# Machine-Learning-Based Column Selection in Column Generation for Railway Crew Planning with Fairness over Time

**Sjoerd Arts**


*Submitted in fulfillment of the requirements for the degree of Master of Science in Business Analytics at Vrije Universiteit Amsterdam*

**July 31, 2026**

# Machine-Learning-Based Column Selection in Column Generation for Railway Crew Planning with Fairness over Time

Sjoerd Arts

2742201

*Master Thesis*

**First supervisor**: dr. René Bekker

**Second reader**: prof. dr. Mark Hoogendoorn

**Company supervisor**: Marije Siemann (Netherlands Railways)

**Vrije Universiteit Amsterdam**

Faculty of Science

Business Analytics

De Boelelaan 1081a

1081 HV Amsterdam

**Host Organisation**

Netherlands Railways (NS)

Digitization of Operations (DigOps)

Laan van Puntenburg 100

3511 ER Utrecht

July 31, 2026

# Preface

This thesis is the result of my six month internship at *Nederlandse Spoorwegen* which started on February 2nd.

Joining the DigOps department has been a great experience. While reading academic papers to become familiar with the problem, I realized that the authors were the same people standing next to me at the coffee machine. Experiencing that level of expertise in real life is truly inspiring.

I would not have been able to complete this report without the help of several people. First, I want to thank my company supervisor Marije. Your expertise is impressive and you are best described as a *duizendpoot*. Thank you for the valuable feedback always being available to help. I also want to thank Ramon for giving me the opportunity and confidence to write my thesis at DigOps. I really appreciated your genuine interest in how I experienced the internship. Gábor, thank you for taking my cynical vocabulary to a new level. Our technical and everyday conversations were a great part of my time here.

To my fellow interns Tijs and Wolf, thank you for the warm welcome during my first month. Lars, thank you for being there for the full ride. It was great to connect, share our experiences and navigate this internship together.

From the Vrije Universiteit, I want to thank my academic supervisor René. Your sharp ideas and constructive feedback consistently pushed me to think more critically. I also really liked that we could speak about personal things because it gave our meetings a very relaxed vibe.

On a personal note, I am really thankful to my family for always asking how my day was and having dinner ready. This gave me the space to fully disconnect and just enjoy my time off. I could not have succeeded without your support at home. To my friends, thank you for the fun weekends and the resulting lack of sleep. Surprisingly, that exhaustion kept me focused for the weeks ahead, even if Mondays were a bit of a struggle.

Combining operations research techniques with machine learning is a relatively new research area, especially within the dynamic context of passenger railway operations at NS. This makes it a challenge, but at the same time it is interesting to contribute to this field. I often had to remind myself that days of struggle simply belong to a journey like this. Looking back, I am really happy with the personal development I gained from navigating these new experiences and challenges. I trust that this research shows the potential of this combination and provides a solid foundation for the many ways still left to explore.

# Abstract

Netherlands Railways (NS) studies an integrated crew planning approach in which daily duties are constructed and assigned while penalties from individual fairness histories steer the planning process over time. The resulting column generation procedure repeatedly solves a restricted master problem (RMP). Pricing can generate many candidate duties, and inserting too many of them increases RMP size and re-optimization time. This thesis investigates whether expert-guided and machine-learning-based column selection can reduce this computational burden while preserving comparable root-node objective quality.

An expert mixed-integer linear program is adapted from the literature to the set-covering and per-template pricing structure at NS. Its decisions provide labels for a graph neural network (GNN) that represents candidate duties and master-problem constraints as a bipartite graph. The selectors are evaluated at the root node and over the full planning day on real three-base and six-base NS instances. The learned model is trained only on the three-base instance and transferred without retraining to the six-base instance. Hybrid full-day variants use the alternative selector at the root node and return to the NS baseline after the first dive.

The expert requires a template-level safety mechanism to prevent individual pricing subproblems from stalling. It reduces the final root-node RMP by 77.29% and 78.49% on the three-base and six-base instances, respectively. However, its minimal selections cause enough additional iterations to increase total root-node time. The GNN achieves a test balanced accuracy of 78.5% and an AUC-PR of 0.580. Inside the solver, it reduces root-node time by 4.65% on the three-base instance and by 16.52% on the six-base instance, while keeping the objective within 0.89% and 0.84% of the baseline. Over the full planning day, the hybrid GNN reduces total time by 10.11% and 16.27%, respectively. Full-day solution-quality differences cannot be attributed solely to column selection because different root-node pools lead to different diving paths.

The results show that a GNN can learn structural column-selection behavior in a fairness-over-time crew planning problem and can improve computational efficiency while preserving comparable root-node quality. The main remaining challenge in this NS setting is to adapt how many columns are inserted to keep templates progressing, so that RMP reduction does not come at the cost of excessive iterations.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Efficient resource allocation is a fundamental challenge in large-scale transportation systems. Railway operators must coordinate infrastructure, rolling stock, and human resources in order to provide reliable and cost-efficient services. Effective crew and resource planning directly impacts operational performance and service quality (Mertens et al., 2023). Among these planning tasks, the management and assignment of railway personnel is particularly complex. This complexity arises from the large number of operational constraints and the need to maintain fair and sustainable working conditions. For a major passenger railway operator such as Netherlands Railways (NS), planning the work of thousands of train drivers and guards across a dense national network results in large combinatorial optimization problems.

In the field of operations research, this problem is typically referred to as crew planning. Crew planning is traditionally divided into two sequential stages. In crew scheduling, individual train trips are grouped into feasible working days, called duties. For example, a duty might combine a morning trip from Utrecht to Amsterdam with a return service in the afternoon, subject to constraints on maximum working time and required break placement. In crew rostering, the generated duties are subsequently assigned to individual crew members over a longer planning period, such that every employee receives a complete and workable schedule over several weeks. This decomposition keeps each stage tractable, but limits the extent to which duties can be tailored to the specific circumstances of individual crew members.

Fairness has long been a core concern in the planning operations of NS. At the company, these considerations are governed by the Sharing-Sweet-and-Sour (SS&S) rules (Abbink et al., 2005). These rules were introduced following nationwide railway strikes in 2001. Their goal is to distribute `sweet` (attractive) and `sour` (unattractive) work fairly among the crew members. For example, in the case of guards, `sweet` work includes trips on modern rolling stock units, while `sour` work involves trains with a higher risk of passenger aggression. NS has 28 crew bases, each corresponding to a large station with a fixed group of crew members who operate trips nationwide but must start and end their work at this station. Traditionally, workload allocation focused on balancing these attributes across each crew base. However, recent analysis has shown that aggregated measures do not guarantee fairness for individual employees, often leading to unequal outcomes among crew members at the same crew base (van Rossum et al., 2024).

To address these limitations, van Rossum et al. (2024) proposed a novel integrated

approach that combines duty construction and individual assignment within a rolling horizon framework. A penalty-based feedback mechanism dynamically steers assignments based on each crew member's accumulated work history, gradually improving individual fairness compliance over time. While this perspective improves the ability to manage fairness over time, it also increases the complexity of the planning problem. The number of feasible duties that can be constructed from a given set of train tasks is extremely large, and the assignment cost of a duty depends on the individual history of the specific crew member to whom it is assigned. Evaluating all possible options directly is infeasible. The planning system therefore relies on column generation, which is a decomposition technique commonly applied to large-scale crew scheduling and crew planning problems with set-covering or set-partitioning formulations (Desaulniers et al., 2005).

Rather than evaluating all options at once, column generation iteratively solves a restricted master problem (RMP) over a limited subset of duties and generates additional candidate duties through a pricing subproblem. Each pricing step can produce many candidates with negative reduced cost, but adding all of them to the master problem is computationally wasteful and can slow convergence due to degeneracy. A selection step therefore determines which candidates are actually inserted. In the NS prototype solver, this step is governed by a task-disjoint heuristic that filters candidates based on reduced cost and task overlap. Although efficient, this heuristic evaluates duties largely in isolation and does not consider their broader impact on the optimization process. Furthermore, van Rossum et al. (2024) observe that a substantial portion of the solving time in their experiments is spent repeatedly re-optimizing the RMP. This highlights the importance of managing the number and quality of columns added during the process, pointing to the column-selection step as a key lever for improving computational efficiency.

The prototype solver implements this integrated planning framework and is used for experimental studies, with the long-term goal of transitioning to such an individualized planning approach in practice. The production solver currently used at NS relies on a different planning approach and does not incorporate this individual feedback mechanism. Improving the column-selection step remains a key research gap, and this thesis investigates alternative selection strategies within the prototype solver environment to address it. Throughout this thesis, the current task-disjoint selector serves as the benchmark against which these alternative strategies are evaluated.

Machine learning offers a promising approach to support the column-selection step. Rather than relying solely on heuristics, data-driven models can learn patterns from previous optimization runs to identify which duties are most likely to improve the solution. The work of Morabit et al. (2021) provides a principled framework for this direction. In their approach, a mathematical expert first identifies a small subset of generated columns that is expected to improve the next RMP solve by evaluating the optimal combination of these columns rather than assessing them in isolation. A supervised learning model is then trained to imitate the expert, replacing the computationally expensive expert with a fast prediction at deployment time. Their study focuses on the root-node linear relaxation, that is, the first linear programming phase of column generation before any integer fixing decisions are made. In that setting, the RMP remains comparable across iterations, and the effect of column selection can be isolated cleanly. Their experiments on vehicle and crew scheduling problems demonstrate meaningful runtime reductions without compromising solution quality. Inspired by these ideas, this research explores the use

of a graph neural network (GNN) to guide column selection within the NS prototype solver. By representing duties and constraints as a graph, the GNN can capture structural relationships in the problem and predict which duties are most valuable to include at each iteration.

This thesis investigates the feasibility and performance of expert-guided and learning-based column selection within the NS setting through a two-part contribution. The first part formulates and evaluates an expert selector within the NS column generation framework. It examines whether the expert can be adapted to the structural characteristics of the NS problem and what insights its selection behavior provides into the column generation process. The second part investigates whether a supervised learning model can imitate the expert's decisions and whether such a learned selector can improve the computational efficiency of the column generation procedure while preserving solution quality. Following the evaluation strategy of Morabit et al. (2021), the primary analysis is conducted at the root node, with additional full-day experiments included to assess broader operational impact. The investigation is guided by the following four research questions, where the first two address the formulation and behavior of the expert, and the latter two evaluate the feasibility and performance of the learning model:

RQ1. Can an expert be formulated and applied to select a smaller subset of generated columns within the column generation framework for the NS crew planning problem with fairness over time?

RQ2. What insights does the expert selection approach provide into the column generation process compared to the task-disjoint baseline, in terms of RMP composition, convergence behavior, computational effort, and solution quality?

RQ3. Can a supervised learning model imitate the expert selection decisions within the NS column generation setting and predict which generated columns should be selected?

RQ4. Does the learned column-selection policy improve computational efficiency while maintaining solution quality, relative to the task-disjoint baseline and the expert?

The remainder of this thesis is structured as follows. Chapter 2 presents an overview of the current crew planning problem with fairness over time at NS. Chapter 3 reviews relevant literature on crew planning, fairness, machine-learning-based methods for optimization, and positions the contribution of this thesis relative to prior work. Chapter 4 details the expert formulation, the machine learning pipeline including feature construction and model architectures, and the experimental design. Chapter 5 presents the experimental results and performance analysis. Chapter 6 discusses the findings in their operational and computational context. Chapter 7 concludes the thesis. Mathematical background on column generation and machine learning is provided in Appendix A.

# Chapter 2

# Problem overview

This chapter explains the planning problem that motivates this thesis. Section 2.1 first positions crew planning within the railway planning process and introduces the transition from the current NS process to the future template-based approach. Section 2.2 defines the operational planning problem alongside the main planning objects. Section 2.3 explains how daily decisions are linked through a rolling history and a penalty-based feedback mechanism. The daily mathematical model is presented in Section 2.4. Finally, Section 2.5 describes the column generation heuristic and identifies the column insertion decision studied in this thesis.

## 2.1 Crew planning at NS

Railway planning consists of multiple interdependent stages, each refining the level of detail of the operational plan. This is illustrated in Figure 2.1. The general objective is to provide reliable and efficient train services while maintaining acceptable working conditions for staff. At the strategic level, network design determines where infrastructure investments are needed to address capacity limitations. Based on this infrastructure, line planning specifies which train services are operated and how frequently they run. These lines are then translated into a detailed schedule, where departure and arrival times are assigned with respect to track and station constraints. Subsequently, rolling stock scheduling assigns train units to the planned trips to ensure sufficient passenger capacity. Crew planning completes this process by assigning personnel to operate the planned services. In practice, these plans are subject to real-time adjustments during operations. These disruptions require modifications to previously constructed schedules.

Crew planning is traditionally divided into crew scheduling and crew rostering (Borndörfer et al., 2018; Caprara et al., 1997). The crew scheduling phase combines timetable tasks into feasible duties. A single task might represent a train trip from Utrecht to Amsterdam at a specified time, whereas a duty combines several such trips into one complete working day. Crew rostering subsequently assigns the generated duties to crew members over a longer planning horizon. For example, a weekly roster might assign an early duty on Monday, a late duty on Tuesday, and a designated rest day on Wednesday. Such a decomposition makes large planning instances easier to solve because duty construction and long-term assignment rules are handled separately.

Separating the two phases also limits the extent to which duties can be tailored to

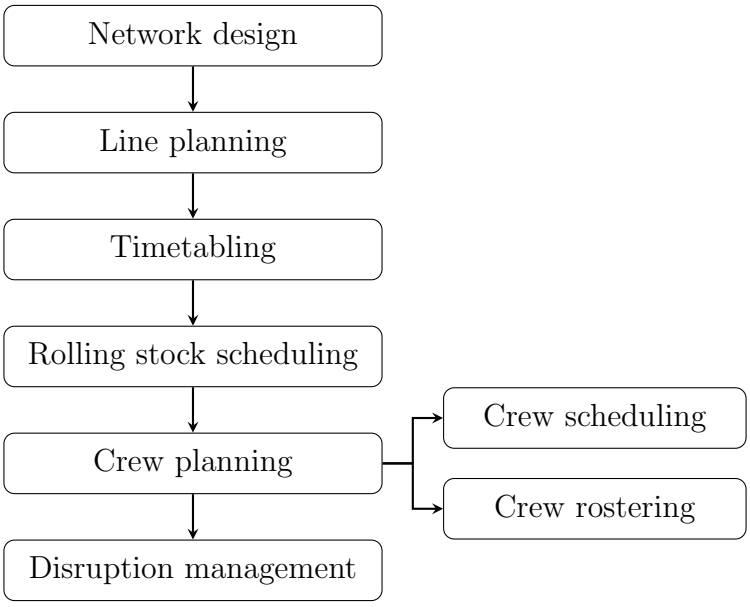



**Figure 2.1:** Overview of the railway planning process, detailing the crew planning decomposition.

individual circumstances. Duties are created before planners know which specific person will perform the work. The later rostering step can exchange complete duties between employees, but the step cannot alter the underlying task content. A crew member who has recently received an accumulation of undesirable work might therefore still have to be assigned one of the fixed duties available in the pool. Constructing a newly tailored duty would lead to a better individual work package, but the traditional sequential approach prevents such dynamic adjustments.

At NS, crew members are organized into 28 geographically distributed crew bases across the Netherlands. The locations of the bases are shown in Figure 2.2. Each crew base relies on its own dedicated workforce, and a regular duty must start and end at the exact same base. The type of work available to different bases varies depending on the specific location within the network. This structure introduces assignment restrictions and leads to regional variation in both workload composition and feasibility. Fairness considerations have traditionally been captured through Sharing-Sweet-and-Sour (SS&S) principles (Abbink et al., 2005). These rules aim to balance desirable (`sweet`) and undesirable (`sour`) work characteristics across crew bases. The rules operate strictly at the aggregate crew-base level. Operating at an aggregate level means that a base can satisfy a fairness rule overall even when individual employees within that base receive substantially different personal work packages. The current planning process introduces further complications because the operational timetable and rolling stock plan can change due to track maintenance, special events, or other system updates. Planned duties might consequently be modified long after publication but before the actual day of operation. Employees therefore receive limited certainty that the detailed duty initially associated with a roster will remain unchanged. Frequent modifications can interfere with personal plans and ultimately cause the realized distribution of work to deviate from the intended crew-base balance.

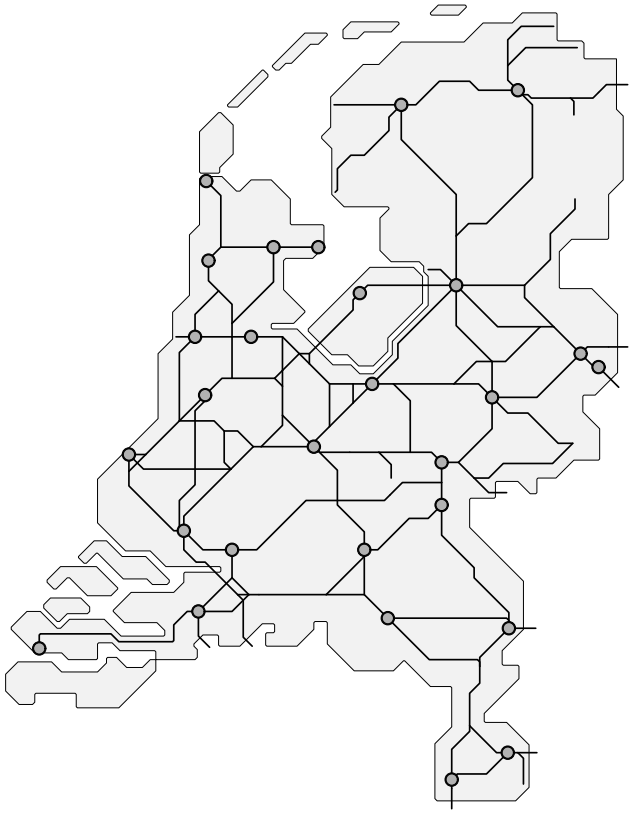


**Figure 2.2:** Location of the 28 NS crew bases across the Dutch railway network. The operated tracks are indicated by black lines. Each crew base is indicated by a small gray circle.

To address the outlined challenges, NS is considering a future planning process that separates early capacity commitments from detailed task allocation. The difference between the current and future processes is illustrated in Figure 2.3, which is based on van Rossum (2025). In the current process, generic duties are constructed first and later assigned through cyclic rosters. In the future process, the tactical phase determines template-based individual rosters instead of fully detailed duties. A template specifies a day and a time window during which a crew member can be assigned work, without fixing the exact tasks in advance. A typical template spans about nine hours, while the duty later placed inside it may last about eight hours. This allows a single template to accommodate multiple feasible duties. That provides additional flexibility and enables duties to be constructed closer to the day of operation using more accurate information. At the same time, the template gives the employee reliable advance information regarding the time period in which the shift will occur.

The hierarchical relationship between rosters, templates, duties, and tasks is shown in Figure 2.4. A complete roster records the assigned work and rest days for a single crew member over a specified period. On a particular workday, the roster contains a template, which acts as an overarching time window. During the subsequent operational planning phase, a concrete duty is constructed inside the allocated window. The duty in turn consists of an ordered sequence of specific train tasks and might also contain required activities such as a meal break. The template framework therefore reserves employee capacity without determining the exact work content.

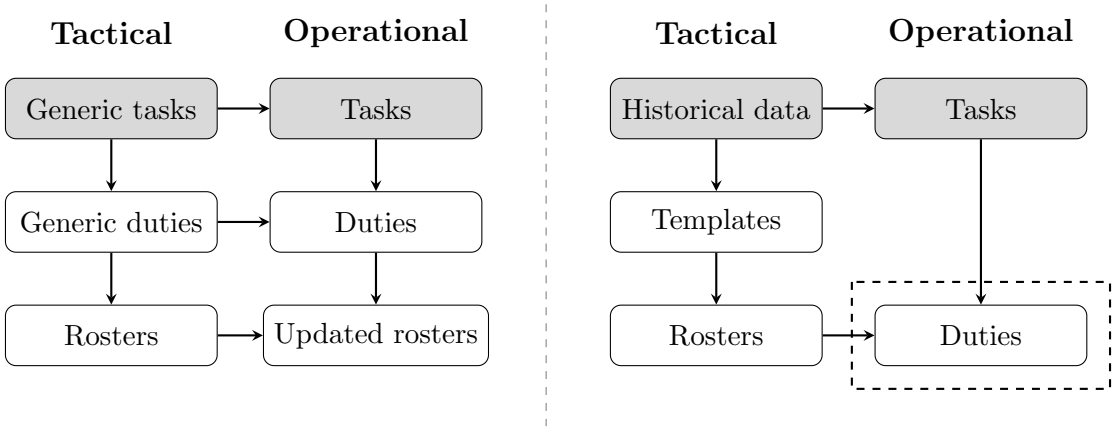



**Figure 2.3:** Comparison of the current (left) and future (right) crew planning processes. Crew planning decisions are indicated by white rectangles, inputs by shaded rectangles. The dashed box around the operational duties highlights the area investigated in this thesis.

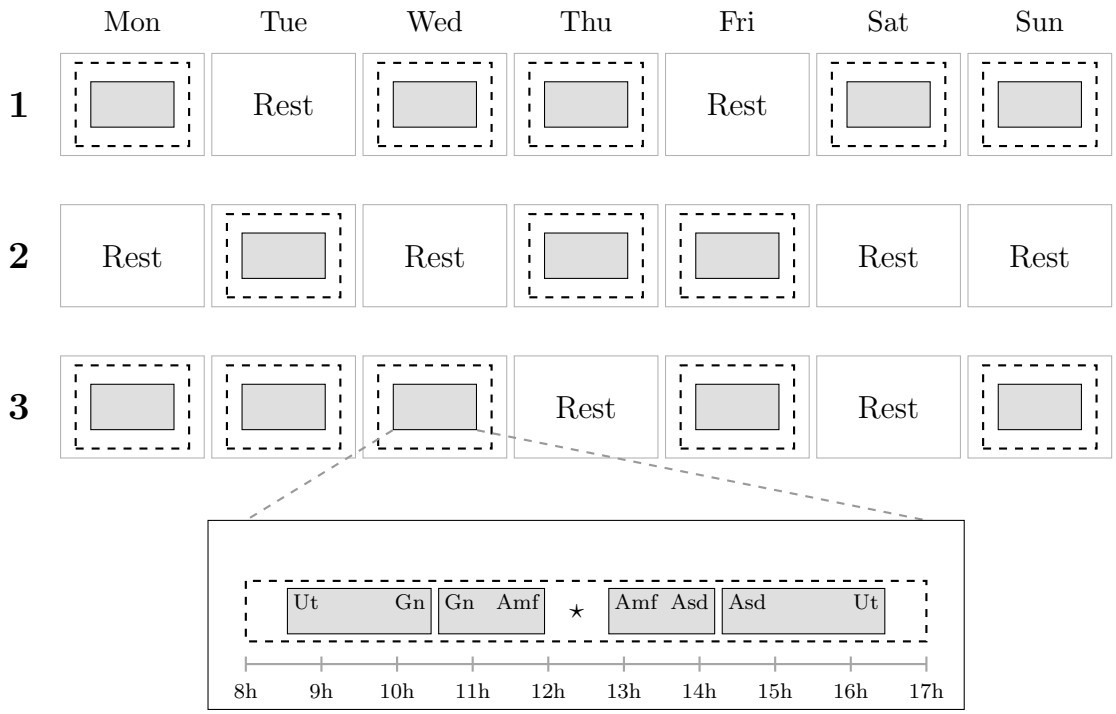



**Figure 2.4:** Example of a three week roster for a crew member based in Utrecht (Ut) showing assignments for each day of the week. The solid upper grey blocks represent specific duties. The dashed block around a duty represents the template. The absence of blocks signifies a rest day. The duty on Wednesday in week three is highlighted and expanded to show its internal structure. The grey blocks represent individual tasks in that zoomed view. For each task the start station is indicated on the top left and the end station on the top right. Other station abbreviations include Amersfoort (Amf), Amsterdam (Asd), and Groningen (Gn). The star marks a scheduled meal break. The time line below the task sequence is aligned exactly with the template time window.

Recent work compares an integrated approach that constructs duties for specific crew members with a sequential benchmark in which duties are first generated and assigned afterwards (van Rossum et al., 2024). On practical NS instances, the integrated approach attains substantially higher individual SS&S satisfaction at little additional computational cost. The larger simulation of van Rossum et al. (2026) subsequently applies the future process to 3,265 guards over the year 2024 and obtains high conformance for most rules. The current process remains the production setting at NS, while the future process is implemented in a prototype solver for research and evaluation. This thesis studies the operational column generation procedure in that prototype and therefore adopts its template-based rosters and individual SS&S setting.

## 2.2 Operational problem setting

The present thesis focuses on the operational phase marked by the dashed box in Figure 2.3. At the operational stage, each crew member already has a template-based roster. The train tasks for the day follow directly from the timetable and rolling stock plan. The goal of the resulting operational planning problem is to transform the template-based rosters into actual rosters by assigning a compatible duty to each template on each day. The resulting roster must ensure that all tasks are covered, all relevant constraints and regulations are satisfied, and the workload is distributed fairly among crew members over time.

Figure 2.4 summarizes the terminology. A task is an individual train trip with a start station, an end station, and corresponding times. A duty is an ordered sequence of tasks that forms one feasible working day. A template is a dated time window in the roster of one crew member and acts as a placeholder for a duty. A roster records the assigned work and rest days of that crew member over the planning period.

Two types of feasibility determine whether a duty can be assigned to a template. First, the duty itself must satisfy the operational rules that define a feasible working day. Second, the duty must be compatible with the template to which it is assigned.

Duty feasibility is governed by strict operational rules. Consecutive tasks must connect logically in both location and time. Each regular duty must start and end at the same crew base, respect a maximum duration, and include a suitably positioned meal break. These requirements ensure that a feasible duty cannot be formed by combining arbitrary tasks.

The template-based rosters are constructed in advance to provide a feasible capacity pattern. Their time windows are arranged such that duties fitting the available templates can be combined without violating the roster structure. In the setting studied here, the relevant link between a daily duty and the roster is therefore template compatibility. The duty must lie within the time window reserved by the template. Rules that determine the sequence of workdays and rest days are handled when the template-based roster is created. They are not modeled again in the daily problem. The operational problem determines the work content inside the available templates rather than redesigning the roster itself.

In addition to feasibility, the distribution of work across crew members is governed by SS&S rules. Each rule is defined by an attribute, a target score, and an evaluation period. Attributes capture `sweet` or `sour` characteristics of work. Scores can be computed in

different ways. Time-averaged attributes measure the fraction of working time associated with a characteristic, duty-averaged attributes are obtained by averaging over duties, and cumulative attributes track occurrences over time. The resulting score is constrained by a lower bound for `sweet` attributes or an upper bound for `sour` attributes. In this setting, the following six attributes are considered.

- *Type-A work* (`sweet`, time-averaged): work on preferred services such as intercity trains.
- *Aggression* (`sour`, time-averaged): work with elevated risk of passenger aggression. This attribute applies to guards only.
- *Double-decker work* (`sour`, time-averaged): physically demanding work on double-decker trains. This attribute applies to guards only.
- *Duty length* (`sour`, duty-averaged): preference for shorter working days.
- *Repetition* (`sour`, cumulative): frequency of traversing the same track segments.
- *Track variation* (`sweet`, cumulative): diversity of covered track segments relative to those accessible from a crew base.

In practice, these rules are not enforced daily, as doing so would be too restrictive. Instead, fairness is evaluated over a longer planning period and becomes a dynamic aspect of the planning problem, since the fairness state of a crew member depends on previous assignments. Most rules are evaluated over the last 45 duties of a crew member, while other attributes use different evaluation horizons. For example, average duty length is assessed over a shorter window, and track variation is evaluated using both a 45-duty rolling window and a yearly rolling window (van Rossum et al., 2026). Consequently, a single duty can temporarily move a fairness score in an unfavorable direction, provided that later assignments restore the long-term balance. This dependence on historical assignments motivates the rolling-horizon procedure described next.

## 2.3 Rolling-horizon planning and feedback

The operational planning process proceeds strictly one day at a time. For each individual day $t$, the available templates, the train tasks that must be covered, and the accumulated work histories of the crew members serve as the primary inputs. A daily mathematical optimization problem assigns a specific duty to every active template. The assignment decisions are then permanently fixed, the corresponding work is added to the crew histories, and the algorithmic procedure advances to day $t + 1$. The continuously updated histories therefore directly influence the structure of the next daily problem. This rolling-horizon approach is essential because fairness depends on accumulated work history, causing the final balance between crew members to emerge gradually over time.

The penalty-based feedback mechanism of van Rossum et al. (2024) connects these daily problems. For an SS&S attribute $a$, the penalty of assigning a piece of work with attribute score $u$ to a crew member with history score $s$ is defined in Equation 2.1.

$$f_a(u, s) = g_a(u) \times h_a(s) \tag{2.1}$$

The function $g_a$ measures how strongly the new work contributes to attribute $a$. The function $h_a$ measures the current fairness state of the crew member based on their history corresponding to attribute $a$. For a `sour` attribute, $h_a(s)$ increases whenever the individual history score approaches or exceeds the designated maximum target. Assigning additional undesirable work with a large value of $g_a(u)$ consequently incurs a high assignment penalty. For a `sweet` attribute, the scoring direction is mathematically inverted so that assigning work that helps a crew member reach the required minimum target is actively favored. As a result, the same duty may have different costs for different crew members, even when the assigned task sequence is identical.

The dynamic interaction creates a closed feedback loop. The duties assigned on day $t$ update the historical attribute scores of the crew members, which in turn determine the assignment penalties for day $t + 1$. Through this rolling mechanism, the daily optimizer gradually steers individual scores toward their corresponding targets without requiring the entire planning horizon to be formulated as a single large-scale mathematical model. This approach is particularly relevant in the guard setting, because they are evaluated on a richer set of attributes. This case is used throughout this thesis, including the experiments.

## 2.4 Mathematical formulation

The daily optimization model follows van Rossum et al. (2024) and van Rossum et al. (2026). Let $t$ denote the day being planned. Let $K_t$ be the set of tasks that must be covered on day $t$, and let $R_t$ be the set of active crew members with a template on that day. For each crew member $r \in R_t$, the set $\Delta_r^t$ contains the feasible duties that fit the template of $r$ on day $t$ and satisfy all duty rules.

For every duty $\delta \in \Delta_r^t$, let $c_{rt}^\delta$ denote the penalty cost of assigning that duty to crew member $r$ on day $t$. This coefficient aggregates the contribution of the duty across all SS&S attributes and incorporates the crew member's historical scores for these attributes through the feedback mechanism in Equation 2.1. Let $a_{tk}^\delta$ equal 1 if duty $\delta$ covers task $k \in K_t$, and 0 otherwise. The binary variable $x_{rt}^\delta$ equals 1 when duty $\delta$ is assigned to crew member $r$ on day $t$.

The daily planning problem can then be formulated as

$$\min \quad \sum_{r \in R_t} \sum_{\delta \in \Delta_r^t} c_{rt}^\delta x_{rt}^\delta \tag{2.2}$$

$$\text{s.t.} \quad \sum_{r \in R_t} \sum_{\delta \in \Delta_r^t} a_{tk}^\delta x_{rt}^\delta \geq 1 \qquad \forall k \in K_t \tag{2.3}$$

$$\sum_{\delta \in \Delta_r^t} x_{rt}^\delta = 1 \qquad \forall r \in R_t \tag{2.4}$$

$$x_{rt}^\delta \in \{0, 1\} \qquad \forall r \in R_t,\ \forall \delta \in \Delta_r^t \tag{2.5}$$

The objective in (2.2) minimizes the assignment penalties on day $t$. It therefore favors

duties that fit the current fairness needs of the crew members. Constraint (2.3) requires every task to be covered at least once. Constraint (2.4) assigns exactly one duty to each active crew member, and (2.5) defines the binary decision variables. High-cost excess duties can be added when the regular templates do not provide enough capacity to cover all tasks. The formulation represents a set-covering model for task coverage combined with assignment constraints that enforce one duty per active crew member.

The selected duties form the plan for day $t$. After these assignments are fixed, their task content is used to update the SS&S history scores of the corresponding crew members. The next daily problem then receives the tasks and templates for day $t+1$, together with these updated history scores.

## 2.5 Solution approach

Even for one planning day, the number of feasible duties is too large to enumerate. Each duty is a feasible sequence drawn from many possible task combinations. The daily model (2.2)-(2.5) is therefore solved with the column generation heuristic of van Rossum et al. (2024). A concise background on linear programming duality, reduced costs, and the generic column generation principle is provided in Appendix A.

The method starts from a restricted master problem (RMP), which contains only a limited subset of duty variables. The binary restrictions in (2.5) are relaxed to $x^{\delta}_{rt} \geq 0$, which yields a linear programming relaxation. Solving this RMP yields dual information for both the task-cover and the assignment constraints. Let $\lambda_{tk}$ be the dual value of (2.3) and $\pi_{rt}$ the dual value of (2.4). The reduced cost of a candidate duty variable $x^{\delta}_{rt}$ can be seen in Equation 2.6.

$$\bar{c}^{\delta}_{rt} = c^{\delta}_{rt} - \sum_{k \in K_t} \lambda_{tk} a^{\delta}_{tk} - \pi_{rt} \tag{2.6}$$

A negative reduced cost indicates that adding the corresponding duty can improve the current linear relaxation.

Pricing searches for duties with negative reduced cost. For each template, the compatible tasks form a directed task network. A path through this network represents an ordered sequence of tasks and therefore a possible duty. Resources along the path maintain the accumulated state of the partial duty, tracking characteristics such as total duty duration, time without a break, and the presence of a valid meal break. These resources impose feasibility constraints, allowing pricing to be formulated as a resource-constrained shortest path problem. The shortest-path objective incorporates both the dual rewards for covering tasks and the history-dependent assignment penalties.

The pricing problems return candidate duties, but not all candidates are inserted into the restricted master problem. The existing baseline selects duties that are adequately task-disjoint, which limits the number of overlapping columns. The methodology presented in the present thesis studies alternative selection mechanisms at the point of insertion, either replacing or functioning alongside the task-disjoint rule.

The root node refers to the first linear relaxation solved before any duty variable has been fixed. At the root node, column generation iteratively improves the restricted master problem by adding duties with negative reduced cost. This process continues until

no improving duties are found or the early stopping criterion is triggered. The early stopping rule terminates the current column generation phase when the relative objective improvement remains below a configured threshold, avoiding excessive computation time spent on marginal improvements near convergence. After this phase, diving is applied to obtain an integer solution. During diving, fixed variables reduce the problem size, and column generation is performed again whenever the reduced restricted master problem is re-solved.

The solution approach relies on several heuristics. Pricing uses a heuristic labeling method, column insertion restricts the candidate set, early stopping limits the amount of column generation performed in each phase, and diving makes irreversible fixing decisions. These heuristic choices are necessary to keep computation times practical on the massive NS instances, but they remove any optimality guarantee and can leave a small number of tasks uncovered.

Algorithm 1 summarizes the daily procedure. Lines 5-7 mark the part studied in this thesis. Pricing produces negative reduced-cost candidates, after which the configured selection mechanism determines which duties enter the restricted master problem. All other components of the algorithm remain unchanged. The history-dependent costs $c^{\delta}_{rt}$ incorporate the feedback mechanism throughout pricing and master optimization. This is how the column generation procedure steers the daily assignment toward fairness over time.

**Algorithm 1** Daily column generation heuristic in the integrated approach

**Require:** Day $t$, task set $K_t$, active crew set $R_t$, feasible duty spaces $\Delta^t_r$, parameters $\epsilon$ (early stop), $\tau$ (diving threshold)
**Ensure:** Integer duty assignment for day $t$

1: Initialize RMP with empty-duty columns for each active crew member (each empty duty belongs to that crew member's feasible duty set), ensuring initial feasibility
2: **repeat**
3: Solve linear program relaxation of current RMP and obtain dual values $(\lambda, \pi)$
4: Solve pricing problems (RCSPPs) for relevant template-employee contexts
5: Collect candidate duties with negative reduced cost
6: Select only sufficiently task-disjoint duties
7: Add selected duties to RMP
8: Apply column management and remove stale columns with consistently poor reduced cost
9: **until** no improving duties are found **or** relative objective decrease is below $\epsilon$ in consecutive iterations
10: **while** current RMP solution is fractional **do**
11: Fix all duty variables with value at least $\tau$
12: **if** no variable satisfies the threshold **then**
13: Fix the variable with highest fractional value
14: **end if**
15: Re-solve reduced RMP with column generation
16: **end while**
17: **return** resulting integer assignment for day $t$

All parameter settings for the feedback mechanism and the column generation heuristic are adopted from the existing NS implementation. These values were calibrated for the NS setting and are therefore kept fixed throughout this study. This ensures that the experiments isolate the effect of the column-selection mechanisms rather than changes in other algorithmic components.

# Chapter 3

# Literature review

This chapter positions the thesis within the broader literature on railway crew planning, fairness-oriented optimization, and machine learning for combinatorial optimization. The aim is not to restate the problem, but to clarify how it relates to prior work and where the current research fits. The proposed contribution of this thesis lies within an already established solution framework, namely the column generation procedure used in railway crew planning. For that reason, the literature is discussed from the same angle. The chapter first discusses decomposition and integration in crew planning (Section 3.1), followed by fairness and attractiveness in crew planning (Section 3.2). Then Section 3.3 reviews machine learning approaches that support optimization algorithms, with particular emphasis on column generation. Finally, Section 3.4 summarizes the research gap addressed by this thesis.

## 3.1 Decomposition and integration in crew planning

Crew planning problems are commonly decomposed into multiple planning stages. In their survey of personnel scheduling, Ernst et al. (2004) describe how scheduling and rostering problems arise across a wide range of industries and explain that decomposition is often necessary to keep optimization models computationally tractable. Railway crew planning follows the same principle. The most common decomposition separates crew scheduling from crew rostering. Crew scheduling constructs feasible duties that cover all timetable tasks, whereas crew rostering assigns these duties to individual crew members or crew groups over a planning horizon.

The separation between scheduling and rostering has long been standard practice in railway crew planning. Early studies already distinguished duty construction from roster construction (Caprara et al., 1997). More recent surveys confirm that this remains the dominant planning framework for large-scale railway systems (Abbink et al., 2018; Heil et al., 2019). The primary motivation is computational. Considering all planning decisions simultaneously quickly leads to optimization models that become prohibitively large. By decomposing the problem, duty feasibility, crew assignment, and longer-term planning requirements can each be addressed in dedicated planning stages.

While decomposition improves tractability, it also masks important interactions between planning stages. A duty that is inexpensive from a scheduling perspective may be difficult to incorporate into a roster. Conversely, a slightly more expensive duty may ulti-

mately produce a more balanced and efficient roster. Integrated planning models seek to capture these dependencies directly. An early example is provided by Ernst et al. (2001), who develop an integrated train crew management model that simultaneously performs crew scheduling and rostering while supporting both cyclic and non-cyclic rosters. Their work demonstrates the conceptual advantages of joint optimization but also highlights the computational challenges of solving integrated models at realistic scales.

As a result, much of the subsequent literature has focused on partial integration rather than fully integrated formulations. For example, Mesquita et al. (2013) study an integrated vehicle, crew, and roster problem with days-off patterns. They use a Benders-decomposition-based heuristic that alternates between daily integrated vehicle-crew scheduling problems and a rostering problem over the planning horizon. Weider et al. (2015) integrate duty scheduling and rostering with the aim of improving driver satisfaction. In a railway re-planning setting, Breugem et al. (2021) use column generation for an integrated problem in which crew duties and recovery decisions interact. These studies point to the same conclusion. Integration can improve planning quality, but only when the solver design controls the additional computational effort.

The recent NS line of work follows this idea of partial integration, but in a different operational setting. Earlier NS studies focus mainly on the construction of duties from timetable tasks on a large scale, while assignment and rostering are handled in later planning steps (Abbink et al., 2005, 2011). An extension of this setting assumes that template-based rosters are already available (van Rossum et al., 2024). The operational model then generates duties and assigns them to individual crew members at the same time. This integrates duty generation with crew-specific assignment decisions, while tactical capacity planning remains outside the operational model. Large-scale simulation shows that this architecture can be used in realistic NS planning experiments (van Rossum et al., 2026). This partially integrated operational architecture forms the planning context for this thesis.

## 3.2 Fairness, attractiveness, and fairness over time

Fairness in mathematical optimization lacks a single universal definition. Depending on the application, it might require an equal allocation of resources among broad demographic groups, or it might ensure that specific individuals receive a minimum guaranteed level of service. Within mathematical models, these requirements can be formalized in various ways. They can be enforced strictly as constraints, optimized as primary objectives, or simply evaluated as post-solution performance measures. Karsu and Morton (2015) review these fairness concepts in optimization and show that different formulations can lead to fundamentally different trade-offs. Chen and Hooker (2023) discuss how such concepts can be practically translated into mathematical models. In the specific context of personnel planning, fairness is closely intertwined with the attractiveness of schedules, as both directly impact how employees experience their resulting work plans (Wolbeck, 2019).

In crew planning, attractiveness refers to the absolute quality of duties or rosters from the employee's perspective. Examples of attractiveness criteria include the amount of weekend work, the number of early or late duties, the length of shifts, and the frequency of less desirable tasks. Fairness, on the other hand, concerns the relative distribution of

this attractive and unattractive work across employees or groups. While often conflated, they measure distinct aspects of a schedule. A set of rosters can be perfectly fair if undesirable duties are distributed evenly. However, if the scheduling process results in all employees receiving an excessive number of those undesirable duties, the rosters remain highly unattractive.

Breugem et al. (2022) study this interaction in an integrated crew rostering setting with both fairness and attractiveness requirements. Their results show that strictly improving equality in the distribution of work can actually reduce overall roster attractiveness. The subsequent work of Breugem et al. (2023) develops heuristic methods for realistic crew rostering instances and confirms that this trade-off remains relevant when practical, real-world constraints are included. These studies highlight that fairness cannot be evaluated purely as an abstract distributional property. The underlying duties and rosters must independently remain acceptable to the employees.

At NS, fairness in crew planning has historically been formalized through Sharing-Sweet-and-Sour rules. Initially, these rules were designed to balance attractive and unattractive work between different crew bases Abbink et al. (2005). However, recent work at NS has shifted the focus from fairness between bases to fairness between individuals over time (van Rossum et al., 2024). This shift aligns with the broader academic concept of dynamic fairness. Lodi et al. (2022) and Lodi et al. (2023) show that enforcing strict fairness in every single scheduling period can be overly restrictive, whereas leveraging historical assignment data allows for more realistic and flexible allocation decisions. Application-oriented work further confirms that this perspective is crucial in practical routing and assignment problems (van Rossum et al., 2023).

The NS integrated operational model incorporates that longitudinal perspective by steering individual SS&S compliance over a work history. Because of this, the attractiveness of a duty is not determined solely by its internal tasks or its position within an overall roster. A duty can also become more or less desirable depending on the past work assigned to the crew member for whom that duty is considered. This historical tracking makes fairness dependent on assignment decisions made across multiple days. Ultimately, it explicitly connects fairness and attractiveness at NS. The organization achieves long-term fairness precisely by continuously balancing the attractiveness of duties within a roster against the accumulated work history of each individual.

## 3.3 Column generation and machine-learning-based assistance

Column generation serves as a standard approach for large-scale optimization problems where the number of possible choices is simply too large to handle directly. In such cases, generating and listing every potential decision beforehand requires excessive computer memory and time. This structure occurs frequently in crew scheduling and routing, where variables often represent feasible duties, pairings, or routes. The method is described in detail in the column generation handbook of Desaulniers et al. (2005). Additionally, Desrosiers and Lübbecke (2006) discuss the method from both a modeling and algorithmic perspective. Lübbecke and Desrosiers (2005) focus on practical implementation issues that arise when column generation is applied in large-scale real-world applications.

In railway crew planning, column generation is well established. Huisman (2007) use column generation for railway crew rescheduling, where duties must be adjusted after sudden disruptions. Potthoff et al. (2010) also apply column generation to railway crew rescheduling and show how the method can support real-time recovery decisions. The cited studies illustrate why column generation is particularly useful in rail applications. Generating the full set of possible duties upfront is far too large a task to handle directly. Instead, a dedicated subproblem known as pricing can search dynamically for only those promising duties that are actually needed to improve the current schedule.

For the present thesis, the important distinction lies between the restricted master problem and the pricing problem. The RMP is the main scheduling model solved over the limited subset of columns that are currently available. The pricing problem then searches for additional duties that might improve the linear relaxation of the RMP. The linear relaxation is a simplified version of the mathematical model where strict integer requirements are temporarily dropped. Dropping integer requirements allows decision variables to take fractional values, providing a fast way to evaluate the potential quality of new columns. In the NS setting, the master problem takes the form of a set-covering model with assignment constraints. Recent integrated crew planning studies apply the same general column generation structure in operational settings that are much richer than traditional static planning models. The settings are richer because duty generation is no longer handled in isolation. The generation phase is instead tightly combined with assignment rules and dynamic fairness considerations (Breugem et al., 2021; van Rossum et al., 2024).

The repeated interaction between pricing and the RMP creates a natural place for machine learning. Pricing can generate many candidate columns, but adding all of those generated columns simultaneously can make the RMP exceptionally large and slow to re-optimize. The issue is not only the number of columns. In set-covering and set-partitioning models, many different columns often cover very similar parts of the scheduling problem. The linear relaxation of the RMP can therefore have several nearly equivalent fractional representations of the same mathematical solution. Such degeneracy can severely slow down algorithmic convergence and can make the dual values passed back to the pricing problem highly unstable. For railway crew scheduling formulated as a set-covering problem, Muroi et al. (2010) report exactly that type of slow convergence. Elhallaoui et al. (2005) study related degeneracy issues in set-partitioning formulations. In the NS setting, van Rossum et al. (2024) observe that repeatedly solving the RMP accounts for a large share of runtime.

Machine learning is increasingly deployed to support decisions inside optimization algorithms rather than to replace the algorithms completely. Bengio et al. (2018) describe that approach as machine learning alongside optimization algorithms. Their survey explains how learned models can assist repeated algorithmic decisions such as branching, heuristic selection, or other choices that are important but computationally expensive to make exactly. Furthermore, Cappart et al. (2022) review graph learning methods for combinatorial optimization and emphasize that many optimization problems have a natural graph representation through variables, constraints, and their interactions.

Within column generation, machine learning has been successfully applied to assist different parts of the algorithm. Kraul et al. (2023) and Shen et al. (2024) use learning models to predict dual information for stabilization purposes, with the aim of making both

pricing and convergence more stable. Other studies formulate column selection directly as a sequential decision problem. Chi et al. (2023) and Yuan et al. (2024) use reinforcement learning to choose columns during successive iterations. Using reinforcement learning directly reflects the fact that selecting specific columns in one iteration influences the shape and complexity of later restricted master problems.

The work most closely related to the present thesis is presented by Morabit et al. (2021). The authors study machine-learning-based column selection for column generation. The approach keeps the traditional column generation loop entirely intact. During the process, the pricing algorithm first generates candidate columns. An expensive one-step look-ahead expert then identifies a specific subset of columns that is mathematically expected to improve the subsequent RMP. A supervised learning model is trained to mimic the decisions of the expert, so that the computationally heavy expert can be replaced by a much faster prediction model during later runs. The research evaluates the proposed idea on an integrated vehicle and crew scheduling problem and on a vehicle routing problem with time windows. The results show that learned column selection can significantly reduce the time spent re-optimizing the RMP while preserving overall solution quality.

## 3.4 Research gap and contribution

The research gap addressed in the present thesis is how to transfer the machine-learning-based column-selection approach of Morabit et al. (2021) to the complex and dynamic crew planning setting at NS. The study by Morabit et al. (2021) demonstrates that a learned selector can imitate a mathematical expert selector in controlled column generation case studies. While those foundational case studies yield promising results, the operational NS setting differs from the literature in three fundamental ways that make transferring the column-selection approach nontrivial.

The first difference concerns the structure of the master problem. The case studies evaluated by Morabit et al. (2021) rely on set-partitioning formulations. In a set-partitioning model, every task must be covered exactly once. Two columns that share a task are mutually exclusive in an integer solution, which also restricts their fractional use during the linear relaxation. Selecting a small subset of relevant columns in such an environment can eliminate many overlapping alternatives. The NS operational model instead utilizes a set-covering formulation with complex assignment constraints. A set-covering model requires tasks to be covered at least once, meaning that over-coverage is mathematically permitted. Overlapping columns are not mutually exclusive. Because column generation operates on the linear relaxation, multiple overlapping duties can easily be selected together in the NS model. Although column selection can still control the physical growth of the restricted master problem, overlapping coverage patterns and algorithmic degeneracy remain part of the problem structure.

The second difference relates to the operational context and the generation of candidate columns. In the case studies of Morabit et al. (2021), duties are generated within a single global pool, meaning the resulting columns are not linked to specific individuals. The quality of a column in that setting is driven by the immediate cost of the covered tasks. By contrast, the NS pricing problem is solved separately for each individual roster template due to the rolling horizon feedback mechanism. Candidate duties are therefore generated within template-specific pools. Generating duties per template is necessary

because the usefulness of a duty extends beyond the tasks assigned on a single day. While column selection is ultimately performed on a combined pool of all generated duties, the composition of that final pool differs from the literature. A column in the NS setting might derive value from efficient task coverage, from fitting a predetermined roster template, or from improving the long-term fairness balance of a specific employee. Those multiple factors make column usefulness more context dependent than in theoretical settings where columns remain detached from individual crew histories.

The third difference involves the computational role of the expert selector and the criteria for algorithmic termination. In the research conducted by Morabit et al. (2021), the column generation loop stops automatically because the algorithm reaches an optimal solution at the root node. The root node represents the first linear relaxation in the solution process before any duty variable has been fixed. Within the context of this thesis, the algorithm lacks that optimal stop point at the root node due to computational time limits. Lacking an optimal stop point represents the main computational difference. The full crew planning procedure must instead comply with an early stop mechanism and subsequent heuristic diving techniques. An expert selector must therefore prove robust when forced to operate alongside such an early stop mechanism.

The core contribution of the present thesis addresses the outlined differences through a two-part investigation. The first part investigates whether the mixed-integer linear programming (MILP) expert selector proposed by Morabit et al. (2021) can be formulated and applied within the more complex NS setting. The investigation evaluates whether the formulated MILP expert can select a minimal subset of generated columns and achieve runtime improvements while preserving overall solution quality. Building upon a successful expert formulation, the second part investigates whether machine learning techniques can imitate the expert selector within the dynamic NS environment. The ultimate aim is to determine whether a learned prediction model can support the column generation process by reducing computational runtime without compromising the quality of the final solution.

# Chapter 4

# Methodology

This chapter presents the methodology used to develop and evaluate expert-guided and learned column-selection rules. The starting point is the column generation procedure introduced in Chapter 2. Its pricing step can generate many duty columns with negative reduced cost, especially because the NS pricing problem is solved separately for each template. Adding all generated columns to the restricted master problem would increase the effort required for repeated re-optimization. A selection step is therefore used to determine which candidates are inserted.

Section 4.1 formulates this decision through an expert mixed-integer linear program (MILP). Section 4.2 then explains the column generation trajectories that motivate the alternative selection configurations in Section 4.3. Section 4.4 describes the representation and features used for supervised learning. The learning models are introduced in Section 4.5. Finally, Section 4.6 presents the experimental controls, data collection procedure, evaluation modes, and parameter settings.

## 4.1 Expert MILP for column selection

Column selection can be formulated as a supervised classification problem (Morabit et al., 2021). At each iteration of column generation, candidate columns are labeled as either selected or not selected. However, these labels are not directly observable from the pricing step. A column does not have an intrinsic notion of quality, and within the column generation process its usefulness is defined only relative to the current restricted master problem, the other candidates, and the constraints they jointly satisfy. An expert MILP provides a principled way to construct these labels. The idea is to temporarily replace the RMP with an enriched optimization problem that considers both the current columns and the newly generated ones. The resulting model identifies a subset of generated columns that achieves a strong improvement in the objective while keeping the number of selected columns small. This is enforced through a selection penalty. Figure 4.1 illustrates how the MILP-based selection procedure is incorporated into the column generation framework. The solid path represents the initial expert selection. The dashed path represents the optional stabilization procedure proposed by Morabit et al. (2021). After the initial subset has been inserted, the RMP is re-optimized and the remaining candidates are reconsidered under the updated dual values. The selection configurations in Section 4.3 specify how this additional path is adapted to the NS setting.

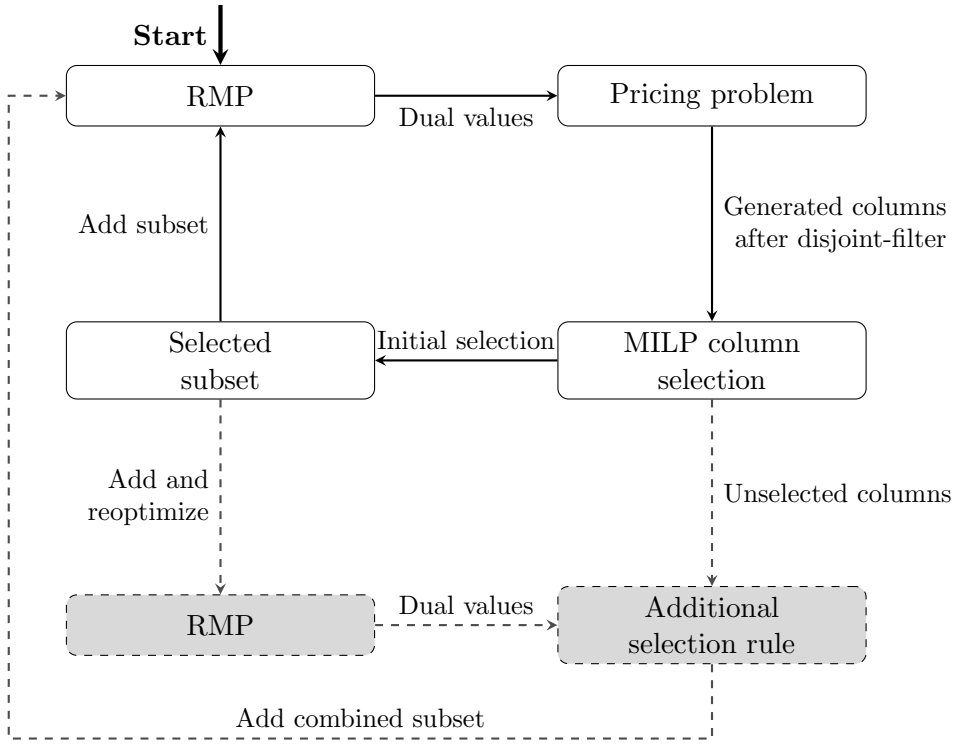



**Figure 4.1:** Overview of the basic column generation algorithm with the addition of the MILP for column selection. The primary loop is indicated by the solid lines and utilizes the MILP to filter an initial subset of promising columns. The secondary stabilization mechanism is indicated by dashed lines and shaded blocks. It processes the remaining unselected columns through the RMP to compile the final combined subset.

Beyond its use for generating labels, the expert MILP can also be executed directly as a selection strategy. In this setting, it controls the growth of the RMP column pool and yields high-quality decisions obtained by solving an optimization problem at each iteration. This provides a strong reference for attainable performance under optimization-driven selection. The drawback is that the MILP must be solved at every relevant column generation iteration, which adds substantial computational effort. In an offline setting, the same procedure is used to generate training labels for supervised learning. A prediction model is then trained to imitate the expert's selection behavior. After labeling, the standard column generation procedure proceeds unchanged.

When this idea is transferred to the NS setting, the expert is built on top of the daily planning problem defined in Section 2.4. The set of tasks on day $t$ remains $K_t$, the set of active crew members remains $R_t$, and the set of feasible duties for crew member $r$ remains $\Delta_r^t$. The binary parameter $a_{tk}^{\delta}$ still equals 1 when duty $\delta$ covers task $k$ and 0 otherwise, and $c_{rt}^{\delta}$ still denotes the assignment cost of duty $\delta$ for crew member $r$ on day $t$. The only difference is that the expert is evaluated on a finite set of duty columns that are actually present in the current column generation iteration, rather than on the full duty sets $\Delta_r^t$.

To make this concrete, consider a single column generation iteration $\ell$. Let $\widetilde{G}_\ell$ denote the raw set of candidate duty columns returned by pricing. The task-disjoint rule normally reduces this set to the candidate pool $G_\ell \subseteq \widetilde{G}_\ell$ that is presented to the configured selector. If the task-disjoint filter is disabled, $G_\ell = \widetilde{G}_\ell$. The set $\Omega_\ell$ collects all duty columns that are already in the RMP at the start of iteration $\ell$. The columns available to the expert

are combined into the union $P_\ell = \Omega_\ell \cup G_\ell$.

Over this enlarged column set, the linear relaxation of the daily model is the same RMP solved in the current column generation and includes three additions. The first addition is a continuous variable $\theta^\delta_{rt} \geq 0$ for each $(r, \delta) \in P_\ell$, which plays the role of the relaxed daily decision variable $x^\delta_{rt}$ on the columns that are actually present. The second addition is a binary variable $y^\delta_{rt} \in \{0, 1\}$ for each candidate $(r, \delta) \in G_\ell$, which indicates whether the expert selects that candidate. Columns in $\Omega_\ell$ are not associated with such a binary variable because they are already part of the RMP. Alongside these variables, a small parameter $\varepsilon > 0$ is included in the objective. It penalizes the number of selected candidates and pushes the expert toward small selected sets when several subsets achieve the same primary objective value.

The expert MILP is then given by:

$$\min \quad \sum_{(r,\delta)\in P_\ell} c^\delta_{rt}\,\theta^\delta_{rt} + \varepsilon \left( \sum_{(r,\delta)\in G_\ell} y^\delta_{rt} \right) \tag{4.1}$$

$$\text{s.t.} \quad \sum_{(r,\delta)\in P_\ell} a^\delta_{tk}\,\theta^\delta_{rt} \geq 1 \qquad \forall k \in K_t \tag{4.2}$$

$$\sum_{\delta\,:\,(r,\delta)\in P_\ell} \theta^\delta_{rt} = 1 \qquad \forall r \in R_t \tag{4.3}$$

$$\theta^\delta_{rt} \leq y^\delta_{rt} \qquad \forall (r, \delta) \in G_\ell \tag{4.4}$$

$$\theta^\delta_{rt} \geq 0 \qquad \forall (r, \delta) \in P_\ell \tag{4.5}$$

$$y^\delta_{rt} \in \{0, 1\} \qquad \forall (r, \delta) \in G_\ell \tag{4.6}$$

Constraints (4.2) and (4.3) correspond to the standard task coverage and assignment requirements of the daily model. They are now enforced only over the columns in $P_\ell$. Constraint (4.4) links the continuous and binary variables associated with the candidate columns. If $y^\delta_{rt} = 0$, it forces $\theta^\delta_{rt} = 0$. This removes the candidate from contributing to the solution. If $y^\delta_{rt} = 1$, the candidate may participate in the RMP solution and behaves like any other column already included in the RMP. Constraints (4.5) and (4.6) state the variable domains. The objective (4.1) consists of the same assignment-penalty sum as before, together with the expert-selection penalty. A candidate is therefore selected only when the resulting decrease in the assignment-penalty sum justifies the additional small per-candidate cost $\varepsilon$.

Solving (4.1)–(4.6) yields the selected subset of candidate columns as defined in Equation 4.7.

$$G^S_\ell = \{\, (r, \delta) \in G_\ell \mid y^\delta_{rt} = 1 \,\} \tag{4.7}$$

This is the set of candidate columns that the expert recommends to insert in iteration $\ell$. For data collection, the corresponding binary values are recorded as target labels for the supervised learning models.

## 4.2 Column generation trajectories

Before specifying the selection configurations, it is useful to understand how column generation develops over a full planning day. The procedure in Algorithm 1 first performs column generation at the root node. It then enters the diving phase, where duty variables are gradually fixed to integer values. Fixing a variable tightens the feasible region and can increase the objective of the linear relaxation. A new column generation segment is subsequently used to reduce the objective under these tighter conditions. Repeating these fixing and re-optimization steps produces a trajectory consisting of several segments, each corresponding to a node in the search process induced by diving, and continues until an integer solution is obtained.

Figure 4.2 illustrates this structure for the task-disjoint baseline and the expert MILP. The baseline is the current NS selector introduced in Section 2.5. It greedily retains negative-reduced-cost columns that are sufficiently disjoint in their task coverage. Its relatively broad selection is expected to reduce the objective quickly within each segment, although the RMP can grow substantially. The expert MILP instead retains a much smaller subset.

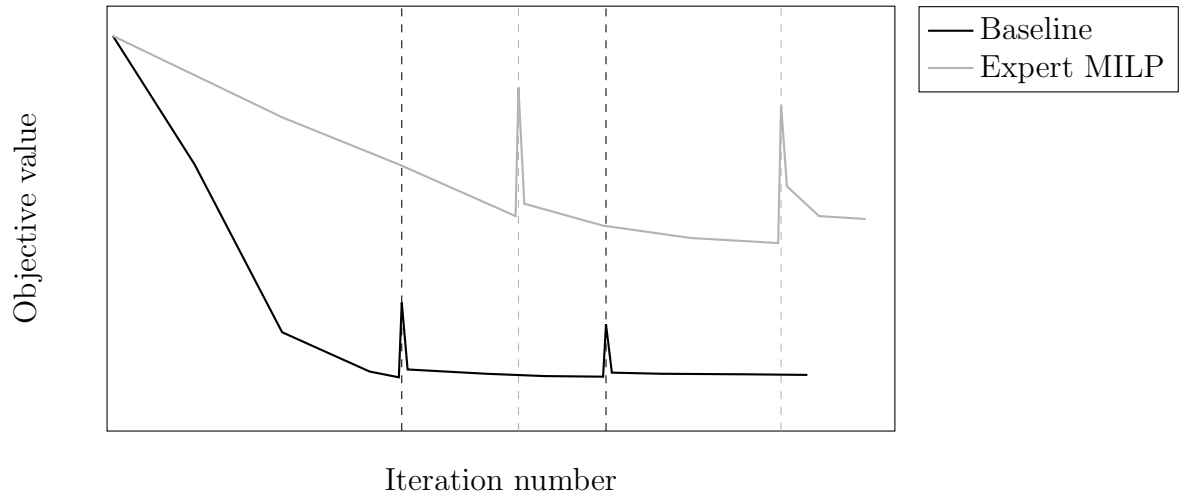



**Figure 4.2:** Schematic illustration of column generation trajectories for the baseline and expert MILP selection strategies over a single planning day. Axis values are indicative only. The dashed vertical lines mark the transitions between column generation segments induced by the diving phase.

The smaller selection of the MILP explains two characteristics of the expert trajectory. First, candidates that are not part of the minimal expert selection are discarded even when they may remain attractive after re-optimization. Similar columns can then be generated again and removed in a later iteration. The dual information may consequently change only marginally and this creates a cycle in which similar columns repeatedly move between pricing and selection. As a result, the algorithm struggles to make further progress, which leads to the stalling behavior reported by Morabit et al. (2021). Second, the smaller RMP provides fewer alternatives after variables have been fixed. The expert trajectory may therefore show a larger objective increase at a diving step and a slower subsequent recovery. The axes in Figure 4.2 contain no measured values, so these trajectories illustrate

the expected mechanisms. These convergence limitations motivate the introduction of supplementary selection mechanisms in Section 4.3.

The qualitative differences described above unfold within a full-day trajectory that is inherently path-dependent. In particular, the early termination rule and the diving rule depend both on the fractional solution available when these rules are performed. As a consequence, even small differences in the columns selected earlier can leave two strategies at different fractional solutions and steer them along different diving paths. The trajectory after each spike is therefore influenced by the preceding sequence of decisions. A complete planning day consequently contains variability that cannot be attributed solely to the column-selection rule. To separate these effects, the experiments defined later in this chapter consider both the full procedure and a root-node setting, where column selection is evaluated prior to any diving. Because the root node can be seen as the first linear relaxation of the day, this setting provides a more controlled environment for isolating the effect of the selection strategy.

In addition, hybrid variants are considered in which an alternative selector, either the expert MILP or the supervised learning model, is used only at the root node and the procedure then switches back to the baseline selector after the first dive. These variants help separate the value of root-node selection from the behavior of a selector deeper in the daily procedure.

## 4.3 Selection configurations

To mitigate the stalling behavior of the expert MILP, supplementary mechanisms are introduced to stabilize column selection and improve convergence. Four configurations are compared in this study. The first corresponds to the current NS loop, where the task-disjoint rule determines which generated candidates are inserted. The other three pass the output of that rule to the expert MILP and differ in the safety mechanism associated with the expert. The four variants are summarized below and explained in more detail immediately afterwards.

- *Baseline*: representing the existing NS loop in which the task-disjoint heuristic is the only selection step.
- *MILP-GN* (global net): in which the expert MILP is followed by Morabit et al. (2021) re-optimization procedure combined with a global reduced-cost safety net.
- *MILP-TN* (template net): in which the expert MILP is followed by the same re-optimization procedure but with a template-specific safety net.
- *MILP-TC* (template constraint): in which the per-template coverage mechanism is incorporated directly into the expert MILP.

The baseline configuration serves as a reference point. It uses the task-disjoint selection heuristic introduced in Section 2.5, following Breugem et al. (2021). Generated columns with negative reduced cost are first sorted by reduced cost, and the column with the most negative reduced cost is selected first. The remaining columns from the same pricing output are then considered in increasing order and are added only when they are sufficiently

disjoint from the columns retained earlier in that selection step. This is measured by a similarity score. For two columns $x_1$ and $x_2$, the similarity is defined as the number of tasks covered by both columns divided by the number of tasks covered by $x_1$. A high similarity score indicates that most tasks of the candidate column $x_1$ are already covered by $x_2$, and therefore that the candidate adds limited additional diversity. In the current NS setup, a column is accepted only if its similarity with every previously retained column from that pricing output remains below a fixed threshold of 0.8. A similarity of 0.8 or higher therefore causes the candidate to be rejected.

The heuristic is a greedy reduction rule rather than an optimization of the combined effect of the selected columns. It removes many highly redundant candidates and promotes diversity in task coverage, but it can still pass a large number of columns to the RMP. The baseline therefore reduces pool growth without eliminating the computational burden caused by repeated RMP re-optimization.

In the MILP-based configurations, the task-disjoint heuristic first performs the same selection step as in the baseline. Its retained columns form the candidate pool $G_\ell$ presented to the expert MILP. This is kept unchanged across all approaches and is consistent with the use of comparable disjoint filtering in Morabit et al. (2021), where columns are grouped into blocks of mutually disjoint elements to promote diversity within the candidate pool. As discussed in the baseline setting, this filtering step is an established component of the procedure that retains a substantial share of the generated candidates while effectively reducing redundancy. This is particularly beneficial for the supervised framework developed later in this chapter, as it reduces the number of near-duplicate columns and thereby improves the quality of the training signal. In addition, it reduces the size of the input used both for label generation and model training.

Although both the MILP and the supervised learning model are capable of handling larger candidate sets, doing so would increase computational cost. The filter therefore offers a practical compromise between maintaining a rich candidate pool and limiting the computational effort required throughout the optimization and learning pipeline.

A potential drawback of this input reduction is that it may exclude columns that the MILP would otherwise select. This effect is evaluated explicitly by solving the MILP both with and without the filter and comparing the resulting selections. The extent to which the outcomes differ is used to assess whether the reduction in input size justifies retaining the filter in the main configurations.

The MILP-GN and MILP-TN configuration follow the two-phase insertion procedure proposed by Morabit et al. (2021), represented by the dashed path in Figure 4.1. The expert MILP first selects $G_\ell^S$ according to Equation 4.7. These columns are inserted to the RMP, which is subsequently re-optimized to obtain updated dual values. Reduced costs are then recomputed for the remaining candidates in $G_\ell \setminus G_\ell^S$. Adding a subset of the columns that remain attractive under the updated dual values helps mitigate the convergence stall described in Section 4.2.

The two variants differ only in how this additional subset is determined. In MILP-GN, the procedure follows Morabit et al. (2021) directly. All candidates with strictly negative recomputed costs are pooled, and the top 50% are added. In MILP-TN, the selection accounts for an important structural characteristic of the NS problem discussed in Section 3.4. Because NS operates a heterogeneous fleet and crew members are subject

to fairness considerations that depend on their individual assignment history, the pricing problem is decomposed into multiple subproblems instead of a single unified one. Each subproblem corresponds to a template, where a template corresponds to a crew member. Applying a single global fraction may leave some templates without any selected columns, which can result in slow convergence by effectively starving parts of the assignment structure. Combined with the minimal selection step, this leads to repeated regeneration of similar columns that are discarded before contributing to progress. To mitigate this behavior, this configuration adds the column with the most negative recomputed reduced cost from every template that still contains a candidate with negative reduced cost after re-optimization. In this way, progress is enforced across all templates rather than focusing only on the globally most attractive columns.

The third MILP configuration, MILP-TC, pursues the same per-template idea but incorporates it directly into the optimization model rather than applying it afterwards. Let $T_\ell$ denote the set of templates for which at least one generated candidate column has negative reduced cost, and let $G_\ell^\tau \subseteq G_\ell$ denote the candidates belonging to template $\tau$. Equation 4.8 shows how to enforce that at least one candidate is selected from every template during the initial solve.

$$\sum_{p \in G_\ell^\tau} y_p \geq 1 \qquad \forall \tau \in T_\ell \text{ with } G_\ell^\tau \neq \emptyset \tag{4.8}$$

Adding this as a constraint to the expert MILP results in (4.1)–(4.6) together with (4.8). Since template coverage is guaranteed by construction, the second re-optimization is, in principle, no longer required. Whether this leads to practical improvements, and how it compares with the two safety-net variants, is assessed experimentally.

To ensure a fair comparison, all four configurations are evaluated under identical experimental conditions.

## 4.4 State representation and data collection for machine learning

The objective of the machine learning component is to train a model that predicts, at every column generation iteration, which of the newly generated candidate columns would be selected by the expert MILP. This requires a dataset that links the state of the algorithm at each iteration to the corresponding expert selection decisions. Constructing such a dataset requires choosing an appropriate representation of the algorithm's state and defining the information that is recorded at each iteration.

### 4.4.1 Graph representation

Before defining the data representation, it is useful to consider the requirements that an ideal learning algorithm should satisfy. Morabit et al. (2021) identify five key requirements that strongly influence the choice of representation.

First, the decision to select a column depends on the other columns available in the same iteration. A duty that substantially improves the RMP on its own may become redundant when another duty covers much of the same work. Consequently, a model

that evaluates columns independently cannot capture these interactions. Second, the number of candidate columns changes from one iteration to the next. This rules out representations that require a fixed-size input vector containing all columns simultaneously. Third, the dual values of the RMP constraints provide valuable information about which tasks are difficult to cover and which crew members are difficult to assign. However, both the number of constraints and the number of constraints in which a column participates vary between instances. Encoding all dual values into a single feature vector would therefore generalize poorly across different problem sizes. Fourth, the model should generalize to planning instances of different sizes. At NS, the number of daily tasks, crew members, and templates varies across planning days and across the two experimental instances considered in this study. Finally, the motivation for replacing the expert MILP is its computational cost. The learned model must therefore produce predictions within a fraction of a second at every column generation iteration.

A bipartite graph satisfies these requirements by representing the RMP without imposing a fixed number of columns or constraints. At iteration $\ell$, one node set represents columns and the other represents constraints. A column node is connected to a constraint node whenever the corresponding duty participates in that constraint. This structure provides a natural way to distribute information across the two types of nodes. Constraint features place the dual information on the relevant constraint nodes instead of requiring one fixed-length vector containing every dual value. The same representation can therefore be constructed for planning days and network instances of different sizes. Each column is informed by the constraints it participates in, which provides a local signal of its relevance in the current solution of the RMP. Through these connections, columns receive information that reflects how tightly the constraints they appear in are currently satisfied and how much flexibility remains in the system. Furthermore, the graph representation enables all candidate columns to be evaluated in a single inference step, making the prediction process efficient even when the number of candidate columns varies between iterations. The representation therefore addresses the varying input size, the relational nature of selection, the use of dual information, transfer across instance sizes, and the need for computationally efficient inference.

### 4.4.2 Logged components and labels

For each column generation iteration $\ell$, four components are stored to represent the state of the algorithm. The relationship between these components and their mapping to the bipartite graph structure is visualized in Figure 4.3.

The first component is the graph topology, represented by a sparse adjacency matrix $E \in \{0,1\}^{n \times m}$, where $n$ is the total number of column nodes and $m$ is the total number of constraint nodes. Crucially, the $n$ column nodes consist of two disjoint sets. These are the newly generated candidate columns $G_\ell$ and the active basis columns $B_\ell$ from the current RMP solution. Thus, the total number of column nodes in iteration $\ell$ is $n = |G_\ell| + |B_\ell|$. The column nodes in the logged graph omit the remaining inactive columns in $\Omega_\ell \setminus B_\ell$. Although these remain available to the expert MILP, including the full column pool would cause the graph to grow continuously as inactive columns accumulate, creating an unnecessary computational bottleneck. The basis columns $B_\ell$ are included because they provide essential context. They represent the current LP solution and indicate which

duties are active and which columns compete for the same tasks or crew assignments.

The second component is the column feature matrix $X_V \in \mathbb{R}^{n \times d}$, where $d$ is the number of column features. This matrix encodes information for all $n$ nodes in the graph. The third is the constraint feature matrix $X_C \in \mathbb{R}^{m \times p}$, where $p$ is the number of constraint features. Although the number of nodes $n$ and $m$ vary across iterations and planning days, the feature dimensions $d$ and $p$ remain fixed. As a result, each instance is represented in a consistent feature space despite differences in graph size.

Finally, the fourth component is the binary label vector $y \in \{0,1\}^{|G_\ell|}$, which records the expert selection decisions for the newly generated candidates. Notice that $y$ has a length of $|G_\ell|$ rather than $n$. The label is defined only for a candidate $(r, \delta) \in G_\ell$, corresponding to the value of the expert variable $y^\delta_{rt}$ from Equation 4.7. Basis columns carry no labels as they are not prediction targets. The procedure used to collect these iteration graphs and labels is detailed in Section 4.6.3.

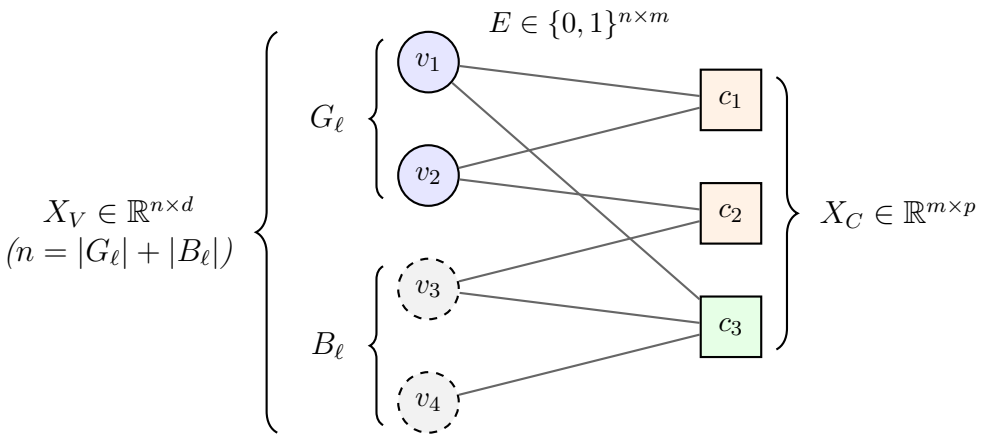



**Figure 4.3:** Bipartite graph representation of the recorded state at iteration $\ell$. The topology is defined by the edge matrix $E$, connecting $n$ column nodes to $m$ constraint nodes. The column nodes comprise newly generated candidates $G_\ell$ (solid blue) and context-providing basis columns $B_\ell$ (dashed gray). Constraint nodes represent either task-covering (orange) or template-assignment (green) constraints. While feature matrices $X_V$ and $X_C$ encode information for all nodes, target labels $y$ apply exclusively to the candidates in $G_\ell$.

### 4.4.3 Feature construction

The recorded state is stored in terms of column nodes and constraint nodes, each described through fixed-length feature vectors. Column features and constraint features are defined in Table 4.1 and Table 4.2, respectively. Together with the labels, these features constitute the dataset used for model training. Each column node $v$ is represented by a feature vector $x_{\text{col},v} \in \mathbb{R}^{15}$. Stacking these vectors for all $n$ column nodes yields the matrix $X_V \in \mathbb{R}^{n \times 15}$. Likewise, each constraint node $c$ has a feature vector $x_{\text{con},c} \in \mathbb{R}^{14}$, and the $m$ vectors form $X_C \in \mathbb{R}^{m \times 14}$.

The column features describe the current optimization value and the work content of a duty. The objective coefficient represents the crew-specific assignment cost of a duty after the feedback mechanism has been applied, while the reduced cost measures whether the candidate can improve the current linear relaxation. For each attribute $a$ introduced

**Table 4.1:** Column node feature vector $x_{\text{col},v} \in \mathbb{R}^{15}$. The `score_a` features encode the duty-level SS&S contribution $g_a(u)$ for each attribute $a$. The `is_new` indicator distinguishes newly generated candidates from basis columns in $B_\ell$.

| Feature | Description |
|---|---|
| `reduced_cost` | Reduced cost $\bar{c}^{\delta}_{rt}$ |
| `objective_coefficient` | Assignment penalty cost $c^{\delta}_{rt}$ |
| `score_duty_length` | SS&S score for Duty length |
| `score_type_a` | SS&S score for Type-A work |
| `score_aggression` | SS&S score for Aggression |
| `score_double_deck` | SS&S score for Double-decker work |
| `score_repetition_all` | SS&S score for Repetition (all trains) |
| `score_repetition_spr` | SS&S score for Repetition (sprinters) |
| `score_track_variation` | SS&S score for Track variation (rolling window) |
| `score_track_variation_yearly` | SS&S score for Track variation (yearly) |
| `is_new` | 1 if generated this iteration, 0 if in basis ($B_\ell$) |
| `column_degree` | Number of incident constraints |
| `primal_value` | RMP primal value (0 if not in RMP solution) |
| `duty_start_time` | Start time (seconds from midnight) |
| `duty_end_time` | End time (seconds from midnight) |

**Table 4.2:** Constraint node feature vector $x_{\text{con},c} \in \mathbb{R}^{14}$. The `saturation_a` features equal the crew-specific history score $s$ divided by the corresponding target score $q_a$ for each attribute $a$. These features are zero on task-covering nodes. Task duration is defined only on task-covering nodes.

| Feature | Description |
|---|---|
| `is_task` | 1 for task-covering constraint, 0 otherwise |
| `is_template_assignment` | 1 for template-assignment constraint, 0 otherwise |
| `dual_value` | Dual value $\lambda_{tk}$ (cover) or $\pi_{rt}$ (assignment) |
| `constraint_degree` | Number of incident column nodes |
| `saturation_duty_length` | History score / target score for Duty length |
| `saturation_type_a` | History score / target score for Type-A work |
| `saturation_aggression` | History score / target score for Aggression |
| `saturation_double_deck` | History score / target score for Double-decker work |
| `saturation_repetition_all` | History score / target score for Repetition (all trains) |
| `saturation_repetition_spr` | History score / target score for Repetition (sprinters) |
| `saturation_track_variation` | History score / target score for Track variation (rolling window) |
| `saturation_track_variation_yearly` | History score / target score for Track variation (yearly) |
| `task_duration` | Task duration for cover nodes, 0 for assignment nodes |
| `slack_status` | Primal value of the constraint's slack variable |

in Section 2.2, the $\texttt{score}_\texttt{a}$ feature records the duty-level contribution $g_a(u)$ from Equation 2.1, where $u$ is the attribute score of the candidate duty. These scores are computed by the same evaluation routines used by the NS solver to construct the assignment costs. They depend on the tasks and structure of the duty, but not on the history of the assigned crew member. Exposing them separately allows a model to distinguish the work characteristics of a duty from the history-dependent effect included in its objective coefficient.

Some attributes are represented by multiple features because the underlying SS&S rules distinguish different rolling windows or train categories. Repetition is therefore tracked separately for all train types and for sprinters only, while track variation is represented by both a 45-duty rolling window and a yearly rolling window. The remaining features indicate whether a column was generated in the current iteration or belongs to the current basis, its degree in the bipartite graph, its value in the RMP solution, and its start and end times.

The constraint features describe the current state of the RMP. The dual values $\lambda_{tk}$ and $\pi_{rt}$ from Section 2.5 quantify the marginal value of the task-covering and template-assignment constraints. They therefore indicate where additional columns can be valuable. On a template-assignment node, the saturation features summarize the current SS&S history of the associated crew member. For each attribute $a$ introduced in Section 2.2, the calculation can be seen in Equation 4.9.

$$\texttt{saturation\_a} = \frac{s}{q_a} \tag{4.9}$$

where $s$ is the current history score of that crew member for attribute $a$, and $q_a$ is the corresponding target score from Section 2.2. Both quantities are therefore attribute-specific, and $s$ is also crew-specific. Each crew member carries a separate history score for every attribute. In the feedback mechanism of Section 2.3, this same history score enters the penalty through the factor $h_a(s)$. The feature therefore does not represent the transformed value $h_a(s)$ itself. It represents the underlying history score $s$, normalized by the attribute's target score. Together, the duty contribution $g_a(u)$ and the normalized history score $s/q_a$ provide the model with the two sources of information underlying the feedback penalty $f_a(u, s)$. Task duration is relevant only to task-covering nodes, while the slack status records the current primal violation represented by the corresponding auxiliary variable.

The interpretation of the saturation values depends on whether an attribute is classified as `sweet` or `sour`. For `sweet` attributes, values above 1.0 indicate that the desired exposure has already been achieved. For `sour` attributes, values above 1.0 indicate that the permitted exposure has been exceeded. In both cases, higher saturation values make assignments that further increase the attribute less attractive. As a result, the saturation features provide the model with a compact representation of the current fairness state.

These records differ from many machine learning datasets in an important way. They are not collected from noisy external measurements and then cleaned afterwards. Each graph is produced inside a controlled column generation run, and every feature is computed from the current iteration. There are no missing values to impute and no labels to correct by hand. Values that might look extreme in a tabular dataset, such as a very high dual value or a saturation score far above one, are not treated as outliers. They describe genuine states of the problem at that iteration. For that reason, no separate filtering or feature-engineering pipeline is applied after logging. The graphs are stored as they are produced and passed on to model training.

## 4.5 Learning models

The objective of the learning task is to predict, for each generated candidate column in $G_\ell$, whether it would be selected by the expert MILP. The different models considered in this study mainly differ in how much of the logged restricted master problem they use to make this decision. The simplest models evaluate each generated column independently based solely on its own feature vector. More advanced models additionally incorporate information from neighboring constraints, while the graph neural networks operate directly on the complete bipartite graph and learn how information should propagate between columns and constraints.

Regardless of their complexity, all models must satisfy two requirements. First, they should reproduce the expert's column-selection decisions as accurately as possible. Second, they must be sufficiently fast to be executed inside the column generation loop, where prediction time directly affects the overall solution time. A brief introduction to supervised learning, binary classification, and neural networks is provided in Section A.2. The remainder of this section introduces the different model families and explains the motivation behind their design.

### 4.5.1 Linear baselines

The first model is logistic regression (LR), introduced by Cox (1958) to formally model the probability of a binary outcome as a function of independent explanatory variables. It is included as a deliberately simple baseline. To establish a baseline of what can be learned strictly from the duties themselves, this initial LR model is configured to use only the column-specific feature vector $x_{\text{col,v}} \in \mathbb{R}^{15}$ of column node $v$. In this minimal setup, the model evaluates each generated candidate column $(r, \delta) \in G_\ell$ independently, without considering its associated constraints or other columns generated in the same iteration. Consequently, LR sees the duty, its cost information, and its duty-level SS&S scores, but lacks broader problem context. For example, two duties with similar reduced costs may look almost identical to LR, even if one covers a task that is already easy to cover and the other covers a task with a high dual value. The benefit of this restriction is that LR is easy to inspect and has no difficulty with variable graph sizes, because the same scoring rule is applied to every generated column separately. The LR computes the linear score according to Equation 4.10.

$$z = w^\top x_{\text{col}} + b \tag{4.10}$$

where $w \in \mathbb{R}^{15}$ is the learned weight vector and $b \in \mathbb{R}$ is the bias. Each feature contributes additively to the score, with its influence determined by the corresponding weight. The score is converted into a probability according to Equation 4.11, where the sigmoid function is defined in Equation 4.12.

$$\hat{y} = \sigma(z) \tag{4.11}$$

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{4.12}$$

An advantage of this additive formulation is that the learned weights are directly inter-

pretable. For example, if the weight associated with the reduced cost is negative, then a more negative reduced cost increases $z$ and therefore the predicted probability of selection. Similarly, a large positive weight for a duty-level SS&S feature indicates that the corresponding attribute strongly contributes to reproducing the expert's choices.

The parameters are estimated by minimizing a weighted binary cross-entropy loss over the training examples. This can be seen in Equation 4.13.

$$\mathcal{L}_{\mathrm{LR}} \;=\; -\frac{1}{N}\sum_{i=1}^{N}[w_{+}\, y_i \log \hat{y}_i \;+\; w_{-}\,(1-y_i)\log(1-\hat{y}_i)] \tag{4.13}$$

where $y_i \in \{0,1\}$ is the expert label and $\hat{y}_i$ is the predicted probability that candidate $i$ is selected. The coefficients $w_+$ and $w_-$ are class weights for the positive (selected) and negative (not selected) classes, respectively. They are computed from the class frequencies in the training set to compensate for the imbalance between selected and non-selected columns. Because the expert selects only a small fraction of the generated columns, the positive class receives a larger weight ($w_+ > w_-$). Therefore, misclassifying a selected column incurs a larger penalty. Without this weighting, a model that simply predicts most columns as not selected could achieve a low loss. To help prevent overfitting, the loss function is augmented with $L_2$ regularization, which penalizes large coefficient magnitudes to help prevent overfitting. This effectively formulates a Ridge regression (Hoerl and Kennard, 1970). The strength of this regularization is controlled by the inverse parameter $C$.

The second model is aggregated logistic regression (Agg-LR). It is included to answer two distinct questions before moving to more complex architectures. The first is whether incorporating local constraint information actually improves predictive performance compared to evaluating columns in isolation. The second is whether simple, linear combinations of raw constraint features are sufficient, or if the selection problem requires the non-linear transformations and complex structural learning provided by a graph neural network.

To test this, Agg-LR retains the interpretable linear classifier but augments the input features. For a candidate column node $v \in G_\ell$, let $\mathcal{N}(v)$ denote the set of its adjacent constraint nodes. The feature vectors $x_{\mathrm{con},c} \in \mathbb{R}^{14}$ of all neighbors $c \in \mathcal{N}(v)$ are pooled using permutation-invariant aggregation operators. If we denote the chosen operator as *agg*, the pooled constraint representation is given by Equation 4.14.

$$x_{\mathrm{pool},v} = \mathrm{agg}_{c\in\mathcal{N}(v)}(x_{\mathrm{con,c}}) \tag{4.14}$$

This pooled summary is then concatenated with the original column feature vector to form an augmented input vector $\tilde{x}_v = [x_{\mathrm{col},v} \,\|\, x_{\mathrm{pool},v}]$. The standard logistic regression equation is then applied to this new vector to compute the selection probability.

Initially, this is tested using only the mean aggregator, which provides a natural baseline by encompassing all neighboring constraints into a single average state and cleanly handling variable node degrees. After establishing this baseline, the aggregation is enhanced to include the minimum, maximum, and sum operators alongside the mean, resulting in a wider concatenated vector $\tilde{x}_v$. This progression is a deliberate design choice to maintain interpretability while capturing distinct structural signals. The mean captures central tendencies, the maximum and minimum highlight extreme bottlenecks, such as a

single task with an exceptionally high dual value or a crew member near a strict fairness bound, and the sum reflects the total accumulated weight of the constraints. By explicitly constructing $\tilde{x}_v$, Agg-LR incorporates vital RMP context without the computational complexity of a full graph neural network architecture.

### 4.5.2 Graph neural network

The second model family considered in this study is the graph neural network (GNN), comprehensively formalized by Scarselli et al. (2009) to extend traditional neural networks by allowing nodes to iteratively update their states based on information from their neighbors. Unlike the previous models, a GNN explicitly captures the relationships between duty columns and the constraints in which they participate. Whether a column should be selected is not solely determined by its own characteristics, but also by its interactions with constraints and with other columns competing for the same tasks or crew member assignments.

These dependencies are naturally represented as a graph. Through message passing, each node repeatedly exchanges information with its neighbors, allowing column embeddings to incorporate information about the surrounding optimization structure. Rather than evaluating columns independently, the model learns representations that reflect both local and relational information. As questioned in the previous section on aggregated logistic regression, it is hypothesized that simple linear summaries of constraints are insufficient for this complex selection task. GNNs address this by applying non-linear transformations to both the node features and the structural graph information.

This class of models is particularly suitable for column generation because it naturally handles variable numbers of columns and constraints. All parameters are shared across nodes, which allows the same model to be applied across instances with different sizes without modification. In addition, the computation is invariant to permutations of nodes, since predictions depend only on the graph structure and not on any ordering of rows in the data.

To make the mechanics of the GNN less abstract for this specific column generation application, the model's operation can intuitively be broken down intro three main steps. First, the raw numerical features of each duty and constraint are translated into a shared internal mathematical representation. In this way, they can interact with each other. Second, information flows through the bipartite graph in multiple message-passing rounds. In these rounds, constraints update their state by looking at all duties that cover them, and duties subsequently update their state by looking at all the constraints they participate in. Third, after exchanging this context, each duty uses its final updated state to predict its probability of being selected by the expert.

#### 4.5.2.1 SAGEConv message-passing framework

The GNN used in this study follows the message-passing framework of Morabit et al. (2021) for bipartite column-selection graphs. The general message-passing is implemented using SAGEConv layers, introduced by Hamilton et al. (2018). SAGEConv learns to generate node embeddings by sampling and aggregating information from a node's local neighborhood. It is used as the reference implementation because prediction must remain fast enough to be useful inside the column generation loop, and because it provides a

flexible framework for studying architectural choices such as message-passing depth and neighborhood aggregation. The model executes the three intuitive steps introduced above.

First, the raw features from Section 4.4.3 are encoded once before any message-passing layer is applied. Because the original column features and constraint features have different lengths, separate linear maps are used to project them into the shared initial embedding dimension $d_h$. Aligning with the notation from the logistic regression baselines, the encoding can be seen in Equation 4.15.

$$\begin{aligned} h_v^{(0)} &= \rho(W_{\text{col}}\, x_{\text{col,v}}) \\ h_c^{(0)} &= \rho(W_{\text{con}}\, x_{\text{con,c}}) \end{aligned} \tag{4.15}$$

where $x_{\text{col},v} \in \mathbb{R}^{15}$ and $x_{\text{con},c} \in \mathbb{R}^{14}$ represent the raw input vectors. The maps $W_{\text{col}} \in \mathbb{R}^{d_h \times 15}$ and $W_{\text{con}} \in \mathbb{R}^{d_h \times 14}$ act as standard dense neural network layers mapping the raw input features to the initial hidden state. In this specific application, $W_{\text{col}}$ learns how to combine a duty's 15 raw features such as its reduced cost and SS&S scores into a meaningful $d_h$-dimensional vector. Similarly, $W_{\text{con}}$ translates a constraint's 14 features such as its dual value into the same dimensional space. These mapping matrices are not predefined. Their weights are initialized randomly and updated during training via backpropagation to minimize the classification error, aligning the internal representations with the expert's decisions. The function $\rho$ is the rectified linear activation (ReLU), defined in Equation 4.16. This operation evaluates each component of the vector and replaces all negative values with zero while leaving positive values unchanged. Applying this simple non-linear transformation is the crucial mechanism that allows the network to capture complex patterns beyond what a simple linear model can learn.

$$\rho(x) = \max(0, x) \tag{4.16}$$

The second step applies $K$ SAGEConv layers on the bipartite graph. Each layer performs two updates in sequence. Constraint nodes first aggregate embeddings from neighboring columns, and column nodes then aggregate from neighboring constraints. To define this formally, the universal message-passing framework is presented first, followed by the specific SAGEConv implementation. A general message-passing update for a node $i$ receives messages from its neighbors through a formal message function $\phi^{(k)}$ and combines them using a permutation-invariant aggregation operator. This can be seen in Equation 4.17.

$$a_i^{(k)} = \text{agg}_{j \in \mathcal{N}(i)} \left\{ \phi^{(k)} \left( h_i^{(k-1)}, h_j^{(k-1)} \right) \right\} \tag{4.17}$$

where $\mathcal{N}(i)$ denotes the neighbors of node $i$. The aggregated neighborhood information $a_i^{(k)}$ is subsequently combined with the node's previous embedding and transformed by an update function $\psi^{(k)}$. This can be seen in Equation 4.18.

$$h_i^{(k)} = \psi^{(k)} \left( \left[ h_i^{(k-1)} \,\|\, a_i^{(k)} \right] \right) \tag{4.18}$$

where $\|$ denotes vector concatenation. This general update process is visualized in Figure 4.4.

To demonstrate exactly how this universal notation translates into concrete SAGEConv operations, one can trace the step-by-step intuition for a single column node during one layer. The column node begins with its current mathematical representation of di-

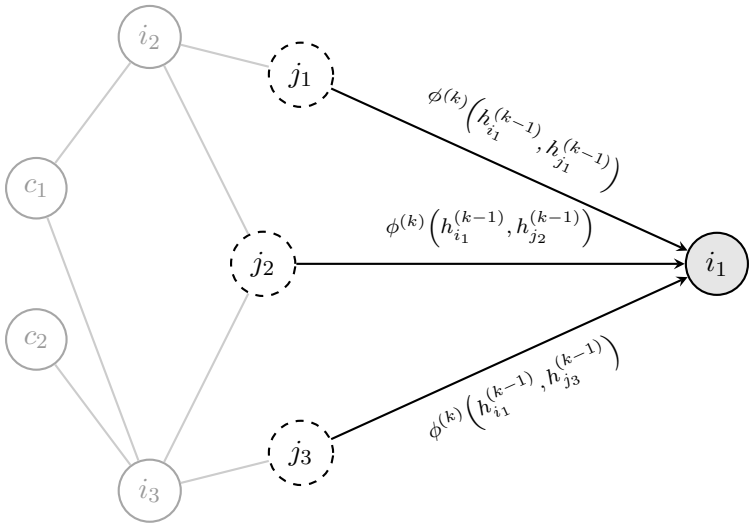



**Figure 4.4:** General message-passing update for target column node $i_1$. The local neighborhood $\mathcal{N}(i_1)$ consists of active constraint nodes (dashed) labeled $j_1, j_2, j_3$, while inactive column nodes $(i)$ and constraint nodes $(c)$ are visually faded. For each active neighbor, a message is generated based on its current embedding. These messages are aggregated and combined with the target node's own embedding to produce its updated state for the next layer.

mension $d_h$. It looks at all the constraint nodes it is connected to and gathers their $d_h$-dimensional embeddings. In this specific architecture, the formal message function $\phi$ simply retrieves the neighbor's current mathematical representation without altering it, allowing the general notation to be simplified in the detailed phase equations later in this section. The column node summarizes these gathered vectors into a single vector using a mathematical operator such as the mean. This summarized vector is the aggregated neighborhood information $a^{(k)}$. Next, the column node concatenates this summarized vector with its own embedding. This means the two vectors are placed side by side to form a single longer vector with a dimension of $2d_h$. At this point, the update function $\psi^{(k)}$ executes its core logic. The learnable mapping matrix $W_V^{(k)}$ multiplies this long $2d_h$ vector to shrink it back down to the original $d_h$ dimension. Finally, this compressed vector is passed through the non-linear activation $\rho$ to produce the proposed update. Through this matrix multiplication and activation, the network learns exactly how to weigh and blend the node's past history with the new neighborhood context.

The exact logic is formalized in two distinct phases because the network operates on a bipartite graph. The mathematical rules for updating a constraint are different from the rules for updating a duty column, requiring distinct mapping matrices. Phase 1 applies this sequence to the constraint nodes using the matrix $W_C^{(k)}$ as defined in Equation 4.19. Phase 2 applies the exact same sequence to the column nodes using the matrix $W_V^{(k)}$ as given in Equation 4.20.

$$\begin{aligned} a_c^{(k)} &= \mathrm{agg}_{v \in \mathcal{N}(c)}\left(h_v^{(k-1)}\right) \\ z_c^{(k)} &= \rho\left(W_C^{(k)} \begin{bmatrix} h_c^{(k-1)} \\ a_c^{(k)} \end{bmatrix}\right) \\ h_c^{(k)} &= \mathrm{LayerNorm}\left(h_c^{(k-1)} + \mathrm{Dropout}_p\left(z_c^{(k)}\right)\right) \end{aligned} \tag{4.19}$$

$$
\begin{aligned}
a_v^{(k)} &= \mathrm{agg}_{c \in \mathcal{N}(v)}\big(h_c^{(k-1)}\big) \\
z_v^{(k)} &= \rho\left(W_V^{(k)} \begin{bmatrix} h_v^{(k-1)} \\ a_v^{(k)} \end{bmatrix}\right) \\
h_v^{(k)} &= \mathrm{LayerNorm}\big(h_v^{(k-1)} + \mathrm{Dropout}_p\big(z_v^{(k)}\big)\big)
\end{aligned}
\tag{4.20}
$$

where in both phases, dropout with probability $p$ is applied to the proposed update during training to limit overfitting. In these specific equations, the proposed update $z^{(k)}$ directly embodies the result of the abstract update function $\psi^{(k)}$, and the raw feature embedding $h^{(k-1)}$ replaces the general message function $\phi^{(k)}$. After this proposed update is calculated, the actual node state is finalized through a residual connection. The node now possesses its old memory vector and the newly proposed update vector $z^{(k)}$. Both are vectors of size $d_h$. Rather than taking an average, the network mathematically adds them together. Adding the proposed update to the original state ensures the node does not forget its initial identity as it absorbs the new neighborhood context. Finally, layer normalization rescales the combined vector to stabilize training. Only $h_c^{(k)}$ and $h_v^{(k)}$ are carried to the next layer. Figure 4.5 illustrates one such layer.

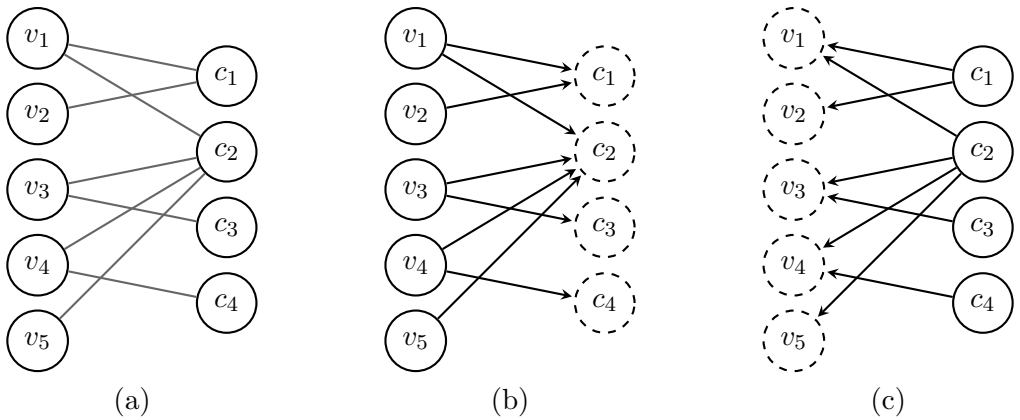



**Figure 4.5:** Bipartite graph model illustrating the two-phase message-passing architecture. (Figure 4.5a) The base graph consists of column nodes $V$ and constraint nodes $C$. (Figure 4.5b) Phase 1 constraint nodes (dashed) aggregate messages from their neighboring column nodes to update their representations. (Figure 4.5c) Phase 2 column nodes (dashed) subsequently aggregate messages from their neighboring constraint nodes.

The number of layers $K$ controls how far information can travel through the graph. With one layer, a column can use information from its adjacent constraints. With two layers, column embeddings can also reflect other columns connected through those constraints. To explore the optimal depth for this specific problem, the experiments in this study will test models with one, two, three, and four message-passing layers. Deeper models are intentionally avoided due to three primary challenges. The first challenge is over-smoothing. When a network becomes too deep, repeated aggregation causes all node embeddings to blend together. If every column repeatedly pulls in information from a wider neighborhood, the final mathematical representations become nearly identical, reducing the model's ability to distinguish good candidate duties from poor ones. A second and related issue is oversquashing. As the neighborhood expands exponentially with each

additional layer, an enormous amount of information from distant nodes must be compressed into a single vector of fixed size. This forces the network to discard potentially critical local details in favor of a blurred global view. The third challenge is computational cost. Each additional layer multiplies the number of neighboring nodes involved in the calculation, drastically increasing the memory and time required during training.

The third step executes the readout, which assigns a selection probability to each generated column. After the $K$ message-passing layers, only the final column embedding $h_v^{(K)}$ enters the readout head. Constraint nodes are updated during message passing because they carry RMP-side context into the column embeddings, but they are not classified directly. The readout head is a small two-layer network, which is given in Equation 4.21.

$$\begin{aligned} q_v &= \rho\big(W_1 h_v^{(K)}\big) \\ \hat{y}_v &= \sigma\big(W_2 \,\mathrm{Dropout}_p(q_v)\big) \end{aligned} \tag{4.21}$$

where $W_1 \in \mathbb{R}^{(d_h/2)\times d_h}$ and $W_2 \in \mathbb{R}^{1\times(d_h/2)}$ act as the final mappings. They project the sophisticated final embedding first to an intermediate representation $q_v \in \mathbb{R}^{d_h/2}$ and then down to a single probability score. Like the previous weights, these mappings are learned by minimizing the binary cross-entropy loss against the true labels. The same dropout probability $p$ is applied here as in the message-passing layers, so the small classification head does not overfit particular dimensions of $h_v^{(K)}$ during training. The function $\sigma$ is the sigmoid from Equation 4.10. Training and evaluation use only the generated columns in $G_\ell$, identified by a boolean mask.

#### 4.5.2.2 Aggregation and attention variants

The SAGEConv formulation leaves a modeling choice in the aggregation operator *agg*. This operator determines how the neighboring embeddings are summarized before they are combined with the node's own embedding. This design directly mirrors the aggregated logistic regression model introduced previously, where the mean operator serves as the primary baseline. For a column node, the mean operator calculates the average state of the constraints it covers. This approach can similarly be extended by applying alternative operators to capture distinct structural signals from the neighborhood. The maximum and minimum operators are particularly well-suited for identifying bottlenecks. Alternatively, the sum operator reflects the total accumulated state of all covered constraints. However, unlike the linear baselines which apply these mathematical summaries directly in a basic equation, the GNN passes these aggregated messages through the non-linear mappings discussed in the previous section. This fundamental difference allows the network to capture complex interactions between a duty and its surrounding constraints rather than just adding their effects together.

A second variant replaces the fixed SAGEConv aggregation by graph attention, using the GATv2 architecture developed by Brody et al. (2022). With fixed aggregation like the mean, all neighbors are combined by a predetermined mathematical rule. Attention instead introduces learnable mappings, allowing the model to dynamically decide how much importance each neighbor should receive. In the context of column selection, this means the network learns to prioritize certain constraints over others when evaluating a duty. For instance, it can heavily weight a constraint that represents a hard-to-cover

task while ignoring redundant ones. GATv2 is specifically chosen because it employs a dynamic attention mechanism. Unlike earlier attention models, it allows the weighting to adapt genuinely to the specific target node being updated rather than ranking neighbors uniformly across the graph. For an edge from node $u$ to node $v$, a typical attention score is given by Equation 4.22.

$$\begin{aligned} e_{uv} &= \theta^{\top} \eta\big(W_{\text{att}} \big[h_u^{(k-1)} \,\|\, h_v^{(k-1)}\big]\big) \\ \alpha_{uv} &= \frac{\exp(e_{uv})}{\sum_{u' \in \mathcal{N}(v)} \exp(e_{u'v})} \end{aligned} \tag{4.22}$$

where $e_{uv}$ is an unnormalized compatibility score and $\alpha_{uv}$ is the normalized attention weight. The subscripts $u$ and $v$ simply denote the direction of the message flowing from a specific neighbor $u$ to a specific target node $v$. The parameters $W_{\text{att}}$ and $\theta$ act as a small neural network dedicated to scoring this specific connection. The matrix $W_{\text{att}}$ first transforms the concatenated node representations into a hidden state. After the non-linear activation $\eta$ is applied, the parameter vector $\theta$ projects this hidden state down to a single numerical score. Although these weight parameters are shared globally across the entire network, they evaluate the combined representations of both the neighbor and the target node together. This mechanism allows the shared network to compute a unique compatibility score for every single connection. During training, backpropagation adjusts these exact parameters based on the classification error. If focusing on constraints with high dual values helps the model predict expert selections accurately, the weights naturally adjust to output a high attention weight $\alpha_{uv}$ whenever that specific mathematical pattern is detected in a neighbor's vector.

This dynamic scoring fundamentally changes how the proposed update $z^{(k)}$ is generated compared to SAGEConv. In the SAGEConv implementation, neighbors are first aggregated into a single vector, which is then transformed by a weight matrix to produce $z^{(k)}$. In contrast, GATv2 first transforms each neighbor's embedding individually using a learnable weight matrix $W_{\text{trans}}$. Each transformed embedding is then multiplied by its corresponding attention weight $\alpha_{uv}$, effectively acting as a volume control that scales the message by its importance. These scaled messages are then summed to produce the final proposed update $z^{(k)}$. This approach is more expressive because it computes unique, importance-weighted features for every neighbor before aggregation. The embedding $h^{(k-1)}$ no longer enters through concatenation, but is kept only through the same residual addition as in SAGEConv, followed by dropout and layer normalization. The two-phase bipartite structure therefore remains unchanged. Multi-head attention performs several such weighted aggregations in parallel, where each head learns a separate attention mechanism. One can think of the heads as several people looking at the same neighborhood before their views are combined. Increasing the number of GAT layers results in repeated attention-based message passing, where each layer computes new attention weights based on the updated node representations. To ensure a comprehensive evaluation, all aggregation and attention variants will be tested across the one to four layer depths previously described.

### 4.5.3 Hyperparameters

Each model class has a set of hyperparameters that must be specified before training. Hyperparameters are settings that control the learning process itself rather than the model weights that are fitted to data. They therefore cannot be estimated from the training set in the same way as the model parameters and instead require a deliberate choice.
For the logistic regression models, the main hyperparameter is the inverse regularization strength $C$, which controls the balance between fitting the training data and keeping the learned weights small. A smaller $C$ imposes stronger regularization and limits how large any individual feature weight can become, which helps the model generalize to unseen planning days. A larger $C$ allows the optimizer more freedom to fit the training data precisely but increases the risk of overfitting. The class weights are not a free hyperparameter in the usual sense. They are computed directly from the class frequencies in the training period and therefore change automatically with the data. Table 4.3 summarizes the settings used for both linear models.

**Table 4.3:** Hyperparameter settings for LR and Agg-LR.

| Hyperparameter | Value | Description |
|---|---|---|
| $C$ | 1.0 | Inverse $L_2$ regularization strength |
| Class weights | computed | Derived from training class frequencies |

For the GNN, the number of trainable parameters and the expressive power of the model depend on several architectural and optimization choices. Table 4.4 lists the initial settings used when evaluating and comparing architectures. The hidden dimension $d_h$ determines the size of the embedding space that every column and constraint node is mapped into by the input projections in Equation 4.15. With $d_h = 64$, each node is represented by a 64-dimensional vector that is updated across the message-passing layers. The readout head in Equation 4.21 then maps this to a score in $[0, 1]$ through an intermediate representation of size $d_h/2 = 32$. Dropout is applied after each activation in the message-passing layers and in the head, randomly setting a fraction $p$ of values to zero during training. This prevents any single dimension of an embedding from being relied on too heavily and acts as a regularizer. The parameters are updated with the Adam optimizer from Kingma and Ba (2017). This is an adaptive gradient-based optimizer that adjusts the effective step size separately for each parameter, while a single global learning rate scales all updates. Weight decay adds a small penalty proportional to the squared weight magnitude, similar in spirit to the $L_2$ regularization used for LR. Training proceeds in epochs, where each epoch corresponds to one complete pass over the shuffled training graphs. The process is capped at a maximum of 50 epochs. For the attention variant, the hidden dimension is additionally divided into four parallel attention heads. Each head operates on a $d_h/4 = 16$-dimensional subspace, and their outputs are concatenated back to $d_h$ after each layer.

These initial settings are used to compare architectures under equal conditions. The number of message-passing layers and the aggregation or attention variants from Section 4.5.2.2 are first compared while keeping all other settings fixed. Once the strongest architecture has been identified, a focused search over the hidden dimension, dropout probability, learning rate, and weight decay is performed to find settings that improve over the initial choice.

**Table 4.4:** Initial hyperparameter settings for the GNN and GNN-Attention models. These values are kept fixed during the architecture comparison.

| Hyperparameter | GNN (SAGEConv) | GNN-Attention | Description |
|---|---|---|---|
| Hidden dimension $d_h$ | 64 | 64 | Node embedding size |
| Number of layers $K$ | 1–4 | 1–4 | Message-passing rounds |
| Aggregation / heads | varied | 4 heads | SAGEConv operator or GATv2 heads |
| Dropout $p$ | 0.1 | 0.1 | Drop probability in layers and head |
| Learning rate | $10^{-3}$ | $10^{-3}$ | Adam step size |
| Weight decay | $10^{-5}$ | $10^{-5}$ | $L_2$ penalty on model weights |
| Batch size | 32 | 32 | Graphs per training update |
| Maximum epochs | 50 | 50 | Maximum number of training passes |

### 4.5.4 Training and evaluation

The data are split chronologically into training, validation, and test periods. The training period is used to estimate the model parameters. The validation period is held out during fitting and used to compare candidate models and guide hyperparameter selection, while the test period remains separate until the final classification comparison. This chronological division prevents information from later planning days from entering model development.

The only preprocessing step is feature standardization. For each feature, the mean and standard deviation are estimated from the training period. The same transformation is then applied to the validation and test periods. LR and Agg-LR standardize their flat input vectors, while the GNN standardizes the column and constraint feature matrices before batching the graphs. No statistics from the validation or test periods are used in this process.

All three model families are trained with the weighted binary cross-entropy loss defined in Equation 4.13. The model architecture determines how the prediction is produced, but not the underlying classification objective. For the GNN, the implementation evaluates this loss directly from the output logits before applying the sigmoid. This is mathematically equivalent to applying the weighted binary cross-entropy to the probabilities, but is more numerically stable. Since the labels are strongly imbalanced, selected columns receive more weight than non-selected columns. The class weights for all models are determined from the training period only.

The GNN is trained on shuffled mini-batches of graphs using the settings from Table 4.4. Gradient clipping limits unusually large parameter updates during training. After each epoch, the model is evaluated on the validation period. A checkpoint is a stored copy of the model parameters at a particular epoch. Whenever the validation loss improves, the current parameters are retained as the new best checkpoint. Training stops when the validation loss has not improved for a fixed number of epochs, after which the parameters from the best checkpoint are restored. This prevents the final model from being determined by a later epoch in which training performance improved while validation performance deteriorated. Alongside early stopping, the learning rate is halved automatically whenever the validation loss fails to improve for several consecutive epochs, allowing the optimizer to take smaller steps as training converges near a good solution. During validation and test, dropout is disabled and predictions are made using the full network

to ensure deterministic outputs.

Once hyperparameter search has identified the best configuration, the final GNN is retrained from scratch on the combined training and validation period. The number of epochs is set to the epoch at which early stopping selected the best checkpoint during the search, rather than running for the full maximum. This uses the additional validation labels while keeping a fixed training budget, since early stopping is no longer available once the validation period is included in training. The feature standardizer is also refit on the combined data. The test period is then used only for the final unbiased evaluation of this deployment model.

Each model returns a predicted selection probability $\hat{y}_i$ for candidate column $i$. At the default decision threshold of 0.5, every candidate with $\hat{y}_i \geq 0.5$ is classified as selected and every candidate with $\hat{y}_i < 0.5$ is classified as not selected. Lowering this threshold selects more columns and generally increases recall at the expense of precision. Raising it has the opposite effect.

Because of the imbalance, standard accuracy is not sufficiently informative, as most candidates are not selected by the expert. A model that rejects nearly all columns could still look accurate, while it would be useless in column generation. Evaluation instead starts from the numbers of true positives TP, false positives FP, true negatives TN, and false negatives FN. A true positive is a column selected by both the expert and the model, while a true negative is rejected by both. A false positive is selected only by the model, while a false negative is selected only by the expert.

Recall measures the share of expert-selected columns recovered by the model, defined in Equation 4.23.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{4.23}$$

Recall is particularly important because a false negative removes a direction that the expert considered useful. The true negative rate measures the share of expert-rejected columns that the model also rejects, given by Equation 4.24.

$$\text{TNR} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{4.24}$$

A high true negative rate helps prevent unnecessary columns from enlarging the RMP. Precision instead evaluates the predicted selections, which can be seen in Equation 4.25

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4.25}$$

The precision reports the share of model-selected columns that were also selected by the expert. Balanced accuracy gives equal importance to recovering selected columns and rejecting non-selected columns, defined in Equation 4.26.

$$\text{Balanced accuracy} = \frac{1}{2}\left(\text{Recall} + \text{TNR}\right) \tag{4.26}$$

These four measures depend on the decision threshold. The area under the precision-recall curve is therefore also reported. The curve evaluates precision and recall over all possible thresholds. In the implementation, its area is summarized by average precision, given by Equation 4.27.

$$\text{AUC-PR} = \sum_{k} (R_k - R_{k-1})\, P_k \tag{4.27}$$

where $P_k$ and $R_k$ are the precision and recall obtained at the $k$-th probability threshold. AUC-PR evaluates how well the model ranks expert-selected columns above non-selected columns without fixing one operating threshold. A random classifier would achieve an AUC-PR equal to the positive rate, as it assigns scores independently of the true labels. Consequently, the expected precision at every level of recall is the fraction of positive samples in the dataset. This makes the positive rate the natural baseline for interpreting AUC-PR, particularly for imbalanced classification problems.

Prediction time is reported alongside these classification metrics. This is necessary because the learned selector is called repeatedly inside column generation. A model that closely reproduces the expert but requires too much prediction time would not provide a useful replacement.

## 4.6 Experimental design

Having defined the selectors and their training procedure, this section describes how they are evaluated within the column generation process. The design separates the effect of column-selection mechanism from differences caused by fairness initialization and accumulated path dependence. All selector evaluations and machine learning data-collection runs use the same history initialization, the same controlled per-day procedure, and the same column generation parameters. Exact instance composition, dates, and parameter values are reported in Section 5.1.

### 4.6.1 Warm start for fairness history

The SS&S penalty mechanism described in Section 2.3 depends on both the accumulated fairness history of each crew member and the characteristics of the duty being assigned. The fairness history evolves over time as duties are assigned and influences the penalty factors attached to future duty costs. Starting from an empty history would make the early fairness scores sensitive to the arbitrary beginning of the experiment because little previous work would be available in the rolling evaluation windows. Such behavior is not representative of actual operations, where crew members enter the planning horizon with an existing work history.

A warm-start period is therefore used before the evaluation window. Its length can influence the initial fairness state if it is too short. The selected length follows the practice advised by NS and allows the rolling fairness measures to stabilize before experimental observations are recorded. The same historical duty records and warm-start length are used for every selection strategy. This removes differences caused by initialization and gives each crew member a realistic and common starting state. The principle follows the history initialization used in the rolling-horizon simulation framework of van Rossum et al. (2026).

### 4.6.2 Per-day experimental control

An additional consideration arises from the path dependence of the diving phase discussed in Section 4.2. If the duty assignments produced by the column generation procedure on one day were used to update the fairness history for the next day, each selection approach would gradually evolve toward its own history state. Over a multi-week evaluation period, these accumulated deviations could produce substantially different fairness landscapes, making the resulting column generation behavior difficult to compare. Any observed performance differences would then reflect not only the selection strategy itself, but also the compounded effects of all preceding assignments.

To avoid this, the fairness history is derived entirely from a fixed folder of historical duty records, as illustrated in Figure 4.6. Following the initial warm start, the corresponding historical records are used to construct the starting conditions for each subsequent day. After the column generation procedure finishes, the resulting assignments are recorded but are not used to update the fairness counters. Instead, the historical duties for that calendar day are read from the same folder to advance the state for the following day. In this way, the fairness evolution remains aligned with actual historical operations while remaining independent of the selection mechanism under evaluation.

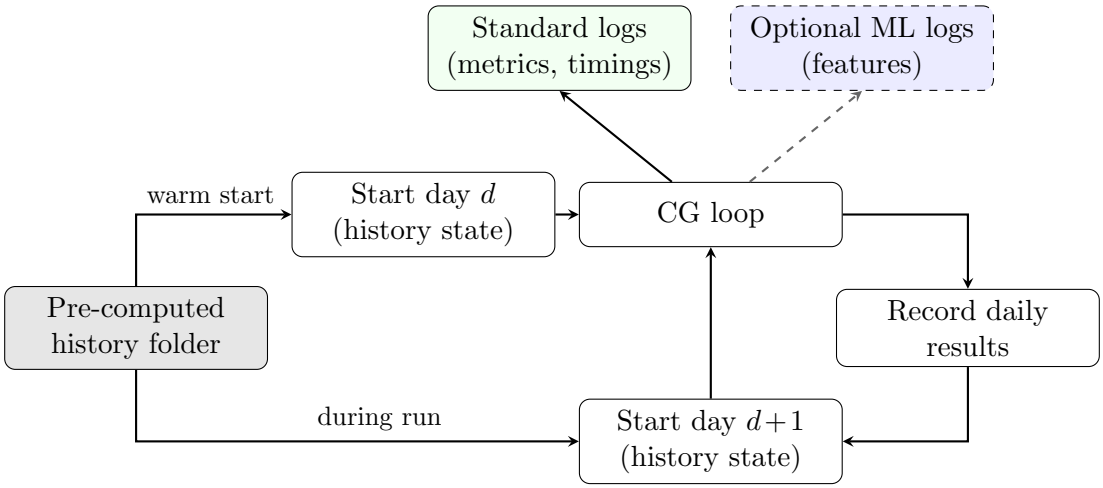



**Figure 4.6:** Controlled per-day procedure. Grey denotes the fixed historical input, green denotes standard solver logs, and dashed blue denotes the optional graphs and expert labels used for machine learning. Experimental assignments and logged outputs do not update the fairness history.

Two logging paths record the experiment without affecting this controlled history. Standard logs are produced for every selector evaluation. They capture iteration-level solver statistics and timings as well as the final outcome of each daily run, and provide the measurements reported in the results. During machine learning data collection, an additional logging path stores the iteration graphs and expert labels. Both paths are outputs of the column generation procedure. Neither changes its decisions nor supplies information to the fairness state of the following day.

### 4.6.3 Machine learning data

The machine learning dataset is collected within the controlled per-day procedure shown in Figure 4.6. Each planning day starts from the fairness state derived from the common historical records, so the logged graphs are not affected by assignments produced on preceding experimental days.

The dataset is generated on the smaller three-base instance, and the models are trained only on graphs and labels from this source instance. This is a methodological choice because repeating the full label-generation procedure on the larger six-base instance would require solving the expert MILP at every recorded iteration. The larger instance is therefore reserved for transfer evaluation, where the trained model is applied without retraining.

Following Morabit et al. (2021), graph logging is restricted to the root-node linear relaxation. At the root node, no duty variables have yet been fixed, so the recorded states are not conditioned on earlier fixing decisions within that day's diving path. Once diving begins, those decisions change the subsequent subproblems and produce path-dependent states, as explained in Section 4.2. Restricting collection to the root node therefore provides a more controlled learning signal. Learning directly from post-dive states is left as a potential extension. This distinction also motivates the root-node and hybrid evaluations. The learned selector is most directly supported at the root node, while applying it throughout the full day forms an additional operational test beyond the setting studied by Morabit et al. (2021).

At every root-node iteration, the expert MILP is solved on the generated candidate set. Its binary values $y^{\delta}_{rt}$ are stored as the labels defined in Equation 4.7. The graph topology and node features shown in Figure 4.3 and defined in Section 4.4.3 are recorded at the same point in the iteration.

### 4.6.4 Selector integration and evaluation modes

The expert and learned selectors are evaluated within the same column generation procedure as the baseline. At each iteration, the current RMP is solved and the pricing problems are solved to generate duties with negative reduced cost. The task-disjoint rule then constructs the candidate pool $G_\ell$ described in Section 4.1. The configured selector determines which of the candidates in $G_\ell$ are inserted into the RMP. The baseline inserts the task-disjoint pool directly. The expert solves the MILP and applies the configured safety mechanism. The learned selector constructs the graph representation and inserts candidates whose predicted probability is equal to or above the selected threshold. The main comparisons use a prediction threshold of 0.5. Alternative thresholds are examined to assess the recall-precision trade-off when the learned selector is integrated into column generation.

Algorithm 2 summarizes how the selector is integrated into the daily procedure from Algorithm 1. Three evaluation modes are distinguished. Root-node evaluation stops before diving and isolates the initial linear relaxation, which is the setting on which the learned selector is trained. Full-day evaluation keeps the configured selector active after diving and tests whether it remains effective in the path-dependent states. Hybrid evaluation uses the expert or learned selector at the root node and switches to the baseline for every later column generation segment. Consequently, all approaches use the same selector after diving. This limits additional differences caused by post-dive selection while

still extending the evaluation to a complete daily solution. The hybrid mode therefore shows whether the column pool created at the root node has a lasting effect on full-day performance without requiring the learned selector to operate outside its training setting.

### 4.6.5 Column generation parameters

Finally, several parameter settings influence the behavior of the column generation algorithm.

The first group concerns acceleration mechanisms. These include the number of columns generated by the pricing problems, the task-disjoint pre-filter threshold, and the column management policy. Together, these balance the richness of the candidate pool with the computational effort required during re-optimization. The second group governs the aggressiveness of the main loop and the diving heuristic. The early termination threshold determines how much relative improvement in the objective is required before the main loop continues. A tighter threshold typically leads to more iterations and a stronger linear program relaxation, but at the cost of additional pricing calls. The diving threshold determines how quickly the algorithm commits to integer values during the diving phase, thereby influencing how strongly early decisions are fixed. The final group defines the fairness penalty function. The SS&S penalty is determined by the attribute-specific target threshold and the scaling factors that translate deviations from those targets into assignment costs.

**Algorithm 2** Daily column generation heuristic with a configurable column selector

**Require:** Day $t$, initialized RMP, selector $S$, evaluation mode $M$, parameters $\epsilon$ (early stop), $\tau$ (diving threshold)
**Ensure:** Integer duty assignment (or root-node solution) for day $t$

Set the active selector $S_{\text{active}} \leftarrow S$
**repeat**
    Solve linear program relaxation of current RMP and obtain dual values $(\lambda, \pi)$
    Solve pricing problems (RCSPPs) and collect candidate duties with negative reduced cost
    Apply the task-disjoint rule to construct candidate pool $G_\ell$
    **if** $S_{\text{active}}$ is the baseline **then**
        Select all candidate duties in $G_\ell$
    **else if** $S_{\text{active}}$ is the expert **then**
        Solve the expert MILP and apply its configured safety mechanism to select duties
    **else**
        Construct the iteration graph and predict selection probabilities
        Select candidate duties whose probability is equal to or above the prediction threshold
    **end if**
    Add selected duties to RMP
    Apply column management and remove stale columns with consistently poor reduced cost
**until** no improving duties are found **or** relative objective decrease is below $\epsilon$ in consecutive iterations
**if** $M$ is root-node evaluation **then**
    **return** root-node solution for day $t$
**end if**
**while** current RMP solution is fractional **do**
    Fix all duty variables with value at least $\tau$
    **if** no variable satisfies the threshold **then**
        Fix the variable with highest fractional value
    **end if**
    **if** $M$ is hybrid evaluation **then**
        Set $S_{\text{active}} \leftarrow$ baseline
    **end if**
    Re-solve reduced RMP with column generation using $S_{\text{active}}$
**end while**
**return** resulting integer assignment for day $t$

# Chapter 5

# Results

This chapter presents the experimental findings for the two-part investigation introduced in Section 3.4. The chapter begins by detailing the experimental setup in Section 5.1. Section 5.2 then evaluates expert-guided column selection through the MILP formulation. Starting from the observed stalling behavior of the basic formulation, the section proceeds through a comparison of the four selection configurations defined in Section 4.3, examines the role of the task-disjoint filter, and closes with a full performance evaluation of the selected expert configuration at both the root node and the full planning day. Section 5.3 turns to the machine learning component. The section evaluates classification performance across candidate model architectures, examines the learned selector inside the column generation loop, and assesses sensitivity to the prediction threshold and transfer to a larger instance.

## 5.1 Experimental setup

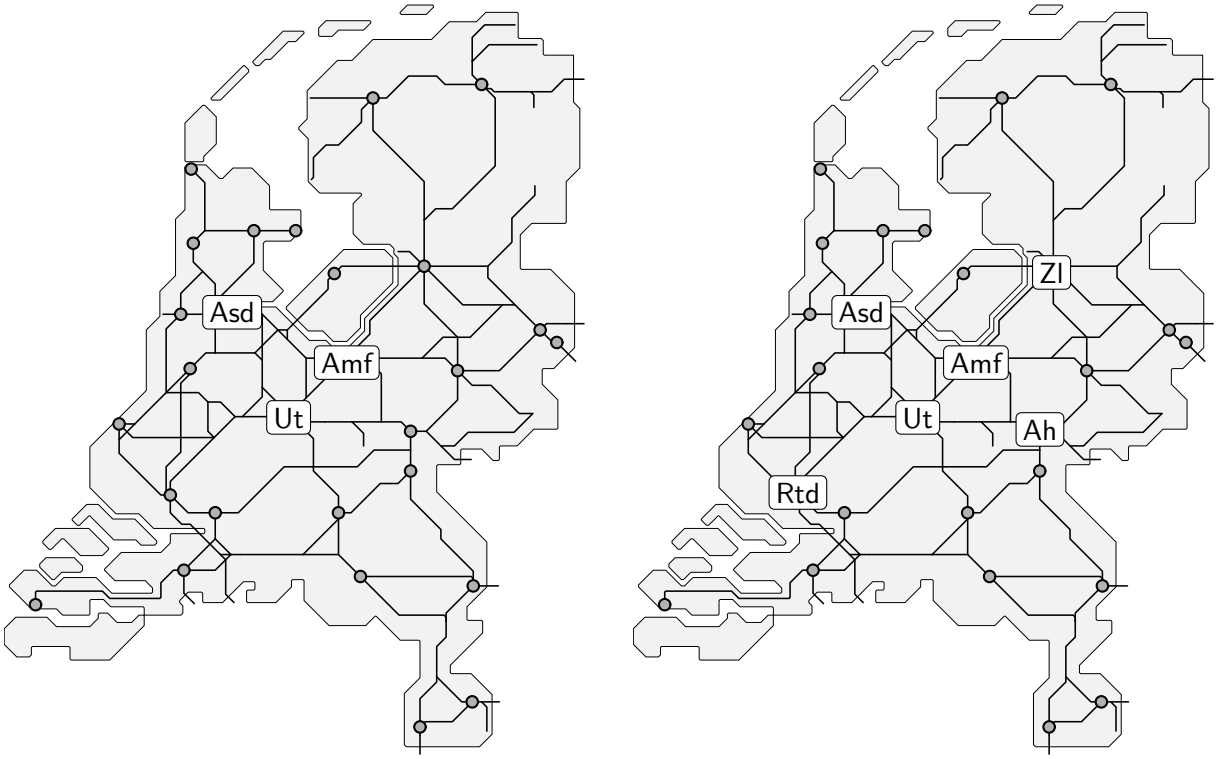
This section specifies the common experimental setting used to compare the column-selection approaches, including the specific planning instances, data periods, parameter configurations, and the computational environment.

### 5.1.1 Experimental instances

In this study, two geographically bounded subsets of the Dutch railway network are used as experimental instances. The first comprises three crew bases, namely Amersfoort (Amf), Amsterdam (Asd), and Utrecht (Ut). The second extends this to six crew bases by additionally including Arnhem (Ah), Rotterdam (Rtd), and Zwolle (Zl). Both instances are shown in Figure 5.1.

The three-base instance corresponds to the `Center` instance used by van Rossum et al. (2024) and is regarded as representative of the operational NS setting. It contains large crew bases connected by busy track segments, capturing the task coverage conflicts and fairness dynamics that characterize the full planning problem. The six-base instance expands the geographical scope, making it possible to examine whether performance carries over to a larger and more complex setting.

Applying the methodology to the complete NS network would require repeated solution of the full planning problem, which is computationally prohibitive for the repeated




**(a)** Instance with three crew bases **(b)** Instance with six crew bases

**Figure 5.1:** Overview of the two instances used in this study. Figure 5.1a depicts the three-base instance containing Amersfoort (Amf), Amsterdam (Asd), and Utrecht (Ut). Figure 5.1b shows the six-base instance, which additionally includes Arnhem (Ah), Rotterdam (Rtd), and Zwolle (Zl). Operated tracks are indicated by black lines and the included crew bases by white labels.

runs performed in this study. Restricting attention to geographically bounded subsets is therefore a common strategy. Following the approach of van Rossum et al. (2026), the daily planning problems within each subset are constructed by grouping all tasks that were historically covered by crew members belonging to the selected bases. This ensures that the resulting problems remain self-contained and preserve the task coverage patterns observed in practice.

To indicate the scale of these real NS instances, Table 5.1 reports the number of tasks and templates for seven consecutive planning days in January 2024. The instances use the guard setting introduced in Section 2.3. A template on a planning day corresponds to one active crew member. The three-base instance contains approximately 1,900 tasks and 190 active crew members per day on average, while the six-base instance contains approximately 3,600 tasks and 350 active crew members. For context, the full operational problem at NS contains more than 10,000 tasks and approximately 1,150 active crew members on a typical weekday. The experimental instances are therefore smaller than the complete operational problem, but remain substantial real-world planning problems.

### 5.1.2 Data periods

All experiments use the controlled history procedure from Section 4.6. For the January experiments, fairness history is initialized from historical duties between 14 November

**Table 5.1:** Number of tasks and templates in the two experimental instances for seven consecutive planning days in January 2024. Templates correspond to active crew members on the respective planning day.

| | Three-base instance | | Six-base instance | |
|---|---|---|---|---|
| Day | Tasks | Templates | Tasks | Templates |
| 08-01 | 1,964 | 191 | 3,725 | 362 |
| 09-01 | 1,995 | 198 | 3,795 | 371 |
| 10-01 | 1,882 | 189 | 3,604 | 357 |
| 11-01 | 2,202 | 218 | 4,176 | 404 |
| 12-01 | 2,052 | 207 | 3,854 | 377 |
| 13-01 | 1,800 | 183 | 3,269 | 332 |
| 14-01 | 1,429 | 147 | 2,678 | 269 |

2022 and 7 January 2024. The first evaluated day is 8 January 2024, which ensures that the accumulated fairness scores have stabilized. This provides each crew member with a realistic assignment history. For later experiments, the same historical records advance the fairness state up to the day preceding the relevant evaluation period. For example, evaluation of the learned selector from 6 May 2024 uses historical duties through 5 May 2024.

The initial expert comparisons cover 8 to 14 January 2024. The selected expert-guided performance comparison extends this window through 21 January. Machine learning data are collected from 8 January to 2 June and split chronologically into training, validation, and test periods. The in-loop evaluation of the learned selector begins with the test period and continues through 30 June 2024. This makes it possible to observe the behavior of the selectors over nearly two months rather than only a few weeks. Table 5.2 summarizes the periods used in this chapter.

**Table 5.2:** Data periods used for expert evaluation, machine learning, and in-loop selector evaluation.

| Purpose | Start date | End date |
|---|---|---|
| Expert configuration comparison | 2024-01-08 | 2024-01-14 |
| Expert performance evaluation | 2024-01-08 | 2024-01-21 |
| Machine learning training | 2024-01-08 | 2024-04-21 |
| Machine learning validation | 2024-04-22 | 2024-05-05 |
| Machine learning test | 2024-05-06 | 2024-06-02 |
| Learned-selector in-loop evaluation | 2024-05-06 | 2024-06-30 |

### 5.1.3 Parameter settings

As noted in Section 2.5, all parameter settings for the column generation heuristic are adopted from the existing NS implementation.

The first group of parameters governs the candidate pool and the restricted master problem size. At most 50 columns are returned per pricing subproblem. The task-disjoint pre-filter accepts a maximum task overlap of 80%. Columns are removed from the restricted master problem when their reduced cost has exceeded 250 in each of the

last five consecutive iterations. The second group controls the main loop and the diving heuristic. The early termination threshold is set to $\epsilon_{\text{term}} = 0.1\%$ and the diving fix threshold is set to $\tau = 0.51$. The third group defines the fairness penalty function and the Sharing-Sweet-and-Sour attribute targets. Unlike the column generation heuristic parameters, which directly control the procedure under study, reporting the complete set of fairness parameter values falls outside the scope of this thesis. The experiments isolate the effect of the column-selection mechanism rather than the design of the fairness feedback itself.

The expert MILP introduces one additional algorithmic parameter. The selection penalty introduced in Section 4.1 is set to $\varepsilon_{\text{MILP}} = 0.01$. In preliminary experiments this value was varied, but these changes had no material effect on the selected columns or on the subsequent column generation trajectories.

### 5.1.4 Computational environment

All experiments are performed on a virtual machine hosted on Microsoft Azure, provided by NS. Specifically, it is a `Standard_D16as_v4` instance running Windows Server 2025. The machine is equipped with 16 virtual CPU cores backed by an AMD EPYC 7763 processor, 64 GiB of RAM, and 32 data disks. The column generation procedure is implemented in Java and uses CPLEX for solving the restricted master problem and the expert MILP. Machine learning training and inference are implemented in Python. Logistic regression models are trained with scikit-learn (Pedregosa et al., 2018), while the graph neural networks are trained with PyTorch and the PyTorch Geometric library (Paszke et al., 2019; Fey and Lenssen, 2019).

During experiments in which a learned model is deployed inside the solver, the Java process requests predictions from a Python inference service running on the same virtual machine. The service is managed through NSSM (Patterson, 2023). This arrangement allows the Java loop to obtain predictions at each relevant iteration without interrupting the optimization flow.

Because the experiments share a virtual machine environment, incidental fluctuations in available compute resources may introduce variation in wall-clock times across runs.

## 5.2 Expert MILP column selection

This section evaluates the expert-guided selector. The primary comparisons are conducted at the root node, where the effect of column selection can be evaluated before diving decisions create different solution paths. The final subsection extends the evaluation to the full planning day. Unless stated otherwise, results are reported for the three-base instance.

Following Chapter 4, results are reported for the baseline selector and the four MILP-based selectors MILP, MILP-GN, MILP-TN, and MILP-TC. MILP uses only the expert MILP, while the other three incorporate a safety-net mechanism. Hereafter, these labels refer to the corresponding selectors.

### 5.2.1 Stalling behavior

The first comparison considers the MILP selector, that is, the expert-guided configuration without a safety net. Table 5.3 reports the final root-node objective values for the baseline and the MILP over the first evaluation week on the three-base instance.

**Table 5.3:** Final root-node objective values for the baseline and the MILP selector.

| Day | Baseline (€) | MILP (€) |
|---|---|---|
| 2024-01-08 | 11,386.20 | 16,406.46 |
| 2024-01-09 | 162,878.44 | 181,525.76 |
| 2024-01-10 | 9,745.42 | 552,922.56 |
| 2024-01-11 | 13,167.34 | 16,585.79 |
| 2024-01-12 | 12,943.64 | 100,977.01 |
| 2024-01-13 | 67,630.87 | 71,124.35 |
| 2024-01-14 | 6,393.47 | 7,030.13 |

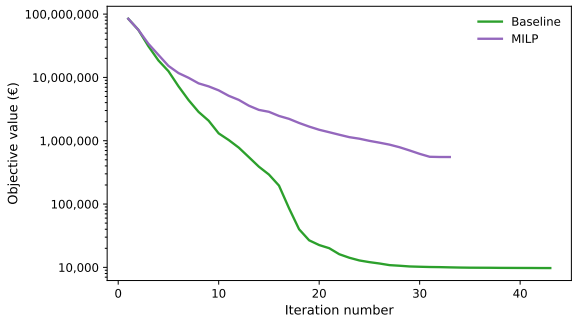
The MILP selector finishes with a higher objective value than the baseline on every day. The size of this difference varies considerably. On 9, 13, and 14 January, both selectors finish in a similar objective range. More substantial gaps occur on 8 and 11 January, while the results on 10 and 12 January stand out most clearly. On 10 January, its final objective is approximately 57 times the baseline objective. On 12 January, it is almost eight times as high. The comparison therefore shows that the MILP selector does not consistently reach the objective level obtained by the baseline within the root-node procedure.

Figure 5.2 provides a more detailed view of this behavior for 10, 12, and 14 January. On 10 and 12 January, the baseline selector continues to reduce the objective after the trajectory of the MILP selector has leveled off. This leaves a visible gap between the configurations at the end of the root node. The trajectory on 14 January has a different shape. Both selectors eventually reach a similar objective range, but the MILP selector takes considerably more iterations to do so. The MILP selector can therefore exhibit either a higher final objective or a longer convergence trajectory, depending on the planning day.

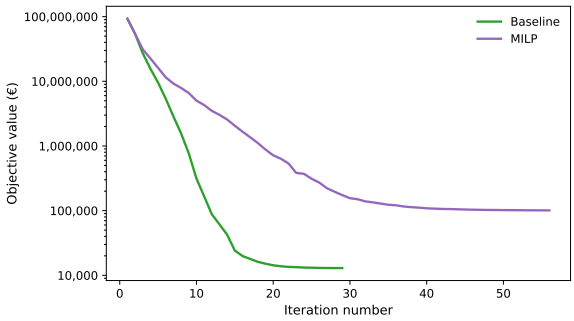
### 5.2.2 Comparison of selection configurations

The comparison above showed that the MILP selector can stall relative to the baseline. The three safety-net variants MILP-GN, MILP-TN, and MILP-TC are therefore compared with that basic configuration next. The comparison again uses the first evaluation week on the three-base instance and focuses on final objective values, RMP growth, and the number of columns selected per iteration. Table 5.4 reports the final root-node objectives for all four MILP-based selectors together with the baseline.

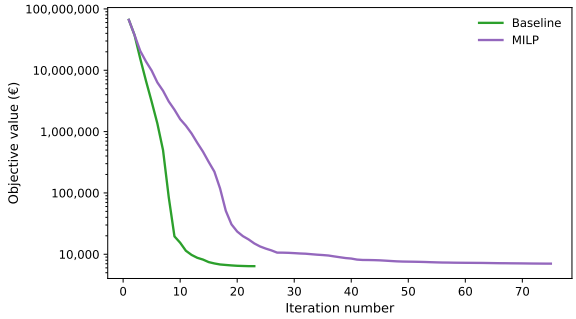
All three safety-net configurations reduce the objective gap relative to the MILP selector. This is particularly visible on 10 and 12 January, where the large differences reported for the MILP selector mostly disappear. On the remaining days, the three safety-net configurations finish close to one another and remain near the baseline objective value. Summed over the week, the final objective is 2.85% above the baseline for MILP-GN, 2.64% for MILP-TN, and 3.15% for MILP-TC. MILP-TN has the smallest aggregate difference, although none of the three configurations is uniformly best on every individual day. The MILP selector is 233.13% above the baseline, primarily due to the results on




**(a)** 10 January 2024




**(b)** 12 January 2024




**(c)** 14 January 2024

**Figure 5.2:** Root-node objective trajectories for the baseline and the MILP selector on three representative days. Objective values are shown on a logarithmic scale.

**Table 5.4:** Final root-node objective values for the baseline and four expert-guided configurations.

| | Objective value (€) | | | | |
|---|---|---|---|---|---|
| Day | Baseline | MILP | MILP-GN | MILP-TN | MILP-TC |
| 2024-01-08 | 11,386.20 | 16,406.46 | 12,148.50 | 11,915.31 | 12,389.43 |
| 2024-01-09 | 162,878.44 | 181,525.76 | 164,254.10 | 164,484.59 | 164,737.64 |
| 2024-01-10 | 9,745.42 | 552,922.56 | 10,639.75 | 10,543.28 | 10,550.65 |
| 2024-01-11 | 13,167.34 | 16,585.79 | 14,475.75 | 14,251.46 | 14,651.11 |
| 2024-01-12 | 12,943.64 | 100,977.01 | 15,012.71 | 14,204.30 | 14,624.35 |
| 2024-01-13 | 67,630.87 | 71,124.35 | 69,055.24 | 69,846.81 | 69,311.92 |
| 2024-01-14 | 6,393.47 | 7,030.13 | 6,655.69 | 6,408.93 | 6,826.92 |

10 and 12 January. Table 5.5 reports the corresponding final RMP sizes and the average number of columns selected per iteration. In this comparison, selected columns are the duties ultimately added to the RMP in an iteration. For MILP-GN and MILP-TN, this includes both the initial expert selection and the later safety-net additions.

The lower objective values of the safety-net configurations coincide with larger selected subsets and larger final RMPs than for the MILP selector. Across the evaluation days, the baseline's average number of selected columns per iteration ranges from approximately 1,100 to 1,500, whereas the corresponding daily averages for the MILP selector lie between 55 and 84. MILP-GN selects the broadest expert-guided subsets, with daily averages between approximately 167 and 296 columns per iteration. MILP-TN and MILP-TC lie

**Table 5.5:** Final RMP size and average number of selected columns per iteration for the baseline and four expert-guided configurations. The GN, TN, and TC denotes the three safety-net configurations.

| | Final RMP size | | | | | Average selected columns | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Day | Baseline | MILP | GN | TN | TC | Baseline | MILP | GN | TN | TC |
| 2024-01-08 | 17021 | 2152 | 4811 | 3050 | 3653 | 1263.6 | 55.1 | 232.3 | 133.2 | 152.8 |
| 2024-01-09 | 14772 | 2760 | 3738 | 3184 | 3574 | 1201.5 | 64.9 | 166.8 | 140.0 | 157.8 |
| 2024-01-10 | 12258 | 2408 | 3804 | 2734 | 3086 | 1123.5 | 72.8 | 200.1 | 124.4 | 148.1 |
| 2024-01-11 | 14150 | 2218 | 5312 | 3003 | 3538 | 1459.2 | 71.9 | 296.3 | 154.5 | 169.9 |
| 2024-01-12 | 19671 | 3031 | 6264 | 3318 | 4061 | 1518.9 | 68.9 | 260.5 | 153.1 | 176.3 |
| 2024-01-13 | 16127 | 1981 | 4491 | 2992 | 3279 | 1509.2 | 83.6 | 218.4 | 151.8 | 168.7 |
| 2024-01-14 | 9694 | 1410 | 3003 | 2031 | 2281 | 1201.2 | 57.1 | 184.9 | 134.7 | 145.5 |

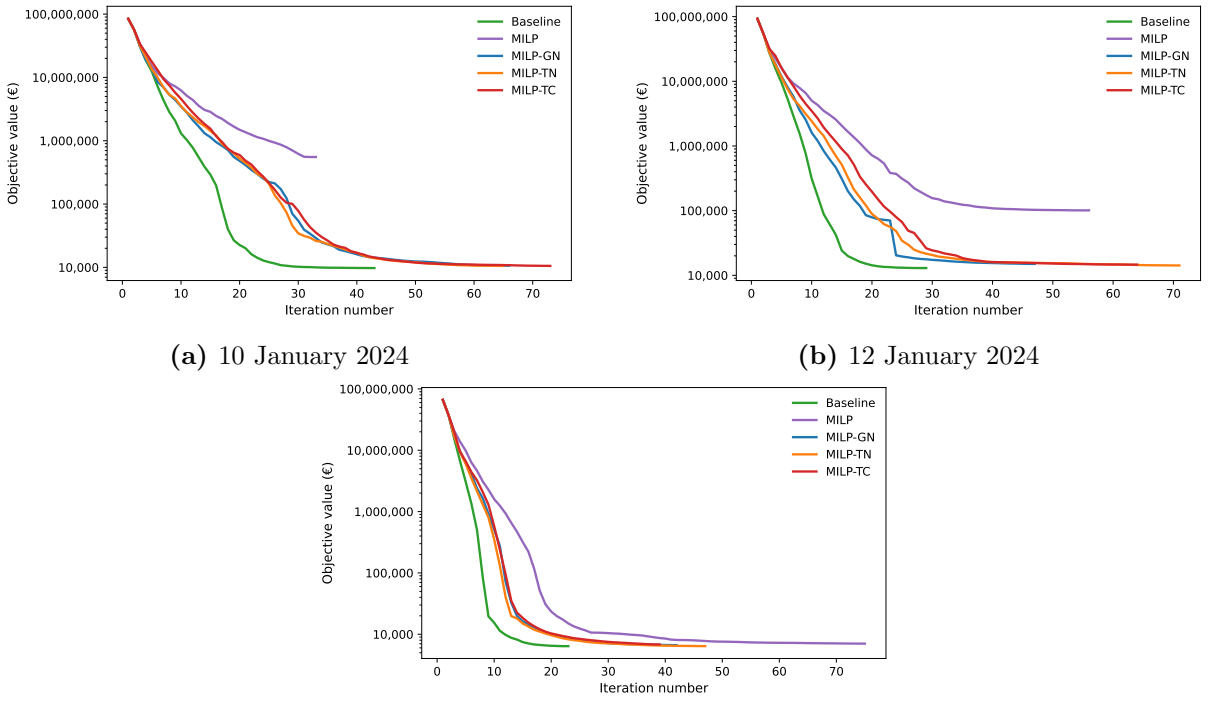
between MILP-GN and the MILP selector. Their final RMP are correspondingly larger than that of the MILP selector, but remain substantially smaller than the baseline RMP on every day. The safety mechanisms therefore recover much of the objective difference without returning the RMP to its baseline size.

Figure 5.3 shows how these aggregate differences develop over the root-node iterations. On 10 January, all three safety-net configurations continue beyond the objective level at which the MILP selector remains for much of the run. Their trajectories then move into the same final objective range as the baseline. On 12 January, the safety-net configurations also avoid the prolonged higher objective level visible for the MILP selector. On 14 January, the final objectives already lie close together, but the safety-net configurations exhibit a shorter tail than the MILP selector.

The total RMP time, shown in Figure 5.4, is higher for each safety-net configuration than for the MILP selector. MILP-GN has the highest RMP time on every day, which is consistent with its larger selected subsets and final RMPs. MILP-TN and MILP-TC are closer to one another across the week. MILP-TN additionally requires the intermediate RMP re-optimization shown by the semi-transparent portion of the bars, while MILP-TC has no corresponding component. Despite this difference in procedure, their final objective values and RMP sizes remain close.

Figure 5.5 illustrates the development of the RMP on 12 January. Figure 5.5a shows that the baseline adds the largest number of columns throughout the root node and consequently maintains the largest RMP. The MILP selector remains at the other end of the comparison, with small additions and the smallest RMP. The three safety-net configurations fall within the range between these two approaches. Although they select more columns per iteration than the MILP selector, as shown in Figure 5.5b, their RMP pools remain considerably smaller than the baseline pool. The downward steps in the RMP-size trajectories occur when column management removes inactive columns.

Taken together, MILP-TN and MILP-TC provide the closest balance between final objective, selected subset size, and RMP time. MILP-TC is retained for the remaining experiments. Its objective values and RMP sizes are comparable to MILP-TN, while it does not require an intermediate re-optimization. Consequently, its labels follow directly from one expert solve and provide a clearer target for supervised learning. In particular, the selected set does not depend on a second round of column additions after the expert solve. This is a step that a supervised model would not observe when imitating the expert.




**(a)** 10 January 2024

**(b)** 12 January 2024

**(c)** 14 January 2024

**Figure 5.3:** Root-node objective trajectories for the baseline, the MILP selector, and the three safety-net configurations. Objective values are shown on a logarithmic scale.

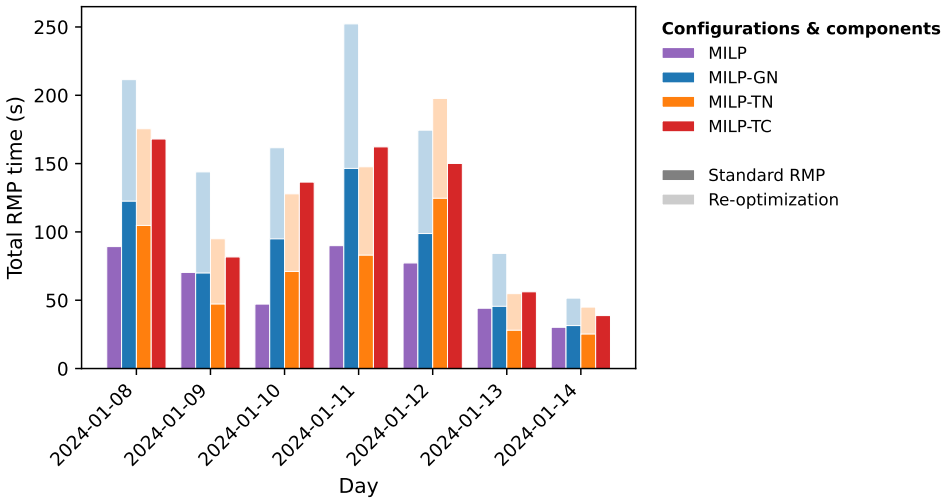
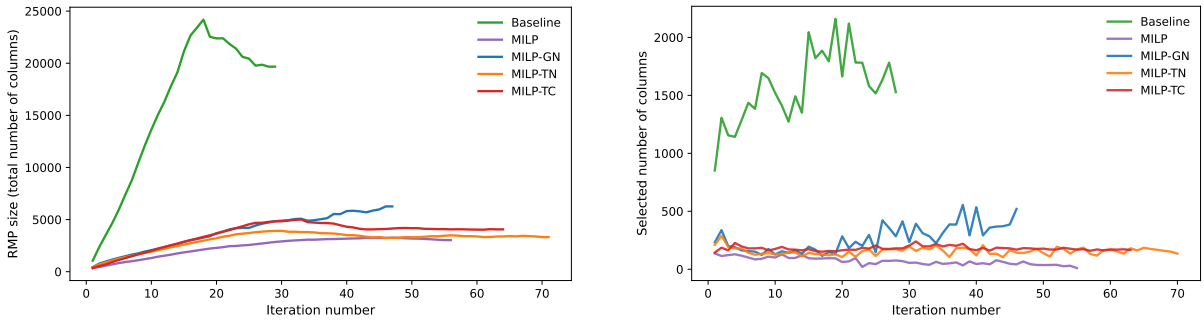



**Figure 5.4:** Total root-node RMP time for the four expert-guided configurations. The semi-transparent portion denotes the intermediate re-optimization used by MILP-GN and MILP-TN.

### 5.2.3 Effect of the task-disjoint filter

Before turning to the longer performance evaluation, the role of the task-disjoint filter is examined for the retained MILP-TC selector. As described in Section 4.3, the filter processes the columns returned by pricing in reduced-cost order. It retains the first column and removes a later candidate when its similarity with a column retained earlier from




**(a)** RMP size **(b)** Selected columns

**Figure 5.5:** Development of the restricted master problem on 12 January 2024 for the baseline and four expert-guided configurations. Figure 5.5a shows the RMP size over iterations and Figure 5.5b the number of selected columns per iteration.

the same pricing output reaches the configured threshold. The retained pricing output is then passed to the expert solve. In the comparison without the filter, the complete raw pricing output is passed to MILP-TC instead, while the expert selection rule itself remains unchanged. Table 5.6 reports the final root-node objectives for MILP-TC with and without the filter over the first evaluation week.

**Table 5.6:** Final root-node objective values for MILP-TC with and without the task-disjoint filter.

| Day | With filter objective (€) | Without filter objective (€) |
|---|---:|---:|
| 2024-01-08 | 12,389.43 | 12,478.56 |
| 2024-01-09 | 164,737.64 | 164,231.35 |
| 2024-01-10 | 10,550.65 | 10,320.24 |
| 2024-01-11 | 14,651.11 | 14,299.65 |
| 2024-01-12 | 14,624.35 | 14,395.82 |
| 2024-01-13 | 69,311.92 | 68,829.36 |
| 2024-01-14 | 6,826.92 | 6,552.83 |

Removing the filter produces a lower objective on six of the seven days. The daily differences are small relative to the objective values, with the largest absolute differences occurring on 9 and 13 January. The objective with the filter is lower only on 8 January. Across the full week, removing the filter decreases the total objective by 0.68%. Table 5.7 complements this comparison with the final RMP size and the average number of selected columns per iteration.

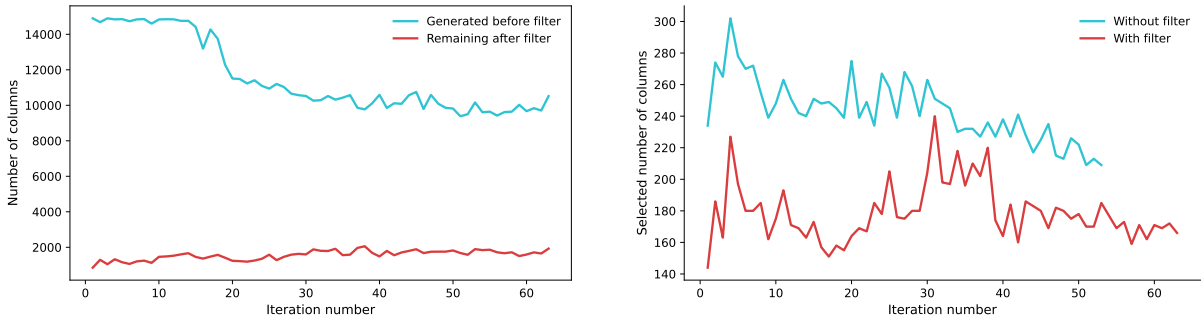
The difference in objective is accompanied by a clearer difference in RMP composition. Without the filter, the average final RMP contains 4,169 columns compared with 3,353 when the filter is applied, an increase of approximately 24%. The average number of selected columns per iteration rises from 163 to 225, an increase of approximately 38%. This pattern occurs on every day in the comparison. The expert that does not use the filter therefore selects a broader subset and ends with a larger RMP, while the corresponding objective reduction remains below 1% over the week.

Figure 5.6 shows these differences over the iterations of 12 January. Figure 5.6a reports the number of columns returned by pricing in each iteration and the number retained after

**Table 5.7:** Final RMP size and average number of selected columns per iteration for MILP-TC with and without the task-disjoint filter.

| | Final RMP | | Average selected columns | |
|---|---|---|---|---|
| Day | With filter | Without filter | With filter | Without filter |
| 2024-01-08 | 3653 | 5083 | 154.90 | 212.10 |
| 2024-01-09 | 3574 | 4333 | 161.70 | 244.00 |
| 2024-01-10 | 3086 | 3897 | 150.20 | 216.90 |
| 2024-01-11 | 3538 | 4326 | 173.20 | 250.70 |
| 2024-01-12 | 4061 | 5210 | 179.10 | 243.40 |
| 2024-01-13 | 3279 | 3910 | 173.40 | 227.60 |
| 2024-01-14 | 2281 | 2424 | 149.30 | 182.50 |

the task-disjoint rule. The rule substantially reduces this candidate pool before it is passed to the expert. Figure 5.6b then compares the number of columns selected per iteration with and without the filter. The expert that does not use the filter selects more columns, and this difference accumulates over the root-node iterations.




**(a)** Candidate columns before and after filtering

**(b)** Selected columns with and without the filter

**Figure 5.6:** Effect of the task-disjoint filter on 12 January 2024 under MILP-TC. Figure 5.6a shows the number of generated columns per iteration before the filter and the number remaining after filtering. Figure 5.6b shows the number of columns selected per iteration with and without the filter.

The task-disjoint filter is retained for the subsequent experiments. The weekly objective difference is limited, while the filter reduces both the number of columns selected by the expert and the size of the candidate set used for expert labeling and model training.

### 5.2.4 Performance of the selected MILP configuration

The preceding comparisons select MILP-TC with the task-disjoint filter as the expert-guided configuration for further evaluation. That configuration is now compared with the baseline over the two-week window from 8 to 21 January 2024, first on the three-base instance and then on the six-base instance.

For compactness, the tables use short labels for the selectors under comparison. B-S denotes the baseline selector. E-S denotes the expert-guided selector, that is, MILP-TC with the task-disjoint filter. For the full-day experiments, H-S denotes a hybrid strategy that applies E-S throughout the root node and returns to B-S after the first dive. Unless

stated otherwise, percentage changes are reported relative to B-S. The root-node analysis remains the primary evaluation. The full-day and hybrid results examine whether the column-selection decisions made at the root remain visible once diving begins.

The timing comparisons include RMP and pricing time. The time required to construct and solve the expert MILP is excluded because the expert is intended to be replaced by a fast learned prediction. Total time therefore equals RMP time plus pricing time. The reported values measure the computational effect of the selected column pools and do not represent the end-to-end runtime of the expert-guided selector.

#### 5.2.4.1 Root node

Table 5.8 compares B-S and E-S at the root node on the three-base instance. The table reports the number of iterations, the final RMP size, and the final objective value for each planning day together with the corresponding percentage differences.

**Table 5.8:** Root-node comparison of the baseline (B-S) and the expert-guided selector (E-S) on the three-base instance. The total row sums iterations and objective values and averages final RMP size. Percentage changes in the total row are recomputed from these total-row values, not averaged over days.

| | Iterations | | | Final RMP size | | | Objective value (€) | | |
|---|---|---|---|---|---|---|---|---|---|
| Day | B-S | E-S | Δ (%) | B-S | E-S | Δ (%) | B-S | E-S | Δ (%) |
| 08-01 | 32 | 72 | 125.00 | 17,021.00 | 3,653.00 | -78.54 | 11,386.20 | 12,389.43 | 8.81 |
| 09-01 | 26 | 42 | 61.54 | 14,772.00 | 3,574.00 | -75.81 | 162,878.44 | 164,737.64 | 1.14 |
| 10-01 | 43 | 73 | 69.77 | 12,258.00 | 3,086.00 | -74.82 | 9,745.42 | 10,550.65 | 8.26 |
| 11-01 | 36 | 54 | 50.00 | 14,150.00 | 3,538.00 | -75.00 | 13,167.34 | 14,651.11 | 11.27 |
| 12-01 | 29 | 64 | 120.69 | 19,671.00 | 4,061.00 | -79.36 | 12,943.64 | 14,624.35 | 12.98 |
| 13-01 | 22 | 37 | 68.18 | 16,127.00 | 3,279.00 | -79.67 | 67,630.87 | 69,311.92 | 2.49 |
| 14-01 | 23 | 39 | 69.57 | 9,694.00 | 2,281.00 | -76.47 | 6,393.47 | 6,826.92 | 6.78 |
| 15-01 | 26 | 48 | 84.62 | 13,684.00 | 3,573.00 | -73.89 | 69,132.42 | 70,846.52 | 2.48 |
| 16-01 | 24 | 39 | 62.50 | 14,267.00 | 3,334.00 | -76.63 | 114,823.88 | 116,202.39 | 1.20 |
| 17-01 | 40 | 54 | 35.00 | 14,461.00 | 3,515.00 | -75.69 | 25,307.83 | 27,439.29 | 8.42 |
| 18-01 | 25 | 41 | 64.00 | 17,634.00 | 4,069.00 | -76.93 | 183,209.00 | 185,405.79 | 1.20 |
| 19-01 | 19 | 32 | 68.42 | 16,451.00 | 3,366.00 | -79.54 | 225,030.10 | 226,533.51 | 0.67 |
| 20-01 | 18 | 28 | 55.56 | 15,877.00 | 3,152.00 | -80.15 | 224,103.60 | 225,672.12 | 0.70 |
| 21-01 | 22 | 42 | 90.91 | 10,726.00 | 2,480.00 | -76.88 | 69,760.46 | 70,142.83 | 0.55 |
| **Total** | 385 | 665 | 72.73% | 14,770.93 | 3,354.36 | -77.29% | 1,195,512.67 | 1,215,334.47 | 1.66% |

E-S produces a smaller final RMP on every planning day. The daily reduction ranges from approximately 74% to 80%, resulting in an average reduction of 77.29%. The final objective is higher under E-S on every day, although the size of the difference varies. The largest daily differences occur during the first week, while most days in the second week remain within 2.5% of B-S. Across the full period, the overall objective is 1.66% higher for E-S. The number of iterations increases on every day and is 72.73% higher in total.

Table 5.9 turns to the computational side of the comparison. Table 5.9a reports average time per root-node iteration, and Table 5.9b the corresponding totals over the full root-node procedure.

The smaller RMP is also visible in the average time per root-node iteration. E-S has a lower RMP time per iteration on every day, with an average reduction of 42.44%. Pricing

**Table 5.9:** Root-node timing for the baseline (B-S) and the expert-guided selector (E-S) on the three-base instance. All times are in seconds. Table 5.9a reports the average time per iteration, where the total row represents average values. Table 5.9b shows the summed time over the root node, where the total row represents summed values. Percentage changes in the total rows are recomputed from the total-row values, not averaged over days.

**(a)** Time per iteration (s)

| | RMP | | | Pricing | | | Total | | |
|---|---|---|---|---|---|---|---|---|---|
| Day | B-S | E-S | Δ (%) | B-S | E-S | Δ (%) | B-S | E-S | Δ (%) |
| 08-01 | 1.78 | 1.20 | -32.58 | 0.80 | 0.83 | 3.75 | 2.58 | 2.03 | -21.32 |
| 09-01 | 1.71 | 0.96 | -43.86 | 0.67 | 0.67 | 0.00 | 2.38 | 1.64 | -31.09 |
| 10-01 | 1.75 | 1.02 | -41.71 | 0.62 | 0.60 | -3.23 | 2.38 | 1.62 | -31.93 |
| 11-01 | 2.18 | 1.25 | -42.66 | 0.93 | 0.89 | -4.30 | 3.11 | 2.15 | -30.87 |
| 12-01 | 2.11 | 1.38 | -34.60 | 0.80 | 0.80 | 0.00 | 2.91 | 2.18 | -25.09 |
| 13-01 | 1.51 | 0.77 | -49.01 | 0.86 | 0.90 | 4.65 | 2.37 | 1.67 | -29.54 |
| 14-01 | 1.02 | 0.56 | -45.10 | 0.60 | 0.64 | 6.67 | 1.62 | 1.20 | -25.93 |
| 15-01 | 1.70 | 1.01 | -40.59 | 0.74 | 0.79 | 6.76 | 2.44 | 1.80 | -26.23 |
| 16-01 | 1.71 | 0.95 | -44.44 | 0.72 | 0.72 | 0.00 | 2.43 | 1.67 | -31.28 |
| 17-01 | 1.71 | 0.90 | -47.37 | 0.61 | 0.65 | 6.56 | 2.32 | 1.55 | -33.19 |
| 18-01 | 2.28 | 1.21 | -46.93 | 0.94 | 0.84 | -10.64 | 3.22 | 2.05 | -36.34 |
| 19-01 | 1.60 | 0.79 | -50.62 | 0.72 | 0.75 | 4.17 | 2.32 | 1.55 | -33.19 |
| 20-01 | 1.30 | 0.60 | -53.85 | 0.83 | 0.81 | -2.41 | 2.14 | 1.41 | -34.11 |
| 21-01 | 1.05 | 0.57 | -45.71 | 0.69 | 0.72 | 4.35 | 1.74 | 1.29 | -25.86 |
| **Total** | 1.72 | 0.99 | -42.44% | 0.75 | 0.75 | 0.00% | 2.46 | 1.74 | -29.27% |

**(b)** Total time (s)

| | RMP | | | Pricing | | | Total | | |
|---|---|---|---|---|---|---|---|---|---|
| Day | B-S | E-S | Δ (%) | B-S | E-S | Δ (%) | B-S | E-S | Δ (%) |
| 08-01 | 56.93 | 86.21 | 51.43 | 25.48 | 60.12 | 135.95 | 82.41 | 146.32 | 77.55 |
| 09-01 | 44.46 | 40.52 | -8.86 | 17.45 | 28.29 | 62.12 | 61.92 | 68.81 | 11.13 |
| 10-01 | 75.45 | 74.45 | -1.33 | 26.75 | 43.82 | 63.81 | 102.19 | 118.27 | 15.74 |
| 11-01 | 78.38 | 67.75 | -13.56 | 33.60 | 48.32 | 43.81 | 111.98 | 116.07 | 3.65 |
| 12-01 | 61.17 | 88.12 | 44.06 | 23.25 | 51.21 | 120.26 | 84.41 | 139.33 | 65.06 |
| 13-01 | 33.33 | 28.60 | -14.19 | 18.88 | 33.17 | 75.69 | 52.20 | 61.77 | 18.33 |
| 14-01 | 23.56 | 21.65 | -8.11 | 13.69 | 24.97 | 82.40 | 37.25 | 46.63 | 25.18 |
| 15-01 | 44.30 | 48.36 | 9.16 | 19.25 | 37.94 | 97.09 | 63.55 | 86.30 | 35.80 |
| 16-01 | 41.09 | 37.12 | -9.66 | 17.30 | 28.11 | 62.49 | 58.39 | 65.23 | 11.71 |
| 17-01 | 68.32 | 48.60 | -28.86 | 24.56 | 34.86 | 41.94 | 92.88 | 83.47 | -10.13 |
| 18-01 | 57.02 | 49.63 | -12.96 | 23.52 | 34.35 | 46.05 | 80.53 | 83.98 | 4.28 |
| 19-01 | 30.33 | 25.41 | -16.22 | 13.77 | 24.04 | 74.58 | 44.10 | 49.45 | 12.13 |
| 20-01 | 23.47 | 16.86 | -28.16 | 15.01 | 22.74 | 51.50 | 38.48 | 39.60 | 2.91 |
| 21-01 | 23.07 | 24.10 | 4.46 | 15.11 | 30.09 | 99.14 | 38.18 | 54.19 | 41.93 |
| **Total** | 660.88 | 657.38 | -0.53% | 287.62 | 502.03 | 74.55% | 948.47 | 1,159.42 | 22.24% |

time per iteration is essentially unchanged. Both selectors average 0.75 seconds. Combining both components, average time per iteration decreases by 29.27%.

The total-time comparison has a different outcome because E-S performs more iterations. Total RMP time varies by day and is only 0.53% lower when summed over the complete period. Total pricing time increases by 74.55%, reflecting the additional pricing calls. As a result, combined RMP and pricing time is 22.24% higher for E-S despite its lower time per iteration.

The same root-node comparison is next repeated on the six-base instance. Table 5.10 reports iterations, final RMP size, and objective value for B-S and E-S over the two-week window.

**Table 5.10:** Root-node comparison of the baseline (B-S) and the expert-guided selector (E-S) on the six-base instance. The total row sums iterations and objective values and averages final RMP size. Percentage changes in the total row are recomputed from these total-row values, not averaged over days.

| | Iterations | | | Final RMP size | | | Objective value (€) | | |
|---|---|---|---|---|---|---|---|---|---|
| Day | B-S | E-S | Δ (%) | B-S | E-S | Δ (%) | B-S | E-S | Δ (%) |
| 08-01 | 67 | 180 | 168.66 | 27,179.00 | 5,924.00 | -78.20 | 1,392.42 | 1,617.43 | 16.16 |
| 09-01 | 31 | 54 | 74.19 | 30,171.00 | 7,072.00 | -76.56 | 66,023.71 | 67,712.53 | 2.56 |
| 10-01 | 42 | 74 | 76.19 | 25,140.00 | 5,768.00 | -77.06 | 8,934.09 | 9,463.44 | 5.93 |
| 11-01 | 34 | 71 | 108.82 | 37,719.00 | 7,473.00 | -80.19 | 19,623.08 | 20,959.84 | 6.81 |
| 12-01 | 37 | 74 | 100.00 | 34,038.00 | 6,648.00 | -80.47 | 19,684.85 | 21,331.56 | 8.37 |
| 13-01 | 26 | 48 | 84.62 | 30,348.00 | 5,947.00 | -80.40 | 118,388.03 | 120,048.64 | 1.40 |
| 14-01 | 29 | 57 | 96.55 | 18,211.00 | 4,104.00 | -77.46 | 8,489.12 | 8,670.11 | 2.13 |
| 15-01 | 29 | 57 | 96.55 | 30,523.00 | 6,787.00 | -77.76 | 68,962.78 | 70,533.55 | 2.28 |
| 16-01 | 32 | 49 | 53.12 | 28,412.00 | 6,497.00 | -77.13 | 74,000.07 | 75,423.42 | 1.92 |
| 17-01 | 34 | 65 | 91.18 | 23,130.00 | 6,036.00 | -73.90 | 25,051.23 | 25,780.05 | 2.91 |
| 18-01 | 29 | 45 | 55.17 | 41,037.00 | 8,383.00 | -79.57 | 296,178.62 | 300,452.38 | 1.44 |
| 19-01 | 25 | 43 | 72.00 | 30,714.00 | 6,795.00 | -77.88 | 141,189.60 | 144,270.92 | 2.18 |
| 20-01 | 23 | 40 | 73.91 | 30,884.00 | 5,819.00 | -81.16 | 133,417.84 | 144,609.59 | 8.39 |
| 21-01 | 28 | 44 | 57.14 | 20,324.00 | 4,469.00 | -78.01 | 24,526.22 | 25,530.56 | 4.09 |
| **Total** | 466 | 901 | 93.35% | 29,130.71 | 6,265.86 | -78.49% | 1,005,861.66 | 1,036,404.02 | 3.04% |

The six-base instance shows the same ordering between the two selectors. E-S reduces the final RMP on every day, with daily reductions between approximately 74% and 81%. The average reduction of 78.49% is close to that observed for the three-base instance. Objective values remain within 10% of B-S on all but the first day and increase by 3.04% overall. The increase in iterations is larger than on the three-base instance. E-S performs 93.35% more iterations over the two-week period.

Table 5.11 reports the corresponding timing results for the root-node procedure on the six-base instance. Table 5.11a shows the average time per iteration, while Table 5.11b reports the total times over the complete procedure.

Average RMP time per iteration decreases by 38.39% on the larger instance. Pricing time per iteration is 11.50% lower, reversing the trend observed on the three-base instance. The combined reduction in time per iteration is 31.46%, consistent with the observations for the three-base instance. This indicates that the smaller expert-guided RMP also reduces the computational effort of an individual iteration at the larger scale.

**Table 5.11:** Root-node timing for the baseline (B-S) and the expert-guided selector (E-S) on the six-base instance. All times are in seconds. Table 5.11a reports the average time per iteration, where the total row represents average values. Table 5.11b shows the summed time over the root node, where the total row represents summed values. Percentage changes in the total rows are recomputed from the total-row values, not averaged over days.

**(a)** Time per iteration (s)

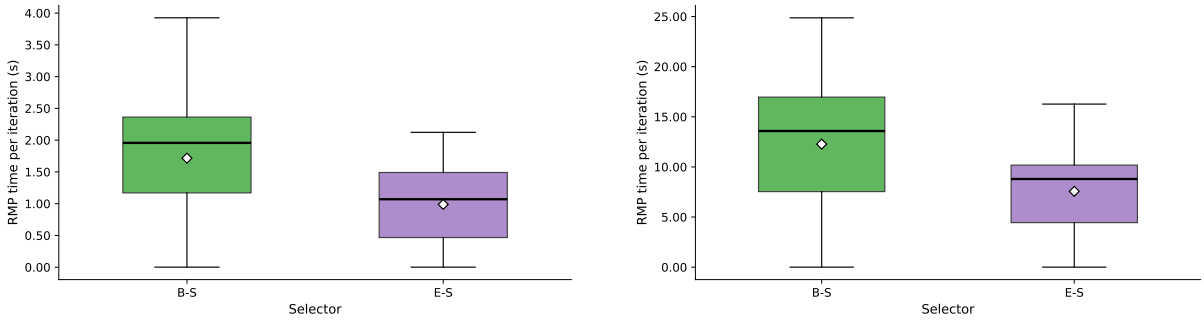
| | RMP | | | Pricing | | | Total | | |
|---|---|---|---|---|---|---|---|---|---|
| Day | B-S | E-S | Δ (%) | B-S | E-S | Δ (%) | B-S | E-S | Δ (%) |
| 08-01 | 15.00 | 8.91 | -40.60 | 3.99 | 4.08 | 2.26 | 18.99 | 13.00 | -31.54 |
| 09-01 | 12.56 | 7.80 | -37.90 | 4.65 | 4.39 | -5.59 | 17.22 | 12.19 | -29.21 |
| 10-01 | 12.11 | 7.26 | -40.05 | 4.20 | 3.28 | -21.90 | 16.31 | 10.55 | -35.32 |
| 11-01 | 16.36 | 10.98 | -32.89 | 5.69 | 4.56 | -19.86 | 22.05 | 15.54 | -29.52 |
| 12-01 | 14.03 | 9.65 | -31.22 | 4.94 | 4.43 | -10.32 | 18.97 | 14.08 | -25.78 |
| 13-01 | 8.77 | 4.89 | -44.24 | 3.96 | 3.38 | -14.65 | 12.73 | 8.27 | -35.04 |
| 14-01 | 6.05 | 3.59 | -40.66 | 2.50 | 2.38 | -4.80 | 8.55 | 5.97 | -30.18 |
| 15-01 | 13.75 | 8.20 | -40.36 | 4.20 | 3.51 | -16.43 | 17.95 | 11.71 | -34.76 |
| 16-01 | 13.50 | 7.71 | -42.89 | 4.80 | 4.15 | -13.54 | 18.30 | 11.85 | -35.25 |
| 17-01 | 10.67 | 6.45 | -39.55 | 4.24 | 3.56 | -16.04 | 14.91 | 10.02 | -32.80 |
| 18-01 | 16.68 | 9.76 | -41.49 | 6.06 | 4.92 | -18.81 | 22.74 | 14.68 | -35.44 |
| 19-01 | 12.16 | 6.88 | -43.42 | 3.95 | 3.39 | -14.18 | 16.11 | 10.27 | -36.25 |
| 20-01 | 8.59 | 4.40 | -48.78 | 3.77 | 3.15 | -16.45 | 12.36 | 7.55 | -38.92 |
| 21-01 | 5.66 | 3.21 | -43.29 | 2.34 | 2.40 | 2.56 | 8.01 | 5.61 | -29.96 |
| **Total** | 12.27 | 7.56 | -38.39% | 4.26 | 3.77 | -11.50% | 16.53 | 11.33 | -31.46% |

**(b)** Total time (s)

| | RMP | | | Pricing | | | Total | | |
|---|---|---|---|---|---|---|---|---|---|
| Day | B-S | E-S | Δ (%) | B-S | E-S | Δ (%) | B-S | E-S | Δ (%) |
| 08-01 | 1,005.10 | 1,604.42 | 59.63 | 267.02 | 735.19 | 175.33 | 1,272.12 | 2,339.61 | 83.91 |
| 09-01 | 389.46 | 421.34 | 8.19 | 144.22 | 236.85 | 64.23 | 533.68 | 658.18 | 23.33 |
| 10-01 | 508.47 | 537.58 | 5.73 | 176.37 | 243.01 | 37.78 | 684.84 | 780.58 | 13.98 |
| 11-01 | 556.35 | 779.71 | 40.15 | 193.49 | 323.80 | 67.35 | 749.85 | 1,103.51 | 47.16 |
| 12-01 | 519.26 | 714.34 | 37.57 | 182.64 | 327.76 | 79.46 | 701.90 | 1,042.10 | 48.47 |
| 13-01 | 228.13 | 234.91 | 2.97 | 102.86 | 162.25 | 57.74 | 330.99 | 397.16 | 19.99 |
| 14-01 | 175.48 | 204.78 | 16.70 | 72.44 | 135.59 | 87.18 | 247.91 | 340.37 | 37.30 |
| 15-01 | 398.88 | 467.33 | 17.16 | 121.78 | 200.09 | 64.30 | 520.65 | 667.42 | 28.19 |
| 16-01 | 432.02 | 377.59 | -12.60 | 153.46 | 203.12 | 32.36 | 585.48 | 580.71 | -0.81 |
| 17-01 | 362.76 | 419.54 | 15.65 | 144.20 | 231.62 | 60.62 | 506.96 | 651.16 | 28.44 |
| 18-01 | 483.63 | 438.99 | -9.23 | 175.72 | 221.60 | 26.11 | 659.35 | 660.59 | 0.19 |
| 19-01 | 303.99 | 295.70 | -2.73 | 98.77 | 145.94 | 47.76 | 402.76 | 441.65 | 9.66 |
| 20-01 | 197.52 | 175.90 | -10.95 | 86.80 | 125.99 | 45.15 | 284.32 | 301.89 | 6.18 |
| 21-01 | 158.56 | 141.26 | -10.91 | 65.60 | 105.44 | 60.73 | 224.15 | 246.70 | 10.06 |
| **Total** | 5,719.61 | 6,813.39 | 19.12% | 1,985.37 | 3,398.25 | 71.16% | 7,704.96 | 10,211.63 | 32.53% |

As on the three-base instance, this per-iteration reduction does not translate into a lower total time. The larger number of iterations increases total RMP time by 19.12% and total pricing time by 71.16%. Consequently, the combined time increases by 32.53%. The larger instance therefore retains the same main pattern of smaller RMPs and faster individual iterations alongside a longer root-node trajectory.

To examine whether the average timing results reflect the underlying iterations, Figure 5.7 pools the RMP time of every root-node iteration over the two-week period. E-S is shifted downward relative to B-S on both instances. Whether the selectors are compared by their means or by their medians, E-S remains lower. The means lie below the medians for both selectors, indicating some lower-tail asymmetry. This is consistent with the presence of early iterations in which the RMP is still small. That asymmetry does not change the conclusion that E-S is faster per iteration. E-S also has a narrower interquartile range, which is consistent with keeping the RMP smaller throughout the root-node procedure.




**(a)** Three-base instance **(b)** Six-base instance

**Figure 5.7:** Boxplots of RMP time per iteration at the root node for B-S and E-S over the two-week evaluation period. Figure 5.7a shows the three-base instance and Figure 5.7b the six-base instance. Each box spans the interquartile range, the black horizontal line marks the median, and the white diamond marks the mean.

#### 5.2.4.2 Full planning day

The root-node results isolate the effect of column selection before diving begins. The comparison is next extended to the full planning day and includes the hybrid strategy H-S. Table 5.12 reports the number of iterations and the final RMP size for B-S, E-S, and H-S on the three-base instance.

E-S maintains a much smaller RMP throughout the full planning day. Its average final RMP is 79.01% smaller than for B-S, and a reduction of at least 71% is observed on every day. This is accompanied by 61.62% more iterations over the evaluation period.

H-S produces a different balance. Since the procedure returns to B-S after the root node, the RMP grows more quickly during the later column generation segments. Its average final RMP is therefore 19.06% smaller than for B-S, which is a more moderate reduction than under E-S. The total number of iterations increases by 32.40%, also substantially less than under E-S.

Table 5.13 reports the corresponding solution-quality measures. The full-day evaluation also permits comparison of uncovered tasks, where a task is counted as uncovered

**Table 5.12:** Full-day iterations and final RMP size for the baseline (B-S), the expert-guided selector (E-S), and the hybrid strategy (H-S) on the three-base instance. The total row sums iterations and averages final RMP size. Percentage changes in the total row are recomputed from these total-row values, not averaged over days.

| | Iterations | | | | | Final RMP size | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Day | B-S | E-S | H-S | $\Delta_E$ (%) | $\Delta_H$ (%) | B-S | E-S | H-S | $\Delta_E$ (%) | $\Delta_H$ (%) |
| 08-01 | 95 | 181 | 135 | 90.53 | 42.11 | 29,134.00 | 7,074.00 | 20,686.00 | -75.72 | -29.00 |
| 09-01 | 63 | 116 | 96 | 84.13 | 52.38 | 29,210.00 | 6,334.00 | 24,550.00 | -78.32 | -15.95 |
| 10-01 | 106 | 155 | 158 | 46.23 | 49.06 | 24,116.00 | 5,657.00 | 21,901.00 | -76.54 | -9.18 |
| 11-01 | 94 | 173 | 112 | 84.04 | 19.15 | 26,626.00 | 7,469.00 | 22,378.00 | -71.95 | -15.95 |
| 12-01 | 96 | 141 | 129 | 46.88 | 34.38 | 37,879.00 | 6,404.00 | 26,184.00 | -83.09 | -30.87 |
| 13-01 | 72 | 92 | 83 | 27.78 | 15.28 | 35,303.00 | 5,727.00 | 25,944.00 | -83.78 | -26.51 |
| 14-01 | 66 | 98 | 86 | 48.48 | 30.30 | 17,855.00 | 3,241.00 | 14,908.00 | -81.85 | -16.51 |
| 15-01 | 76 | 127 | 106 | 67.11 | 39.47 | 25,371.00 | 5,650.00 | 23,849.00 | -77.73 | -6.00 |
| 16-01 | 77 | 111 | 94 | 44.16 | 22.08 | 28,778.00 | 6,078.00 | 23,670.00 | -78.88 | -17.75 |
| 17-01 | 97 | 145 | 122 | 49.48 | 25.77 | 26,028.00 | 5,862.00 | 22,659.00 | -77.48 | -12.94 |
| 18-01 | 75 | 131 | 79 | 74.67 | 5.33 | 32,529.00 | 7,656.00 | 28,077.00 | -76.46 | -13.69 |
| 19-01 | 50 | 80 | 67 | 60.00 | 34.00 | 29,769.00 | 5,892.00 | 22,753.00 | -80.21 | -23.57 |
| 20-01 | 42 | 74 | 61 | 76.19 | 45.24 | 29,658.00 | 4,791.00 | 21,105.00 | -83.85 | -28.84 |
| 21-01 | 62 | 107 | 90 | 72.58 | 45.16 | 16,813.00 | 3,825.00 | 16,230.00 | -77.25 | -3.47 |
| **Total** | 1071 | 1731 | 1418 | 61.62% | 32.40% | 27,790.64 | 5,832.86 | 22,492.43 | -79.01% | -19.06% |

when no assigned duty covers it in the final integer solution. Besides that number, the table includes the adjusted objective, obtained by removing the large slack cost of uncovered tasks from the solver objective. These measures are reported separately because full-day trajectories are path-dependent and can finish with different numbers of uncovered tasks. As the table shows, the uncovered-task counts differ between selectors on almost every day. Including their large slack costs in the objective comparison would therefore allow day-to-day differences in task coverage to dominate the comparison of assignment penalties. The adjusted objective instead makes it possible to examine the fairness-related assignment costs independently, while the uncovered-task count retains the coverage information.

These day-to-day differences accumulate over the two weeks. E-S leaves 44 tasks uncovered compared with 33 for B-S. Its adjusted objective is 20.39% higher overall and exceeds the B-S value on every day, indicating a higher assignment-penalty cost under the fairness feedback. H-S leaves 29 tasks uncovered, four fewer than B-S, while its adjusted objective is 9.21% higher. The daily H-S adjusted objective can be either above or below B-S, but its overall difference is less than half that of E-S. The hybrid results are therefore closer to the baseline solution quality than the results obtained when the expert-guided selector remains active throughout the day.

Table 5.14 reports the corresponding timing results for the full-day procedure on the three-base instance. Table 5.14a shows the average time per iteration, while Table 5.14b reports the total times over the complete procedure.

Both expert-guided strategies reduce the computational time of an individual iteration. Average combined time per iteration decreases by 50.21% for E-S and by 24.90% for H-S. Most of this reduction comes from the RMP component, which decreases by 61.81% for E-S and 33.67% for H-S. Pricing time per iteration increases slightly for both strategies.

**Table 5.13:** Full-day solution quality for the baseline (B-S), the expert-guided selector (E-S), and the hybrid strategy (H-S) on the three-base instance. The adjusted objective removes the slack cost of €50,000 per uncovered task from the solver objective, leaving the assignment penalties from the fairness feedback. The total row sums all values across planning days. Percentage changes in the total row are recomputed from these total-row values, not averaged over days.

| | Uncovered tasks | | | Adjusted objective value (€) | | | | |
|---|---|---|---|---|---|---|---|---|
| Day | B-S | E-S | H-S | B-S | E-S | H-S | $\Delta_E$ (%) | $\Delta_H$ (%) |
| 08-01 | 2 | 2 | 0 | 11,937.17 | 26,647.09 | 12,243.03 | 123.23 | 2.56 |
| 09-01 | 3 | 4 | 5 | 13,211.12 | 15,461.59 | 16,363.39 | 17.03 | 23.86 |
| 10-01 | 1 | 1 | 0 | 11,177.89 | 18,757.05 | 10,658.05 | 67.80 | -4.65 |
| 11-01 | 0 | 2 | 1 | 14,116.42 | 16,121.62 | 14,579.48 | 14.20 | 3.28 |
| 12-01 | 3 | 3 | 0 | 13,260.16 | 16,366.51 | 15,137.92 | 23.43 | 14.16 |
| 13-01 | 4 | 6 | 5 | 17,696.77 | 22,484.97 | 18,918.89 | 27.06 | 6.91 |
| 14-01 | 0 | 0 | 0 | 6,752.95 | 8,867.02 | 10,511.95 | 31.31 | 55.66 |
| 15-01 | 1 | 2 | 1 | 20,090.76 | 21,356.40 | 21,861.57 | 6.30 | 8.81 |
| 16-01 | 4 | 3 | 2 | 15,731.95 | 18,232.19 | 16,193.58 | 15.89 | 2.93 |
| 17-01 | 1 | 1 | 2 | 25,904.72 | 28,259.01 | 34,112.99 | 9.09 | 31.69 |
| 18-01 | 3 | 6 | 3 | 34,360.67 | 38,754.84 | 35,191.04 | 12.79 | 2.42 |
| 19-01 | 5 | 5 | 5 | 25,307.27 | 26,719.22 | 26,701.16 | 5.58 | 5.51 |
| 20-01 | 5 | 4 | 4 | 24,943.07 | 27,136.16 | 25,393.39 | 8.79 | 1.81 |
| 21-01 | 1 | 5 | 1 | 20,379.82 | 21,670.07 | 20,472.27 | 6.33 | 0.45 |
| **Total** | 33 | 44 | 29 | 254,870.74 | 306,833.74 | 278,338.71 | 20.39% | 9.21% |

Over the complete evaluation period, E-S reduces total RMP time by 38.27%, while its additional iterations increase total pricing time by 72.39%. The net result is a 19.35% reduction in combined time. H-S reduces total RMP time by 12.60% and increases pricing time by 57.96%, leaving its combined time 0.54% below B-S.

**Table 5.14:** Full-day timing for the baseline (B-S), the expert-guided selector (E-S), and the hybrid strategy (H-S) on the three-base instance. All times are in seconds. Table 5.14a reports the average time per iteration, where the total row represents average values. Table 5.14b shows the summed time over the full planning day, where the total row represents summed values. Percentage changes in the total rows are recomputed from the total-row values, not averaged over days.

**(a)** Time per iteration (s)

| | RMP | | | | Pricing | | | | Total | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Day | B-S | E-S | H-S | $\Delta_E$ | $\Delta_H$ | B-S | E-S | H-S | $\Delta_E$ | $\Delta_H$ | B-S | E-S | H-S | $\Delta_E$ | $\Delta_H$ |
| 08-01 | 2.19 | 0.93 | 1.50 | -57.53 | -31.51 | 0.45 | 0.52 | 0.59 | 15.56 | 31.11 | 2.64 | 1.45 | 2.09 | -45.08 | -20.83 |
| 09-01 | 2.34 | 0.70 | 1.34 | -70.09 | -42.74 | 0.39 | 0.38 | 0.40 | -2.56 | 2.56 | 2.73 | 1.08 | 1.74 | -60.44 | -36.26 |
| 10-01 | 1.78 | 0.88 | 1.12 | -50.56 | -37.08 | 0.38 | 0.41 | 0.38 | 7.89 | 0.00 | 2.16 | 1.28 | 1.50 | -40.74 | -30.56 |
| 11-01 | 2.28 | 0.93 | 1.52 | -59.21 | -33.33 | 0.55 | 0.50 | 0.58 | -9.09 | 5.45 | 2.83 | 1.43 | 2.10 | -49.47 | -25.80 |
| 12-01 | 2.58 | 1.06 | 1.81 | -58.91 | -29.84 | 0.38 | 0.51 | 0.55 | 34.21 | 44.74 | 2.96 | 1.57 | 2.35 | -46.96 | -20.61 |
| 13-01 | 1.90 | 0.61 | 1.33 | -67.89 | -30.00 | 0.45 | 0.58 | 0.64 | 28.89 | 42.22 | 2.34 | 1.19 | 1.96 | -49.15 | -16.24 |
| 14-01 | 1.02 | 0.39 | 0.76 | -61.76 | -25.49 | 0.37 | 0.32 | 0.39 | -13.51 | 5.41 | 1.39 | 0.71 | 1.16 | -48.92 | -16.55 |
| 15-01 | 2.10 | 0.72 | 1.50 | -65.71 | -28.57 | 0.38 | 0.43 | 0.51 | 13.16 | 34.21 | 2.48 | 1.15 | 2.02 | -53.63 | -18.55 |
| 16-01 | 2.25 | 0.85 | 1.55 | -62.22 | -31.11 | 0.33 | 0.38 | 0.43 | 15.15 | 30.30 | 2.58 | 1.23 | 1.99 | -52.33 | -22.87 |
| 17-01 | 1.55 | 0.71 | 1.09 | -54.19 | -29.68 | 0.38 | 0.35 | 0.40 | -7.89 | 5.26 | 1.93 | 1.06 | 1.49 | -45.08 | -22.80 |
| 18-01 | 2.71 | 0.92 | 1.89 | -66.05 | -30.26 | 0.48 | 0.43 | 0.60 | -10.42 | 25.00 | 3.19 | 1.34 | 2.49 | -57.99 | -21.94 |
| 19-01 | 2.23 | 0.60 | 1.21 | -73.09 | -45.74 | 0.36 | 0.40 | 0.47 | 11.11 | 30.56 | 2.60 | 1.00 | 1.68 | -61.54 | -35.38 |
| 20-01 | 1.47 | 0.39 | 0.81 | -73.47 | -44.90 | 0.50 | 0.48 | 0.51 | -4.00 | 2.00 | 1.97 | 0.87 | 1.32 | -55.84 | -32.99 |
| 21-01 | 1.05 | 0.40 | 0.67 | -61.90 | -36.19 | 0.35 | 0.41 | 0.46 | 17.14 | 31.43 | 1.40 | 0.81 | 1.14 | -42.14 | -18.57 |
| **Total** | 1.99 | 0.76 | 1.32 | -61.81% | -33.67% | 0.41 | 0.44 | 0.49 | 7.32% | 19.51% | 2.41 | 1.20 | 1.81 | -50.21% | -24.90% |

**(b)** Total time (s)

| | RMP | | | | | Pricing | | | | | Total | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Day | B-S | E-S | H-S | $\Delta_E$ | $\Delta_H$ | B-S | E-S | H-S | $\Delta_E$ | $\Delta_H$ | B-S | E-S | H-S | $\Delta_E$ | $\Delta_H$ |
| 08-01 | 208.37 | 167.45 | 202.65 | -19.64 | -2.75 | 42.43 | 94.30 | 79.29 | 122.25 | 86.87 | 250.80 | 261.75 | 281.94 | 4.37 | 12.42 |
| 09-01 | 147.44 | 81.42 | 128.59 | -44.78 | -12.78 | 24.53 | 43.66 | 38.67 | 77.99 | 57.64 | 171.96 | 125.08 | 167.26 | -27.26 | -2.73 |
| 10-01 | 188.97 | 135.77 | 176.43 | -28.15 | -6.64 | 40.06 | 63.11 | 59.82 | 57.54 | 49.33 | 229.03 | 198.88 | 236.25 | -13.16 | 3.15 |
| 11-01 | 213.91 | 161.41 | 170.19 | -24.54 | -20.44 | 52.15 | 86.48 | 65.50 | 65.83 | 25.60 | 266.05 | 247.89 | 235.69 | -6.83 | -11.41 |
| 12-01 | 247.51 | 149.43 | 233.01 | -39.63 | -5.86 | 36.95 | 71.33 | 70.69 | 93.04 | 91.31 | 284.46 | 220.76 | 303.70 | -22.39 | 6.76 |
| 13-01 | 136.73 | 56.02 | 110.19 | -59.03 | -19.41 | 32.07 | 53.43 | 52.76 | 66.60 | 64.52 | 168.80 | 109.44 | 162.95 | -35.17 | -3.47 |
| 14-01 | 67.40 | 38.15 | 65.75 | -43.40 | -2.45 | 24.56 | 31.77 | 33.84 | 29.36 | 37.79 | 91.97 | 69.92 | 99.60 | -23.98 | 8.30 |
| 15-01 | 159.55 | 91.06 | 159.29 | -42.93 | -0.16 | 29.16 | 54.56 | 54.40 | 87.11 | 86.56 | 188.71 | 145.62 | 213.69 | -22.83 | 13.24 |
| 16-01 | 173.32 | 94.37 | 146.08 | -45.55 | -15.72 | 25.10 | 42.40 | 40.76 | 68.92 | 62.39 | 198.42 | 136.76 | 186.84 | -31.08 | -5.84 |
| 17-01 | 150.49 | 102.87 | 133.29 | -31.64 | -11.43 | 37.04 | 51.20 | 48.36 | 38.23 | 30.56 | 187.54 | 154.07 | 181.65 | -17.85 | -3.14 |
| 18-01 | 203.09 | 120.17 | 149.30 | -40.83 | -26.49 | 35.78 | 55.89 | 47.49 | 56.20 | 32.73 | 238.88 | 176.06 | 196.80 | -26.30 | -17.62 |
| 19-01 | 111.72 | 48.02 | 81.33 | -57.02 | -27.20 | 18.12 | 32.29 | 31.43 | 78.20 | 73.45 | 129.84 | 80.30 | 112.76 | -38.15 | -13.15 |
| 20-01 | 61.89 | 29.13 | 49.66 | -52.93 | -19.76 | 20.98 | 35.27 | 31.07 | 68.11 | 48.09 | 82.87 | 64.40 | 80.74 | -22.29 | -2.57 |
| 21-01 | 65.14 | 42.94 | 60.70 | -34.08 | -6.82 | 21.43 | 43.46 | 41.53 | 102.80 | 93.79 | 86.57 | 86.41 | 102.22 | -0.18 | 18.08 |
| **Total** | 2,135.53 | 1,318.21 | 1,866.46 | -38.27% | -12.60% | 440.36 | 759.15 | 695.61 | 72.39% | 57.96% | 2,575.90 | 2,077.34 | 2,562.09 | -19.35% | -0.54% |

The full-day comparison is repeated on the six-base instance. Table 5.15 again reports iterations and final RMP size for B-S, E-S, and H-S.

**Table 5.15:** Full-day iterations and final RMP size for the baseline (B-S), the expert-guided selector (E-S), and the hybrid strategy (H-S) on the six-base instance. The total row sums iterations and averages final RMP size. Percentage changes in the total row are recomputed from these total-row values, not averaged over days.

| | Iterations | | | | | Final RMP size | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Day | B-S | E-S | H-S | $\Delta_E$ (%) | $\Delta_H$ (%) | B-S | E-S | H-S | $\Delta_E$ (%) | $\Delta_H$ (%) |
| 08-01 | 168 | 403 | 353 | 139.88 | 110.12 | 72,455.00 | 16,879.00 | 61,322.00 | -76.70 | -15.37 |
| 09-01 | 139 | 214 | 159 | 53.96 | 14.39 | 71,585.00 | 15,829.00 | 58,110.00 | -77.89 | -18.82 |
| 10-01 | 157 | 259 | 191 | 64.97 | 21.66 | 55,312.00 | 13,624.00 | 46,444.00 | -75.37 | -16.03 |
| 11-01 | 115 | 255 | 192 | 121.74 | 66.96 | 75,507.00 | 16,268.00 | 63,605.00 | -78.45 | -15.76 |
| 12-01 | 120 | 222 | 192 | 85.00 | 60.00 | 67,488.00 | 14,289.00 | 72,286.00 | -78.83 | 7.11 |
| 13-01 | 95 | 168 | 113 | 76.84 | 18.95 | 69,626.00 | 11,610.00 | 59,876.00 | -83.33 | -14.00 |
| 14-01 | 126 | 224 | 148 | 77.78 | 17.46 | 38,276.00 | 7,592.00 | 31,472.00 | -80.17 | -17.78 |
| 15-01 | 118 | 225 | 168 | 90.68 | 42.37 | 65,531.00 | 15,512.00 | 54,048.00 | -76.33 | -17.52 |
| 16-01 | 129 | 170 | 136 | 31.78 | 5.43 | 61,758.00 | 13,591.00 | 58,128.00 | -77.99 | -5.88 |
| 17-01 | 125 | 236 | 151 | 88.80 | 20.80 | 44,864.00 | 13,408.00 | 45,962.00 | -70.11 | 2.45 |
| 18-01 | 93 | 199 | 117 | 113.98 | 25.81 | 89,930.00 | 17,117.00 | 63,026.00 | -80.97 | -29.92 |
| 19-01 | 88 | 141 | 118 | 60.23 | 34.09 | 64,802.00 | 12,849.00 | 53,415.00 | -80.17 | -17.57 |
| 20-01 | 82 | 119 | 101 | 45.12 | 23.17 | 60,163.00 | 10,122.00 | 53,266.00 | -83.18 | -11.46 |
| 21-01 | 98 | 211 | 119 | 115.31 | 21.43 | 35,157.00 | 9,422.00 | 31,565.00 | -73.20 | -10.22 |
| **Total** | 1653 | 3046 | 2258 | 84.27% | 36.60% | 62,318.14 | 13,436.57 | 53,751.79 | -78.44% | -13.75% |

Table 5.16 reports uncovered tasks and the adjusted objective for the same three strategies. As on the three-base instance, the uncovered-task counts differ between selectors on almost every day. These day-to-day differences accumulate over the two weeks. E-S leaves 57 tasks uncovered compared with 43 for B-S. Its adjusted objective is higher on every day and increases by 19.35% overall. H-S leaves 44 tasks uncovered, only one more than B-S, and increases the adjusted objective by 10.19%. Its daily adjusted objective is below B-S on four days and above it on the remaining ten. As on the smaller instance, the hybrid strategy remains closer to the baseline solution quality than the strategy that applies expert-guided selection throughout the full day.

Table 5.17 reports the corresponding timing results for the full-day procedure on the six-base instance. Table 5.17a shows average time per iteration, while Table 5.17b reports the total times over the complete procedure.

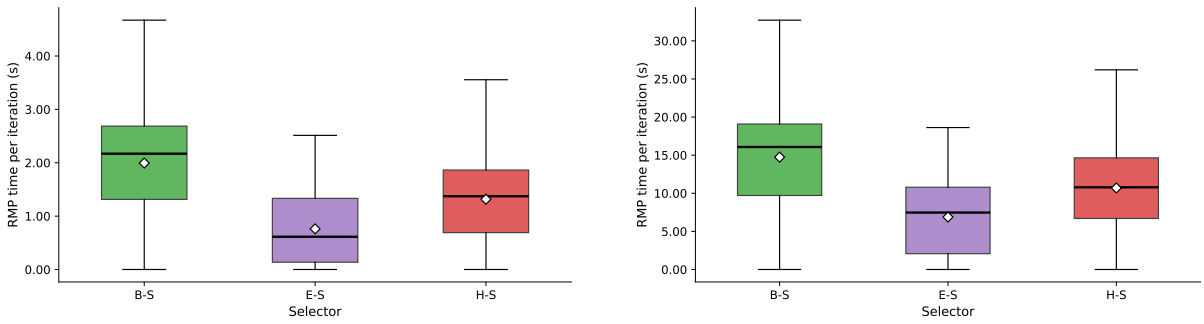
Average combined time per iteration decreases by 46.46% for E-S and by 23.05% for H-S. The RMP component again accounts for most of the reduction. Average pricing time per iteration remains close to B-S for E-S and is 7.66% higher for H-S.

Across the two weeks, E-S reduces total RMP time by 13.91%, while total pricing time increases by 87.02%. These changes result in a 1.38% reduction in combined time. H-S reduces total RMP time by less than 1% and increases pricing time by 47.25%, producing a 5.12% increase in combined time. The larger instance therefore shows the same reductions in RMP size and per-iteration time, while the full-day totals remain close to the baseline.

**Table 5.16:** Full-day solution quality for the baseline (B-S), the expert-guided selector (E-S), and the hybrid strategy (H-S) on the six-base instance. The adjusted objective removes the slack cost of €50,000 per uncovered task from the solver objective, leaving the assignment penalties from the fairness feedback. The total row sums all values across planning days. Percentage changes in the total row are recomputed from these total-row values, not averaged over days.

| | Uncovered tasks | | | Adjusted objective value (€) | | | | |
|---|---|---|---|---|---|---|---|---|
| Day | B-S | E-S | H-S | B-S | E-S | H-S | $\Delta_E$ (%) | $\Delta_H$ (%) |
| 08-01 | 7 | 2 | 1 | 5,298.33 | 8,431.43 | 3,915.63 | 59.13 | -26.10 |
| 09-01 | 4 | 4 | 1 | 17,674.11 | 21,998.49 | 29,756.05 | 24.47 | 68.36 |
| 10-01 | 1 | 6 | 1 | 21,893.90 | 26,423.48 | 10,980.24 | 20.69 | -49.85 |
| 11-01 | 1 | 3 | 3 | 20,975.79 | 29,559.25 | 21,419.78 | 40.92 | 2.12 |
| 12-01 | 3 | 1 | 3 | 20,940.91 | 27,344.62 | 25,358.05 | 30.58 | 21.09 |
| 13-01 | 7 | 4 | 3 | 20,067.62 | 28,518.71 | 20,400.18 | 42.11 | 1.66 |
| 14-01 | 3 | 1 | 1 | 9,546.97 | 12,113.70 | 10,522.11 | 26.89 | 10.21 |
| 15-01 | 3 | 8 | 4 | 21,508.36 | 27,089.85 | 22,065.17 | 25.95 | 2.59 |
| 16-01 | 3 | 4 | 6 | 25,666.23 | 28,615.09 | 29,580.68 | 11.49 | 15.25 |
| 17-01 | 1 | 2 | 3 | 31,457.28 | 32,322.59 | 28,596.85 | 2.75 | -9.09 |
| 18-01 | 5 | 10 | 7 | 52,132.65 | 52,984.73 | 50,251.68 | 1.63 | -3.61 |
| 19-01 | 3 | 8 | 4 | 43,582.25 | 51,013.53 | 44,147.38 | 17.05 | 1.30 |
| 20-01 | 2 | 4 | 6 | 37,726.29 | 48,087.75 | 65,673.79 | 27.46 | 74.08 |
| 21-01 | 0 | 0 | 1 | 26,761.43 | 29,452.06 | 28,755.45 | 10.05 | 7.45 |
| **Total** | 43 | 57 | 44 | 355,232.12 | 423,955.28 | 391,423.04 | 19.35% | 10.19% |

Figure 5.8 gives the corresponding per-iteration distributions for the full planning day. As at the root node, the mean and median again agree closely. On both instances, E-S lies below H-S, and H-S lies below B-S, under both measures. The E-S and H-S distributions also have generally narrower interquartile ranges than B-S. This supports the interpretation from the timing tables. Limiting RMP growth lowers the computational effort of an iteration, with the strongest effect under E-S.




**(a)** Three-base instance

**(b)** Six-base instance

**Figure 5.8:** Boxplots of RMP time per iteration over the full planning day for B-S, E-S, and H-S over the two-week evaluation period. Figure 5.8a shows the three-base instance and Figure 5.8b the six-base instance. Each box spans the interquartile range, the black horizontal line marks the median, and the white diamond marks the mean.

**Table 5.17:** Full-day timing for the baseline (B-S), the expert-guided selector (E-S), and the hybrid strategy (H-S) on the six-base instance. All times are in seconds. Table 5.17a reports the average time per iteration, where total row represents average values. Table 5.17b shows the summed time over the full planning day, where the total row represents summed values. Percentage changes in the total rows are recomputed from the total-row values, not averaged over days.

**(a)** Time per iteration (s)

| | RMP | | | | Pricing | | | | Total | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Day | B-S | E-S | H-S | $\Delta_E$ | $\Delta_H$ | B-S | E-S | H-S | $\Delta_E$ | $\Delta_H$ | B-S | E-S | H-S | $\Delta_E$ | $\Delta_H$ |
| 08-01 | 17.34 | 8.66 | 11.22 | -50.06 | -35.29 | 2.33 | 3.06 | 2.98 | 31.33 | 27.90 | 19.67 | 11.72 | 14.21 | -40.42 | -27.76 |
| 09-01 | 16.83 | 6.44 | 11.60 | -61.73 | -31.08 | 1.81 | 2.42 | 2.33 | 33.70 | 28.73 | 18.64 | 8.86 | 13.93 | -52.47 | -25.27 |
| 10-01 | 14.38 | 6.46 | 10.31 | -55.08 | -28.30 | 2.30 | 1.69 | 1.93 | -26.52 | -16.09 | 16.68 | 8.15 | 12.24 | -51.14 | -26.62 |
| 11-01 | 20.91 | 12.10 | 15.47 | -42.13 | -26.02 | 3.10 | 2.51 | 2.62 | -19.03 | -15.48 | 24.01 | 14.62 | 18.09 | -39.11 | -24.66 |
| 12-01 | 16.67 | 8.08 | 12.81 | -51.53 | -23.16 | 2.75 | 2.52 | 2.53 | -8.36 | -8.00 | 19.41 | 10.60 | 15.34 | -45.39 | -20.97 |
| 13-01 | 12.34 | 5.32 | 7.29 | -56.89 | -40.92 | 1.91 | 2.12 | 2.15 | 10.99 | 12.57 | 14.24 | 7.44 | 9.44 | -47.75 | -33.71 |
| 14-01 | 6.90 | 2.09 | 5.07 | -69.71 | -26.52 | 1.07 | 1.08 | 1.53 | 0.93 | 42.99 | 7.96 | 3.17 | 6.60 | -60.18 | -17.09 |
| 15-01 | 16.71 | 7.84 | 12.14 | -53.08 | -27.35 | 1.95 | 2.10 | 1.85 | 7.69 | -5.13 | 18.67 | 9.94 | 13.99 | -46.76 | -25.07 |
| 16-01 | 16.22 | 6.89 | 13.25 | -57.52 | -18.31 | 2.07 | 2.09 | 2.27 | 0.97 | 9.66 | 18.29 | 8.98 | 15.53 | -50.90 | -15.09 |
| 17-01 | 11.62 | 6.23 | 9.07 | -46.39 | -21.94 | 1.97 | 1.97 | 2.22 | 0.00 | 12.69 | 13.59 | 8.20 | 11.29 | -39.66 | -16.92 |
| 18-01 | 22.32 | 10.65 | 15.14 | -52.28 | -32.17 | 3.09 | 2.50 | 2.72 | -19.09 | -11.97 | 25.41 | 13.15 | 17.86 | -48.25 | -29.71 |
| 19-01 | 15.38 | 5.77 | 9.44 | -62.48 | -38.62 | 1.88 | 1.85 | 1.83 | -1.60 | -2.66 | 17.27 | 7.62 | 11.27 | -55.88 | -34.74 |
| 20-01 | 10.42 | 3.66 | 7.24 | -64.88 | -30.52 | 1.67 | 1.84 | 1.80 | 10.18 | 7.78 | 12.10 | 5.50 | 9.03 | -54.55 | -25.37 |
| 21-01 | 6.21 | 1.97 | 4.61 | -68.28 | -25.76 | 1.16 | 0.97 | 1.39 | -16.38 | 19.83 | 7.37 | 2.94 | 6.00 | -60.11 | -18.59 |
| **Total** | 14.74 | 6.89 | 10.70 | -53.26% | -27.41% | 2.09 | 2.12 | 2.25 | 1.44% | 7.66% | 16.83 | 9.01 | 12.95 | -46.46% | -23.05% |

**(b)** Total time (s)

| | RMP | | | | | Pricing | | | | | Total | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Day | B-S | E-S | H-S | $\Delta_E$ | $\Delta_H$ | B-S | E-S | H-S | $\Delta_E$ | $\Delta_H$ | B-S | E-S | H-S | $\Delta_E$ | $\Delta_H$ |
| 08-01 | 2,913.20 | 3,488.73 | 3,962.35 | 19.76 | 36.01 | 391.27 | 1,234.42 | 1,053.42 | 215.49 | 169.23 | 3,304.47 | 4,723.15 | 5,015.77 | 42.93 | 51.79 |
| 09-01 | 2,338.83 | 1,378.57 | 1,844.03 | -41.06 | -21.16 | 251.71 | 517.42 | 370.96 | 105.56 | 47.38 | 2,590.53 | 1,895.99 | 2,214.98 | -26.81 | -14.50 |
| 10-01 | 2,257.62 | 1,674.16 | 1,970.00 | -25.84 | -12.74 | 361.25 | 436.43 | 367.69 | 20.81 | 1.78 | 2,618.88 | 2,110.59 | 2,337.69 | -19.41 | -10.74 |
| 11-01 | 2,404.24 | 3,085.90 | 2,969.95 | 28.35 | 23.53 | 356.83 | 641.16 | 503.61 | 79.68 | 41.13 | 2,761.07 | 3,727.06 | 3,473.56 | 34.99 | 25.80 |
| 12-01 | 1,999.88 | 1,793.10 | 2,460.04 | -10.34 | 23.01 | 329.46 | 560.42 | 486.14 | 70.10 | 47.56 | 2,329.33 | 2,353.52 | 2,946.18 | 1.04 | 26.48 |
| 13-01 | 1,172.07 | 893.86 | 823.22 | -23.74 | -29.76 | 181.11 | 355.50 | 243.39 | 96.29 | 34.39 | 1,353.18 | 1,249.35 | 1,066.60 | -7.67 | -21.18 |
| 14-01 | 869.30 | 468.07 | 750.82 | -46.16 | -13.63 | 134.25 | 242.07 | 226.16 | 80.31 | 68.46 | 1,003.56 | 710.14 | 976.98 | -29.24 | -2.65 |
| 15-01 | 1,972.26 | 1,764.66 | 2,040.15 | -10.53 | 3.44 | 230.69 | 472.38 | 310.17 | 104.77 | 34.45 | 2,202.94 | 2,237.04 | 2,350.32 | 1.55 | 6.69 |
| 16-01 | 2,091.91 | 1,171.13 | 1,802.04 | -44.02 | -13.86 | 267.67 | 354.88 | 309.37 | 32.58 | 15.58 | 2,359.58 | 1,526.01 | 2,111.41 | -35.33 | -10.52 |
| 17-01 | 1,452.38 | 1,471.30 | 1,369.96 | 1.30 | -5.67 | 246.80 | 464.96 | 334.54 | 88.40 | 35.55 | 1,699.17 | 1,936.25 | 1,704.50 | 13.95 | 0.31 |
| 18-01 | 2,075.82 | 2,119.92 | 1,771.10 | 2.12 | -14.68 | 287.41 | 497.36 | 318.14 | 73.05 | 10.69 | 2,363.23 | 2,617.28 | 2,089.25 | 10.75 | -11.59 |
| 19-01 | 1,353.82 | 814.09 | 1,113.80 | -39.87 | -17.73 | 165.60 | 260.16 | 216.28 | 57.10 | 30.60 | 1,519.42 | 1,074.25 | 1,330.09 | -29.30 | -12.46 |
| 20-01 | 854.74 | 435.78 | 730.80 | -49.02 | -14.50 | 137.18 | 219.14 | 181.47 | 59.75 | 32.29 | 991.93 | 654.92 | 912.27 | -33.98 | -8.03 |
| 21-01 | 608.32 | 415.31 | 548.40 | -31.73 | -9.85 | 113.57 | 204.72 | 165.90 | 80.26 | 46.08 | 721.89 | 620.02 | 714.30 | -14.11 | -1.05 |
| **Total** | 24,364.39 | 20,974.58 | 24,156.66 | -13.91% | -0.85% | 3,454.80 | 6,461.02 | 5,087.24 | 87.02% | 47.25% | 27,819.18 | 27,435.57 | 29,243.90 | -1.38% | 5.12% |

The expert evaluation on both instances yields a consistent picture. E-S reduces the root-node RMP by approximately 77–78% and lowers time per iteration by approximately 29–31%. The overall root-node objective remains within 4% of the baseline, but the number of iterations and total root-node time increase. During the full-day evaluation, E-S retains a similarly small RMP and lowers time per iteration, yet these computational gains come with a clear quality cost. Its adjusted objective is approximately 19–20% above the baseline, and it also leaves more tasks uncovered than B-S on both instances. H-S produces smaller reductions in final RMP size, while its adjusted objective gap remains limited to approximately 9–10% and its uncovered-task counts stay close to the baseline. The hybrid strategy therefore recovers much of the computational benefit of expert-guided selection at the root without the same degradation in solution quality over the full planning day.

## 5.3 Machine learning column selection

This section evaluates whether a supervised model can imitate the expert-guided selector and whether the resulting learned policy improves column generation. The models are trained on decisions from root-node iterations of the three-base instance. The in-loop experiments then evaluate both root-node deployment and deployment over the full planning day.

The training data consist of bipartite column–constraint graphs collected from expert-guided runs. Each graph corresponds to one root-node iteration, and each candidate column receives the binary label implied by the expert selection. The collection period from 8 January to 2 June 2024 contains 147 planning days and 6,896 graphs in total. These graphs contain 9,502,314 candidate columns, with an average of 1,378 candidates per iteration. The expert selects 1,162,015 columns, which corresponds to a positive rate of 12.2%. Following the chronological split in Table 5.2, the training, validation, and test sets comprise 4,953, 621, and 1,322 graphs respectively, covering 15, 2, and 4 weeks with no gap between consecutive periods. Classification metrics in this section are reported on the training and validation graphs. The held-out test graphs are reserved for inference-time measurement and are not used for model selection.

### 5.3.1 Initial model comparison

The initial comparison considers plain logistic regression, two aggregated logistic regression variants, and GNNs with one to four message-passing layers. For both aggregated LR and GNN, models are evaluated with mean aggregation and with the combined mean, maximum, minimum, and sum variant denoted by M-M-M-S. The GNN comparison additionally includes attention-based variants. Table 5.18 reports classification performance on the training and validation periods. Recall, true negative rate (TNR), precision, and balanced accuracy (bal. acc.) use a prediction threshold of 0.5. AUC-PR is independent of this threshold and is particularly informative for the imbalanced labels in this experiment.

All GNN variants improve substantially over logistic regression. Validation AUC-PR increases from 0.164 for LR and 0.254 for the stronger aggregated LR model to values between 0.421 and 0.606 for the GNNs. Increasing message-passing depth is accompanied by higher validation AUC-PR within each GNN family. At the same depth, M-M-M-S

**Table 5.18:** Training and validation classification metrics under the initial hyperparameter settings.

| | Recall (%) | | TNR (%) | | Precision (%) | | Bal. acc. (%) | | AUC-PR | |
|---|---|---|---|---|---|---|---|---|---|---|
| Model | Train | Val | Train | Val | Train | Val | Train | Val | Train | Val |
| LR | 53.6 | 52.2 | 61.6 | 61.2 | 16.4 | 15.7 | 57.6 | 56.7 | 0.171 | 0.164 |
| Agg-LR (mean) | 64.3 | 64.7 | 59.6 | 58.6 | 18.3 | 17.8 | 62.0 | 61.6 | 0.214 | 0.209 |
| Agg-LR (M-M-M-S) | 69.5 | 67.7 | 61.3 | 63.4 | 20.2 | 20.4 | 65.4 | 65.6 | 0.256 | 0.254 |
| 1L-mean | 72.8 | 71.2 | 71.7 | 73.0 | 26.6 | 26.7 | 72.2 | 72.1 | 0.424 | 0.421 |
| 2L-mean | 74.9 | 75.8 | 75.3 | 73.9 | 29.9 | 28.7 | 75.1 | 74.9 | 0.518 | 0.511 |
| 3L-mean | 76.3 | 77.3 | 77.4 | 76.3 | 32.2 | 31.1 | 76.8 | 76.8 | 0.554 | 0.551 |
| 4L-mean | 78.6 | 78.8 | 76.7 | 76.6 | 32.2 | 31.8 | 77.6 | 77.7 | 0.569 | 0.567 |
| 1L-M-M-M-S | 72.2 | 70.7 | 72.7 | 73.9 | 27.1 | 27.3 | 72.4 | 72.3 | 0.431 | 0.426 |
| 2L-M-M-M-S | 75.2 | 75.6 | 77.0 | 76.2 | 31.5 | 30.5 | 76.1 | 75.9 | 0.534 | 0.528 |
| 3L-M-M-M-S | 78.1 | 78.4 | 78.7 | 78.3 | 34.1 | 33.3 | 78.4 | 78.3 | 0.582 | 0.577 |
| 4L-M-M-M-S | 81.2 | 81.5 | 77.7 | 77.8 | 33.9 | 33.7 | 79.5 | 79.6 | 0.607 | 0.606 |
| 1L-attention | 71.4 | 70.0 | 72.9 | 74.2 | 27.1 | 27.3 | 72.2 | 72.1 | 0.426 | 0.423 |
| 2L-attention | 74.0 | 75.3 | 77.4 | 75.5 | 31.5 | 29.9 | 75.7 | 75.4 | 0.528 | 0.521 |
| 3L-attention | 77.4 | 79.1 | 77.8 | 76.2 | 33.0 | 31.5 | 77.6 | 77.6 | 0.570 | 0.568 |
| 4L-attention | 78.5 | 78.8 | 78.4 | 78.7 | 33.9 | 33.8 | 78.5 | 78.7 | 0.587 | 0.590 |

and attention variants score higher than mean aggregation, but these differences remain small relative to the step from logistic regression to GNN. Train and validation metrics remain close across architectures, with no pronounced train–validation gap.

The learned selector is called once per column generation iteration. Its classification performance must therefore be considered together with inference time. Table 5.19 reports the forward-pass latency of the initial models on the held-out test graphs. The labels of these graphs are not used for this measurement. Using the held-out period to measure runtime therefore does not influence model selection through classification performance.

Logistic regression variants are substantially faster than the GNNs. The mean and median are close for every model, indicating that the central latency estimate is not driven by a small number of slow calls. Among the GNN architectures, inference time increases with depth and depends strongly on the aggregation mechanism. At four message-passing layers, the mean-aggregation model averages 12.56 milliseconds per iteration, compared with 32.55 milliseconds for the attention variant and 65.59 milliseconds for M-M-M-S.

Figure 5.9 displays the trade-off between mean inference time and validation AUC-PR. The four-layer mean model reaches a validation AUC-PR of 0.567 at 12.56 milliseconds per iteration. The four-layer attention model raises AUC-PR to 0.590 while increasing mean latency to 32.55 milliseconds. The highest AUC-PR of 0.606 is obtained by the four-layer M-M-M-S model at 65.59 milliseconds per iteration.

The four-layer mean model, indicated by the blue diamond marker, is selected for tuning. It provides most of the validation improvement obtained by the more elaborate architectures while retaining a considerably lower inference time. Even the more complex models require only tens of milliseconds for prediction, which is small relative to the RMP and pricing times per iteration. Their additional predictive improvement may therefore still translate into useful in-loop behavior. Nevertheless, the four-layer mean model provides a suitable starting point for tuning. Tuning each more complex architecture would

**Table 5.19:** Inference latency in milliseconds of the initial models on the test graphs. Times are measured per graph and therefore per column generation iteration.

| Model | Mean (ms) | Median (ms) |
|---|---|---|
| LR | 0.21 | 0.21 |
| Agg-LR (mean) | 0.22 | 0.22 |
| Agg-LR (M-M-M-S) | 0.32 | 0.28 |
| 1L-mean | 3.45 | 3.55 |
| 2L-mean | 6.68 | 6.86 |
| 3L-mean | 9.54 | 9.85 |
| 4L-mean | 12.56 | 12.93 |
| 1L-M-M-M-S | 20.16 | 20.95 |
| 2L-M-M-M-S | 34.85 | 35.88 |
| 3L-M-M-M-S | 49.49 | 51.42 |
| 4L-M-M-M-S | 65.59 | 67.88 |
| 1L-attention | 8.38 | 8.22 |
| 2L-attention | 16.53 | 16.29 |
| 3L-attention | 24.39 | 24.24 |
| 4L-attention | 32.55 | 32.61 |

require additional computational effort, while their gains in validation AUC-PR over the four-layer mean model remain limited. Evaluating whether those gains justify deploying the more complex models inside the solver remains a useful direction for further investigation.

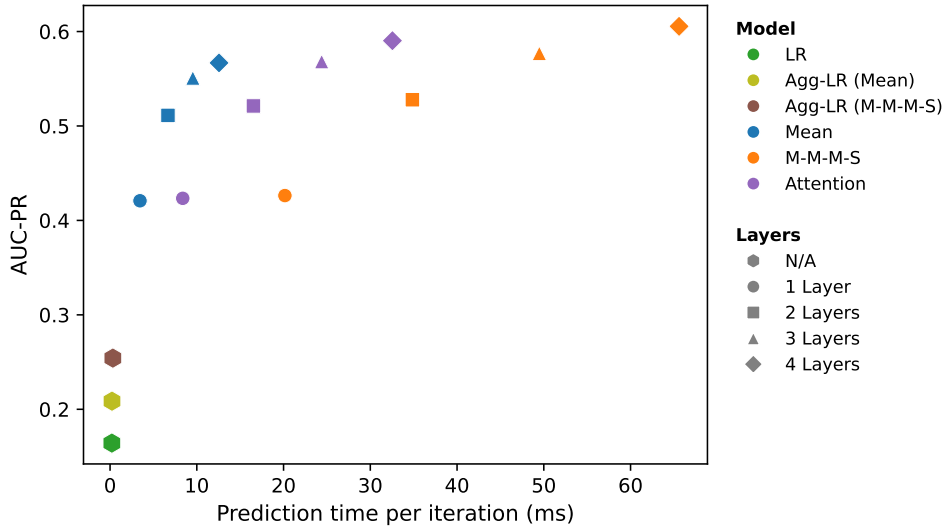



**Figure 5.9:** Mean inference time per column generation iteration and validation AUC-PR for the initial models. Color identifies the model family and marker shape identifies message-passing depth.

Figure 5.10 shows the training and validation loss of the selected architecture. Both losses decrease over training and remain close to one another.

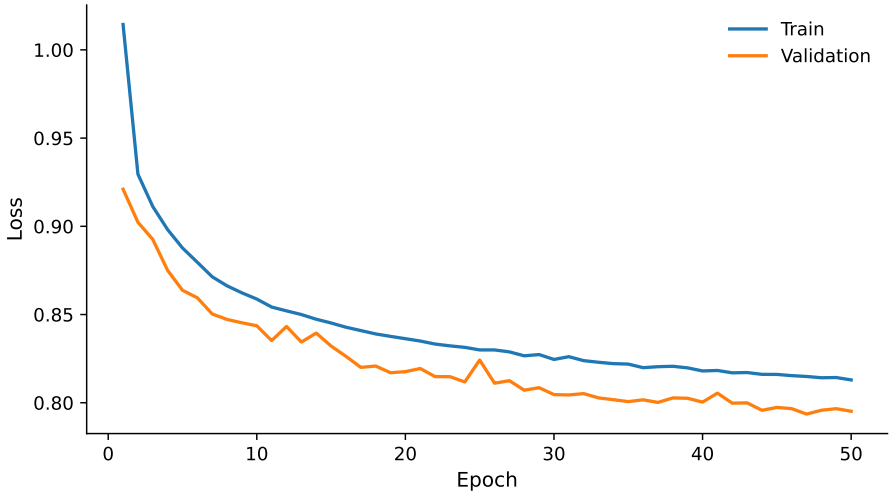



**Figure 5.10:** Training and validation loss of the four-layer mean GNN under the initial hyperparameter settings.

### 5.3.2 Tuned model performance

A focused search is performed for the selected four-layer mean architecture. Table 5.20 lists the hyperparameter values with which this architecture is tested. Selection is based only on validation performance. The chosen values are shown in bold.

**Table 5.20:** Hyperparameter values investigated for the four-layer mean GNN, with chosen values shown in bold.

| Hyperparameter | Investigated values |
|---|---|
| Hidden dimension $d_h$ | 64, **128** |
| Dropout probability $p$ | **0.1**, 0.2, 0.3 |
| Learning rate | $5 \times 10^{-4}$, $\mathbf{10^{-3}}$ |
| Weight decay | $\mathbf{10^{-5}}$, $10^{-4}$ |

The only modification relative to the initial hyperparameter settings is the increase of the hidden dimension from 64 to 128. The resulting model has a mean inference time of 23.81 milliseconds per iteration and a median of 24.46 milliseconds, which is 1.85 times slower than the initial configuration.

The final model is retrained from scratch on the combined training and validation period. As described in Section 4.5.4, early stopping is no longer possible because there is no separate validation period during this final fit. The fixed training budget is therefore set to 49 epochs, which is the epoch at which early stopping selected the best checkpoint during hyperparameter search. The test period remains held out until this final model has been fitted.

Table 5.21 reports the final classification scores at threshold 0.5. Performance on the combined training and validation data is close to performance on the held-out test period. The test AUC-PR of 0.580 and the test balanced accuracy of 78.5% both exceed the training and validation scores of the initial four-layer mean model in Table 5.18.

Figure 5.11 shows the training loss during the final 49-epoch fit. No validation curve

**Table 5.21:** Training and test classification metrics of the tuned four-layer mean GNN.

| Set | Recall (%) | TNR (%) | Precision (%) | Bal. acc. (%) | AUC-PR |
|---|---|---|---|---|---|
| Train | 82.9 | 74.2 | 31.1 | 78.5 | 0.591 |
| Test | 81.9 | 75.0 | 30.5 | 78.5 | 0.580 |

is available for this run because the former validation period is included in the training data. The loss decreases steadily and follows a similar trajectory to the training curve in Figure 5.10, which suggests that optimization during final retraining proceeded in a comparable way despite the absence of a separate validation split.

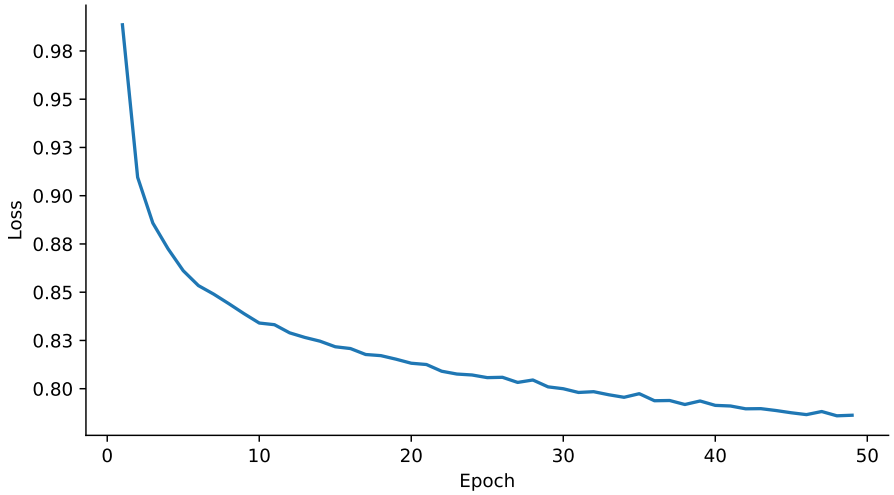



**Figure 5.11:** Training loss of the tuned four-layer mean GNN during the final fit on the combined training and validation period.

The prediction threshold determines the balance between recovering expert-selected columns and retaining columns that the expert rejects. Table 5.22 reports this trade-off on the combined training and validation data and on the test data. AUC-PR is omitted because it is calculated from the complete ranking and does not change with the threshold.

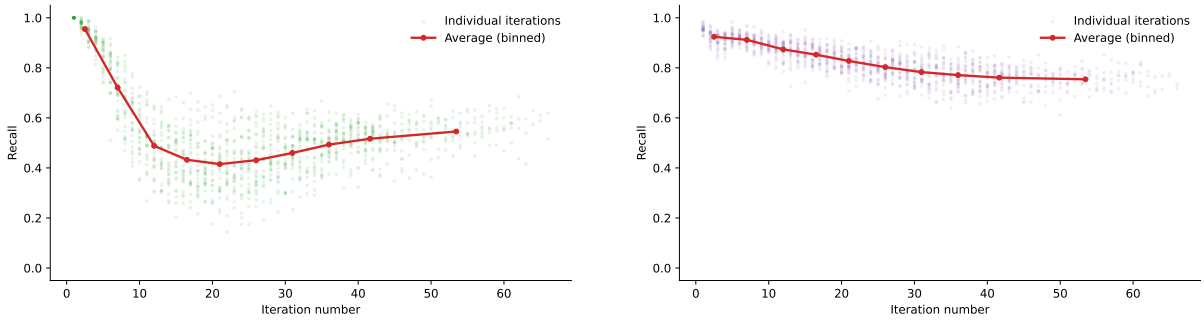
Lower thresholds recover a larger share of the columns selected by the expert, but they also accept more negative examples. Test recall decreases from 99.0% at threshold 0.1 to 29.4% at threshold 0.9. Conversely, both test TNR and precision increase with the threshold, from 28.5% to 98.7% and from 15.7% to 74.7% respectively. Bal. acc. is highest at threshold 0.5 for both reported sets.

Recall directly measures the fraction of expert-selected columns that is recovered by the model. It is therefore the clearest signal of how closely the learned model imitates the positive decisions of the expert. Figure 5.12 reports recall at threshold 0.5 over the root-node iteration sequence for LR and the tuned GNN. Each point represents one planning-day and iteration pair for which the expert selects at least one column. The solid curves show the mean within quantile bins of the iteration number, grouping iterations at similar positions in the sequence so that the trend over the course of a solve can be read more easily than from the individual points alone. LR recall drops sharply after the

**Table 5.22:** Training and test classification metrics of the tuned four-layer mean GNN at different prediction thresholds.

| | | Prediction threshold | | | | |
|---|---|---|---|---|---|---|
| Set | Metric | 0.1 | 0.3 | 0.5 | 0.7 | 0.9 |
| Train | Recall (%) | 99.2 | 93.8 | 82.9 | 63.9 | 29.7 |
| | TNR (%) | 27.0 | 54.3 | 74.2 | 89.0 | 98.7 |
| | Precision (%) | 16.1 | 22.4 | 31.1 | 45.0 | 75.7 |
| | Bal. acc. (%) | 63.1 | 74.1 | 78.5 | 76.5 | 64.2 |
| Test | Recall (%) | 99.0 | 93.2 | 81.9 | 63.1 | 29.4 |
| | TNR (%) | 28.5 | 55.6 | 75.0 | 89.4 | 98.7 |
| | Precision (%) | 15.7 | 22.0 | 30.5 | 44.3 | 74.7 |
| | Bal. acc. (%) | 63.8 | 74.4 | 78.5 | 76.2 | 64.0 |

first root-node iterations and subsequently recovers toward the end of the observed range. The lowest part of this curve identifies a section of the trajectory in which LR recovers relatively few expert-selected columns. The tuned GNN follows a smoother pattern. Its recall gradually decreases as the iteration number rises and does not show the same pronounced drop.




**(a)** Plain logistic regression **(b)** Tuned four-layer mean GNN

**Figure 5.12:** Test recall at prediction threshold 0.5 over the root-node iteration sequence. Figure 5.12a shows plain logistic regression and Figure 5.12b the tuned four-layer mean GNN. Points represent individual planning-day and iteration pairs. The solid curves show binned mean recall.

Figure 5.13 aggregates recall at the same threshold of 0.5 by test day. Both models remain reasonably stable across the test period. The GNN curve lies above the LR curve on the test days, consistent with the higher overall recall reported for the tuned model.

### 5.3.3 Learned selector in the column generation loop

The tuned model is next evaluated inside the column generation loop over the period from 6 May to 30 June 2024. The tables use the same compact notation as the expert-guided evaluation. B-S denotes the baseline selector and E-S denotes the expert-guided selector. G-S denotes the GNN selector, where candidate columns are added when their predicted probability is at least 0.5. H-E-S applies E-S throughout the root node and returns to B-S after the first dive. H-G-S follows the same hybrid construction with G-S at the root

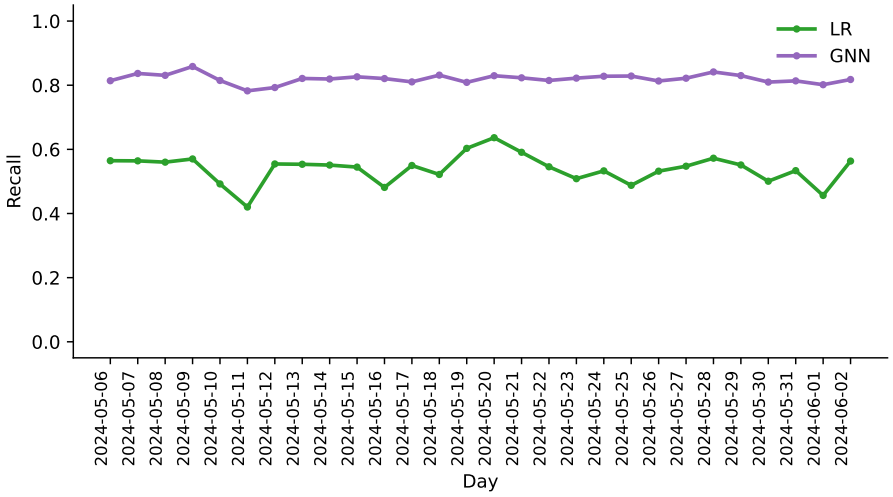



**Figure 5.13:** Test recall at prediction threshold 0.5 per planning day for plain logistic regression and the tuned four-layer mean GNN.

node. Percentages report the change from B-S for every metric, except uncovered tasks, which use absolute differences.

The root-node comparison includes B-S, E-S, and G-S. The full-day comparison includes B-S, E-S, H-E-S, and H-G-S. This distinction evaluates the learned selector directly in the setting on which it was trained and then tests whether its root-node decisions remain visible when baseline selection resumes during diving.

#### 5.3.3.1 Root node

Table 5.23 reports period totals for iterations and total time, together with averages of final RMP size and of RMP, pricing, and combined time per iteration. E-S increases the number of iterations by 75.61%, while G-S limits this increase to 17.48%. Both selectors reduce the final RMP size. The reduction is 77.91% for E-S and 64.35% for G-S.

**Table 5.23:** Root-node efficiency over the learned-selector evaluation period on the three-base instance. Time per iteration and final RMP size are averages. Other values are period totals.

| Metric | B-S | E-S | G-S | $\Delta_E$ (%) | $\Delta_G$ (%) |
|---|---|---|---|---|---|
| Number of iterations | 1,648 | 2,894 | 1,936 | 75.61 | 17.48 |
| Final RMP size | 22,454.98 | 4,960.82 | 8,005.98 | -77.91 | -64.35 |
| RMP time per iteration (s) | 1.95 | 1.16 | 1.39 | -40.51 | -28.72 |
| Pricing time per iteration (s) | 0.86 | 0.87 | 0.88 | 1.16 | 2.33 |
| Total time per iteration (s) | 2.81 | 2.03 | 2.28 | -27.76 | -18.86 |
| Total RMP time (s) | 3,207.84 | 3,365.81 | 2,698.24 | 4.92 | -15.89 |
| Total pricing time (s) | 1,418.79 | 2,509.04 | 1,713.31 | 76.84 | 20.76 |
| Total time (s) | 4,626.73 | 5,874.87 | 4,411.61 | 26.98 | -4.65 |

The smaller RMPs reduce RMP time per iteration for both alternative selectors. Total time per iteration decreases by 27.76% for E-S and by 18.86% for G-S. The additional iterations under E-S raise its total time by 26.98%. G-S instead reduces total time by

4.65% because its smaller reduction in time per iteration is accompanied by a much smaller increase in iterations.

Table 5.24 reports the root-node objective separately from the computational metrics. E-S increases the period total by 2.98%. The objective under G-S is 0.89% above B-S.

**Table 5.24:** Root-node objective values over the learned-selector evaluation period on the three-base instance.

| Metric | B-S | E-S | G-S | $\Delta_E$ (%) | $\Delta_G$ (%) |
|---|---|---|---|---|---|
| Objective value (€) | 9,897,338.87 | 10,192,486.91 | 9,985,835.28 | 2.98 | 0.89 |

Figure 5.14 shows the root-node RMP-time distributions over the longer learned-selector evaluation period. For all three selectors, the mean is close to the median. Under both measures, E-S is below G-S and G-S is below B-S. Both E-S and G-S also have a narrower interquartile range than B-S, with the largest reduction under E-S. This is consistent with the relation between smaller RMPs and lower time per iteration.

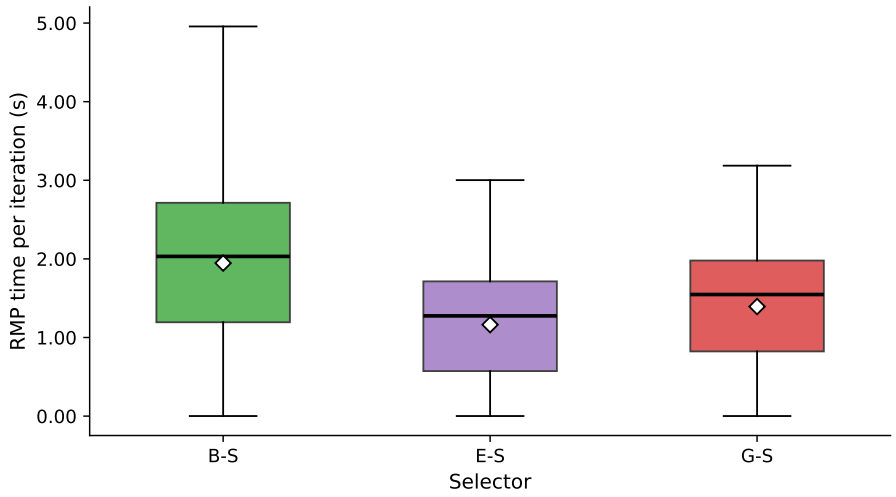



**Figure 5.14:** Boxplots of RMP time per iteration at the root node for B-S, E-S, and G-S over the learned-selector evaluation period on the three-base instance. Each box spans the interquartile range, the black horizontal line marks the median, and the white diamond marks the mean.

#### 5.3.3.2 Full planning day

Table 5.25 extends the comparison over the full planning day. E-S performs 81.12% more iterations than B-S. The increase falls to 40.69% for H-E-S and 12.91% for H-G-S. E-S retains the smallest final RMP. Both hybrid strategies finish with an RMP that is approximately one quarter smaller than the baseline RMP.

All three alternatives lower total time per iteration. E-S has the largest reduction at 48.08%, followed by H-E-S at 26.28% and H-G-S at 20.19%. Over the complete period, E-S reduces total time by 5.90% and H-G-S by 10.11%. H-E-S increases total time by 3.58%.

**Table 5.25:** Full-day efficiency over the learned-selector evaluation period on the three-base instance. Time per iteration and final RMP size are averages. Other values are period totals.

| Metric | B-S | E-S | H-E-S | H-G-S | $\Delta_E$ (%) | $\Delta_{H\text{-}E}$ (%) | $\Delta_{H\text{-}G}$ (%) |
|---|---|---|---|---|---|---|---|
| Number of iterations | 4,269 | 7,732 | 6,006 | 4,820 | 81.12 | 40.69 | 12.91 |
| Final RMP size | 39,412.66 | 8,241.34 | 29,301.32 | 29,779.34 | -79.09 | -25.66 | -24.44 |
| RMP time per iteration (s) | 2.63 | 1.11 | 1.72 | 1.96 | -57.79 | -34.60 | -25.48 |
| Pricing time per iteration (s) | 0.49 | 0.52 | 0.58 | 0.53 | 6.12 | 18.37 | 8.16 |
| Total time per iteration (s) | 3.12 | 1.62 | 2.30 | 2.49 | -48.08 | -26.28 | -20.19 |
| Total RMP time (s) | 11,225.90 | 8,553.87 | 10,350.48 | 9,433.67 | -23.80 | -7.80 | -15.97 |
| Total pricing time (s) | 2,104.06 | 3,989.32 | 3,456.68 | 2,548.79 | 89.60 | 64.29 | 21.14 |
| Total time (s) | 13,329.97 | 12,543.16 | 13,807.17 | 11,982.49 | -5.90 | 3.58 | -10.11 |

Solution quality is reported separately in Table 5.26. The adjusted objective removes the penalty assigned to uncovered-task slack. E-S leaves 278 tasks uncovered compared with 223 for B-S and raises the adjusted objective by 8.32%. H-E-S leaves 233 tasks uncovered and has an adjusted objective that is 1.69% higher than B-S. H-G-S leaves 217 tasks uncovered and its adjusted objective is 0.12% lower than the baseline total.

**Table 5.26:** Full-day solution quality over the learned-selector evaluation period on the three-base instance. The adjusted objective removes the slack cost of €50,000 per uncovered task from the solver objective, leaving the assignment penalties from the fairness feedback. The $\Delta$ columns report absolute differences for uncovered tasks and percentage change relative to B-S for adjusted objective.

| Metric | B-S | E-S | H-E-S | H-G-S | $\Delta_E$ | $\Delta_{H\text{-}E}$ | $\Delta_{H\text{-}G}$ |
|---|---|---|---|---|---|---|---|
| Uncovered tasks | 223 | 278 | 233 | 217 | 55 | 10 | −6 |
| Adjusted objective (€) | 3,232,910.80 | 3,501,798.38 | 3,287,431.65 | 3,229,040.73 | 8.32% | 1.69% | -0.12% |

Figure 5.15 shows the full-day RMP-time distributions over the same evaluation period. As at the root node, the mean and median nearly coincide for every selector. Under both measures, E-S is lowest, followed by H-E-S, then H-G-S, and then B-S. E-S, H-E-S, and H-G-S also have narrower interquartile ranges than B-S. This is consistent with the extent to which each selector limits RMP growth and with the reductions in RMP time per iteration reported in the table.

### 5.3.4 Prediction threshold sensitivity

The classification results in Table 5.22 show how the prediction threshold changes recall, TNR, precision, and balanced accuracy outside the solver. The same thresholds are evaluated inside the column generation loop to determine how this classification trade-off affects computation and solution quality. At the root node the comparison uses G-S at each threshold. Over the full planning day it uses the corresponding hybrid H-G-S configuration, so that baseline selection resumes after the first dive. The tables place the prediction thresholds in the columns. The results at threshold 0.5 are the same G-S and H-G-S runs reported in the preceding learned-selector comparison and are repeated here to make the sensitivity pattern visible.

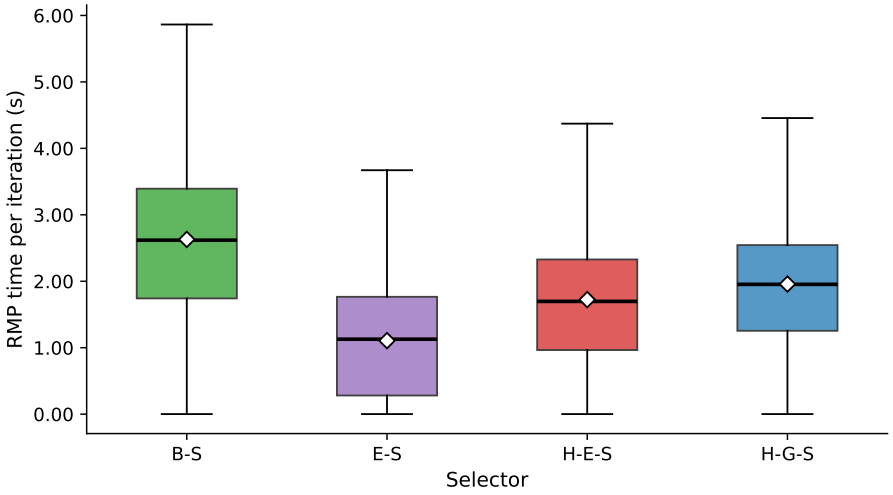



**Figure 5.15:** Boxplots of RMP time per iteration over the full planning day for B-S, E-S, H-E-S, and H-G-S over the learned-selector evaluation period on the three-base instance. Each box spans the interquartile range, the black horizontal line marks the median, and the white diamond marks the mean.

#### 5.3.4.1 Root node

Table 5.27 shows that raising the threshold reduces the final RMP size and the time per iteration. At threshold 0.1 the final RMP is already 25.77% smaller than under B-S, and the reduction grows to 83.44% at threshold 0.9. Total time per iteration decreases steadily from −8.19% at threshold 0.1 to −35.94% at threshold 0.9. The number of iterations moves in the opposite direction. It increases only slightly at the lower thresholds and then rises sharply, reaching +183.68% at threshold 0.9. Period total time therefore does not improve monotonically. Thresholds 0.1, 0.3, and 0.5 reduce total time by 6.01%, 8.06%, and 4.65% respectively, whereas thresholds 0.7 and 0.9 raise it by 10.69% and 81.68%.

The corresponding root-node objectives are reported in Table 5.28. The objective also rises with the threshold. The increase remains below 1% for thresholds 0.1, 0.3, and 0.5. At threshold 0.7 it reaches 3.05%, and at threshold 0.9 it reaches 26.76%.

#### 5.3.4.2 Full planning day

The full-day pattern in Table 5.29 is similar. Higher thresholds continue to shrink the final RMP and to reduce time per iteration, while the number of iterations increases. Thresholds 0.3 and 0.5 give the largest reductions in period total time, namely 10.08% and 10.11%. Threshold 0.1 reduces total time by only 1.45%, threshold 0.7 by 1.32%, and threshold 0.9 increases it by 19.69%.

Solution quality is reported in Table 5.30. Thresholds 0.1, 0.3, 0.5, and 0.7 leave between 217 and 224 tasks uncovered, compared with 223 under B-S. Their adjusted objectives also remain close to B-S, with differences between −0.12% and 1.75%. Threshold 0.9 stands apart, as it leaves 237 tasks uncovered and raises the adjusted objective by 7.50%. The uncovered-task counts and adjusted objectives do not change monotonically with the threshold. This again reflects path dependence in the full-day procedure. Differ-

**Table 5.27:** Root-node sensitivity to the GNN prediction threshold on the three-base instance.

| Metric | B-S | 0.1 | 0.3 | 0.5 | 0.7 | 0.9 |
|---|---|---|---|---|---|---|
| Number of iterations | 1,648 | 1,683 | 1,758 | 1,936 | 2,372 | 4,675 |
| Δ number of iterations (%) | | 2.12 | 6.67 | 17.48 | 43.93 | 183.68 |
| Final RMP size | 22,454.98 | 16,667.71 | 11,522.39 | 8,005.98 | 5,464.16 | 3,718.62 |
| Δ final RMP size (%) | | -25.77 | -48.69 | -64.35 | -75.67 | -83.44 |
| RMP time per iteration (s) | 1.95 | 1.73 | 1.54 | 1.39 | 1.25 | 0.87 |
| Δ RMP time per iteration (%) | | -11.28 | -21.03 | -28.72 | -35.90 | -55.38 |
| Pricing time per iteration (s) | 0.86 | 0.86 | 0.88 | 0.88 | 0.91 | 0.93 |
| Δ pricing time per iteration (%) | | 0.00 | 2.33 | 2.33 | 5.81 | 8.14 |
| Total time per iteration (s) | 2.81 | 2.58 | 2.42 | 2.28 | 2.16 | 1.80 |
| Δ total time per iteration (%) | | -8.19 | -13.88 | -18.86 | -23.13 | -35.94 |
| Total RMP time (s) | 3,207.84 | 2,903.74 | 2,702.56 | 2,698.24 | 2,961.54 | 4,058.96 |
| Δ total RMP time (%) | | -9.48 | -15.75 | -15.89 | -7.68 | 26.53 |
| Total pricing time (s) | 1,418.79 | 1,445.02 | 1,551.03 | 1,713.31 | 2,159.63 | 4,346.89 |
| Δ total pricing time (%) | | 1.85 | 9.32 | 20.76 | 52.22 | 206.38 |
| Total time (s) | 4,626.73 | 4,348.77 | 4,253.61 | 4,411.61 | 5,121.18 | 8,405.89 |
| Δ total time (%) | | -6.01 | -8.06 | -4.65 | 10.69 | 81.68 |

**Table 5.28:** Root-node objective sensitivity to the GNN prediction threshold on the three-base instance.

| Metric | B-S | 0.1 | 0.3 | 0.5 | 0.7 | 0.9 |
|---|---|---|---|---|---|---|
| Objective value (€) | 9,897,338.87 | 9,907,993.71 | 9,914,531.23 | 9,985,835.28 | 10,198,725.92 | 12,545,702.99 |
| Δ objective (%) | | 0.11 | 0.17 | 0.89 | 3.05 | 26.76 |

ent root-node selections can lead to different diving trajectories and therefore to different uncovered-task counts at the end of the day.

**Table 5.29:** Full-day sensitivity to the GNN prediction threshold on the three-base instance.

| Metric | B-S | 0.1 | 0.3 | 0.5 | 0.7 | 0.9 |
|---|---|---|---|---|---|---|
| Iterations | 4,269 | 4,446 | 4,486 | 4,820 | 5,567 | 7,828 |
| Δ iterations (%) | | 4.15 | 5.08 | 12.91 | 30.41 | 83.37 |
| Final RMP size | 39,412.66 | 35,169.55 | 31,572.68 | 29,779.34 | 29,174.96 | 27,225.79 |
| Δ final RMP size (%) | | -10.77 | -19.89 | -24.44 | -25.98 | -30.92 |
| RMP time per iteration (s) | 2.63 | 2.47 | 2.16 | 1.96 | 1.81 | 1.36 |
| Δ RMP time per iteration (%) | | -6.08 | -17.87 | -25.48 | -31.18 | -48.29 |
| Pricing time per iteration (s) | 0.49 | 0.49 | 0.51 | 0.53 | 0.56 | 0.68 |
| Δ pricing time per iteration (%) | | 0.00 | 4.08 | 8.16 | 14.29 | 38.78 |
| Total time per iteration (s) | 3.12 | 2.95 | 2.67 | 2.49 | 2.36 | 2.04 |
| Δ total time per iteration (%) | | -5.45 | -14.42 | -20.19 | -24.36 | -34.62 |
| Total RMP time (s) | 11,225.90 | 10,961.65 | 9,684.05 | 9,433.67 | 10,048.57 | 10,667.70 |
| Δ total RMP time (%) | | -2.35 | -13.73 | -15.97 | -10.49 | -4.97 |
| Total pricing time (s) | 2,104.06 | 2,175.04 | 2,302.75 | 2,548.79 | 3,106.11 | 5,286.47 |
| Δ total pricing time (%) | | 3.37 | 9.44 | 21.14 | 47.62 | 151.25 |
| Total time (s) | 13,329.97 | 13,136.65 | 11,986.75 | 11,982.49 | 13,154.65 | 15,954.16 |
| Δ total time (%) | | -1.45 | -10.08 | -10.11 | -1.32 | 19.69 |

**Table 5.30:** Full-day solution-quality sensitivity to the GNN prediction threshold on the three-base instance. The adjusted objective removes the slack cost of €50,000 per uncovered task from the solver objective, leaving the assignment penalties from the fairness feedback. The Δ uncovered-tasks row reports absolute differences relative to B-S and the Δ adjusted-objective row reports percentage changes relative to B-S.

| Metric | B-S | 0.1 | 0.3 | 0.5 | 0.7 | 0.9 |
|---|---|---|---|---|---|---|
| Uncovered tasks | 223 | 218 | 224 | 217 | 217 | 237 |
| Δ uncovered tasks | | −5 | 1 | −6 | −6 | 14 |
| Adjusted objective (€) | 3,232,910.80 | 3,234,215.80 | 3,249,813.63 | 3,229,040.73 | 3,289,333.98 | 3,475,459.37 |
| Δ adjusted objective (%) | | 0.04 | 0.52 | -0.12 | 1.75 | 7.50 |

### 5.3.5 Generalizability to larger instance

The learned model is trained only on root-node iterations from the three-base instance. To examine whether its behavior transfers to a larger problem without retraining, the in-loop comparison from the preceding subsections is repeated on the six-base instance over the same period from 6 May to 30 June 2024. The GNN selector uses a prediction threshold of 0.5.

#### 5.3.5.1 Root node

Table 5.31 reports the root-node efficiency results. E-S increases the number of iterations by 75.04%, whereas G-S remains closer to B-S with an increase of 9.14%. Both selectors substantially reduce the final RMP. The reductions are 79.43% for E-S and 62.50% for G-S.

**Table 5.31:** Root-node efficiency over the learned-selector evaluation period on the six-base instance. Time per iteration and final RMP size are averages. Other values are period totals.

| Metric | B-S | E-S | G-S | $\Delta_E$ (%) | $\Delta_G$ (%) |
|---|---|---|---|---|---|
| Number of iterations | 1,783 | 3,121 | 1,946 | 75.04 | 9.14 |
| Final RMP size | 45,849.79 | 9,430.21 | 17,191.88 | -79.43 | -62.50 |
| RMP time per iteration (s) | 13.87 | 8.38 | 9.87 | -39.58 | -28.84 |
| Pricing time per iteration (s) | 4.38 | 4.22 | 4.09 | -3.65 | -6.62 |
| Total time per iteration (s) | 18.25 | 12.61 | 13.96 | -30.90 | -23.51 |
| Total RMP time (s) | 24,736.19 | 26,162.09 | 19,209.29 | 5.76 | -22.34 |
| Total pricing time (s) | 7,807.59 | 13,184.36 | 7,957.13 | 68.87 | 1.92 |
| Total time (s) | 32,543.73 | 39,346.44 | 27,166.41 | 20.90 | -16.52 |

The reduction in RMP size again lowers the computational effort of an individual iteration. Total time per iteration decreases by 30.90% for E-S and by 23.51% for G-S. The additional iterations cause E-S to increase period total time by 20.90%. G-S instead reduces total time by 16.52%. This reduction is larger than the 4.65% observed on the three-base instance and is driven by G-S remaining closer to B-S in its number of iterations.

Table 5.32 reports the objective separately. E-S raises the period total by 2.58%, while G-S increases it by 0.84%.

**Table 5.32:** Root-node objective values over the learned-selector evaluation period on the six-base instance.

| Metric | B-S | E-S | G-S | $\Delta_E$ (%) | $\Delta_G$ (%) |
|---|---|---|---|---|---|
| Objective value (€) | 13,736,529.92 | 14,090,410.59 | 13,851,487.99 | 2.58 | 0.84 |

#### 5.3.5.2 Full planning day

Table 5.33 extends the transfer comparison over the full planning day. H-E-S performs 34.37% more iterations than B-S, while the increase under H-G-S is 6.52%. Both hybrid strategies finish with an RMP approximately 20% smaller than under B-S.

**Table 5.33:** Full-day efficiency over the learned-selector evaluation period on the six-base instance. Time per iteration and final RMP size are averages. Other values are period totals.

| Metric | B-S | H-E-S | H-G-S | $\Delta_{H\text{-}E}$ (%) | $\Delta_{H\text{-}G}$ (%) |
|---|---|---|---|---|---|
| Number of iterations | 6,069 | 8,155 | 6,465 | 34.37 | 6.52 |
| Final RMP size | 92,151.45 | 74,057.52 | 73,790.43 | -19.63 | -19.92 |
| RMP time per iteration (s) | 19.38 | 13.73 | 14.87 | -29.15 | -23.27 |
| Pricing time per iteration (s) | 2.17 | 2.45 | 2.08 | 12.90 | -4.15 |
| Total time per iteration (s) | 21.56 | 16.17 | 16.94 | -25.00 | -21.43 |
| Total RMP time (s) | 117,633.31 | 111,951.06 | 96,109.25 | -4.83 | -18.30 |
| Total pricing time (s) | 13,187.16 | 19,939.04 | 13,426.53 | 51.20 | 1.82 |
| Total time (s) | 130,820.48 | 131,890.11 | 109,535.75 | 0.82 | -16.27 |

The per-iteration pattern is consistent with the three-base instance. Total time per iteration decreases by 25.00% for H-E-S and by 21.43% for H-G-S. Over the complete period, H-E-S remains close to B-S with a total-time increase of 0.82%. H-G-S reduces total time by 16.27%, compared with 10.11% on the three-base instance.

The corresponding solution-quality measures are shown in Table 5.34. H-E-S leaves 33 more tasks uncovered than B-S and has an adjusted objective that is 1.36% higher. H-G-S leaves 19 more tasks uncovered and raises the adjusted objective by 1.73%.

**Table 5.34:** Full-day solution quality over the learned-selector evaluation period on the six-base instance. The adjusted objective removes the slack cost of €50,000 per uncovered task from the solver objective, leaving the assignment penalties from the fairness feedback. The $\Delta$ columns report absolute differences for uncovered tasks and percentage change relative to B-S for adjusted objective.

| Metric | B-S | H-E-S | H-G-S | $\Delta_{H\text{-}E}$ | $\Delta_{H\text{-}G}$ |
|---|---|---|---|---|---|
| Uncovered tasks | 309 | 342 | 328 | 33 | 19 |
| Adjusted objective (€) | 5,327,420.92 | 5,400,098.58 | 5,419,602.33 | 1.36% | 1.73% |

On the three-base instance, H-G-S left slightly fewer tasks uncovered than B-S and had a slightly lower adjusted objective. Both differences change direction on the six-base instance. This variation is consistent with the path dependence of the full-day procedure. Without having been trained on the six-base instance, G-S again limits RMP growth,

lowers time per iteration, and stays closer to B-S in the number of iterations than E-S. The resulting reduction in total time is visible at both the root node and over the full planning day.

# Chapter 6

# Discussion

This chapter discusses whether the machine-learning-based column-selection framework of Morabit et al. (2021) can be transferred to the NS crew planning problem with fairness over time. The investigation formulates and evaluates an expert MILP selector within the NS column generation framework. It then trains a graph neural network to imitate its decisions inside the solver loop. The sections below address the four research questions in turn and conclude with limitations and directions for future research.

## 6.1 Expert formulation in the NS setting

The expert MILP can be formulated and applied within the NS column generation framework, but this requires structural adaptations beyond a direct translation of the original approach. The plain expert MILP without a safety net stalls on certain planning days and finishes with a root-node objective up to 57 times the baseline value. This pattern is consistent with the observation of Morabit et al. (2021) that the basic expert requires a supplementary re-optimization step to prevent stalling. In the NS setting, however, the standard global safety net used in the original work does not transfer well.

The reason lies in the per-template structure of NS pricing. Because duties are generated separately for each crew member's roster template, the combined candidate pool is a union of many template-specific subsets rather than a single global pool. A global safety net that adds the top fraction of candidates by recomputed reduced cost can leave individual templates without any newly inserted columns in a given iteration. When a template receives no new column, its assignment dual barely changes, and that template's pricing subproblem will regenerate the same rejected candidates in the next iteration. Rather than progressing, the algorithm cycles at the template level, while the global objective appears to move forward for the templates that did receive new columns.

The MILP-TC formulation addresses this issue by incorporating the per-template coverage requirement directly into the expert MILP. By requiring at least one column from each template that still has candidates with negative reduced cost, the expert guarantees progress across all active templates in a single solve. This achieves the same per-template idea as MILP-TN, but without the second insertion phase that follows re-optimization in the alternative safety-net variants. The resulting selection therefore comes from one expert solve, which provides a cleaner target for supervised learning. Objective values and RMP sizes remain comparable to the other safety-net variants. The design follows

directly from the NS pricing decomposition, in which pricing operates on many parallel template-specific pools rather than a single unified pool.

## 6.2 Insights from expert-guided selection

The expert-guided E-S reduces the final root-node RMP by 77.29% on the three-base instance and 78.49% on the six-base instance. The reduction occurs on every planning day in the two-week evaluation window. This confirms that NS column generation contains substantial redundancy even after the task-disjoint filter has removed highly overlapping candidates. In the three-base configuration comparison, the baseline adds roughly 1,100 to 1,500 columns per iteration, whereas MILP-TC selects an average of 163 columns per iteration while reaching a similar root-node objective.

Most baseline additions are therefore not present in the final LP solution with positive weight. They are not without value to the column generation process, however. When a new column enters the LP basis fractionally during an intermediate iteration, it shifts which constraints are tight at the optimum. The dual values are shadow prices of those tight constraints. A basis change therefore alters the dual vector and that altered vector determines the reduced costs used in the next pricing step.

The broader baseline selection creates more opportunities for columns to enter the basis and alter the dual vector. The much smaller expert selection is more likely to produce only a small basis change. Pricing then operates against a similar cost structure and can regenerate the same candidates. Many baseline columns are eventually displaced and end up in the final RMP with zero weight, which explains the large redundancy at convergence. Their value may therefore lie in changing the trajectory during earlier iterations rather than in the final LP solution.

The smaller column pool reduces combined time per iteration by 29.27% on the three-base instance and 31.46% on the six-base instance. Most of this reduction comes from the RMP component, whose solution time decreases by 42.44% and 38.39%, respectively. The boxplots confirm that E-S has a lower mean and median than B-S in each instance. E-S also has a narrower interquartile range. The average reduction is therefore not driven by a few unusually fast early iterations. Pricing time per iteration is unchanged at 0.75 seconds on the three-base instance. On the six-base instance it decreases by 11.50%, possibly because the labeling algorithm computes path costs from arc reduced costs that depend on the dual vector. When duals change little between iterations, the cost landscape is more predictable and the procedure may terminate earlier. Even so, the pricing reduction is secondary to the RMP savings.

The per-iteration savings do not translate into lower total root-node time. E-S requires 72.73% more iterations on the three-base instance and 93.35% more on the six-base instance, raising total root-node time by 22.24% and 32.53%, respectively. Although MILP-TC requires progress on every active template, it still selects only a few columns per template each iteration. The large increase in iterations is consistent with marginal basis changes and insufficient dual movement to steer pricing toward new candidates. Pricing can then regenerate the same rejected duties, and the additional iterations outweigh the per-iteration savings.

In a set-partitioning formulation of the type studied by Morabit et al. (2021), this cycling is less persistent. Task coverage constraints are equalities, so over-coverage is

structurally impossible. Every task coverage constraint is therefore binding in a feasible LP solution, meaning it always carries a nonzero dual by complementary slackness. Even a sparse column selection therefore creates competition around tight coverage equalities. In the NS set-covering formulation, over-coverage at the task level is permitted. With a thin column pool, task constraints can remain feasible with positive surplus at the LP optimum. By complementary slackness, that non-binding inequality then carries a zero dual and is removed from the pricing signal. When more competing columns are available, as under baseline selection, those constraints can instead be driven to their bound and are then able to contribute to the dual signal. The crew assignment constraints are equalities in both formulations. They remain binding and are not forced to zero by complementary slackness. The structural difference therefore originates at the task coverage level.

The per-template pricing decomposition deepens this effect. Each subproblem sees only its own portion of the dual vector. A stuck template therefore cannot be rescued by progress elsewhere. The crew assignment equality for each template remains binding, but its dual signal alone is insufficient when task-level duals are zero. The larger iteration increase on the six-base instance reflects this at greater scale. More template-specific subproblems can cycle simultaneously, so the instance may require a broader column budget to maintain forward progress.

The preceding analysis concerns the root node. The full-day evaluation asks how the same thin-pool conditions behave once diving begins. After a duty variable is fixed, the problem is more constrained. A sparse column pool can then leave fewer backup duties for the segments that follow, and the limited task-level pricing signal may persist. E-S trajectories have more uncovered tasks and higher adjusted objectives than B-S, which is consistent with this explanation. These quality differences describe the realized diving paths. They are not a controlled ranking of selectors by full-day solution quality. On the computational side, E-S still reduces time per iteration. Total full-day time falls by 19.35% on the three-base instance but only by 1.38% on the six-base instance.

The hybrid strategy (H-S) applies E-S at the root node and returns to baseline selection during diving. It therefore keeps a compact root-node pool while replenishing alternatives when the problem becomes more constrained. Its adjusted-objective increases are smaller than under E-S, and its uncovered-task counts remain close to B-S. Total full-day time stays near the baseline, at -0.54% on the three-base instance and +5.12% on the six-base instance.

## 6.3 Learning to imitate the expert

The supervised GNN model can imitate expert selection decisions in the NS setting with a test balanced accuracy of 78.5% and a test AUC-PR of 0.580. The improvement over plain logistic regression is substantial. AUC-PR rises from 0.164 to 0.580. Most of this gain is already captured by the simplest single-layer GNN. This confirms that the bipartite graph structure contains information that column features alone cannot encode.

The graph represents the column-constraint relationships in the LP directly. Column nodes connect to the constraint nodes they participate in. Each constraint aggregates information about all columns competing for the same task or crew assignment. The model can therefore distinguish whether a duty covers a hard-to-fill constraint, faces heavy competition, or would relieve a tight assignment situation. Logistic regression, which

evaluates each column independently, cannot capture this relational context. Two duties with similar features and reduced costs can look indistinguishable to logistic regression. One may cover a nearly uncovered task with a high dual value while the other covers an abundantly covered task with a near-zero dual. The GNN resolves this ambiguity through message-passing.

Performance improves with message-passing depth through the four layers evaluated. The attention and multi-aggregation variants provide modest additional improvement at higher inference latency, although all predictions still require only tens of milliseconds. The four-layer mean model was selected as a practical starting point for tuning. The more elaborate architectures were not tuned or evaluated inside the solver, so their modest validation gains may still lead to useful in-loop behavior.

The model generalizes well to the held-out test period. Train and test metrics remain close despite the months separating them. This is practically important because fairness histories evolve continuously. The model must therefore make reliable predictions across fairness states that differ from the training distribution. The stability of recall across planning days supports the conclusion that the model has learned structural properties of the column-selection problem rather than memorizing patterns tied to specific dates or histories. The transfer experiment on the six-base instance provides a stronger test of this conclusion.

A balanced accuracy of 78.5% does not imply that every disagreement with the expert produces an unusable selection. The expert MILP itself often faces several near-equivalent columns competing for the same assignment slots. A slightly different column from the same template can still be structurally useful even though it counts as a binary mismatch with the expert label. The in-loop analysis in the next section confirm that imperfect imitation can be beneficial in this setting.

## 6.4 Computational efficiency of the learned selector

The GNN selector (G-S) achieves a 4.65% reduction in total root-node time while keeping the objective increase to 0.89% above the baseline. It also reduces the final root-node RMP by 64.35%. The expert selector, by contrast, raises total root-node time by 26.98% due to its larger iteration increase of 75.61% versus 17.48% for G-S. The model's imperfect imitation passes through somewhat more columns than the strict MILP minimum. This produces more meaningful dual movement per iteration. It also steers per-template pricing toward new candidates rather than cycling on the same rejected ones. Each extra column beyond the minimum contributes to the dual shift that can break the cycling pattern described earlier.

The transfer to the six-base instance strengthens the evidence that the GNN learns structural selection behavior. Without retraining, G-S reduces the final root-node RMP by 62.50% and time per iteration by 23.51%. Its iteration count is only 9.14% above B-S, compared with 75.04% for E-S. Total root-node time consequently falls by 16.52%, while the objective remains 0.84% above B-S. Together with the held-out test-period results, this transfer shows that the model has learned more than instance-specific patterns. The larger time saving than on the three-base instance arises because G-S stays closer to the baseline iteration count while still keeping a substantially smaller RMP. That again reflects a less minimal selection than the strict expert, now visible at larger scale.

The full-day results show that this computational benefit persists after the procedure returns to baseline selection during diving. H-G-S reduces total time by 10.11% on the three-base instance and by 16.27% on the six-base instance. Time per iteration decreases by 20.19% and 21.43%, respectively. The number of iterations remains only 12.91% and 6.52% above B-S. These runtime reductions are the defensible full-day result. Full-day quality remains path-dependent. On the three-base instance, H-G-S leaves six fewer tasks uncovered than B-S and reduces the adjusted objective by 0.12%. On the six-base instance these differences reverse. H-G-S leaves 19 more tasks uncovered and raises the adjusted objective by 1.73%. The selectors also differ in uncovered tasks on most individual days. Root-node quality remains the controlled comparison. There G-S stays close to B-S on both instances.

The prediction threshold analysis provides further insight into the trade-off between how many columns the model passes to the solver and overall solver efficiency. Raising the threshold passes fewer columns to the solver. This reduces RMP size and time per iteration. Iteration count rises at the same time, following the same trade-off between a smaller inserted column set and slower overall progress that was argued for sparse expert selection earlier. Total time therefore does not improve monotonically. The minimum is achieved around thresholds 0.3 and 0.5. At threshold 0.9, individual iterations are fast, but iteration count and the root-node objective increase sharply. The 26.76% objective increase matches the stalling pattern of the basic MILP in the configuration comparison, confirming that the same sparse-selection mechanism is at work. The optimal threshold is likely to vary by instance and planning period. Full-day quality also shows no clear trend with the threshold. Thresholds from 0.1 to 0.7 produce between 217 and 224 uncovered tasks despite steadily changing how many columns are passed to the solver. This sensitivity amplifies the path-dependence argument. Small root-node differences can lead to different diving paths and different end-of-day coverage. The results nevertheless identify a middle range in which the GNN reduces computation time while keeping the root-node objective close to B-S.

## 6.5 Limitations and future research

The most fundamental limitation of this study is that it does not resolve the iteration trade-off. It demonstrates only that the learned selector handles it better than the strict MILP minimum. Neither the expert nor the current GNN provides an explicit mechanism to detect and respond to template-level stagnation. An adaptive column injection policy could address this, for example by extending the expert so that stuck templates receive additional columns while the remaining selection stays minimal. The same idea can be realized by temporarily broadening the inserted set when pricing progress stalls, which would restore dual movement without abandoning sparse selection altogether. The threshold results already hint at such a mechanism. At threshold 0.5, the GNN effectively implements a soft form of adaptive injection. It lets through more columns when prediction confidence is spread across many candidates. The six-base transfer results support this direction. There, G-S admits more columns than the strict expert, stays much closer to the baseline iteration count, and still substantially reduces the RMP.

A second limitation is that all experiments are conducted in the heuristic prototype solver described. Unlike the case studies of Morabit et al. (2021), the NS procedure

terminates the root node through early stopping rather than to a proven LP optimum. The main performance comparison also retain the task-disjoint filter as a common pre-filter. Within that setting, the primary analysis and the training of the learned model are restricted to root-node iterations. Keeping the expert active throughout the full planning day already shows the cost of the thin pool after diving. Hybrid strategies offer a practical extension to the full day by returning to the baseline after the first dive. Training or adapting a selector for the post-dive states themselves remains future work.

Several other directions for future research follow from the results. Dual stabilization is a classical response to degeneracy in large master problems, and it was not evaluated here. Selective column insertion already tries to reduce that degeneracy by keeping the RMP much smaller, but stabilization remains relevant for the baseline. For sparse selection, the open issue is the other side of the trade-off. The higher iteration counts under sparse selection, together with the set-covering principle that over-covered task constraints carry a zero dual, suggest that thin pools weaken the pricing signal. Dual stabilization is designed to dampen oscillating duals. It cannot create a pricing signal for constraints that have positive surplus and therefore a true zero dual. A small penalty on over-coverage is one way to keep more task constraints competitive under sparse selection.

The learning task could also be extended beyond single-step column classification. The current model cannot reason about how the current column pool affects the trajectory over subsequent iterations. It also cannot reason about the quality of the final integer solution. A reinforcement learning formulation could be rewarded on total root-node time or full-day solution quality. Such a policy could learn strategic trade-offs across iterations. This thesis provides a necessary foundation by demonstrating that a GNN can represent the relevant problem structure and produce useful per-iteration predictions.

Another direction is dual value prediction. If a model predicts the dual vector accurately enough, pricing can use those predictions and some RMP solves can be skipped. The bipartite graph logs already contain the required inputs, and the duals returned by the LP solver provide continuous regression targets. This is a harder task than binary column classification. The same predictions could also support learned dual stabilization in the baseline setting.

# Chapter 7

# Conclusion

This thesis investigated whether a machine-learning-based column-selection framework can be transferred to the NS crew planning problem with fairness over time. It also examined whether the resulting approach can improve the computational efficiency of the daily column generation while preserving solution quality. The investigation produced four main findings.

The expert MILP can be formulated and successfully applied in the NS setting, but only with structural adaptations tailored to the NS problem. The per-template pricing decomposition, which arises from the fairness-over-time feedback mechanism, demands that the expert's safety mechanism operates at the template level rather than globally. Without such a mechanism, templates can receive no new columns and the procedure stalls. MILP-TC prevents this by requiring coverage of every active template in the expert optimization problem. It provides a consistent target for supervised learning and reduces the root-node RMP by 77.29% on the three-base instance and 78.49% on the six-base instance. The corresponding objective increases remain limited to 1.66% and 3.04%.

The strict expert is not an effective deployed selector by itself. Its smaller RMP makes individual iterations faster, but the number of iterations rises enough to increase total root-node time. This trade-off is consistent with sparse insertion producing little dual movement and causing pricing to regenerate rejected duties. The NS set-covering formulation strengthens this mechanism. Over-covered tasks carry a true zero dual, so they drop out of the pricing signal. Per-template pricing deepens the effect, because a stuck template cannot be rescued by progress elsewhere. The expert thus reveals both the amount of removable redundancy and the risk of reducing the column pool too aggressively.

A graph neural network trained on expert-labeled data can imitate the expert's column-selection decisions with a test balanced accuracy of 78.5% and a test AUC-PR of 0.580. The improvement over logistic regression is large and arises primarily from the bipartite graph representation. By encoding the relationships between duties and constraints directly in the network architecture, the GNN can represent the task coverage, crew assignment, and template-specific structure of the fairness-over-time planning problem. This information is unavailable to a model that evaluates each column independently. Stable performance across held-out months and varying fairness histories indicates that the network learns structural properties rather than patterns tied to specific planning

days. Its successful transfer from the three-base training instance to the six-base instance without retraining provides stronger evidence of this generalization.

The learned selector achieves a better balance. At the root node, G-S reduces total time by 4.65% on the three-base instance and by 16.52% on the six-base instance. The corresponding RMP reductions are 64.35% and 62.50%, while the objectives remain 0.89% and 0.84% above B-S. The larger time saving on the six-base instance arises because G-S stays closer to the baseline iteration count. The GNN admits more columns than the strict expert and therefore retains more of the dual movement needed for progress. The threshold results confirm that the best computational balance lies in a middle range. Selecting too few columns reduces time per iteration but eventually causes the iteration count and root-node objective to rise sharply.

The hybrid learned selector extends these savings over the full planning day by using G-S at the root node and returning to B-S after the first dive. It reduces total time by 10.11% on the three-base instance and by 16.27% on the six-base instance. These runtime reductions are the defensible full-day result. Final solution quality cannot be used to rank the selectors because different root-node column pools lead to different diving paths. On the three-base instance H-G-S ends with slightly fewer uncovered tasks and a slightly lower adjusted objective than B-S. On the six-base instance both differences reverse. Threshold sensitivity shows the same non-monotonic variation in uncovered tasks. These outcomes describe the realized trajectories, not a causal quality advantage of one selector. Root-node quality remains the controlled comparison.

For NS, three practical implications follow. First, use the expert primarily to generate labels and to expose structural weaknesses of sparse selection, rather than deploying its minimal selection as the operational selector. Even if that selection trade-off were more favorable, repeatedly solving the expert MILP would remain too costly for routine use. Second, prefer the learned selector at the root node. It keeps a much smaller RMP, reduces time per iteration, and improves total root-node runtime while remaining close in objective. Third, if the selector is extended beyond the root node, the hybrid strategy is the safer full-day setup. It preserves the root-node time savings without requiring the model to operate outside its training setting. Full-day solution quality should not be treated as the decision criterion here, because of path dependence. Deployment should retain a moderate prediction threshold and monitor iteration growth rather than optimizing only RMP size.

More broadly, the study shows that machine learning can support a repeated decision inside a combinatorial optimization algorithm without replacing the optimization procedure itself. A graph representation makes this possible in an NS crew planning problem where duty value depends on task coverage, crew assignment, and fairness history. All main performance findings were obtained in the heuristic prototype solver with early stopping, diving, and the task-disjoint pre-filter. Extending training to post-dive states and testing under proven LP convergence remain open. The most direct next step is adaptive column injection, which can add columns when template-level progress stalls while retaining sparse selection elsewhere. Dual prediction, learned stabilization, and longer-horizon learning provide further research directions.

# References


E. Abbink, M. Fischetti, L. Kroon, G. Timmer, and M. Vromans. Reinventing Crew Scheduling at Netherlands Railways. *Interfaces*, 35(5):393–401, 2005. URL `http://www.jstor.org/stable/20141324`.

E. Abbink, L. Albino, T. Dollevoet, D. Huisman, J. Roussado, and R. Saldanha. Solving Large Scale Crew Scheduling Problems in Practice. *Public Transport*, 3(2):149–164, 2011. doi: 10.1007/s12469-011-0045-x.

E. Abbink, D. Huisman, and L. Kroon. *Railway Crew Management*. International Series in Operations Research & Management Science. Springer-Verlag, 2018. doi: 10.1007/978-3-319-72153-8_11.

Y. Bengio, A. Lodi, and A. Prouvost. Machine Learning for Combinatorial Optimization: A Methodological Tour d'Horizon. *CoRR*, abs/1811.06128, 2018. URL `http://arxiv.org/abs/1811.06128`.

C. Bishop. *Pattern Recognition and Machine Learning*, volume 16, pages 140–155. 2006. doi: 10.1117/1.2819119.

R. Borndörfer, T. Klug, L. Lamorgese, C. Mannino, M. Reuther, and T. Schlechte, editors. *Handbook of Optimization in the Railway Industry*. International Series in Operations Research and Management Science. Springer, 2018. doi: 10.1007/978-3-319-72153-8.

T. Breugem, B. van Rossum, T. Dollevoet, and D. Huisman. A Column Generation Approach for the Integrated Crew Re-Planning Problem. *Omega*, 107:102555, 2021. doi: 10.1016/j.omega.2021.102555.

T. Breugem, T. Dollevoet, and D. Huisman. Is Equality Always Desirable? Analyzing the Trade-Off Between Fairness and Attractiveness in Crew Rostering. *Management Science*, 68:2619–2641, 2022. doi: 10.1287/mnsc.2021.4005.

T. Breugem, T. Schlechte, C. Schulz, and R. Borndörfer. A Three-Phase Heuristic for the Fairness-Oriented Crew Rostering Problem. *Computers & Operations Research*, 154: 106186, 2023. doi: 10.1016/j.cor.2023.106186.

S. Brody, U. Alon, and E. Yahav. How Attentive Are Graph Attention Networks?, 2022. URL `https://arxiv.org/abs/2105.14491`.

Q. Cappart, D. Chételat, E. Khalil, A. Lodi, C. Morris, and P. Veličković. Combinatorial Optimization and Reasoning With Graph Neural Networks, 2022. URL `https://arxiv.org/abs/2102.09544`.

A. Caprara, M. Fischetti, P. Toth, D. Vigo, and P. Guida. Algorithms for Railway Crew Management. *Mathematical Programming*, 79, 1997. doi: 10.1007/BF02614314.

V. Chen and J. Hooker. A Guide to Formulating Fairness in an Optimization Model. *Annals of Operations Research*, 326:581–619, 2023. doi: 10.1007/s10479-023-05264-y.

C. Chi, A. Aboussalah, E. Khalil, J. Wang, and Z. Sherkat-Masoumi. A Deep Reinforcement Learning Framework for Column Generation, 2023. URL `https://arxiv.org/abs/2206.02568`.

D. Cox. The Regression Analysis of Binary Sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 20(2):215–232, 1958. doi: 10.1111/j.2517-6161.1958.tb00292.x.

G. Desaulniers, J. Desrosiers, and M. Solomon, editors. *Column Generation*. Springer Books. Springer, 2005. doi: 10.1007/b135457.

J. Desrosiers and M. Lübbecke. *A Primer in Column Generation*, pages 1–32. 2006. doi: 10.1007/0-387-25486-2_1.

I. Elhallaoui, D. Villeneuve, F. Soumis, and G. Desaulniers. Dynamic Aggregation of Set-Partitioning Constraints in Column Generation. *Operations Research*, 53:632–645, 2005. doi: 10.1287/opre.1050.0222.

A. Ernst, H. Jiang, M. Krishnamoorthy, H. Nott, and D. Sier. An Integrated Optimization Model for Train Crew Management. *Annals of Operations Research*, 108:211–224, 2001. doi: 10.1023/A:1016019314196.

A. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier. Staff Scheduling and Rostering: A Review of Applications, Methods and Models. *European Journal of Operational Research*, 153(1):3–27, 2004. doi: 10.1016/S0377-2217(03)00095-X.

M. Fey and J. Lenssen. Fast Graph Representation Learning With PyTorch Geometric, 2019. URL `https://arxiv.org/abs/1903.02428`.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. URL `http://www.deeplearningbook.org`.

W. Hamilton, R. Ying, and J. Leskovec. Inductive Representation Learning on Large Graphs, 2018. URL `https://arxiv.org/abs/1706.02216`.

J. Heil, K. Hoffmann, and U. Buscher. Railway Crew Scheduling: Models, Methods and Applications. *European Journal of Operational Research*, 283, 2019. doi: 10.1016/j.ejor.2019.06.016.

A. Hoerl and R. Kennard. Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics*, 12(1):55–67, 1970. doi: 10.1080/00401706.1970.10488634.

D. Huisman. A Column Generation Approach for the Rail Crew Re-Scheduling Problem. *European Journal of Operational Research*, 180:163–173, 2007. doi: 10.1016/j.ejor.2006.04.026.

G. James, D. Witten, T. Hastie, R. Tibshirani, and J. Taylor. *An Introduction to Statistical Learning: With Applications in Python*. Springer Texts in Statistics. Springer International Publishing, 2023. URL https://books.google.nl/books?id=bQgD0AEACAAJ.

Ö. Karsu and A. Morton. Inequity Averse Optimization in Operational Research. *European Journal of Operational Research*, 173, 2015. doi: 10.1016/j.ejor.2015.02.035.

D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization, 2017. URL https://arxiv.org/abs/1412.6980.

S. Kraul, M. Seizinger, and J. Brunner. Machine Learning–Supported Prediction of Dual Variables for the Cutting Stock Problem With an Application in Stabilized Column Generation. *INFORMS Journal on Computing*, 35, 2023. doi: 10.1287/ijoc.2023.1277.

A. Lodi, P. Olivier, G. Pesant, and S. Sankaranarayanan. Fairness Over Time in Dynamic Resource Allocation With an Application in Healthcare. *Mathematical Programming*, 203:285–318, 2022. doi: 10.1007/s10107-022-01904-6.

A. Lodi, S. Sankaranarayanan, and G. Wang. A Framework for Fair Decision-Making Over Time With Time-Invariant Utilities. *European Journal of Operational Research*, 319, 2023. doi: 10.1016/j.ejor.2023.11.030.

M. Lübbecke and J. Desrosiers. Selected Topics in Column Generation. *Operations Research*, 53:1007–1036, 2005. doi: 10.1287/opre.1050.0234.

L. Mertens, L. Wolbeck, D. Rößler, L. Xie, and N. Kliewer. An Overview of Optimization Approaches for Scheduling and Rostering Resources in Public Transportation, 2023. URL https://arxiv.org/abs/2310.13425.

M. Mesquita, M. Moz, A. Paias, and M. Pato. A Decomposition Approach for the Integrated Vehicle-Crew-Roster Problem With Days-Off Pattern. *European Journal of Operational Research*, 229:318–331, 2013. doi: 10.1016/j.ejor.2013.02.055.

M. Morabit, G. Desaulniers, and A. Lodi. Machine-Learning–Based Column Selection for Column Generation. *Transportation Science*, 55, 2021. doi: 10.1287/trsc.2021.1045.

Y. Muroi, T. Nishi, and M. Inuiguchi. Improvement of Column Generation Method for Railway Crew Scheduling Problems. *IEEJ Transactions on Electronics, Information and Systems*, 130:275–283, 2010. doi: 10.1541/ieejeiss.130.275.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library, 2019. URL https://arxiv.org/abs/1912.01703.

I. Patterson. NSSM: The Non-Sucking Service Manager, 2023. URL https://nssm.cc.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, A. Müller, J. Nothman, G. Louppe, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay. Scikit-Learn: Machine Learning in Python, 2018. URL https://arxiv.org/abs/1201.0490.

D. Potthoff, D. Huisman, and G. Desaulniers. Column Generation With Dynamic Duty Selection for Railway Crew Rescheduling. *Transportation Science*, 44:493–505, 2010. doi: 10.2307/25769516.

D. Rumelhart, G. Hinton, and R. Williams. Learning Representations by Back-Propagating Errors. *Nature*, 323(6088):533–536, 1986.

F. Scarselli, M. Gori, A. Tsoi, M. Hagenbuchner, and G. Monfardini. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

Y. Shen, Y. Sun, X. Li, Z. Cao, A. Eberhard, and G. Zhang. Adaptive Stabilization Based on Machine Learning for Column Generation, 2024. URL `https://arxiv.org/abs/2405.11198`.

B. van Rossum. *Optimising Fair Work Allocation: Applications in Railway Crew Planning*. PhD thesis, Erasmus University Rotterdam, 2025.

B. van Rossum, R. Chen, and A. Lodi. Optimizing Fairness Over Time With Homogeneous Workers. In *23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023)*, volume 115, pages 17:1–17:6. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi: 10.4230/OASIcs.ATMOS.2023.17.

B. van Rossum, T. Dollevoet, and D. Huisman. Railway Crew Planning With Fairness Over Time. *European Journal of Operational Research*, 318(1):55–70, 2024. doi: 10.1016/j.ejor.2024.04.029.

B. van Rossum, W. van Dijk, T. Dollevoet, D. Huisman, M. Siemann, and J. van 't Wout. Simulating a New Crew Planning Process at Netherlands Railways. *Journal of Rail Transport Planning & Management*, 38:100584, 2026. doi: 10.1016/j.jrtpm.2026.100584.

S. Weider, C. Schulz, R. Borndörfer, and S. Seidl. Integration of Duty Scheduling and Rostering to Increase Driver Satisfaction, 2015.

L. Wolbeck. Fairness Aspects in Personnel Scheduling, 2019.

H. Yuan, L. Fang, and S. Song. A Reinforcement-Learning-Based Multiple-Column Selection Strategy for Column Generation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(8):8209–8216, 2024. doi: 10.1609/aaai.v38i8.28661.

# Appendix A

# Preliminaries

This chapter introduces the fundamental concepts that form the basis of the methodology developed in this thesis. It is intended to provide a concise and accessible overview of the main techniques. The chapter begins with linear programming and column generation. These form the optimization framework. Then it gives a brief introduction to machine learning, covering supervised learning, binary classification, and neural networks.

## A.1 Column generation

Column generation is designed for linear programs with an extremely large number of variables, such as those arising in routing, scheduling, or planning problems. In these applications, each variable can correspond to a feasible plan or route, leading to an exponential number of possibilities. Enumerating all variables explicitly is computationally infeasible.

To address this, the full linear program is referred to as the master problem (MP), which contains all possible variables. In practice, a restricted master problem (RMP) is solved that contains only a small subset of variables. This initial subset is typically chosen to ensure that the RMP has a feasible solution, often generated through a simple heuristic or by including artificial variables. Additional variables, called columns, are generated iteratively through a pricing subproblem. The dual variables of the RMP provide insight into which columns could potentially improve the objective. This decomposition into RMP and subproblems allows large-scale problems to be solved efficiently without enumerating all variables.

Conceptually, column generation does not create new decisions, but gradually reveals variables that were already part of the full problem but not yet included. Adding a column introduces an additional degree of freedom in the primal problem. From the dual perspective, each primal variable corresponds to a dual constraint. Hence, adding a column to the RMP also adds a previously omitted dual constraint. The method can therefore be viewed equivalently as expanding the primal model or tightening the dual model. A comprehensive overview of the method and its applications can be found in Desaulniers et al. (2005) and Desrosiers and Lübbecke (2006).

### A.1.1 Linear programming and duality

Linear programming provides a mathematical framework for modeling decision problems with linear objectives and constraints. A standard linear program can be written down as shown in Equation A.1.

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & c^\top x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned} \tag{A.1}$$

where $x \in \mathbb{R}^n$ denotes the decision variables, $c \in \mathbb{R}^n$ the cost vector, $A \in \mathbb{R}^{m \times n}$ the constraint matrix, and $b \in \mathbb{R}^m$ the right-hand side of the constraints. The feasible region defined by these constraints forms a convex polyhedron, and optimal solutions occur at its extreme points.

Every linear program, which is called the primal, has an associated dual problem. The dual provides a complementary perspective on the same optimization task by assigning a value to each constraint. This value indicates how the objective would change if that constraint were relaxed. The dual for the linear program above is presented in Equation A.2.

$$\begin{aligned} \max_{\pi \in \mathbb{R}^m} \quad & b^\top \pi \\ \text{s.t.} \quad & A^\top \pi \leq c \end{aligned} \tag{A.2}$$

The dual variables $\pi$, often called shadow prices, measure the sensitivity of the objective to changes in the constraints. For instance, in a minimization problem, if $\pi_1 = 5$ for the first constraint, increasing the resource $b_1$ by one unit would reduce the total cost by 5 units. On the other hand, decreasing it by one unit would increase the cost by 5. This provides a concrete interpretation of the value of each constraint.

Strong duality ensures that, under mild conditions, the optimal values of the primal and dual coincide. In other words, $c^\top x^* = b^\top \pi^*$. At such an optimum, complementary slackness relates each primal inequality constraint to its own dual variable. If a $\leq$ constraint has positive slack (unused right-hand side $b$), or a $\geq$ constraint has positive surplus (excess over the right-hand side $b$), then the dual of that constraint must be zero. In the other direction, a nonzero dual is possible only if the constraint is binding. That implication does not reverse, since a binding constraint may still have a zero dual when the optimum is degenerate. Equality constraints have no slack or surplus, so they are not forced to a zero dual by this rule. These dual values are the inputs to the pricing step discussed next.

### A.1.2 Pricing and reduced costs

A key concept in column generation is the reduced cost. The reduced cost of a variable indicates whether the introduction of this variable into the current solution could improve the objective value. For a minimization problem, only variables with negative reduced cost are of interest.

Given dual variables $\pi$ associated with the constraints of the RMP, the reduced cost of a column $j$ with cost $c_j$ and column vector $a_j$ is defined in Equation A.3.

$$\bar{c}_j = c_j - \pi^\top a_j \tag{A.3}$$

The column vector $a_j$ describes how a variable interacts with the constraints of the master problem. In many scheduling and routing applications, its entries are binary and indicate whether a column contributes to a given constraint. More generally, this column vector represents the extent to which a variable satisfies each constraint. Combined with the dual variables, the term $\pi^\top a_j$ measures the total value of the constraints addressed by the column. A nonzero dual is possible only for a binding constraint and indicates that this constraint is among the most restrictive parts of the current solution. By complementary slackness, a constraint with positive slack or surplus carries a zero dual and does not contribute to the reduced cost.

Importantly, even when a constraint is binding, adding a new column can improve the objective without changing the constraint's right-hand side, $b$. The constraint remains satisfied, but the way it is satisfied can change. The dual value reflects the marginal cost of satisfying that constraint given the current set of variables. A column with a negative reduced cost reveals that this cost can be reduced. Equivalently, a negative reduced cost indicates that the corresponding dual constraint is violated. This means that the dual value of the covered constraints exceeds the cost of the column.

The pricing subproblem is responsible for identifying such columns. Rather than enumerating all possible variables, it solves Equation A.4.

$$\min_{a \in \mathcal{A}} \; c(a) - \pi^\top a \tag{A.4}$$

The $\mathcal{A}$ denotes the set of all feasible columns and $c(a)$ represents the cost associated with generating column $a$. If the optimal value is negative, an improving column is found and added to the RMP. If no such column exists, then no dual constraint is violated. By strong duality, the current solution is optimal for the full master problem.

The structure of $\mathcal{A}$ allows the pricing problem to be solved efficiently using specialized algorithms, such as shortest path problems with resource constraints in routing and scheduling settings.

### A.1.3 Illustrative example

Consider a scheduling setting in which each column represents a feasible sequence of tasks. After solving the RMP, tasks that are difficult to cover correspond to constraints with high dual values. A sequence that includes such tasks becomes attractive if it provides a cheaper way of covering them. The pricing problem constructs such sequences while respecting feasibility requirements.

## A.2 Machine learning

Machine learning studies methods that learn patterns from data. Instead of specifying all decision rules by hand, a model is fitted from examples and then used to make predictions on new observations. General introductions to the field are given by Bishop (2006); James et al. (2023); Goodfellow et al. (2016). These references cover both the statistical foundations and the neural network methods that underlie many modern applications.

### A.2.1 Learning settings

Machine learning methods are commonly divided into supervised learning, unsupervised learning, and reinforcement learning.

In supervised learning, each training example consists of an input together with its desired output. The objective is to learn a mapping from inputs to outputs that generalizes well to unseen data. A simple example is predicting whether an email is spam based on its text, where historical emails provide both the text and the correct label.

In unsupervised learning, no output labels are available. Instead, its objective is to discover patterns or structures within the data itself. For example, a retailer may group customers based on purchasing behavior without knowing beforehand which groups should exist.

In reinforcement learning, an agent interacts with an environment and learns by receiving rewards for its actions. The objective is to maximize the cumulative reward over time. A common example is a game-playing agent that learns which actions lead to a high final score.

### A.2.2 Supervised learning and classification

In supervised learning, the training data consist of input-output pairs $(x_i, y_i)$, where $x_i$ denotes the input features and $y_i$ denotes the observed output. The objective is to learn a function $f$ that maps inputs to predictions. If the output variable is continuous, the problem is called regression. Predicting travel time or cost is a typical example of a regression problem. If the output is a categorical label, the problem is called classification. Binary classification is the special case with only two classes, commonly denoted by 0 and 1.

A classifier produces a prediction for the class of each observation. In binary classification, this prediction is often represented as a probability $\hat{p}_i \in (0, 1)$ that an observation belongs to class 1. A threshold can then be applied to convert this probability into a class prediction. A standard loss function for binary classification is the binary cross-entropy (BCE) loss, which can be seen in Equation A.5.

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right] \tag{A.5}$$

where $y_i \in \{0, 1\}$ is the true label and $\hat{y}_i$ is the predicted probability. The loss is minimized when the model assigns high probability to the correct class and large penalties are incurred for confident incorrect predictions. During training, the model parameters are adjusted to minimize this loss. An example of a binary classifier is logistic regression, which uses a linear combination of the input features followed by a sigmoid function to produce a probability estimate.

Good performance on the training data alone is not sufficient. A model may memorize the training examples and perform poorly on unseen data. This phenomenon is known as overfitting. A common approach to detect overfitting is to monitor performance on a validation set that is not used for parameter updates. If the training loss continues to decrease while validation performance no longer improves, the model is likely beginning to overfit.

### A.2.3 Neural networks

A neural network is a parametric model composed of layers of simple computational units called neurons. Each neuron receives inputs, forms a weighted sum, adds a bias term, and applies a nonlinear activation function. For an input vector $x$, the operation performed by a neuron can be expressed as shown in Equation A.6.

$$a = \phi(w^\top x + b) \tag{A.6}$$

where $w$ denotes the weight vector and $b$ indicates the bias. The term $\phi$ represents the activation function, which enables the network to learn nonlinear relationships. Without nonlinear activations, stacking multiple layers would still be equivalent to a single linear transformation. Common activation functions include the rectified linear unit (ReLU) and the sigmoid function, provided in Equation A.7 and Equation A.8, respectively.

$$\mathrm{ReLU}(z) = \max(0, z) \tag{A.7}$$

$$\sigma(z) = \frac{1}{1 + \exp(-z)} \tag{A.8}$$

Figure A.1 shows a simple feedforward neural network. The input layer receives the features, the hidden layer creates intermediate representations, and the output layer produces the prediction. In binary classification, the final output is often passed through a sigmoid function to be able to interpret it as a probability.

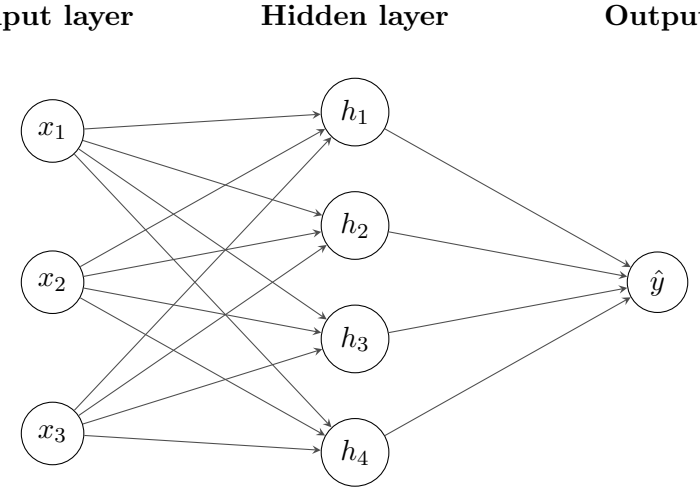



**Figure A.1:** Architecture of a standard feedforward neural network. The network comprises an input layer with three features, a single hidden layer with four nodes, and an output node for the final prediction. Each directed edge represents a learnable weight parameter that transforms the signal as it propagates through the network.

The parameters of a neural network are learned by minimizing a loss function. Backpropagation, introduced by Rumelhart et al. (1986), computes the gradient of the loss with respect to each model parameter by applying the chain rule through the network layers. These gradients are then used by an optimization algorithm to update the parameters. The Adam optimizer proposed by Kingma and Ba (2017) is a widely used example

because it adapts the update step separately for different parameters.

Neural networks can represent highly flexible nonlinear functions, but this flexibility also increases the risk of overfitting. Regularization methods are therefore commonly used. Dropout randomly deactivates hidden units during training, which discourages the model from relying too strongly on a small subset of units. Early stopping terminates training when validation performance no longer improves. The feedforward neural network architecture can be extended to data with more complex structures. For graph-structured data, graph neural networks replace conventional layers with message-passing operations that allow information to be exchanged between connected nodes.