison between MIP and CP metrics for the multiple runways scenario

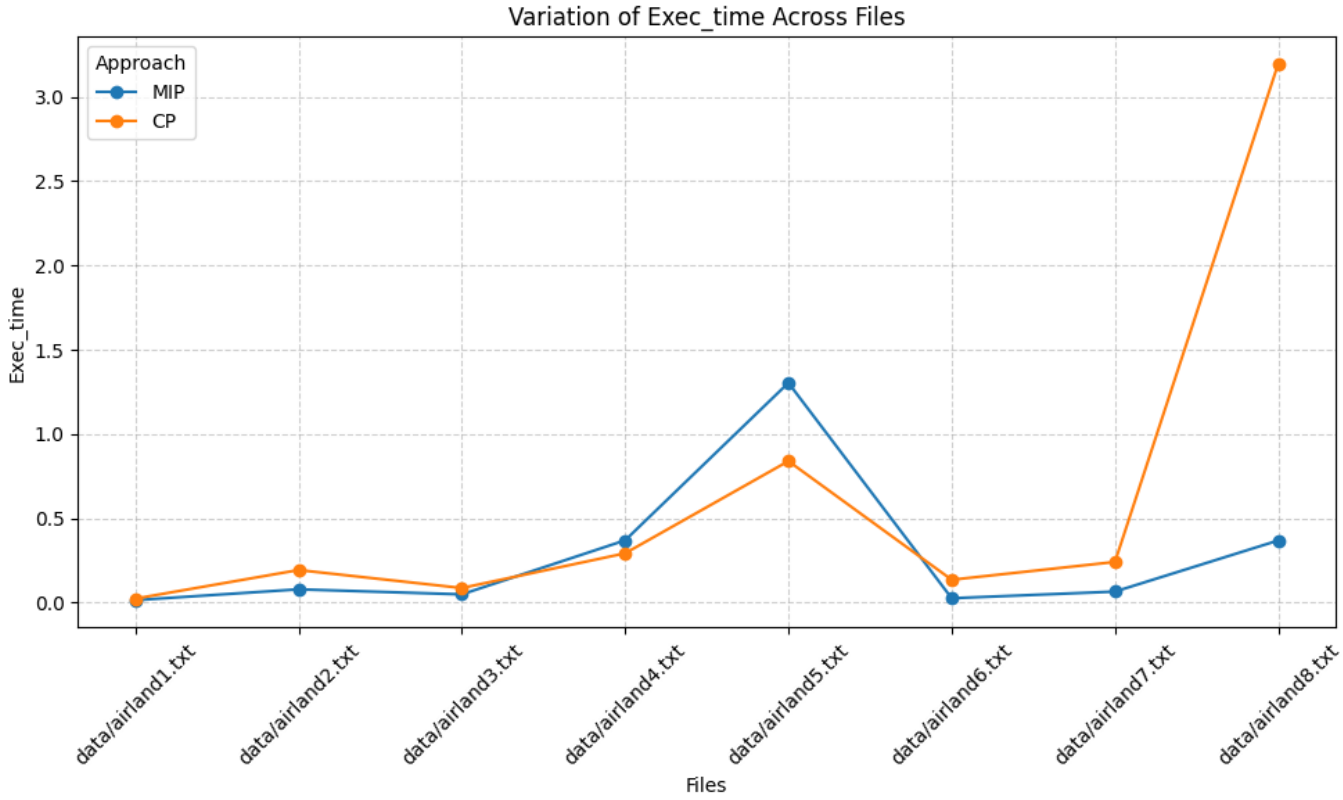

Figure 7: Variation of execution time across files

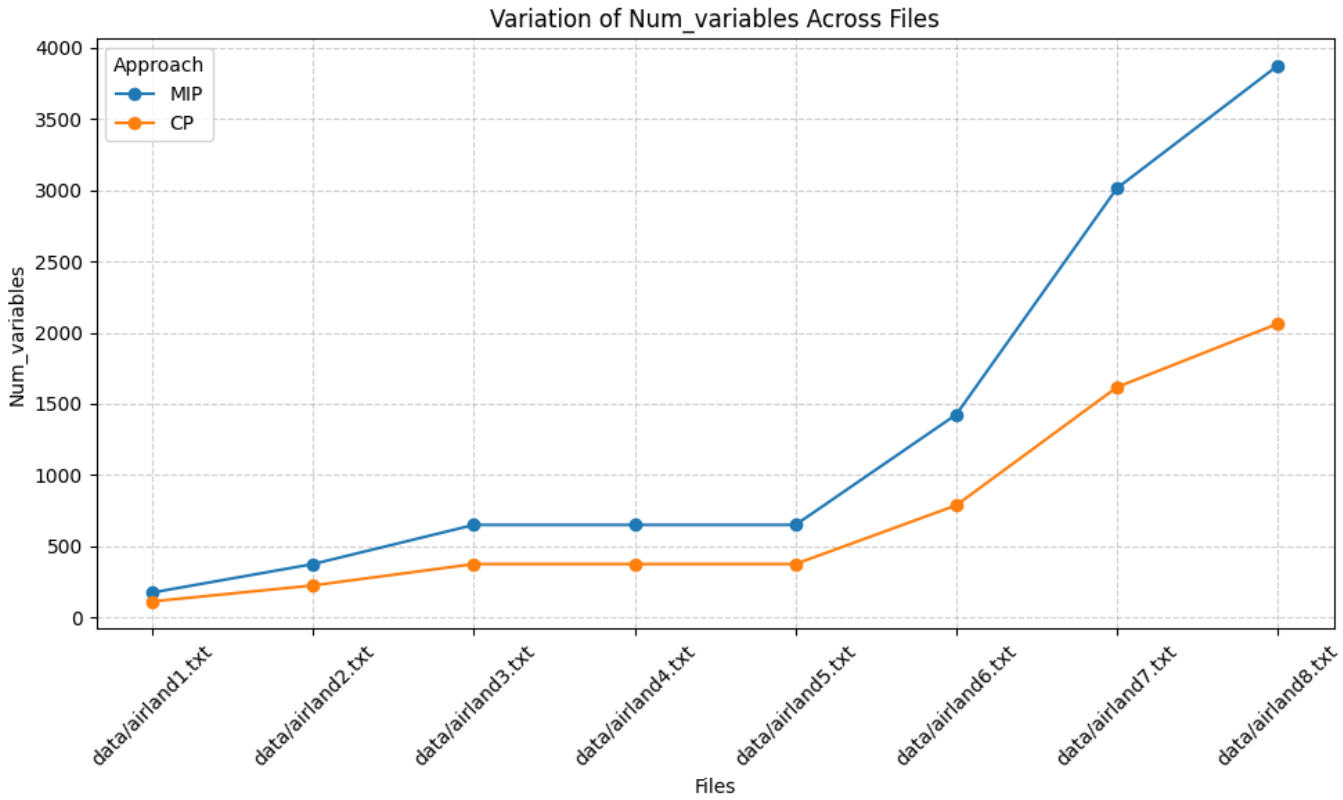

Figure 8: Variation of number of variables across files

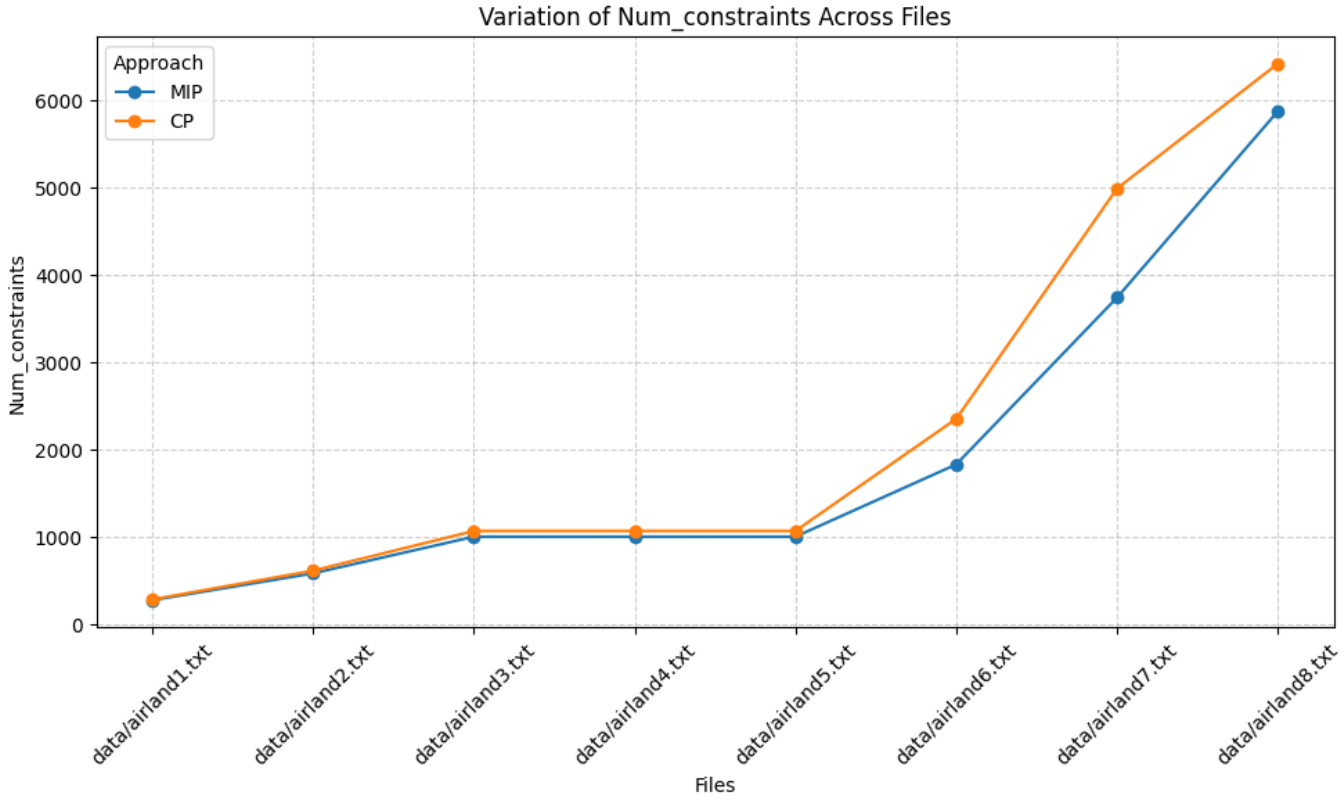

Figure 9: Variation of number of constraints across files

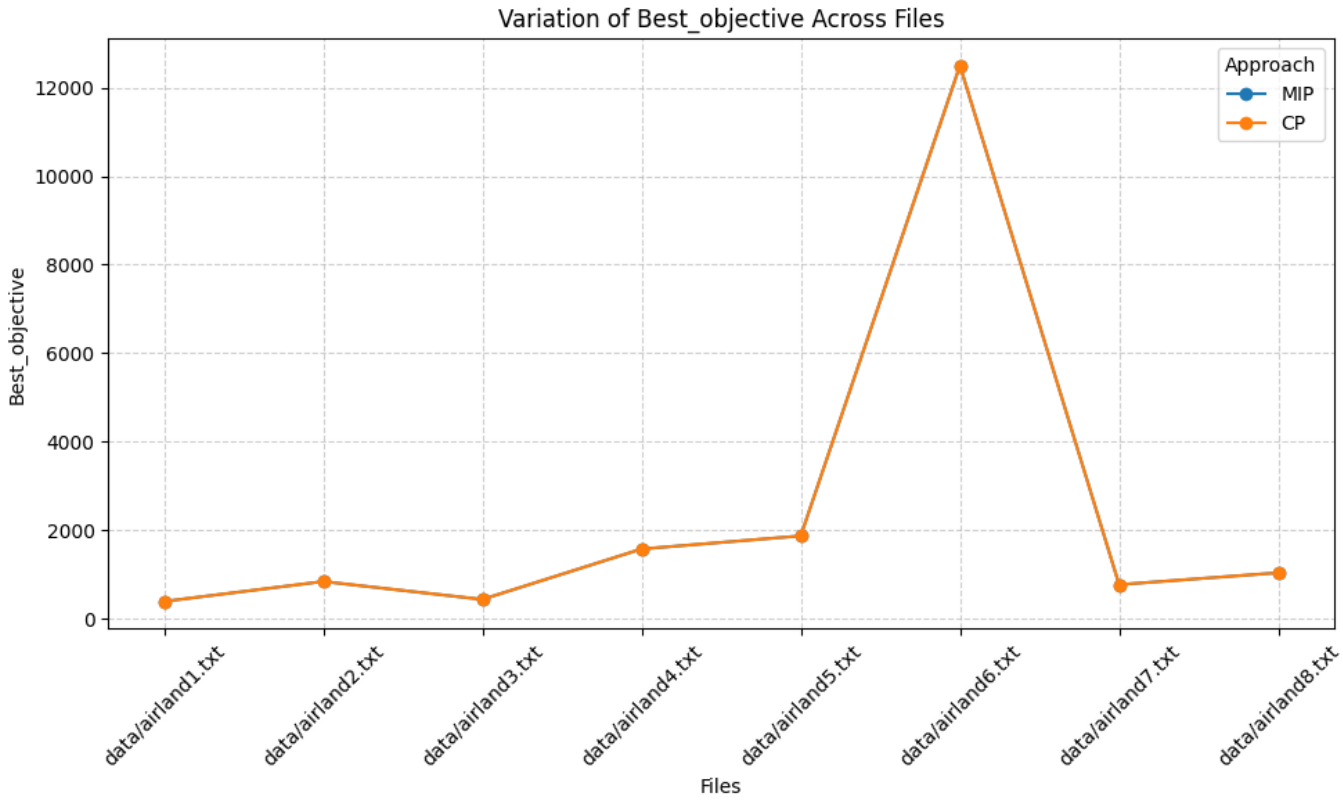

Figure 10: Variation of best objective bound across files

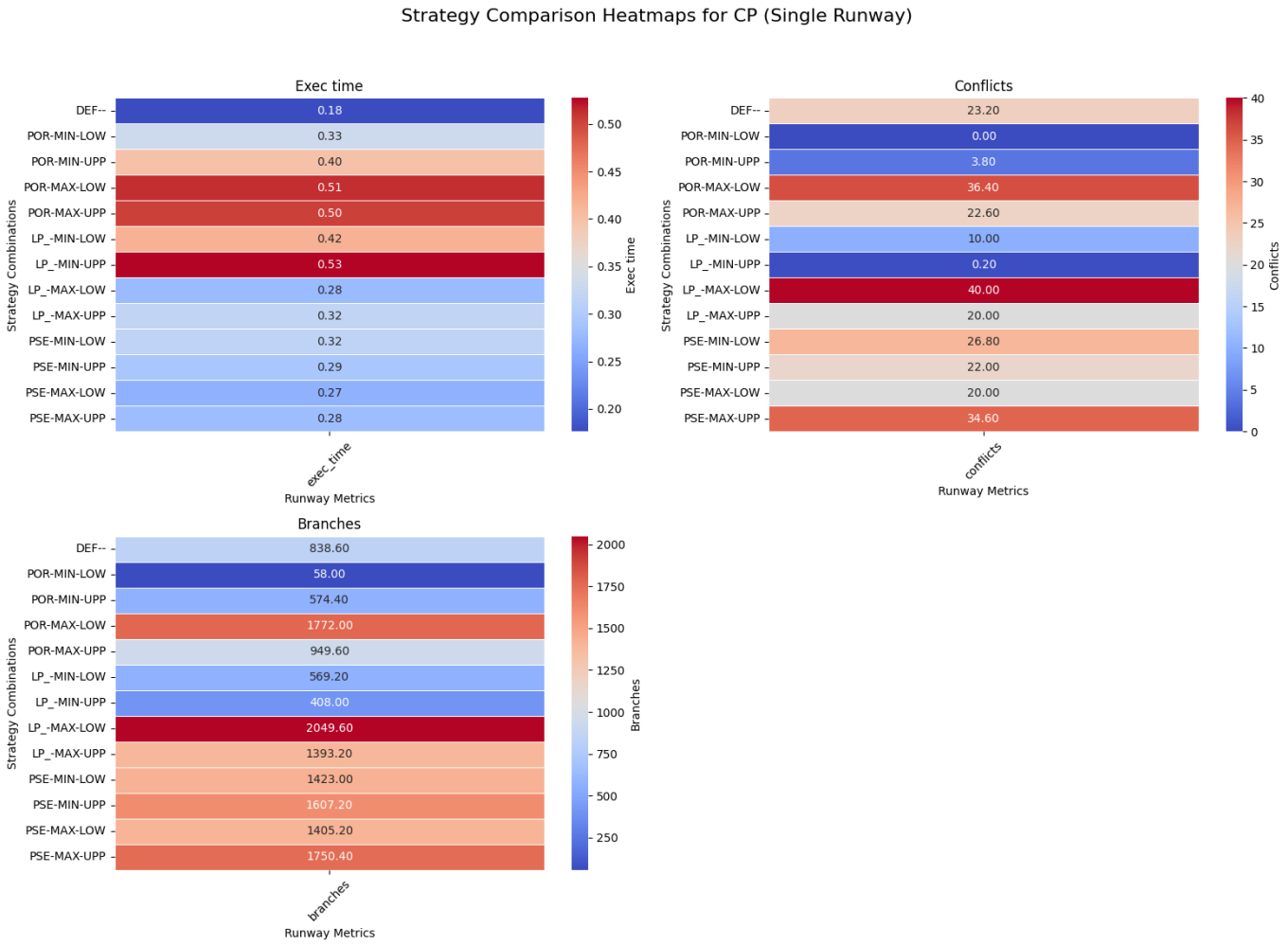

Figure 11: Strategy comparison heatmap for CP in the single runway scenario

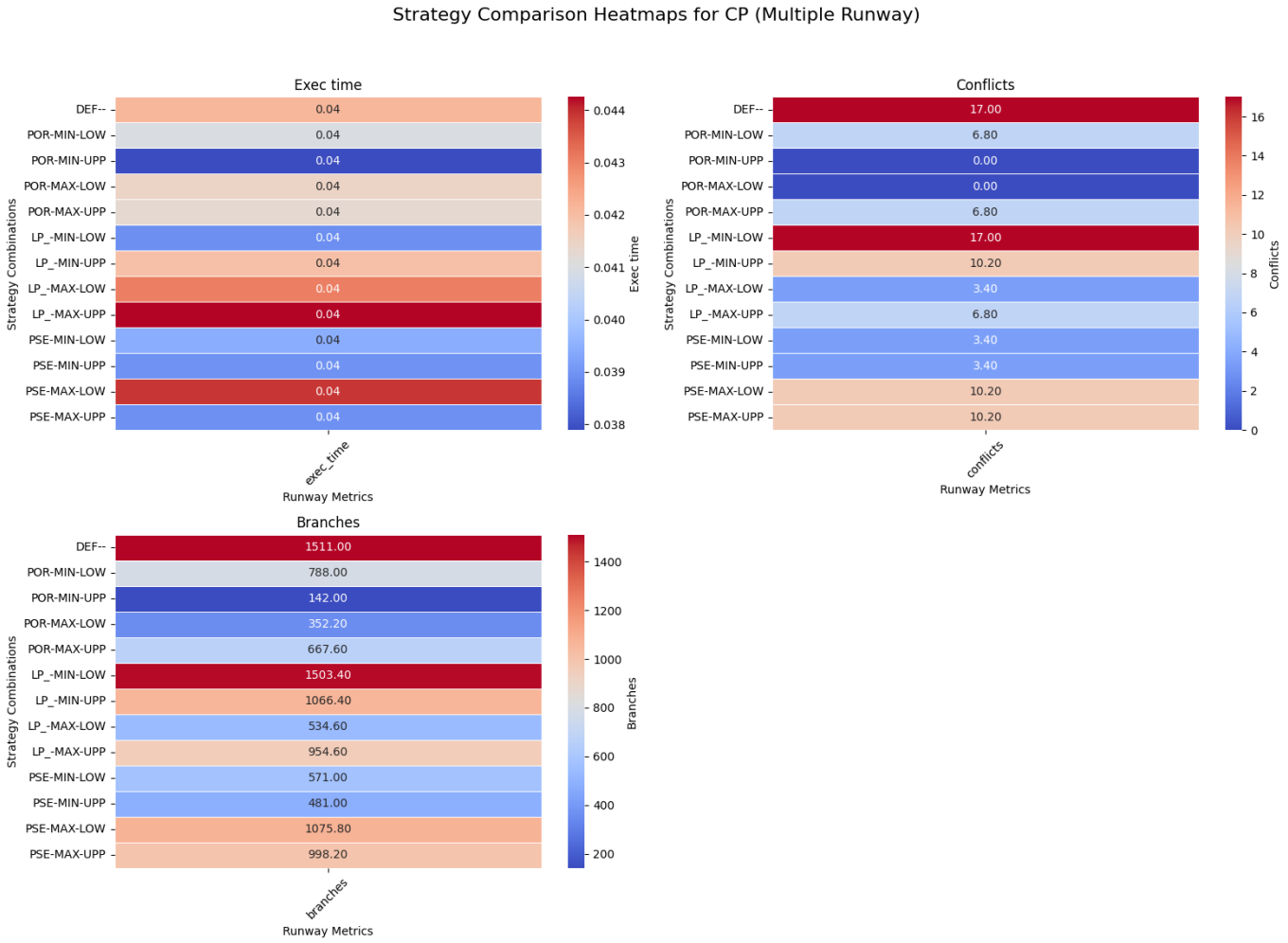

Figure 12: Strategy comparison heatmap for CP in the multiple runways scenario

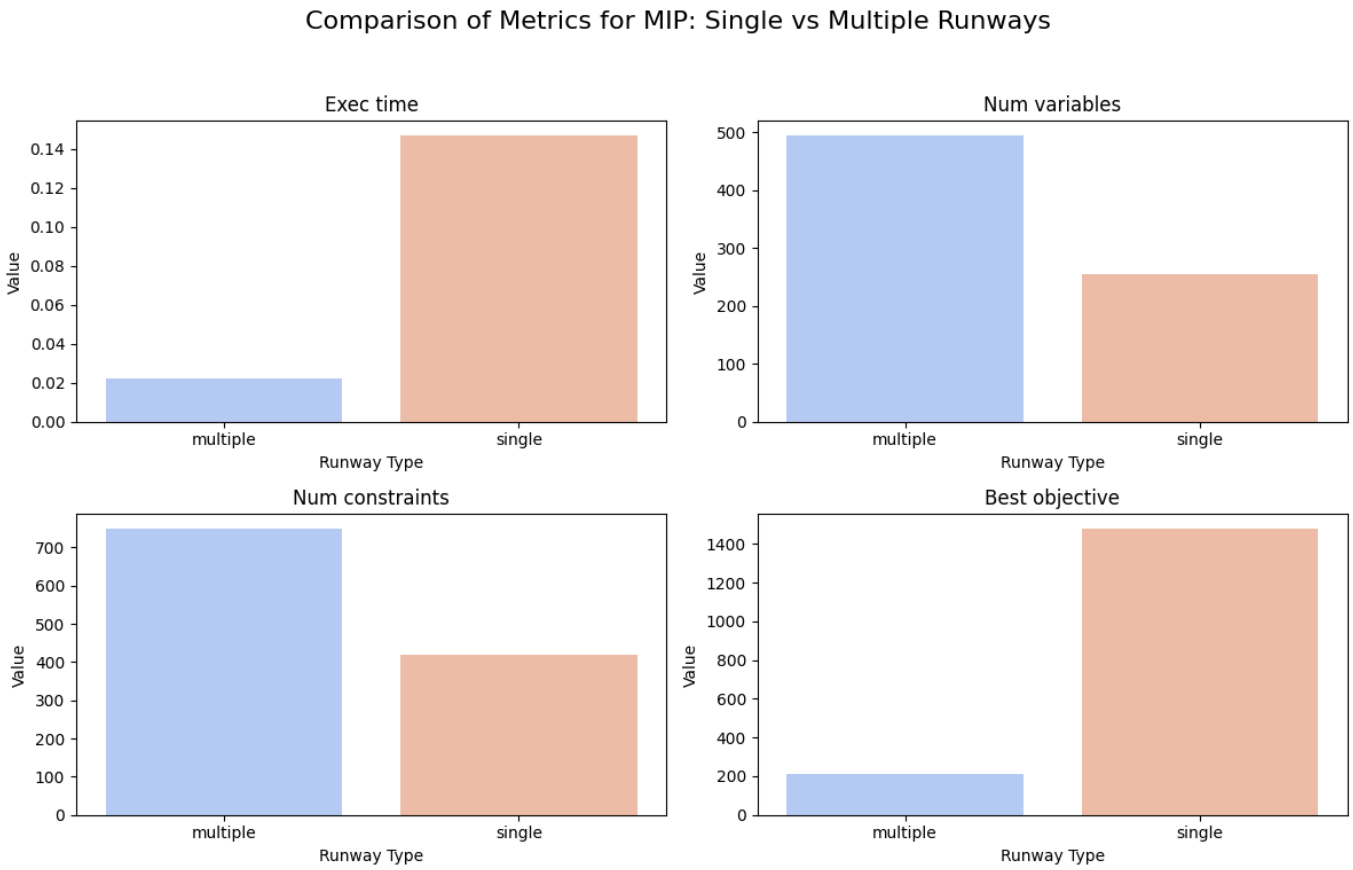

Figure 13: Performance comparison of MIP models in the single and multiple runways scenarios

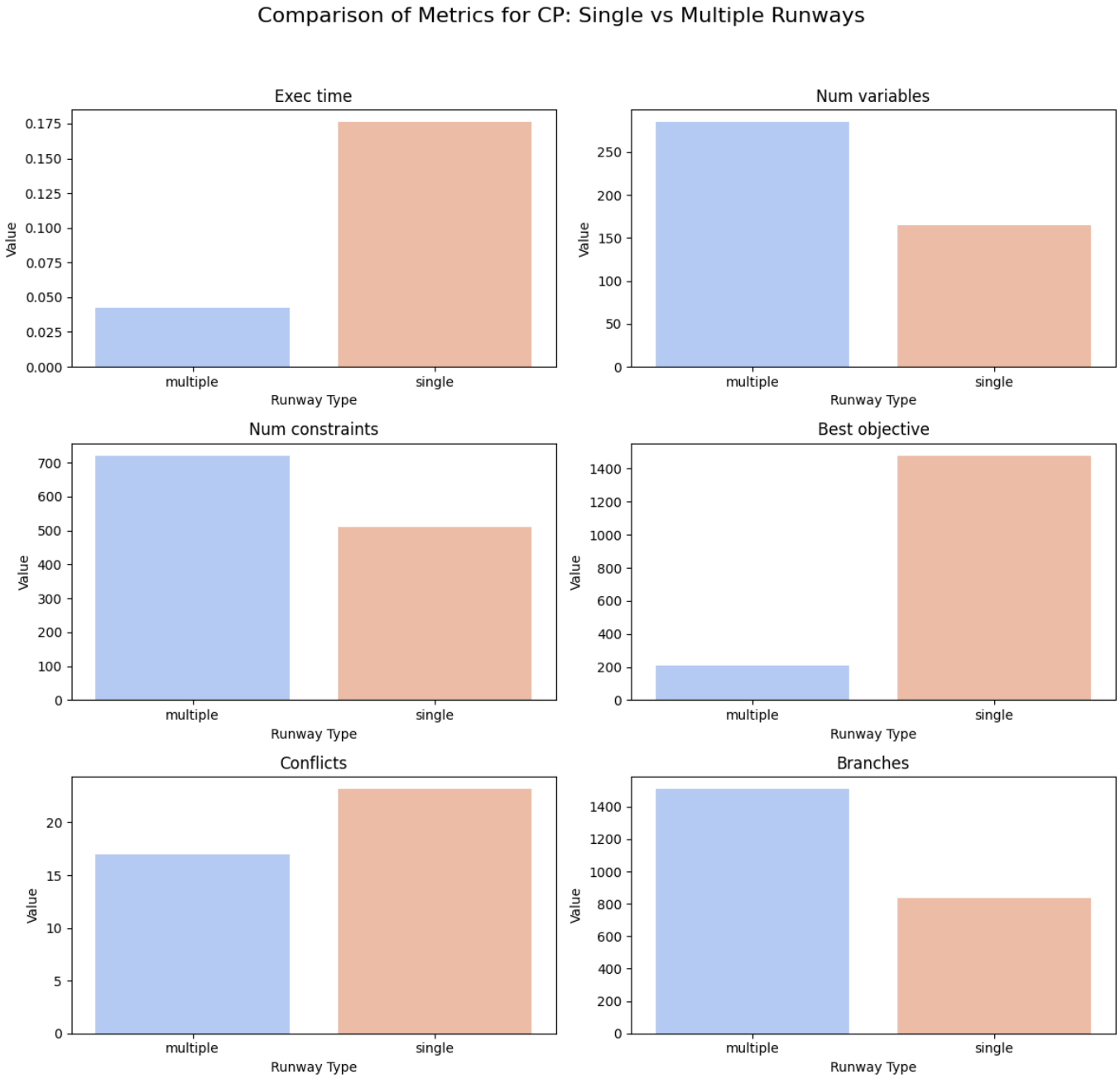

Figure 14: Performance comparison of CP models in the single and multiple runways scenarios