Transportation Research Part E 209 (2026) 104764

Contents lists available at ScienceDirect

Transportation Research Part E

journal homepage: www.elsevier.com/locate/tre

Rapid re-optimization via learning-enhanced column generation for vehicle routing with driver break scheduling

Ning Xue [a], Tianxiang Cui [a,*], Shi Cheng [b]

[a] School of Computer Science, University of Nottingham, Ningbo China 199 Taikang East Road Ningbo, 315100, Zhejiang, China
[b] School of Computer Science, Shaanxi Normal University, 710119, Xi'an, China

ARTICLE INFO

Keywords:
Vehicle routing
Driver break scheduling
Hours of service regulations
Heuristics
Machine learning

ABSTRACT

In many parts of the world, the road freight transportation sector is subject to stringent legal requirements regarding driver hours. These regulations present a significant challenge for developing practical and efficient schedules. The simultaneous optimization of vehicle routing and driver break schedules constitutes a major computational problem. In practice, many real-life vehicle routing problems require periodic planning and must adapt to sudden demand changes; this necessitates efficient re-optimization capabilities. This paper addresses this need by proposing a rapid re-optimization framework for the Vehicle Routing with Driver Break Scheduling. Our method integrates a column generation algorithm with a machine learning heuristic specifically designed for fast re-optimization. We evaluate the proposed approach under European Union Regulation (EC)561/2006 and Directive 2002/15/EC using two sets of benchmark instances, one based on a synthetic data and the other on a real-life data. The results demonstrate that the ML-enhanced approach is substantially faster than the implementation without the ML prediction component, reducing runtime by more than 96% (to just a matter of seconds), while increasing routing cost by less than 2% and yielding a final solution gap of 2.7% above the estimated lower bound.

1. Introduction

The regulation of driving and working hours for truck and public transportation drivers is a critical component of public policy aimed at enhancing urban mobility and ensuring the safety of passengers and freight transport. These regulations, which are enforced in numerous jurisdictions worldwide, are designed to mitigate the risks associated with driver fatigue, thereby promoting safer and more efficient transportation systems. For instance, in the European Union and Great Britain, regulations such as (EC)561/2006 and 2002/15/EC establish stringent limits on driving hours and mandate rest periods. Similarly, in Australia, frameworks such as Standard Hours, Basic Fatigue Management, and Advanced Fatigue Management are implemented to regulate driver fatigue. New Zealand's Work Time Requirements of the Road Code and the United States' Hours of Service Regulations also impose similar guidelines, with significant penalties for non-compliance, underscoring the global emphasis on safety and operational efficiency. In China, there are specific regulations governing rest requirements for long-distance passenger vehicles, tourist coaches, and freight trucks. For example, drivers must take at least a 20-min break after driving continuously for more than four hours. Such measures are integral to maintaining driver alertness and reducing the likelihood of accidents, thereby contributing to safer urban mobility.

* Corresponding author.
E-mail address: tianxiang.cui@nottingham.edu.cn (T. Cui).

https://doi.org/10.1016/j.tre.2026.104764
Received 24 September 2025; Received in revised form 12 January 2026; Accepted 10 February 2026
Available online 16 February 2026

Simultaneously addressing the routing problem and optimizing driver break schedules presents a significant computational challenge. Under Regulation (EC)561/2006, driver breaks are categorized into two parts: a first 15-min break followed by a subsequent 30-min break. If break division is permitted, a single trip may necessitate as many as four breaks, raising the number of possible scheduling combinations to $\mathcal{O}(n^4)$, where n represents the number of visits on the route. This complexity is compounded by the requirement to align with Directive 2002/15/EC on working time, as break schedules must satisfy both regulatory frameworks.

In this paper, both Regulation (EC)561/2006 and Directive 2002/15/EC, limited to a single day of operation, serve as the regulatory framework for the case study, due to the short-term nature of urban logistics planning which typically involves next-day route optimization, especially for the sake of the requirement of re-optimization, which usually limits the planning horizon even within a few hours. However, the algorithm proposed in this paper is designed to be adaptable to other regulations as well. Simply put, Regulation (EC)561/2006 establishes standards for driving times and mandatory breaks for drivers, whereas Directive 2002/15/EC regulates working time requirements, including the mandatory breaks stipulated therein. In other regions, similar regulations have different names. For clarity and generality, in this paper, the term *driving breaks* regulation refers to the break requirements under Regulation (EC)561/2006, while *duty breaks* regulation denotes the break obligations under Directive 2002/15/EC. Generally speaking, driving break regulations set standards for driving times and mandatory breaks. For a single day's schedule, these regulations specify a maximum daily driving time of 9 h and require breaks totaling at least 45 min after every 4.5 h of driving, with options for splitting the break into smaller intervals. Duty break regulations govern working time for drivers, including both driving and other work-related activities. They mandate breaks of at least 15 min after 6 h of work, 30 min for work periods between 6 to 9 h, and 45 min for work exceeding 9 h. Further details regarding these regulations are provided in Xue et al. (2025a).

Building upon traditional routing problems like the Pickup and Delivery Problem (PDP), the Dial-a-Ride Problem (DARP), and the Vehicle Routing Problem (VRP), the Generalized Pickup and Delivery Problem (GPDP) (Savelsbergh and Sol, 1995) offers a more realistic framework for modeling logistics operations. Unlike these simpler variants, which assume fixed depots, the GPDP handles scenarios where vehicles may have diverse starting and ending locations, and where loads may require multiple pickup or delivery points (Gómez-Lagos et al., 2022). The aim is to find routing solutions that meet all transportation needs while keeping each load intact. The GPDP's ability to handle complex constraints makes it more realistic for practical applications, though more computationally challenging than conventional routing problems (Wang et al., 2023).

Many GPDPs and driver break scheduling problems require repetitive or periodic arrangements, as seen across various industries. For instance, supermarkets and convenience stores regularly receive goods from suppliers, while breweries deliver fresh beer to bars on fixed schedules (e.g., specific weekdays) and collect used barrels for recycling. In e-commerce, companies like Amazon handle recurring deliveries of pre-ordered items, and fast-food chains frequently promote and distribute their products. Similarly, bike-sharing companies regularly relocate bikes to different points to meet demand. These use cases demonstrate that repeatedly solving the GPDP is a common and widespread operational requirement. However, even with a stable set of regular customers, operational plans are rarely static. Demand quantities can fluctuate significantly, and last-minute changes or urgent requests often occur with short notice. Consequently, the initial periodic plan frequently requires rapid and efficient re-optimization to maintain feasibility, cost-effectiveness, and compliance with regulations. Addressing these challenges efficiently is essential for reducing operational costs, enhancing service quality, and ensuring customer satisfaction, making them a critical focus in modern supply chain management. From an operational planning perspective, dispatchers often assign familiar routes or areas to experienced drivers, as they demonstrate greater efficiency in navigating known road conditions and interacting with regular customers (Quirion-Blais and Chen, 2021). Moreover, consistent break patterns offer drivers flexibility in planning meals and selecting rest locations, thereby improving both operational efficiency and job satisfaction (Kraul et al., 2024).

Fig. 1 illustrates the driver shift schedules for a 12-h period in a real-life vehicle routing and break scheduling problem. Each shift is characterized by its start time, end time, and break patterns (defined by the start time and duration of each break assigned to a driver on a vehicle route). Fig. 2 depicts several common break patterns for shifts shorter than 12 h. In Fig. 1, the horizontal axis represents the shift ID, while the vertical axis indicates the frequency of each shift in the solution. The figure presents solutions obtained by randomly adjusting customer demand by ±30% from the original problem instance. Shifts with identical IDs (though their frequencies may differ) in both the original and modified demand scenarios are plotted; non-matching shifts are excluded from the figure, leaving gaps in the sequence where they would otherwise appear. These results reveal notable similarities between solutions, even with demand variations of up to 30%. This suggests that predicting and fixing partial solutions that remain stable despite demand fluctuations is feasible for the routing and driver break scheduling problem, which is consistent with the results reported by Morabit et al. (2024). By providing these pre-identified patterns to the solver before the optimization process, we can enhance computational efficiency and achieve faster convergence to an optimal solution.

Recent algorithmic advances have addressed dynamic dispatching challenges in logistics, including the Monte Carlo approach (Okulewicz and Mańdziuk, 2020), Markov decision processes (Zhang et al., 2023), and reinforcement learning methods (Zhou et al., 2023). Despite these developments, existing research has rarely considered the simultaneous rapid re-optimization of routing and driver break scheduling. Although research indicates that data-driven optimization holds significant promise for solving dynamic routing problems, logistics companies have been slow to adopt these approaches in practice, likely due to concerns regarding interpretability, difficulty in debugging unsatisfactory solutions, and the high cost of implementing entirely new frameworks. Consequently, it is important to enhance currently deployed methods, such as heuristic algorithms or mathematical programming-based methods, to enable rapid re-optimization without major infrastructural changes.

To our knowledge, current approaches to routing and driver break scheduling do not adequately address the repetitive nature of real-world logistics operations or provide mechanisms for swift re-optimization in response to sudden demand changes. This gap underscores the need for advanced solution methods that leverage historical patterns to improve computational efficiency. In practice,

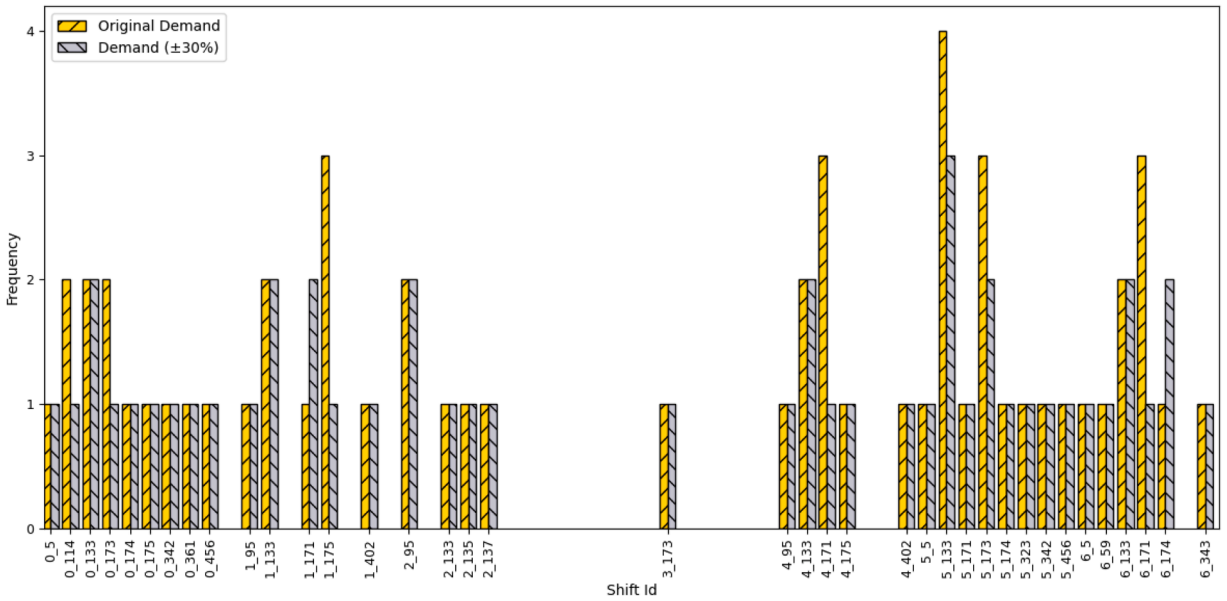

Fig. 1. Shift comparison with a 30% change in demand.

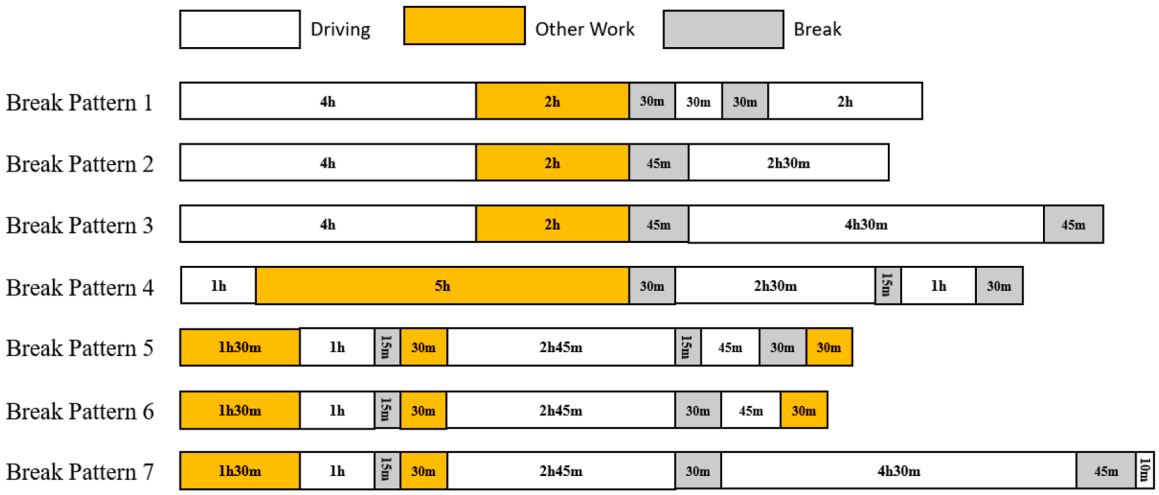

Fig. 2. Driver breaks patterns illustrations.

many routing and break scheduling problems must be solved repeatedly, such as daily or even at shorter intervals, where only the data (rather than the problem structure or objectives) changes between instances.

To address the aforementioned challenges, including (1) the computational complexity of routing and driver break scheduling, (2) the need for rapid re-optimization in response to demand fluctuations, and (3) concerns regarding interpretability and debuggability, we propose a machine learning-aided heuristic column generation approach. This hybrid approach leverages historical expert-generated schedules to rapidly identify stable solution components that remain unchanged when demand varies. Furthermore, it utilizes dual values from mathematical programming to guide the heuristic search toward promising solutions; the idea is to utilize the signal obtained from the rigorous mathematical programming to efficiently guide the heuristic for faster convergence. More specifically, the key contributions of this paper are as follows:

1. *ML-Enhanced Heuristic Column Generation:* We propose a machine learning-augmented column generation framework for the GPDPTW-DBS that uses dual values from the linear programming relaxation to heuristically guide neighborhood search, significantly accelerating the optimization process.
2. *Prediction of Stable Solution Components:* Our methodology employs machine learning to identify persistent routing edges and break patterns in near-optimal solutions. This enables the re-optimization process to concentrate exclusively on areas affected by demand fluctuations instead of generating complete solutions de novo.

3. *Search Space Reduction via Variable Fixing:* We reduce computational complexity through strategic variable fixing, implementing a dual approach: (1) fixing predicted stable routing edges to narrow the solution space, and (2) predicting high-level break strategies from which break patterns are derived, thereby reducing the computational time required to update time labels.

Our approach bridges the gap between theoretical advances and practical logistics applications by enhancing existing optimization paradigms (e.g., MIP solvers and heuristics) with machine learning. This enables practical solution times, increased interpretability, and manageable implementation costs, which are key considerations that have previously limited the adoption of data-driven methods in industry. Furthermore, the proposed framework is extensible to various repetitive optimization problems in other domains, wherever historical solution patterns can guide future decision-making.

The remainder of this paper is structured as follows: Section 2 provides a comprehensive review of the relevant literature. Section 3 offers a detailed description of the problem under investigation. Subsequently, Section 4 presents the column generation formulation developed to address the problem. Section 5 elaborates on the proposed solution framework. The experimental analysis and results are discussed in Section 6. Finally, the paper concludes with a summary of key findings and their implications in Section 7.

2. Literature review

This section reviews the work most relevant to our study, focusing on two connected research streams. First, we examine vehicle routing methods that incorporate driver break regulations. Second, we review the domain of learning-enhanced column generation (CG), which seeks to improve the efficiency and scalability of CG for large-scale scheduling.

2.1. Driver break scheduling in vehicle routing

Research on vehicle routing problems incorporating driver break regulations underscores the challenges of ensuring route feasibility and optimizing break schedules. Archetti and Savelsbergh (2009) highlight the inherent difficulty in verifying route feasibility under break constraints, particularly when attempting to identify optimal break arrangements. Goel (2009) develop a labeling-based feasibility check for routes under driving break restrictions, optimizing customer insertions/deletions without considering split breaks. Goel and Irnich (2017) enhance this by introducing dominance rules for efficient route selection in column generation, strictly focusing on driving breaks. Goel (2018) later incorporate duty break constraints, presenting a scheduling algorithm to validate break compliance for fixed routes. However, their approach limits breaks to either a single 45-min period or a 15-min break followed by a 30-min break. This method involves evaluating numerous sequences and durations of driver activities, prioritizing the maximization of driving periods and minimization of off-duty periods to comply with regulations (Goel, 2009). Despite these advancements, the approach only partially addresses driving break regulation and excludes the possibility of break splitting.

Prescott-Gagnon et al. (2010) apply a labeling method to detect viable partial paths containing breaks and rests on arcs, representing path states via resource vectors. Although this approach accounts for both driving break regulation and duty break regulation, it assumes a single 45-min break and does not incorporate split breaks or multi-day rest periods. Goel and Irnich (2017) build upon this work by explicitly modeling driver activities and introducing a method to select optimal paths without evaluating all possible activity combinations. This is achieved through a parameter-free labelling model integrated into a column generation framework. Further advancements are made by Tilk and Goel (2020), who propose a bidirectional approach to improve computational efficiency. The work of Sartori et al. (2022) solves the truck driver scheduling problem for interdependent routes through a label propagation approach that enforces driving break regulations. However, their approach also excludes break splitting, though they suggest potential extensions to incorporate duty break regulation and allow for split breaks.

Recent studies have also explored the impact of adjusting vehicle departure times, assuming that compact duty schedules reduce costs (Archetti and Savelsbergh, 2009; Kok et al., 2010). Most recently, Xue et al. (2025a) introduce a flexible routing approach that incorporates driver breaks into the planning process, seeking to minimize total route duration. Their method demonstrates adaptability by being applicable to individual routes at any scheduling stage, providing a practical framework for real-world implementation.

2.2. Learning-based enhancement for column generation

Recent years have witnessed a proliferation of studies integrating machine learning (ML) to address NP-hard combinatorial optimization (CO) problems (Bengio et al., 2021; Cappart et al., 2023; Chung et al., 2025). Learning-based approaches in CO generally fall into two paradigms. The first is supervised learning, where models are trained to emulate expert behavior, whether from humans or algorithms. The second is reinforcement learning (RL), which utilizes trial-and-error exploration to optimize predefined objectives through interaction with the problem environment.

Building on these advances, recent research has increasingly focused on applying ML and RL to enhance CG, an iterative algorithm designed to solve large-scale linear programs with exponentially many columns (variables). It adds promising columns to a restricted master problem (RMP) until optimality is reached. However, CG converges slowly for large problems. Solving its NP-hard pricing subproblems and selecting impactful columns grow computationally expensive. Learning-enhanced CG methods aim to mitigate core limitations of classical CG, including inefficient column selection, computationally expensive pricing subproblems, and poor scalability on large instances.

To systematically analyze the state-of-the-art in learning-enhanced CG, Table 1 presents a structured comparison of key studies. Methods are organized by their core CG mechanism, and the degrees of ML integration with CG are highlighted across 4 dimensions, listed below:

Table 1
Comparison of ML/RL-enhanced column generation methods.

Reference	Dual Used	Column Selection	Arc/Node Reduction	ML Deviation
ML-Guided Column Generation				
Morabit et al. (2021)	✓	✓	–	–
Morabit et al. (2023)	✓	–	✓	–
Gerbaux et al. (2025)	✓	–	✓	–
Quesnel et al. (2022)	✓	✓	–	–
Shen et al. (2022)	✓	✓	–	–
Xia and Zhang (2024)	✓	✓	–	–
Huang et al. (2023)	✓	–	✓	–
Tahir et al. (2021)	✓	–	✓	✓
This Study	✓	–	✓	✓
ML-Direct Column Prediction				
Xue et al. (2025b)	–	✓	–	–
Babaki et al. (2021)	✓	✓	–	–
Hijazi et al. (2024)	✓	–	–	–
RL-Guided Column Generation				
Chi et al. (2022)	✓	✓	–	–
Xu et al. (2025)	✓	✓	✓	–

Legend: ✓ = Yes; – = No; ML Deviation = Allowed to deviate from ML predictions

1. **Dual Used:** Dual values are fundamental to classical CG, as they reflect the marginal value of each constraint for the optimal objective function value of the RMP. These values update iteratively with each solve of the RMP and evolve dynamically as the CG process progresses. This feature indicates whether dual values from the RMP are leveraged to guide the CG process, specifically for solving the subproblem and identifying potential columns for inclusion in the RMP.
2. **Column Selection:** Denotes whether ML or RL is employed to prioritize or filter columns for inclusion in the RMP. This mechanism accelerates CG convergence by identifying which columns (e.g., crew pairings, delivery routes) to add.
3. **Arc/Node Reduction:** Captures whether ML or RL is used to prune non-promising arcs or nodes from the pricing subproblem's network representation. Unlike column selection, this mechanism focuses on retaining only those arcs or nodes that contribute constructively to column generation, thus reducing computational complexity.
4. **ML Deviation:** Indicates whether the algorithm allows for deviation from ML predictions in *column selection* or *arc/node retention*. Specifically, it assesses if arcs/nodes predicted by ML to be retained, or columns predicted by ML to be prioritized for RMP inclusion, can be temporarily excluded and reintroduced in subsequent CG iterations. This flexibility allows the algorithm to temporarily ignore ML predictions to mitigate the risk of omitting critical columns or network components that are essential for convergence to high-quality solutions. In Table 1, a check mark under the *ML Deviation* column signifies that ML deviation is allowed. An unchecked mark denotes the algorithm's strict adherence to ML predictions, with no provision to revise the decisions inferred by the learning model during the CG process.

As can be seen from Table 1, the evolution of learning-enhanced CG can be broadly categorized into three primary research streams, alongside additional innovative variants expanding the field. The first and most prominent stream, **ML-guided column generation**, uses ML to augment heuristic pricing and improve both column quality and computational efficiency. In this approach, the pricing subproblem remains the primary source for generating columns, while the ML model serves to enhance the process. Specifically, the model may prune arcs, rank candidate columns, or filter out non-promising options, but the generation of columns fundamentally still relies on the underlying pricing solver. As an illustration, Morabit et al. (2021) developed a supervised learning-based column selection strategy that accelerates CG convergence. Similarly, Morabit et al. (2023) proposed an ML-driven arc selection method for constrained shortest path problems, effectively pruning non-promising edges to reduce pricing network size without significantly sacrificing solution quality. Gerbaux et al. (2025) applied ML-based network pruning for bus scheduling, using supervised models to identify critical arcs that contribute to near-optimal columns. In crew rostering, Quesnel et al. (2022) utilized deep learning to rank candidate duties, guiding subproblem to generate columns aligned with long-term optimality. Shen et al. (2022) integrated ML with labeling algorithms for graph coloring. For vehicle routing with two-dimensional loading, Xia and Zhang (2024) proposed a neural CG algorithm that integrates attention and recurrence mechanisms to predict column feasibility. In air cargo recovery, Huang et al. (2023) employed a ML model to predict promising connections, enabling the filtering of non-valuable flight delay decisions during column-and-row generation, thereby accelerating solution efficiency while maintaining cost and load utilization balance. Tahir et al. (2021) demonstrated this for aircrew pairing problems, where a deep convolutional neural network first predicts flight connection probabilities to prune non-promising arcs from the shortest path problem with resource constraints (SPPRC) network, generating reduced subproblems to efficiently produce candidate columns; full SPPRCs (with all feasible arcs) are used as a fallback when successive column generation iterations fail to achieve sufficient solution improvements, ensuring no critical columns are missed.

The second stream, **ML-direct column prediction**, bypasses traditional pricing subproblems by using ML to directly generate feasible columns. In contrast to the first stream, this approach circumvents the pricing subproblem entirely. The ML model is directly

integrated into the CG process and helps produce complete and feasible columns, such as job sequences, task groups, or routes, often without relying on an exact solution to the pricing subproblem. These generated columns are then validated for feasibility. The conventional pricing subproblem is either omitted entirely or used only as a fallback rather than serving as the primary source for column generation. For example, in trailer-swapping truck scheduling, Xue et al. (2025b) used a ML model to filter valid task groups within heuristic CG, accelerating the solution process without sacrificing feasibility. Babaki et al. (2021) framed column selection for the capacitated VRP as a sequential decision-making task, using imitation learning and neural architectures that encode both instance data and CG state. The model predicts feasible dual values via differentiable optimization, ensuring compliance with constraints, and then decodes duals into a candidate-route distribution. Tahir et al. (2021) developed a neural network-enhanced CG strategy for aircrew pairing problems, minimizing total pairing cost. The method uses a deep convolutional neural network to predict flight connection probabilities, generating reduced subproblems with high-probability arcs to produce candidate columns and replacing most full subproblem solves. Full subproblems (with all feasible arcs) are invoked only if successive iterations fail to sufficiently improve the solution, ensuring no critical columns are missed while reducing overall computation time.

Beyond these two streams, a prominent variant is **RL-guided column generation**, which introduces sequential decision-making to select optimal low-level pricing heuristics. Chi et al. (2022) proposed a deep RL framework for column selection, leveraging graph neural networks to encode the bipartite variable-constraint structure of the RMP. Their reward function incentivizes RMP objective improvement and penalizes extra iterations, enabling their agent to outperform greedy column selection on both the cutting stock problem and VRP with time windows. Xu et al. (2025) developed an RL-based hyper-heuristic that selects among five low-level heuristics per CG iteration. Their framework prunes pricing networks to speed up solving the shortest path problem with resource constraints.

Beyond structured CG enhancements, RL has also proven useful when directly applied to routing problems, thereby informing CG design. James et al. (2019) formulated online VRP as a sequential decision task, using graph-embedded pointer networks to iteratively generate routes, modeling how RL can inform column structures in dynamic routing contexts. For the multi-vehicle capacitated VRP, Nazari et al. (2018) demonstrated that RL can outperform traditional heuristics and OR-Tools, illustrating the learning potential from intrinsic problem structure. Early foundational work by Khalil et al. (2017) and further scaling by Li et al. (2018) established graph-based learning as a scalable approach for RL-guided CG.

A core challenge in integrating ML with CO is balancing solution quality with the flexibility to correct ML-driven errors. In our study, ML-predicted elements such as temporarily fixed arcs or driver break patterns are incorporated in a way that allows ML deviation. These elements are not permanently enforced; for example, arcs initially selected by the ML model can be broken and reintroduced as needed during the optimization process. The motivation for this design is to minimize the risk of missing important columns due to prediction errors. From Table 1, it can be seen that while Tahir et al. (2021) also supports ML deviation, their mechanism differs fundamentally. Their “partial pricing with full pricing fallback” strategy initially restricts the subproblem to high-probability flight connections using ML and invokes the complete problem only if no improvement is achieved. In this approach, ML decisions can be overridden entirely when necessary. In contrast, our approach does not substitute the entire subproblem structure but instead applies targeted, dynamic adjustments through a destructive repair heuristic. This enables selective removal and re-integration of ML-predicted components, maintaining overall algorithmic flexibility while leveraging ML guidance. Collectively, these distinctions emphasize the importance of flexible integration strategies to robustly translate ML predictions into effective optimization decisions.

3. Problem definition

The GPDPTW-DBS can be defined on a directed graph $G = (N, A)$, where $N = P \cup D \cup W$ represents the set of nodes, and A denotes the set of arcs connecting the nodes. Here, P is the collection of all pickup locations, D represents all delivery locations, and $W = M^+ \cup M^-$ represents the set of start and end locations for vehicles. Each directed edge $(i, j) \in A$ is characterized by a driving duration δ_{ij} from point i to point j.

A request $k \in K$ is represented as a pair of nodes (p, d), where $p \in P$ and $d \in D$, indicating that goods must be transported from p to d. Each node $i \in N$ has an associated time window $[e_i, l_i]$, where e_i denotes the earliest possible starting time and l_i represents the latest allowable starting time. A vehicle may arrive prior to e_i and wait until service begins, but it must not arrive after l_i. The demand at node i is denoted by q_i, such that for each request (p, d), $q_p \geq 0$ and $q_d = -q_p$.

A feasible solution consists of a collection of selected routes $R \subseteq \Omega$ that satisfies all (or nearly all) customer requests. Every route $r \in \Omega$ represents an ordered sequence of n visits $\{0, 1, \ldots, n-1\}$ accompanied by a break schedule comprising a set of breaks, each defined by its start time and duration. The break schedule for route r can be formally represented as $B_r = \{(s_b, d_b) \mid b = 1, \ldots, |B_r|\}$, where each pair (s_b, d_b) denotes the start time and duration of the bth break in the schedule. The solution is subject to these constraints:

- All service must occur within the specified time intervals $[e_i, l_i]$ for each location.
- The total load must satisfy $\sum_{i \in r} q_i \leq Q$ at all times during each route r.
- For each request $k = (p_k, d_k) \in K$, the visit sequence must satisfy $\psi(p_k) < \psi(d_k)$, where $\psi(i)$ denotes the position of node i in the route.
- Each pickup-delivery pair (p, d) must be served by the same vehicle route.
- Every vehicle route r must begin at its origin depot $o_r \in M^+$ and terminate at its destination depot $d_r \in M^-$.

The objective of the GPDPTW-DBS is to minimize the total cost, which is the sum of the cost c_r of all selected routes $r \in R$. The cost c_r of a route is measured by the total duty time. For a route with n visits, the total duty time is calculated as $a_n - a_0$, where a_i represents the arrival time at location i.

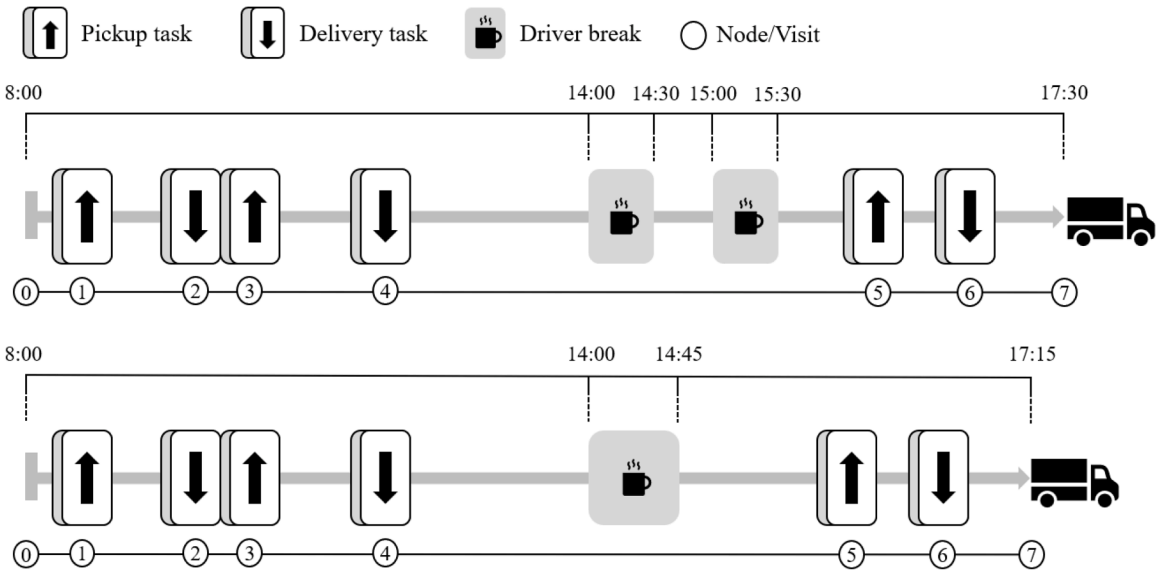

Fig. 3. Illustration of GPDPTW-DBS route execution.

To illustrate how break scheduling impacts route feasibility and efficiency in the GPDPTW-DBS, Fig. 3 compares two distinct break strategies for the same set of pickup-delivery requests $k \in K$ on a single vehicle route $r \in \Omega$. This route consists of $n = 8$ ordered visits (nodes) $\{0, 1, 2, 3, 4, 5, 6, 7\}$. It starts at the origin depot (node 0) and ends at the destination depot (node 7). The horizontal time axis tracks the execution of two task types: *pickup tasks* (i.e., upward arrows at visits 1, 3, 5) and *delivery tasks* (i.e., downward arrows at visits 2, 4, 6). Driving durations accumulate between consecutive visits.

The first schedule (top) follows a compliant driver break strategy required by regulation (EC)561/2006. After 4.5 h of cumulative driving, which completes visits 0–4 (corresponding to two pickup-delivery requests), the driver takes 2 breaks: $B_r = \{(s_1 = 14:00, d_1 = 30), (s_2 = 14:30, d_2 = 30)\}$. These breaks are scheduled between nodes and do not correspond to any specific node. The first 30-min break is required by Directive 2002/15/EC, which mandates a 30-min break after 6 h of work. Here, the driver has worked from 8:00 to 14:00, reaching the 6-h threshold. After driving for another 30 min, a second 30-min break is scheduled. This second break is mandated by Regulation (EC) 561/2006, which requires a 45-min break after every 4.5 h of driving. The regulation permits two smaller breaks: one of at least 15 min followed by another of at least 30 min. Since the driver has already taken a 30-min break, a second 30-min break satisfies this requirement. After these breaks, the driver resumes driving to complete visits 5–6 (one additional request) and arrives at the terminal (node 7) at 17:30. The total duty time (route cost c_r) is $a_7 - a_0 = 9.5$ h.

This schedule can be improved, as shown in the second schedule (bottom). After completing the same initial visits 0–4, the split breaks are replaced by a single 45-min driver break: $B_r = \{(s_1 = 14:00, d_1 = 45)\}$. This single break satisfies both regulatory constraints while reducing downtime. It allows the driver to complete visits 5–6 and arrive at the terminal at 17:15. The total duty time is $a_7 - a_0 = 9.25$ h. The improvement occurs because the 45-min break, taken when the duty time reaches 6 h, also resets the driving hour counter according to Regulation (EC)561/2006. Consequently, an additional 30-min break after 4.5 h of driving is no longer required.

This example demonstrates that while break schedules B_r must adhere to regulatory constraints, their pattern directly influences the total duty time, which is the objective of the GPDPTW-DBS. It also underscores the value of flexible break scheduling within feasible routes, where breaks can be optimally inserted between visits to enhance efficiency.

4. Column generation formulation with driver break constraints

Column generation (CG) is recognized as an exact algorithm for addressing a wide range of optimization problems, such as routing and scheduling. The method operates iteratively, alternating between two core steps: (1) solving a simplified version of the original linear relaxation, referred to as the restricted master problem (RMP) (discussed in Section 4.1), and (2) solving a pricing subproblem (explained in Section 4.2), which seeks to discover new columns with negative reduced costs (for minimization objectives) to incorporate into the RMP. The CG procedure completes when no further improving columns can be identified.

CG is especially powerful for tackling large-scale linear programming problems, particularly those embedded within a branch-and-price framework to solve integer programming counterparts. This characteristic makes it an ideal choice for the problem formulation outlined in Section 4.1, where the set of feasible routes is exceedingly large, leading to a model with a massive number of columns, even though the optimal solution usually relies on only a small fraction of them.

In the context of vehicle routing, CG is an exact approach whose pricing problem can be modeled as an Elementary Shortest Path Problem with Resource Constraints (ESPPRC). However, solving the ESPPRC is computationally prohibitive for large-scale instances,

Table 2
Notations used in the problem formulation.

Notation	Description
Sets	
K	Set of transportation requests.
N	Set of all nodes (locations).
A	Set of arcs connecting pairs of nodes.
B	Set of break points.
Ω	Set of all feasible routes.
V	Set of available vehicles (fleet).
Variables	
λ_r	Binary variable indicating whether route r is selected (1) or not (0).
a_{kr}	Binary parameter indicating whether request $k \in K$ is served by route r (1) or not (0).
x_{ij}	Binary variable indicating whether arc $(i, j) \in A$ is used in the route (1) or not (0).
a_i	Arrival time at node $i \in N$.
L_b^{duty}	Accumulated duty time at break point $b \in B$ since the last duty break reset.
L_b^{driving}	Accumulated driving duration at break point $b \in B$ since the last driving reset.
$L_b^{\text{break\|work1st}}$	The total break duration until the first threshold for assigning a duty break.
$L_b^{\text{break\|work2nd}}$	The total break duration until the second threshold for assigning a duty break.
Δ^{break}	Duration of a single, continuous break period.
Δ_1^{break}	Duration of the first part of a split break.
Δ_2^{break}	Duration of the second part of a split break.
Data Constants	
c_r	Total duty time for route r, computed as $a_n - a_0$ for a route with n visits.
c_{ij}	Duty time between arc $(i, j) \in A$.
π_k	Dual value from the master problem corresponding to request $k \in K$.
ν	Dual value from the master problem corresponding to the fleet size constraint.
e_i	Earliest arrival time at node $i \in N$.
l_i	Latest arrival time at node $i \in N$.
δ_{ij}	Driving time between nodes i and j.
$l^{\text{break\|drive}}$	Maximum driving time without a break (default: 270 min).
$dutyBreakRequirement1$	Minimum break duration for 6-9 h of work (default: 30 min).
$dutyBreakRequirement2$	Minimum break duration for more than 9 h of work (default: 45 min).
$l^{\text{break\|work1st}}$	First threshold for assigning a duty break (default: 360 min).
$l^{\text{break\|work2nd}}$	Second threshold for assigning a duty break (default: 540 min).
l^{break}	The minimum duration of a break (set by default to 45 min).
$l^{\text{break\|1st}}$	Minimum duration of the first part of a split break (default: 15 min).
$l^{\text{break\|2nd}}$	Minimum duration of the second part of a split break (default: 30 min).
q_i	Demand at node $i \in N$ (positive for pickups, negative for deliveries).
Q	Vehicle capacity.

especially in re-optimization settings where computational efficiency is of great importance. To accelerate the process, we employ a heuristic column generation strategy. This approach is motivated by two key insights: first, the ability to efficiently reuse high-quality columns (routes) from prior solutions within the column pool; and second, the utilization of dual values obtained from the RMP to effectively guide the heuristic search. Unlike pure heuristics, which may exhibit uncontrolled randomness, this dual guidance ensures a targeted exploration of the solution space, promoting the generation of columns that are most likely to improve the objective function, thereby leading to significantly faster convergence without substantial sacrifice in solution quality. The notations used in the model formulation are provided in Table 2, with default configurations adhering to driver break and duty break regulations. Furthermore, the parameter settings for driver break regulations in Australia (Standard Hours (SH) and Basic Fatigue Management (BFM)) and New Zealand (Work Time Requirement (WTR)) are outlined in Table A.1 in the Appendix.

The column generation framework for this problem is formally defined by two interconnected components: the *Master Problem* and the *Pricing Sub-Problem*. The Master Problem, detailed in Section 4.1, is a linear relaxation that selects an optimal combination of routes from a restricted pool to satisfy all requests at minimal cost. To improve this solution, the Pricing Sub-Problem, described in Section 4.2, is solved to generate new, cost-effective routes with negative reduced cost, which are then added to the RMP for the next iteration. This iterative process continues until no further improving columns can be found, indicating optimality for the linear relaxation.

4.1. Master problem

Assuming that we are given the set of all feasible routes Ω, the problem can be modeled using the following set partitioning formulation:

$$\min \quad \sum_{r \in \Omega} c_r \lambda_r \tag{1}$$

$$\text{s.t.} \quad \sum_{r \in \Omega} a_{kr} \lambda_r = 1 \quad \forall k \in K \tag{2}$$

Table 3
Constraints of the master and pricing problems.

Problem / Category	Description
Master Problem	**Fleet Size:** The number of vehicles used must not exceed the number available.
Pricing Sub-Problem	$\sum_{r\in\Omega} \lambda_r \le \lvert V\rvert$ **1. Route Structure** • **Elementarity:** Visit each node $i \in N$ at most once. • **Depot Start/End:** Start at depot o and end at depot d. **2. Pickup & Delivery** • **Precedence:** For request $k \in K$, visit p_k before d_k ($a_{p_k} \le a_{d_k}$). • **Pairing:** The same vehicle must serve both p_k and d_k. **3. Temporal** • **Time Windows:** $e_i \le a_i \le l_i$ for all nodes $i \in N$. **4. Capacity** • **Vehicle Capacity:** $0 \le L_i^{\text{load}} \le Q$ at any node i. **5. Driver Scheduling** Let Δ^{break} be the duration of a single, continuous break period. For a split break, let Δ_1^{break} be the duration of the first part and Δ_2^{break} be the duration of the second part. **Regulation (EC)561/2006 (Driving Regulation):** • **Maximum Daily Driving:** The total driving time must not exceed 9 h. • **Driving Breaks:** Let L_b^{driving} be the driving time accumulated since the last break reset at break point $b \in B$. - If $L_b^{\text{driving}} \le l^{\text{break\|drive}}$: Driving may continue. - If $L_b^{\text{driving}} > l^{\text{break\|drive}}$: A break must be taken before driving can continue. The break must be either: 1. A single continuous break of duration $\Delta^{\text{break}} \ge l^{\text{break}}$, **or** 2. A split break consisting of a first period of duration $\Delta_1^{\text{break}} \ge l^{\text{break\|1st}}$ followed by a second period of duration $\Delta_2^{\text{break}} \ge l^{\text{break\|2nd}}$. A break satisfying either of the above conditions resets the driving clock: $L_{b+1}^{\text{driving}} \leftarrow 0$. **Directive 2002/15/EC (Duty Regulation):** • **Maximum Work Without a Break:** A driver must not work more than $l^{\text{break\|work1st}}$ consecutive minutes (i.e., 6 h) without a break. Any break taken to satisfy this rule must be at least $l^{\text{break\|1st}}$ min long. • **Total Work Break Requirements:** The total accumulated working time during the day determines the minimum total break duration required: - If $0 < L_b^{\text{duty}} \le l^{\text{break\|work1st}}$ (work up to 6 h): No additional total break duration is required beyond satisfying the maximum continuous work rule above. - If $l^{\text{break\|work1st}} < L_b^{\text{duty}} \le l^{\text{break\|work2nd}}$ (work between 6 and 9 h): The driver must take a total break duration of at least $dutyBreakRequirement1$ min (i.e., 30 min) during the day. - If $L_b^{\text{duty}} > l^{\text{break\|work2nd}}$ (work more than 9 h): The driver must take a total break duration of at least $dutyBreakRequirement2$ min (i.e., 45 minutes) during the day. Any required break may be split into periods of at least 15 min each. A break of sufficient total duration ($\Delta^{\text{break}} \ge$ required duration) resets the work accumulators: $L_{b+1}^{\text{break\|work1st}} \leftarrow 0$, $L_{b+1}^{\text{break\|work2nd}} \leftarrow 0$.

$$\sum_{r\in\Omega} \lambda_r \le |V| \tag{3}$$

$$\lambda_r \in \{0,1\} \quad \forall r \in \Omega \tag{4}$$

In this model, the binary decision variables λ_r indicate whether a feasible route $r \in \Omega$ is selected ($\lambda_r = 1$) in the solution or not ($\lambda_r = 0$). The objective function (1) minimizes the cumulative duration of all selected routes. The coefficients a_{kr} are binary parameters that state whether a specific transportation request $k \in K$ is served by route r ($a_{kr} = 1$) or not ($a_{kr} = 0$). Consequently, constraint (2) ensures that each request k is covered by exactly one route. The constraint (3) limits the number of routes selected to the size of the available vehicle fleet $|V|$. Finally, constraint (4) enforces the binary nature of the decision variables.

4.2. Pricing sub-problem

The reduced cost of a route r is given by:

$$\bar{c}_r = c_r - \sum_{k\in K} \pi_k a_{kr} - \nu \tag{5}$$

where $c_r = \sum_{(i,j)\in r} c_{ij}$ is the total cost of route r, $a_{kr} = 1$ if route r serves request k, π_k is the dual value from the master problem's partitioning constraint for request $k \in K$, and ν is the dual variable associated with the fleet size constraint.

5. The proposed solution framework

Prior to solving the RMP for the first iteration, dual information is not available, necessitating an initial set of columns (routes) to initiate the procedure. After solving the linear programming relaxation (LPR) to optimality, the dual values are utilized to compute

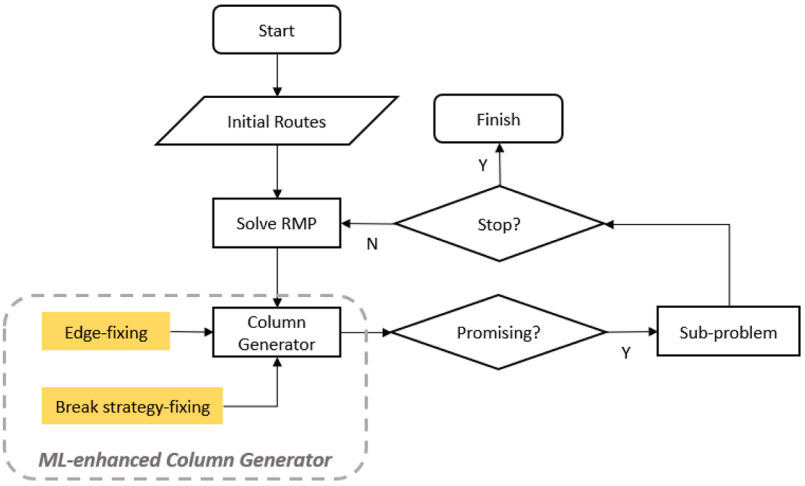

Fig. 4. Framework of the proposed column generation process.

the reduced costs of routes in the pricing subproblem. Initial experiments reveal that solving the RMP accounts for the majority of the computational effort. To enhance convergence, the algorithm was adjusted to incorporate multiple routes with negative reduced costs in each iteration, a method aligned with prior research (Xue et al., 2021). Since the pricing problem is solved iteratively within the column generation framework, it is imperative to employ an efficient solution strategy for this subproblem.

To solve the pricing problem optimally, we can employ dynamic programming by formulating it as an ESPPRC, as proposed by Goel and Irnich (2017) and Goel (2018). These methods can offer provably optimal solutions within a column generation scheme, ensuring the strong lower bounds required for branch-and-price algorithms. However, this optimality guarantee incurs significant computational costs. The state space and resource constraints, particularly those for complex hours-of-service rules, can trigger exponential growth in the number of labels. This restricts applicability to large-scale or real-time scenarios. As noted, the main objective of this paper is to prioritize rapid re-optimization, which necessitates a trade-off between optimality and efficiency.

We therefore adapted the ESPPRC from Goel (2018), which serves as a reliable benchmark for evaluating other methods, by modifying its state space and transition rules for our specific problem context. This tailored ESPPRC (see Section A.1 in the appendix for details) will serve as a benchmark to assess the method we propose in this paper. Given this balance between optimality and computational feasibility, we advocate for metaheuristic techniques outlined in Xue et al. (2021) to solve the pricing problem. This choice is motivated by the need to handle the combined complexity of routing and driver break scheduling within an efficient metaheuristic framework. The column generation process terminates based on a predefined limit on the number of RMP iterations, striking a deliberate balance between computational efficiency and solution quality, and iterates until stopping conditions are met. Finally, to derive integer solutions, the relaxed constraints on λ_r are reinstated during the final resolution of the RMP. The overall solution framework is depicted in Fig. 4, with a step-by-step explanation provided subsequently.

5.1. Initial set of routes

To generate this initial set, we utilize a straightforward route initialization technique. A fundamental requirement for creating a basic route set is to guarantee that every pickup and delivery request is covered by at least one route. The most basic strategy involves constructing a dedicated route for each request, where an unloaded truck starts from the origin depot, proceeds to the pickup point, transports the goods to the designated delivery location, and finally returns to the destination depot. Driver break schedules are generated using the compliant scheduling method from Xue et al. (2025a), which ensures full adherence to driving and duty time regulations.

Nevertheless, this method may lead to an infeasible solution regarding the constraint on the maximum number of vehicles. To resolve this, we initially assign the number of available vehicles to be equal to the number of requests in the problem instance. As the column generation process advances, the number of trucks is progressively decreased to align with the number of columns generated in each RMP iteration, thereby maintaining feasibility and enhancing the efficiency of the subsequent solving process.

5.2. Edge-fixing

Let $\mathcal{P}_o$ and $\mathcal{S}_o$ denote the original problem instance and its corresponding solution, respectively. For a perturbed instance $\mathcal{P}_m$ of the original problem $\mathcal{P}_o$, our approach aims to determine which components of the known solution $\mathcal{S}_o$ persist in the unknown solution $\mathcal{S}_m$, avoiding complete reoptimization. Let E be the edge set comprising all arcs employed in $\mathcal{S}_o$. The goal is to predict which edges $e \in$ E have a high probability of being part of $\mathcal{S}_m$. Here, an edge e is defined as the connection between two requests. By fixing these edges, we aim to significantly reduce the problem's complexity, thereby greatly decreasing the computational time required.

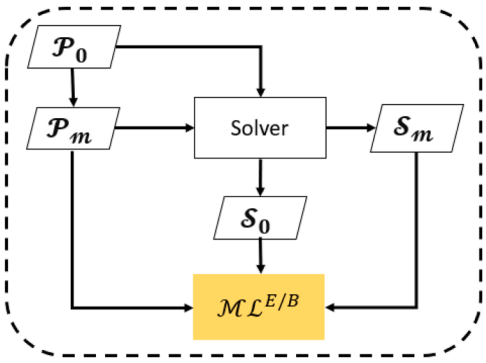

Fig. 5. Data generation process.

We trained a supervised binary classification model, denoted as $\mathcal{ML}^E$, which predicts the probability of edge retention in modified problem instances. The first step involves collecting sufficient data for training. The training process for the learned model, denoted as $\mathcal{ML}^E$, is illustrated in Fig. 5. Given a tuple of original and modified instances along with their solutions ($\mathcal{P}_o$, $\mathcal{S}_o$, $\mathcal{P}_m$, $\mathcal{S}_m$), a labeled dataset $\mathcal{D} = \{\{X_e, y_e\} \mid \forall e \in \mathrm{E}(\mathcal{S}_o)\}$ is constructed. Each entry in $\mathcal{D}$ represents an edge in the original solution, where $X_e \in \mathbb{R}^n$ corresponds to the edge's feature vector, n is the number of features, and $y_e \in \{0, 1\}$ is the binary label indicating whether the edge is part of $\mathcal{S}_m$.

Due to the need to generate a large number of modified instances and their corresponding solution pairs, it is computationally infeasible to use an exact approach. Instead, we employ the heuristic solver proposed in Xue et al. (2025a), which is based on a combination of local search and large neighbourhood search approaches. Since this is a supervised learning problem, both solutions $\mathcal{S}_o$ and $\mathcal{S}_m$ are required to build the dataset. The labels are assigned by checking the overlap between the edges in $\mathcal{S}_o$ and $\mathcal{S}_m$ as follows:

$$y_e = \begin{cases} 1 & \text{if } e \in (\mathrm{E}(\mathcal{S}_o) \cap \mathrm{E}(\mathcal{S}_m)), \\ 0 & \text{otherwise.} \end{cases}$$

The feature vector X_e captures the characteristics of each edge. To identify relevant features, we conducted an extensive feature engineering process informed by the work of Arnold and Sörensen (2019) and Morabit et al. (2024). We first created a comprehensive set of potential features to capture the distinctive characteristics of edge patterns, then refined the selection using Recursive Feature Elimination (RFE).

This study prioritizes parsimony and problem alignment over exhaustive feature inclusion. Several potentially relevant features, such as route length and vehicle load on edges, were omitted for three principal reasons. First, these features demonstrated low importance and were eliminated during the RFE process due to their minimal contribution to classifier performance. Second, the reoptimization context involves changes only to demands while regular customer locations, fleet size, and vehicle capacities remain fixed. Consequently, many candidate features are either irrelevant or redundant, as they are already captured by existing features such as edge costs and spatial coordinates. Finally, the classifier focuses on edge level predictions rather than route level attributes. This approach requires avoiding features that introduce dependencies on other edges, thereby preserving the simplicity and isolation of the prediction task. Eventually, the following key features were selected:

- The (x, y) coordinates of the delivery node $d_k \in D$ (end location of request k) and the pickup node $p_l \in P$ (start location of request l), for requests $k, l \in K$;
- The cost c_{ij} of edge $(i, j) \in A$;
- The old and new demands of nodes (p_k, d_k) and (p_l, d_l) for requests $k, l \in K$;
- A binary indicator for depot edges (1 if either i or j is a depot node, where $(i, j) \in A$);
- A binary indicator for demand changes (1 if demand for request k, l, or both has changed, where $k, l \in K$);
- The rank of d_k relative to p_l (and vice versa) based on neighbor distances (e.g., rank 1 if p_l is d_k's nearest neighbor);
- A binary indicator for arc intersection (1 if arc (p_k, d_k) intersects with arc (p_l, d_l)).

The selected features effectively support the prediction of edge retention or modification during pickup-delivery routing reoptimization. In this scenario, pickup (p_l) and delivery (d_k) locations remain fixed, while demands are perturbed. The features focus on core factors that influence edge stability and responsiveness to changes. Spatial details such as the (x, y) coordinates of d_k and p_l, along with the neighbor distance rank between these nodes, capture fixed geographic proximity. Edges connecting geographically close pickup and delivery points are less susceptible to disruption from demand shifts, as their spatial efficiency remains consistent. The edge cost c_{ij} directly reflects route optimality, ensuring that low-cost edges, which are critical for minimizing total travel expenses, are prioritized for retention.

The old and new demands of p_k, d_k, p_l, and d_l, combined with the binary indicator for demand changes, isolate the primary dynamic perturbation in the problem. These features identify edges associated with requests that have altered demands, which are at

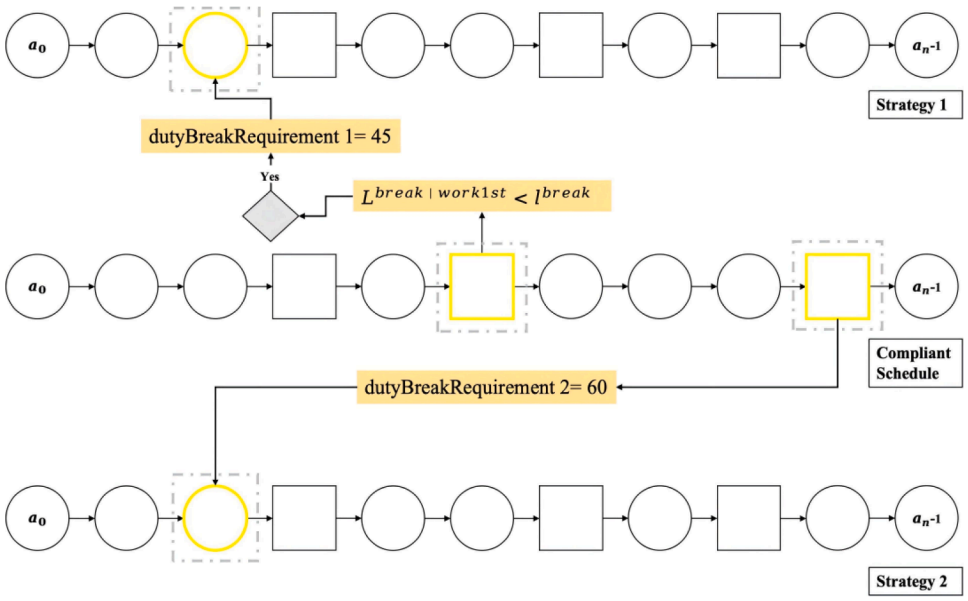

Fig. 6. Compliant schedule and optimization strategies.

higher risk of removal, as well as those with stable demands, which are more likely to be retained. The binary indicator for depot edges accounts for structural constraints, recognizing that edges connected to the depot are inherently more stable, since all routes must begin or end at the depot. Additionally, the binary indicator for arc intersection addresses practical routing challenges. Intersecting arcs, such as (p_k, d_k) and (p_l, d_l), can introduce inefficiencies, so this feature helps identify edges that may require adjustment through re-optimization even when their spatial or demand attributes appear stable.

5.3. Break strategy-fixing

The core concept of applying the learned heuristic to break strategy fixing is to predict driver break schedules that are likely to remain consistent across different routes. By identifying these stable components, the break strategy can be applied directly, eliminating the need for frequent backtracking and updates of time-related variables along the entire route. This reduces the search space and accelerates the solution process. This method capitalizes on insights from prior solutions, enabling the reuse of stable break patterns. Fixing these patterns significantly lowers the problem's complexity, resulting in a notable decrease in computational time.

Let B denote the set of driver breaks included in the known solution S_o. Our objective is to predict the break strategy $b \in \mathrm{B}$ for a given route $r \in S_m$. The study (Xue et al., 2025a) proposed a compliant approach for scheduling driver breaks, along with two advanced strategies designed to optimize route duration by strategically adjusting the timing for resetting driving hours. These strategies are outlined below:

- **Strategy 1**: When the accumulated break duration reaches the initial threshold ($l^{\text{break|work1st}} = 30$ min) for duty break qualification, and if any break sequence results in $L_i^{\text{break|work1st}} < 45$ min (whether through one 30-min break or multiple 15-min breaks), the `dutyBreakRequirement1` must be raised from 30 to 45 min.
- **Strategy 2**: If the cumulative break duration attains the secondary threshold $l^{\text{break|work2nd}}$ and the break count satisfies $|B| > 1$, then `dutyBreakRequirement2` must be increased from 45 to 60 min.

The break scheduling methodologies corresponding to Strategies 1 and 2 are visually represented in Fig. 6. Within the diagram, circular nodes denote customer visits, whereas square markers represent planned driver breaks. From an initial compliant schedule, the process spawns new branches when multiple breaks exist, modifying the first break and propagating time adjustments backward through the route. When the compliant scheduling process yields a solution containing more than two breaks ($|B| > 2$), the algorithm automatically invokes Strategy 2.

The algorithm has been provided in detail in Xue et al. (2025a); for the completeness of this paper, the main logic process is as follows. The process first assesses if the initial duty break, of duration $dutyBreakRequirement1$, is required, which is triggered when the accumulated working time surpasses the first threshold $l^{break|work1st}$. The system also verifies that there is adequate time available in the schedule to accommodate this pause without conflict. If these criteria are satisfied, the break is officially inserted into the driver's itinerary. Scheduling this break causes a delay, which subsequently shifts the entire remaining timeline for the duty period later by the duration of the break ($dutyBreakRequirement1$); this ensures all future calculations are based on the updated, realistic schedule. Following this, a second assessment is made for the subsequent duty break of duration $dutyBreakRequirement2$. This evaluation uses

the newly adjusted timeline to check if the working time has now also exceeded the second threshold $l^{break|work2nd}$ and confirms there is sufficient time to place this longer interruption.

To resolve regulatory violations, the procedure inserts additional driver breaks. It uses the compliant schedule as a default but may branch to Strategy 1 or Strategy 2 based on specific conditions. The insertion process starts at the node where a perturbation occurs (e.g., a change in a time window), even if the current solution remains feasible at that point. All subsequent time-related variables are then recalculated and updated from this node onward due to the forward-propagating nature of schedule updates. A set of breaks B' dominates another set B when the resulting route r' has shorter duration than route r while both routes satisfy all time window constraints for their respective visits.

As mentioned, although the driver break optimization algorithm demonstrates efficiency through selective evaluation of time-related variables at critical points, it remains computationally intensive due to inherent repetitive calculations. As illustrated in Fig. 6, the algorithm escalates `dutyBreakRequirement2` from 45 to 60 min upon satisfying two conditions: (1) reaching the secondary break duration threshold, and (2) having multiple breaks ($|B| \geq 2$). Such adjustments mandate recalculation of preceding nodes' time variables, creating computational inefficiencies. Our proposed machine learning framework aims to substantially reduce this overhead.

Specifically, we employ a supervised learning approach with a ternary classification model $\mathcal{ML}^B$. The procedure for preparing the training data, which is the same as the one illustrated in Fig. 5, follows a methodology similar to that described in Section 5.2. The objective is to predict the break strategy (Compliant, Strategy 1, or Strategy 2) and, consequently, generate the most suitable break patterns that are likely to be part of a route, given the features of the route, rather than updating the labels (both duty- and driving time-related, such as L_b^{driving}, $L_b^{\text{break|work1st}}$, and $L_b^{\text{break|work2nd}}$) at each node and checking if the conditions for switching to different break strategies are met. It is important to clarify that while we refer to this method as `break strategy-fixing`, the machine learning prediction focuses on the break strategy rather than the break patterns directly. This is because a single break strategy can produce more than one break pattern, as demonstrated in Xue et al. (2025a). Since there are two break strategies, together with the compliant schedule, resulting in three categories in total, we employ a ternary classification model (Compliant, Strategy 1, or Strategy 2) to predict the break strategy.

The selected features for the classifier $\mathcal{ML}^B$ are designed to predict the optimal break scheduling strategy by capturing the core temporal and regulatory constraints that govern driver breaks. Unlike the edge-fixing scenario, where class balance in the training data is inherently tied to solution similarity (detailed in Section 6.2.1), the training data for break strategy-fixing was explicitly curated to ensure a balanced distribution across the three strategy labels, mitigating potential model bias and improving learning efficacy.

The feature set is constructed from the evolving state of a route to directly reflect the rules of Regulation (EC) 561/2006 and Directive 2002/15/EC:

- Total route duration
- Driving/Non-driving time within $l^{\text{break|drive}}$
- Driving/Non-driving time within $l^{\text{break|work1st}}$
- Driving/Non-driving time within $l^{\text{break|work2nd}}$

These features collectively model the state of the driver according to the two intertwined regulations. The driving-time-based features directly dictate compliance with driving break rules, while the working-time-based features control duty break requirements.

5.4. ML-enhanced heuristic column generator

The pricing subproblem can, in principle, be formulated and solved exactly as an ESPPRC. However, this exact approach is computationally prohibitive for large instances, as will be demonstrated in the experimental results in Section 6.1. Instead, heuristic methods are commonly adopted in commercial routing applications (Hall and Partyka, 2016), due to their scalability and flexibility in handling complex, real-life constraints such as driver break scheduling. Despite their efficiency, naive heuristics often fail to consistently produce high-quality columns with negative reduced cost. Also, as demonstrated in Section 1, we show that even with demand variation of up to 30%, there is still significant similarity between route schedules, which suggests that predicting and fixing partial solutions that remain stable despite demand fluctuations is feasible for the reoptimization problem.

To overcome these challenges, we employ a hybrid *ML-enhanced heuristic column generator*. First, ML models are trained for edge prediction ($\mathcal{ML}^E$) and break strategy prediction ($\mathcal{ML}^B$), identifying structural elements of the solution that are spatially efficient, regulatorily compliant, and robust to demand perturbations. While these predicted components are not guaranteed to appear in every optimal solution, they typically provide a backbone that significantly reduces the search space. This backbone can be incrementally refined in the subsequent component of our approach. Second, a dual-guided VNS metaheuristic is integrated. Using metaheuristics to find columns with negative reduced cost is well-established and has demonstrated its efficacy in numerous studies, such as Xue et al. (2021). In this work, the VNS neighborhood functions are specifically designed to permit deviations from ML predictions when justified by evolving dual values. At each RMP iteration, the dual-guided VNS systematically explores neighborhoods, leveraging dual values from the current LP relaxation to adaptively refine candidate solutions. This mechanism ensures the algorithm remains flexible, adjusts to the optimization context, and avoids over-reliance on potentially suboptimal ML predictions.

By combining structural insights from ML predictions with guidance from dual values from the RMP, our design maintains the mathematical rigor of column generation while achieving practical computational scalability. This establishes an adaptive feedback

Table 4
Notation for variables in the VNS column generator.

Symbol	Description
$\mathcal{S}$	Current solution set
$\mathcal{R}_s$	Set of routes in solution $\mathcal{S}$
n	Index of the current neighborhood structure
$n_{\max}$	Index of the largest (last) neighborhood function
ν, π_k	Dual variables from the RMP
$\bar{c}_r$	Reduced cost of route r, see Eq. (5)
$c_{\min}$	Best (minimum) reduced cost found in current VNS iteration
$c'_{\min}$	Minimum reduced cost in the newly generated neighborhood $\mathcal{R}'$
$maxIteration$	Maximum column generation iterations
$maxColumns$	Maximum number of routes in column pool
$columnPool$	Set of best routes (columns) for RMP
$\mathcal{P}_a$	ML-predicted component patterns (only for ML-enhanced VNS)
$useML$	Boolean flag: True = ML-enhanced VNS, False = basic VNS

loop in which ML provides guidance on promising regions of the solution space, and VNS dynamically responds to evolving dual values, thereby generating high-quality columns that respect all GPDPTW-DBS constraints. The approach is particularly advantageous in reoptimization scenarios, which are the primary focus of our work and involve incremental modifications to problem instances.

Algorithm 1 Column generation main loop (ML/Non-ML variants).

Require: Initial solution $\mathcal{S}$, $maxIteration$, $useML$ (boolean flag)
$j \leftarrow 0$
while $j < maxIteration$ **do**
  Solve RMP($\mathcal{S}$) to obtain dual variables ν, π_k ▷ Get dual prices
  if $useML$ **then**
    $\mathcal{P}a \leftarrow$ MLPredict($\mathcal{S}$) ▷ Get predicted patterns
    $columnPool \leftarrow$ VNS($\mathcal{S}, \nu, \pi_k, \mathcal{P}a, nmax, maxColumns$) ▷ ML-enhanced VNS
  else
    $columnPool \leftarrow$ VNS($\mathcal{S}, \nu, \pi_k, \emptyset, nmax, maxColumns$) ▷ Basic VNS (no ML)
  end if
  if $columnPool$ contains no new columns with $\bar{c}_r < 0$ **then**
    break ▷ No improving columns found; LP optimal
  end if
  $\mathcal{S} \leftarrow$ RMP($\mathcal{S} \cup columnPool$) ▷ Update RMP with new columns
  $j \leftarrow j + 1$
end while
return $\mathcal{S}$

Algorithm 2 Dual-guided VNS for column generation (ML/Non-ML variants).

Require: Current solution $\mathcal{S}$, dual variables ν, π_k, $\mathcal{P}_a$ (ML predictions), $n_{\max}$, $maxColumns$
$n \leftarrow 1$, update $\mathcal{R}_s$ from $\mathcal{S}$, $c_{\min} \leftarrow 0$
while $n \leq n_{\max}$ **do**
  $c'_{\min} \leftarrow$ minReducedCost($\mathcal{R}', \nu, \pi_k$) ▷ Finds best $\bar{c}_r$ using ν, π_k
  if $c'_{\min} < c_{\min}$ **then** ▷ Dual-based acceptance rule
    $n \leftarrow 1$, $c_{\min} \leftarrow c'_{\min}$ ▷ Intensify: Reset to the first neighborhood
    $columnPool \leftarrow$ sortByReducedCost($\mathcal{R}', \nu, \pi_k, maxColumns$)
    $columnPool \leftarrow columnPool \cup \mathcal{R}_s$
  else
    $n \leftarrow n + 1$ ▷ Diversify: switch to next neighborhood
  end if
end while
return $columnPool$

5.4.1. Neighborhood operators and ML integration

The complete algorithmic procedure is formally presented in Algorithm 1, with supporting notation detailed in Table 4. Our VNS implementation has two variants: *basic VNS* (no ML prediction) and *ML-enhanced VNS* (integrates ML-predicted fixed edges/driver

break strategy). Both variants employ four fundamental neighborhood operators: `Insert`, `Move`, `Swap`, and `Two-Move`, each designed to systematically explore different dimensions of the solution space. Our designed operators, though original in their implementation, build on well-established neighborhood structures used in solving classic VRP and PDPTW. They keep the same type of topological changes (e.g., adjusting route segments) as these established structures, while also adapting to the unique traits of our problem. Below are the definitions of the four neighborhood functions (with ML-specific behavior highlighted for ML-enhanced VNS):

- `Insert`: This operation inserts an unassigned request entity into an active vehicle route. In ML-enhanced VNS, a request entity is defined as either a single unassigned request or a pair of unassigned requests connected by a fixed edge predicted by $\mathcal{ML}^E$. In basic VNS, entities are limited to individual unassigned requests (no ML-predicted fixed edges). The entity is evaluated for insertion at all feasible positions in existing routes. If no feasible insertion exists, a new route is created to accommodate the entity.
- `Move`: This operator removes a request from its current route and evaluates its insertion into a different route. In both the ML-enhanced VNS and the basic VNS, the operator employs a first-improvement strategy: for a randomly selected request, feasible insertion positions across all other routes are evaluated, and the search stops at the first position that yields an improvement in solution quality (i.e., reduces total route duration). The distinction between the variants lies in the position evaluation. In the ML-enhanced VNS, the target insertion position is constrained by the highest-probability fixed edge predicted by $\mathcal{ML}^E$, meaning it is inserted immediately before or after a request with which it shares a predicted strong connection. In the basic VNS, insertion positions are not constrained by the predictions made by $\mathcal{ML}^E$. In both variants, if a request becomes unassigned during the move (e.g., due to constraint violations), it is subsequently reinserted by the `Insert` operator.
- `Swap`: The algorithm locates two requests in separate routes, removes each from its current route, and inserts them into the other's original route, subject to constraint satisfaction.
- `Two-Move`: The algorithm locates three requests from different routes, removes each from its current route, and inserts them into the subsequent route in the rotation sequence (1→2→3→1), subject to constraint validation. More specifically, the three-route rotation operates as follows: given three distinct routes r_1, r_2, and r_3, a request k_1 is selected from r_1 and a request k_2 from r_2. The operator first removes k_1 from r_1 and attempts to insert it into r_2, following the insertion logic of the `Move` operator. It then removes k_2 from r_2 and attempts to insert it into r_3. The movement thus follows the rotation sequence $k_1 \rightarrow r_2$, $k_2 \rightarrow r_3$, $k_3 \rightarrow r_1$. All insertions are subject to the feasibility validation of all constraints.

To ensure complete coverage of all requests during the search, a substantial penalty cost M is assigned to any unassigned request. This makes any move that would leave a request unassigned highly undesirable in the objective function, unless no feasible insertion position can be found.

In the ML-enhanced variant, the search is guided by the model's predictions while allowing deviations through the design of neighborhood functions. The `Move` operator is constrained to evaluate only the insertion position adjacent to the request with the highest predicted connection probability from $\mathcal{ML}^E$. If this specific move is infeasible or does not improve the solution, the request may become unassigned and is subsequently handled by the `Insert` operator. The `Insert` operator considers request entities that may be individual requests or ML-predicted pairs, treating each entity as a unit for insertion. The `Swap` operator intentionally operates independently of $\mathcal{ML}^E$ predictions, providing an explicit override mechanism. Thus, this design allows deviations from ML predictions and ensures continuous adaptation to the evolving LP solution: the acceptance of any move, including those that diverge from ML predictions, is ultimately governed by the reduced cost $\bar{c}_r$, computed using the current dual values (π_k, ν). As a result, the algorithm dynamically adjusts its search direction in response to changes in the dual values throughout the column generation process.

As an integral component of the solution process, the break scheduling module activates during route construction whenever accumulated duty or driving time exceeds regulatory thresholds ($L_b^{duty} \geq l^{break|work1st}$ or $L_b^{driving} \geq l^{break|drive}$). In the ML-enhanced VNS variant, when a route requires break assignment, its features are computed and passed to $\mathcal{ML}^B$, which predicts the break strategy. In the basic VNS variant, break schedules are generated using a deterministic heuristic rule (described in Section 5.3). Both variants handle constraint violations consistently. If adding a required driver break causes a route to exceed the maximum allowed daily driving time, a new route is created, and the last request from the violated route is reinserted using the `Insert` operator.

The ML predictions provide structural guidance that is both spatially efficient and regulatorily compliant. The edges predicted by $\mathcal{ML}^E$ typically connect geographically proximate nodes, while the break strategies predicted by $\mathcal{ML}^B$ not only satisfy mandatory driving regulations, but also predict the optimal break strategy given the route features. These patterns offer a stable foundation that reduces the initial search space while remaining robust to demand perturbations. Although not guaranteed to appear in every optimal solution, they provide a high-quality starting point that the optimization framework can efficiently refine.

Both $\mathcal{ML}^E$ and $\mathcal{ML}^B$ make predictions based solely on static features independent of the optimization state. Specifically, edge prediction features (Section 5.2) depend only on instance data and potential demand perturbations, while break strategy features (Section 5.3) depend only on route temporal structure and regulatory parameters. This independence ensures predictions remain valid throughout the column generation process, regardless of evolving dual values.

5.4.2. Column generation with dual-guided VNS algorithm

Our VNS approach explicitly leverages dual information from the LPR to guide the heuristic search process, representing a fundamental departure from traditional VNS implementations that typically rely on random selection mechanisms to explore the search space. As illustrated in Algorithm 1, our VNS algorithm is specifically designed to identify feasible routes with negative reduced costs for the RMP, rather than to solve the entire problem directly. Consequently, we omit the conventional *shaking* phase, since dual-guided reduced costs inherently direct the search toward promising regions of the solution space, rendering additional random

perturbations unnecessary and potentially detrimental to search efficiency. The efficacy of this targeted search strategy is empirically validated by the experiments presented in Section 6.1.

The column generation loop starts by solving the RMP to obtain dual variables π_k for the request coverage constraints and ν for the fleet size constraint (Algorithm 1, line 3). A boolean flag $useML$ determines whether machine learning predictions are integrated into the process. If $useML$ is set to True, the model predicts fixed components (Algorithm 1, line 6). These pairs are passed as $\mathcal{P}_a$ to the VNS subroutine (Algorithm 2). If $useML$ is set to False, the set $\mathcal{P}_a$ remains empty (Algorithm 1, line 8). Within this subroutine, each candidate route generated by applying the nth neighborhood structure has its reduced cost calculated, which quantifies its potential improvement over the current LP solution.

The search direction is governed by a dual-based acceptance rule. Routes from $\mathcal{R}'$ generated by neighborhood structure n are considered for acceptance based on their reduced costs. The neighborhood exploration is accepted (i.e., intensification occurs) only if the minimum reduced cost among routes in $\mathcal{R}'$ is lower than the best value found in the current VNS call ($c'_{\min} < c_{\min}$, as indicated in Algorithm 2, line 5). If this criterion is met, the algorithm accepts the improved routes, resets to explore from the first neighborhood structure ($n \leftarrow 1$), and updates the column pool with selected routes from $\mathcal{R}'$ sorted by reduced cost. When no improving route is found in $\mathcal{R}'$, the algorithm advances to the next neighborhood structure ($n \leftarrow n + 1$).

6. Computational experiments

To ensure a rigorous evaluation of our methods' effectiveness and solution quality, we used a dataset built from established benchmark instances (Xue et al., 2025a). The original test cases come from two sources: Li and Lim (2001) and Sartori and Buriol (2020).

The instances from Li and Lim (2001) are synthetic. Their 'c' instances feature clustered customer locations, while their 'r' instances have randomly dispersed customer locations. In contrast, the instances from Sartori and Buriol (2020) were generated using realistic geographic locations, offering scenarios closer to real-world applications. For detailed information regarding demand distributions and vehicle constraints, please refer to the original papers.

Our experimental setup includes 20 carefully chosen instances: 10 from each of the adapted datasets, equally divided between 100-request and 200-request problems. We preserved the original customer and depot coordinates and demand values from the Xue et al. (2025a) instances. However, we modified transportation times, service durations, and vehicle capacities to generate routes sufficiently long to trigger the diverse break requirements mandated by driving and duty regulations. Additionally, vehicle start and end locations were adapted to permit non-depot positions, and the number of requests was lowered to ensure a feasible assignment within the available vehicle fleet.

To ensure a rigorous and unbiased evaluation of the algorithm's base performance, all experimental runs in this study employ a *cold start* initialization strategy. This approach initializes the column pool exclusively using the basic routes generated through the method described in Section 5.1, deliberately excluding any prior solution knowledge. This methodological choice provides a conservative performance baseline, even though real-world reoptimization scenarios typically benefit from *warm starts*, which reuse columns from previous solutions to obtain better solution quality more quickly.

Since the solution process involves two distinct machine learning models (i.e., $\mathcal{ML}^E$ and $\mathcal{ML}^B$), each incorporating a machine learning component, we employed the Decision Tree algorithm from the scikit-learn (Pedregosa et al., 2011) library to train these models. Note that the predictions generated by the learned model need not be 100% accurate, as misclassifications may occur. The primary objective is to extract maximal knowledge from established schedules to enhance decision-making. These schedules encompass both routing and driver break assignments. The parameters of the classification algorithm were manually fine-tuned to optimize performance. We chose Decision Trees for this classification task due to their fast inference capability, even though they may not achieve the highest accuracy compared to other algorithms such as Neural Networks, Naive Bayes, or Support Vector Machines. However, since the inference process must be compatible with our column generator, minimizing inference time was a critical consideration. The quality score for a prediction is a probability value between 0 and 1, calculated by determining the proportion of each class in the leaf node where the instance ends up after traversing the decision tree. These trained decision trees were subsequently hard-coded into the proposed algorithm to ensure seamless integration and efficient execution.

For each instance, we generated a set of modified instances by randomly altering the demand values across all requests. Each modified instance $(\mathcal{P}_m^i)_\alpha^j$, where $i = 1, \ldots, 15$ and $j = 1, \ldots, 100$ was created by perturbing demand values uniformly within $[d_i(1-\alpha), d_i(1+\alpha)]$, where d_i is the original demand and $\alpha \in (0, 0.3)$ controls the perturbation level, ensuring $d_i(1-\alpha) > 0$. The 30% variation ($\alpha = 0.3$) represents the maximum realistic demand fluctuation.

We processed the data as follows: For each problem instance and each value of α, 100 modified instances were generated. Among these, 95 instances were allocated for the ML phase, which includes training and hyperparameter tuning, while the remaining 5 instances were reserved for testing (i.e., the optimization phase, where the ML model is integrated into the CG algorithm). During the generation of the training set, we recorded the break strategy (Compliant, Strategy 1, or Strategy 2) used to schedule driver breaks along each route. To train $\mathcal{ML}^B$ and ensure a balanced dataset (while mitigating potential model bias), we used downsampling: we selected an equal number of data instances from each of the three break strategy categories. Experiments were conducted on all 20 aforementioned instances, denoted as $\mathcal{P}_o^1, \ldots, \mathcal{P}_o^{20}$. Additionally, we generated 100 modified versions of each instance, represented as $\{(\mathcal{P}_m^i)_\alpha^1, \ldots, (\mathcal{P}_m^i)_\alpha^{100}\}$, where $i \in \{1, 2, \ldots, 20\}$. For each problem instance d_i, the value d_i was randomly adjusted by a factor of α.

Table 5
Summarized performance comparison with estimated lower bounds.

Instance	This Work (Heur. CG)		Xue et al. (2025)		This Work (Est. LB)	
	Cost	Time(s)	Cost	Time(s)	Cost	Time(s)
lc101-DBS	90.12 ± 0.11	705 ± 43	90.33 ± 0.18	986 ± 62	89.50	3123
lc102-DBS	95.38 ± 0.17	340 ± 14	95.72 ± 0.23	474 ± 38	94.68	3176
lc103-DBS	83.55 ± 0.25	218 ± 53	85.62 ± 0.25	347 ± 32	82.95	3322
lc104-DBS	64.10 ± 0.14	525 ± 24	65.94 ± 0.20	791 ± 55	63.65	3628
lc105-DBS	89.50 ± 0.15	282 ± 53	90.08 ± 0.19	465 ± 40	88.80	3434
lr203-DBS	38.92 ± 0.14	528 ± 12	40.06 ± 0.11	755 ± 58	38.68	4402
lr204-DBS	34.77 ± 0.05	432 ± 21	35.77 ± 0.10	669 ± 49	34.55	4228
lr205-DBS	65.08 ± 0.11	702 ± 34	65.93 ± 0.16	1146 ± 75	64.60	4534
lr206-DBS	59.78 ± 0.09	978 ± 51	59.69 ± 0.12	1459 ± 92	59.35	4982
lr207-DBS	55.66 ± 0.21	928 ± 43	55.86 ± 0.10	1428 ± 88	55.27	4815
ber-n100-3-DBS	193.78 ± 0.17	932 ± 64	194.29 ± 0.38	1272 ± 95	192.20	4027
ber-n100-4-DBS	174.12 ± 0.42	838 ± 43	174.71 ± 0.33	1167 ± 85	172.70	3903
ber-n100-5-DBS	310.75 ± 0.53	1018 ± 62	311.15 ± 0.60	1567 ± 110	308.10	5133
ber-n100-6-DBS	49.51 ± 0.12	328 ± 21	49.23 ± 0.08	443 ± 35	49.15	3018
ber-n100-7-DBS	57.17 ± 0.26	380 ± 52	57.39 ± 0.11	556 ± 42	56.75	3192
bar-n200-3-DBS	520.88 ± 0.67	703 ± 13	521.18 ± 0.97	946 ± 72	516.45	5236
bar-n200-5-DBS	431.85 ± 0.53	592 ± 53	432.61 ± 0.80	861 ± 65	428.05	4775
bar-n200-6-DBS	885.70 ± 1.36	1342 ± 74	885.76 ± 1.60	1872 ± 135	877.75	5258
bar-n200-7-DBS	374.28 ± 0.43	968 ± 81	374.77 ± 0.70	1534 ± 115	370.98	5027
ber-n200-6-DBS	209.18 ± 0.33	765 ± 62	210.11 ± 0.40	1074 ± 82	207.48	4956
Average	**194.20 ± 0.31**	**675.20 ± 44**	**194.81 ± 0.36**	**990.60 ± 73.8**	**192.58**	**4208**

6.1. Performance of the proposed column generation approach (Without ML enhancement)

The computational efficiency of our column generation approach is governed by a critical trade-off in setting the parameter `maxColumns`. Since solving the restricted master problem (RMP) accounts for the majority of computational expense, we introduce multiple columns per iteration to amortize this cost. However, an excessively small `maxColumns` value necessitates more frequent RMP solves, while an overly large value increases each RMP's solution time, in the limiting case where all feasible columns are included, the RMP becomes computationally equivalent to solving the original problem. Empirical results from Section A.6 indicate that parameter values `maxColumns` = 1000 and `maxIteration` = 30 achieve an effective balance between solution quality and computational effort. Furthermore, our methodology permits temporary violation of the vehicle fleet size constraint during initial exploration phases to facilitate broader solution space coverage. Our implementation maintains this constraint relaxation only during the exploratory phase, with full compliance restored before solution output. For RMP optimization, we employed Gurobi 11 with default settings and ran it on an 8-core Intel i7-2.99 GHz processor with 16 GB RAM. We used the same machine for all the experiments in this study. All experiments in this paper are based on 30 independent runs, performed on the same machine.

The performance of the proposed heuristic CG approach is evaluated against the benchmark method of Xue et al. (2025a). To justify the approximation introduced by our heuristic pricing strategy and provide a lower-bound guarantee on solution quality, we implemented an exact pricing problem solved via the ESPPRC method (see Appendix for details). The results of this exact pricing analysis are reported in the column labeled `Est. LB` of Table 5. To ensure a fair and consistent comparison with established benchmarks in the literature (Kok et al., 2010; Prescott-Gagnon et al., 2010; Goel, 2018), the hours-of-service rules in this experimental study are simplified. Specifically, a driving break is defined strictly as either a single off-duty period of at least 45 min or a sequence of two off-duty periods where the first is at least 15 min and the second is at least 30 min. This modeling choice guarantees compliance with the minimum requirements of Regulation (EC)561/2006. However, it does not exploit the full scheduling flexibility allowed by a broader interpretation of break patterns, which could yield more efficient or realistic schedules. A detailed discussion of this limitation and an illustrative example are provided in Section A.1 in the appendix.

The results in Table 5 demonstrate the effectiveness of the proposed heuristic CG approach. A paired t-test confirms that our method achieves statistically significant, marginally better solution quality than the benchmark by Xue et al. (2025a) ($p < .01$), with a mean cost of 194.20 compared to their mean of 194.81. Computationally, our heuristic CG is also significantly more efficient, completing in approximately 1.5 times less time on average (675.20s vs. 990.60s). To quantify the quality of these solutions, we computed estimated lower bounds (Est. LB) by solving the pricing problem exactly via the ESPPRC method. The results show that the estimated lower bounds are, on average, approximately 0.84% below the cost of our solutions ($p < .05$). This provides a certificate of quality, indicating our solutions are within a small, well-bounded gap of optimality. However, generating this bound is computationally intensive, requiring an average of 4208 s. The significant difference between the runtime of our heuristic CG approach (675.20s) and the bound calculation time (4208s) ($p < .001$) underscores the fundamental trade-off between solution quality and computational effort in optimization.

6.2. Performance comparison of the ML-enhanced column generation approach

This section presents a performance comparison of our ML-enhanced column generation approach, based on three experimental rounds. Each round employed the heuristic column generator after applying different ML components to assess their individual and combined effectiveness: (1) *Round 1: Edge-Fixing Algorithm* evaluated the effectiveness of the edge-fixing component in enhancing the column generator's ability to produce high-quality solutions compared to the benchmark (the standalone heuristic column generator); (2) *Round 2: Break Strategy-Fixing Algorithm* assessed the contribution of the break strategy-fixing component to solution quality; and (3) *Round 3: Full ML-Enhanced Algorithm* tested the integrated approach, applying both ML components to evaluate the complete algorithm's performance in improving both solution quality and computational efficiency, representing the main contribution of this work.

As mentioned earlier, for each instance, 100 modified instances were generated, resulting in a total of 20 instances × 100 modified instances = 2000 generated instances. Upon obtaining S_o and S_m, we extract the requisite training ans testing sets, features and labels according to the methodology presented in Section 5. Tables 6 and 7 summarize the results obtained for the test instances derived from the modified synthetic benchmark (Li and Lim, 2001) and the real-life benchmark (Sartori and Buriol, 2020), respectively. Each row represents the average values computed across the five test instances.

6.2.1. Similarity between S_o and S_m of edge-fixing

The first column of Tables 6 and 7 (which share the same structure) lists the original problem instances $\mathcal{P}_o$, while the second column quantifies the similarity between their solutions S_o and the modified solutions S_m, computed using the following formula:

$$\text{Similarity}(S_o, S_m) = \frac{|S_o \cap S_m|}{|S_o|}$$

The similarity metric, which corresponds to the percentage of edges that are fixed (i.e., those labeled as 1), was obtained from Morabit et al. (2024). This metric plays a significant role in influencing both the quality of the solution and the computational time required to obtain it. When certain edges are not fixed despite being necessary (false negatives), the computational time may increase, as the column generator might still need to incorporate these edges into the final solution. On the other hand, edges labeled as 0 that are inaccurately classified (false positives) can lead to a larger optimality gap, potentially compromising solution quality. Note that the weighting of edges can be adjusted based on the desired trade-off between solution quality and computational efficiency. Assigning a higher weight to edges labeled as 0 can reduce false positives, resulting in fewer edges being fixed. While this approach may improve solution quality, it could also lead to longer computational times. Conversely, assigning a higher weight to edges labeled as 1 increases the likelihood of fixing more edges, which reduces computational time but may negatively impact solution quality. In our approach, we employ a balanced weighting scheme between the two classes to achieve a reasonable compromise between solution quality and computational efficiency. A more detailed analysis of these results can be found in Section A.3 in the appendix. However, in practical applications, the choice of weighting should be guided by the specific objectives of the task, aiming to strike an optimal balance tailored to the problem at hand.

6.2.2. Accuracy of the ML models: $\mathcal{ML}^E$ vs. $\mathcal{ML}^B$

The next two columns adjacent to `Similarity` report the test accuracy (denoted as `Accuracy` in the table) of the machine learning models $\mathcal{ML}^E$ and $\mathcal{ML}^B$. The ten-fold cross-validation accuracy rates for the decision tree across the various datasets for $\mathcal{ML}^E$ range from 73% to 87%, while the average accuracy of $\mathcal{ML}^B$ ranges from 75% to 87%. However, a closer examination of individual instances reveals variance and a slight positive correlation between accuracy and similarity. Specifically, the similarity increases as the amplitude of the demand changes decreases; this amplitude is governed by the parameter α, which dictates the difference between the original and modified instances. Instances with higher similarity are easier to predict, leading to higher accuracy, which aligns with expectations. Furthermore, by fixing the edges proposed by the model, we find that solution quality follows a similar trend to similarity. Greater changes in demands result in lower similarity, an increased likelihood of misclassifications, and a subsequent decline in solution quality. Consequently, instances with higher similarity tend to exhibit better solution quality compared to those with lower similarity.

6.2.3. Solution quality and computational efficiency

The next two columns following the `Accuracy` column report the cost (summarized as travel duration in hours) and elapsed time in seconds for the column generation heuristic (denoted as `CG`) without Edge-fixing or Break strategy-fixing. The results using the Edge-fixing algorithm followed by the heuristic column generator are summarized under the column `Edge-fixing+CG`. Similarly, the results using the Break strategy-fixing algorithm followed by the heuristic column generator are summarized under the column `Break-fixing+CG`. Finally, the results where both the Edge-fixing algorithm and Break strategy-fixing were applied are summarized under the column `Edge&Break-fixing+CG`.

As can be seen from Tables 6 and 7, the metrics evaluated are solution quality (`cost`, measured as the cost in travel duration) and computational efficiency (`Time(s)`, measured as the elapsed time in seconds). For the Li and Lim (2001) dataset, the baseline CG achieves an average cost of 67.68 with an elapsed time of 563.8 s. `Edge-fixing+CG` increases the average cost to 68.88 (a 1.8% increase) while reducing the elapsed time to 20.9 s (a 96.3% reduction). `Break-fixing+CG` results in an average cost of 68.62 (a 1.4% increase) and an elapsed time of 40.9 s (a 92.7% reduction). The combined approach (`Edge&Break-fixing+CG`) achieves an average cost of 68.94 (a 1.9% increase) and reduces the elapsed time to 17.4 s (a 96.9% reduction). The improvement in computational time for all enhanced methods over the baseline is statistically significant ($p < .001$), as confirmed by a paired t-test.

Table 6
Performance metrics across different instances (Li and Lim, 2001).

Instance	Similarity	Accuracy		CG		Edge-fixing + CG		Break-fixing + CG		Edge& Break-fixing + CG	
		$\mathcal{ML}^E$	$\mathcal{ML}^B$	Cost	Time(s)	Cost	Time(s)	Cost	Time(s)	Cost	Time(s)
lc101-DBS	61%	77%	76%	90.12 ± 0.11	705 ± 43	91.75 ± 0.19	27.9 ± 4.8	91.21 ± 0.18	47.2 ± 5.7	91.84 ± 0.19	23.5 ± 3.6
lc102-DBS	68%	80%	78%	95.38 ± 0.17	340 ± 14	97.07 ± 0.24	15.0 ± 2.9	96.77 ± 0.23	26.4 ± 4.5	97.18 ± 0.25	11.1 ± 2.2
lc103-DBS	55%	81%	82%	83.55 ± 0.15	218 ± 53	85.03 ± 0.15	6.2 ± 1.8	84.79 ± 0.19	15.8 ± 2.1	85.12 ± 0.16	5.9 ± 1.3
lc104-DBS	58%	74%	80%	64.10 ± 0.14	525 ± 42	65.22 ± 0.16	22.3 ± 4.9	64.93 ± 0.15	40.1 ± 6.2	65.29 ± 0.17	17.8 ± 3.9
lc105-DBS	55%	75%	78%	89.50 ± 0.15	282 ± 53	91.08 ± 0.19	8.7 ± 2.3	90.79 ± 0.18	18.4 ± 4.1	91.17 ± 0.21	8.5 ± 2.0
lr203-DBS	68%	73%	75%	38.92 ± 0.14	528 ± 43	39.59 ± 0.12	14.7 ± 3.7	39.38 ± 0.11	38.0 ± 5.8	39.64 ± 0.13	13.4 ± 3.1
lr204-DBS	64%	79%	86%	34.77 ± 0.05	432 ± 35	35.37 ± 0.11	11.1 ± 3.2	35.26 ± 0.10	31.2 ± 4.8	35.40 ± 0.12	10.6 ± 2.8
lr205-DBS	72%	84%	86%	65.08 ± 0.11	702 ± 57	66.20 ± 0.17	29.6 ± 5.1	65.96 ± 0.16	54.7 ± 7.3	66.27 ± 0.18	22.7 ± 4.2
lr206-DBS	81%	82%	81%	59.78 ± 0.10	978 ± 80	60.88 ± 0.16	44.6 ± 6.8	60.64 ± 0.15	70.8 ± 8.9	60.97 ± 0.17	33.2 ± 5.5
lr207-DBS	73%	82%	85%	55.66 ± 0.09	928 ± 76	56.62 ± 0.15	29.0 ± 5.4	56.45 ± 0.14	66.8 ± 7.6	56.68 ± 0.16	26.9 ± 4.7
Average	**65.5%**	**79.7%**	**80.7%**	**67.68 ± 0.21**	**563.8 ± 49.6**	**68.88 ± 0.16**	**20.9 ± 4.1**	**68.62 ± 0.16**	**40.94 ± 5.7**	**68.96± 0.17**	**17.4± 3.3**

For the Sartori and Buriol (2020) dataset, the baseline CG achieves an average cost of 320.72 with an elapsed time of 786.6 s. `Edge-fixing+CG` increases the average cost to 326.30 ($SD = 0.64$; a 1.7% increase) while reducing the elapsed time to 35.7 s (a 95.5% reduction). `Break-fixing+CG` results in an average cost of 325.34 (a 1.4% increase) and an elapsed time of 56.1 s (a 92.9% reduction). The combined approach achieves an average cost of 326.64 (a 1.9% increase) and reduces the elapsed time to 24.2 s (a 96.9% reduction). The observed reductions in runtime are highly significant ($p < .001$).

The results show that both Edge-fixing and Break strategy-fixing dramatically reduce computational time across both datasets. The combined approach yields the most substantial time savings, with an average reduction exceeding 96%. These findings confirm that the proposed method effectively identifies high-quality columns for solving the GPDPTW-DBS. All enhanced methods maintain solution quality within a 2% cost increase relative to the baseline CG, which is below a practically meaningful threshold. The solutions generated are, on average, approximately 2.7% above the estimated lower bounds for the Li and Lim dataset and approximately 1.8% above for the Sartori and Buriol dataset. This indicates that the ML-guided fixing strategies achieve remarkable computational speedups while producing solutions very close to optimality. Both Edge-fixing and Break-fixing accelerate the solving process from distinct angles. Edge-fixing reduces the size of the problem, while Break-fixing cuts down on time-related variable updates at nodes. This efficiency gain remains attributed to the problem-describing features that align with Regulation (EC)561/2006 and Directive 2002/15/EC. Edge-fixing's effectiveness still stems from the large number of modified training instances, which is evident in the stronger accuracy and quality of higher-similarity instances. Current route initialization uses simple heuristics, but practical deployments could further boost computational performance by leveraging frequently observed routes and break patterns from historical data.

6.2.4. Runtime distribution and online inference

The solution process incorporates two distinct machine learning models: $\mathcal{ML}^E$ for edge prediction and $\mathcal{ML}^B$ for break strategy prediction. Both models were trained using the Decision Tree algorithm from the scikit-learn library (Pedregosa et al., 2011). The training phase was efficient, taking approximately 290 s for $\mathcal{ML}^E$ (trained on 319,826 edge instances) and 50 s for $\mathcal{ML}^B$ (trained on 47,683 route instances) on an Intel i7 2.40 GHz processor with 4 GB RAM.

During the online ML-enhanced CG phase, the trained models are used only for inference. Their integration into the algorithm, however, differs due to the nature of their predictions. For $\mathcal{ML}^E$, edge predictions are computed once as a pre-processing step before the VNS begins. The model provides a probability score for each possible connection between requests. Since a request cannot have multiple immediate successors in a valid route, we implement a mutual exclusivity rule by selecting only the connection with the highest predicted probability. This results in a dictionary where each request is paired with at most one successor, forming a set of fixed request pairs. This static set of predicted pairs is used throughout the column generation process without further modification.

For $\mathcal{ML}^B$, predictions must be made dynamically during route construction, since features such as route duration depend on the evolving state of the route. To minimize overhead, the learned decision tree is integrated directly into the column generation logic as a series of hard-coded if-else conditions.

The computational cost of these ML components during the online phase is minimal across all tested instances. As shown in Table 8, which details the runtime breakdown for all 20 instances across 30 independent runs, the combined time for $\mathcal{ML}^E$ and $\mathcal{ML}^B$ inference consistently accounts for only 0.9% to 2.2% of the total runtime, with standard deviations of 0.1–0.4% across runs. The vast majority of computational effort is spent on the core optimization components: solving the RMP and the neighborhood search within VNS. The distribution remains stable across instances with different sizes and characteristics: RMP solving consumes 69.1–78.9% and VNS consumes 19.4–29.3% of the time, with standard deviations of 0.4–1.4% for both components.

6.2.5. Sensitivity analysis

We conducted a sensitivity analysis to evaluate the impact of demand change magnitude (α) on the performance of our ML-assisted column generation framework, specifically comparing the edge-fixing ($\mathcal{ML}^E$) and break strategy-fixing ($\mathcal{ML}^B$) strategies. The key observation is that the efficacy of the edge-fixing approach is highly sensitive to the value of α, with higher disturbance

Table 7
Performance metrics across different instances (Sartori and Buriol, 2020).

Instance	Similarity	Accuracy		CG		Edge-fixing + CG		Break-fixing + CG		Edge&Break-fixing + CG	
		$\mathcal{ML}^E$	$\mathcal{ML}^B$	Cost	Time(s)	Cost	Time(s)	Cost	Time(s)	Cost	Time(s)
ber-n100-3-DBS	64%	78%	78%	193.78 ± 0.17	932 ± 64	197.16 ± 0.39	46.4 ± 5.9	196.58 ± 0.38	67.7 ± 7.4	197.34 ± 0.40	27.5 ± 3.8
ber-n100-4-DBS	71%	81%	80%	174.12 ± 0.42	838 ± 43	177.16 ± 0.35	32.3 ± 4.5	176.64 ± 0.34	61.7 ± 7.2	177.34 ± 0.36	27.1 ± 4.1
ber-n100-5-DBS	68%	83%	81%	310.75 ± 0.53	1018 ± 62	316.13 ± 0.64	36.0 ± 5.2	315.20 ± 0.62	74.9 ± 8.9	316.44 ± 0.65	32.9 ± 4.6
ber-n100-6-DBS	71%	75%	80%	49.51 ± 0.12	328 ± 21	50.47 ± 0.12	15.1 ± 2.8	50.22 ± 0.11	21.3 ± 3.5	50.51 ± 0.13	9.7 ± 1.9
ber-n100-7-DBS	68%	78%	79%	57.17 ± 0.26	380 ± 52	58.18 ± 0.14	19.3 ± 3.4	58.01 ± 0.13	26.0 ± 4.2	58.23 ± 0.15	11.9 ± 2.3
bar-n200-3-DBS	71%	76%	76%	520.88 ± 0.67	703 ± 13	530.19 ± 1.01	37.2 ± 5.6	528.62 ± 0.99	46.4 ± 6.3	530.70 ± 1.02	21.6 ± 3.5
bar-n200-5-DBS	72%	81%	87%	431.85 ± 0.53	592 ± 53	439.28 ± 0.85	29.9 ± 4.8	437.98 ± 0.83	44.0 ± 6.1	439.71 ± 0.86	19.5 ± 3.2
bar-n200-6-DBS	75%	87%	86%	885.70 ± 1.36	1342 ± 74	900.90 ± 1.71	51.8 ± 7.5	898.25 ± 1.68	98.2 ± 10.7	901.79 ± 1.72	41.5 ± 5.9
bar-n200-7-DBS	74%	85%	80%	374.28 ± 0.43	968 ± 81	380.75 ± 0.75	47.2 ± 6.9	379.63 ± 0.73	66.8 ± 8.4	381.13 ± 0.76	28.0 ± 4.5
ber-n200-6-DBS	76%	81%	86%	209.18 ± 0.33	765 ± 62	212.85 ± 0.45	41.4 ± 6.2	212.23 ± 0.43	53.9 ± 7.6	213.16 ± 0.46	22.6 ± 3.8
Average	**71.0%**	**80.5%**	**81.3%**	**320.72 ± 0.48**	**786.6 ± 52.5**	**326.30 ± 0.64**	**35.7 ± 5.3**	**325.34 ± 0.63**	**56.1 ± 7.03**	**326.64 ± 0.65**	**24.2 ± 3.8**

Table 8
Runtime distribution (percentage) of algorithm components across all instances (Mean ± standard deviation over 30 runs).

Instance	ML^E (%)	ML^B (%)	VNS (%)	RMP (%)
lc101-DBS	0.7 ± 0.1	0.5 ± 0.1	24.6 ± 0.8	74.2 ± 0.9
lc102-DBS	0.8 ± 0.1	0.6 ± 0.1	26.3 ± 1.0	72.3 ± 1.1
lc103-DBS	0.9 ± 0.2	0.7 ± 0.1	28.5 ± 1.2	69.9 ± 1.3
lc104-DBS	0.6 ± 0.1	0.5 ± 0.1	23.8 ± 0.7	75.1 ± 0.8
lc105-DBS	0.8 ± 0.1	0.6 ± 0.1	25.4 ± 0.9	73.2 ± 1.0
lr203-DBS	1.0 ± 0.2	0.5 ± 0.1	20.8 ± 0.6	77.7 ± 0.7
lr204-DBS	1.1 ± 0.2	0.6 ± 0.1	19.4 ± 0.5	78.9 ± 0.6
lr205-DBS	1.3 ± 0.2	0.7 ± 0.1	22.1 ± 0.7	75.9 ± 0.8
lr206-DBS	1.5 ± 0.3	0.6 ± 0.1	21.6 ± 0.7	76.3 ± 0.8
lr207-DBS	1.2 ± 0.2	0.6 ± 0.1	21.6 ± 0.7	76.6 ± 0.8
ber-n100-3-DBS	0.6 ± 0.1	0.5 ± 0.1	27.8 ± 1.2	71.1 ± 1.1
ber-n100-4-DBS	0.7 ± 0.1	0.6 ± 0.1	26.9 ± 1.1	71.8 ± 1.2
ber-n100-5-DBS	0.9 ± 0.2	0.7 ± 0.1	29.3 ± 1.3	69.1 ± 1.4
ber-n100-6-DBS	0.5 ± 0.1	0.4 ± 0.1	24.2 ± 0.8	74.9 ± 0.9
ber-n100-7-DBS	0.6 ± 0.1	0.5 ± 0.1	25.7 ± 0.9	73.2 ± 1.0
bar-n200-3-DBS	1.1 ± 0.2	0.5 ± 0.1	20.2 ± 0.6	78.2 ± 0.7
bar-n200-5-DBS	1.2 ± 0.2	0.6 ± 0.1	21.5 ± 0.7	76.7 ± 0.8
bar-n200-6-DBS	1.4 ± 0.3	0.6 ± 0.1	22.8 ± 0.8	75.2 ± 0.9
bar-n200-7-DBS	1.3 ± 0.2	0.5 ± 0.1	21.9 ± 0.7	76.3 ± 0.8
ber-n200-6-DBS	1.3 ± 0.2	0.6 ± 0.1	20.5 ± 0.7	77.6 ± 0.8
Average	0.98 ± 0.17	0.58 ± 0.07	23.74 ± 2.45	74.73 ± 2.61

levels leading to decreased solution similarity, lower prediction accuracy, and consequently higher computational time and solution cost. In contrast, the break strategy prediction model demonstrates remarkable robustness, maintaining stable accuracy and solution quality across all α values, as it operates independently of the underlying routing structure. This divergence highlights the distinct mechanistic roles of the two strategies: edge-fixing directly constraints the solver's search space and is thus vulnerable to problem instability, while break strategy-fixing acts as an efficient auxiliary process that remains reliable regardless of demand variations. We also conducted sensitivity analysis on the key algorithm parameters *maxColumns* and *maxIterations* of the algorithm. For more detailed results and discussions, please refer to the Section A.3 in the Appendix.

7. Conclusion

This paper presents a rapid reoptimization approach that integrates routing and driver break scheduling. The method generates routes with minimal duration while explicitly addressing mandatory driver breaks. Our heuristic column generation framework, enhanced with machine learning, is applicable to diverse problem variants where duration-related costs are critical. The framework excludes mileage-based or labor-related payments, making it suitable for applications such as supermarket, convenience store, and e-commerce deliveries.

The ML-enhanced approach ensures compliance with break requirements while achieving significant computational gains. Runtime is reduced by over 96% to just seconds, with routing costs increasing by less than 2% and a solution gap of only 2.7% above the estimated lower bound. The framework's strength lies in its ability to effectively manage the complexity of simultaneous routing and break scheduling.

In practice, many GPDPTW-DBS instances are solved repeatedly with unchanged objectives and scales but updated data. By training ML models for edge- and break strategy-fixing, our method identifies unchanged solution components and reduces the search space. This results in substantial computational time savings, as demonstrated on both synthetic and real-world benchmarks. The three break strategy labels, derived from previous studies, can be generalized as long as the strategies are enumerable for classifier training. The classifier for edge-fixing assigns equal weights to the labels. In practice, assigning a higher weight to edges labeled as 1 increases the likelihood of fixing more edges. While this can reduce computational time, it may also negatively impact solution quality, presenting a trade-off that merits further exploration in real-world applications.

Ideally, the machine learning model should be trained on optimal solutions. However, due to the complexity of solving the GPDPTW-DBS, it was infeasible to generate a sufficient volume of optimal training data for this study. Consequently, we utilized near-optimal solutions instead. Although the ML prediction is only one component of our solution framework and is subsequently refined by the heuristic column generation process, we posit that the quality of the final solutions and the computational efficiency of the process could be further enhanced if optimal solutions were used to train the ML models.

Future research directions include enhancing edge-fixing features through expanded feature engineering to improve prediction accuracy and solution quality. Additionally, by leveraging instance similarity, we propose defining features for individual columns. These columns represent routes with break schedules that can be filtered and prioritized to identify promising solutions for the restricted master problem. This approach could further improve efficiency and scalability while providing valuable insights for broader applications.

Acknowledgements

This Project is Supported by the Ningbo Science and Technology Bureau (Ningbo Commonweal Programme, Project ID 2024S057, 2025S114; Ningbo Young Scientific and Technological Innovation Leading Talent Scheme, Project ID 2025QL055).

Author Statement

There is no conflict of interest for this study.

CRediT authorship contribution statement

Ning Xue: Writing – original draft, Visualization, Validation, Methodology, Investigation, Formal analysis, Data curation, Conceptualization; **Tianxiang Cui:** Writing – review & editing, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization; **Shi Cheng:** Writing – review & editing, Resources.

Data availability

Data will be made available on request.

Declaration of interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Appendix

Minutes of service regulations in Australia and New Zealand

Adaptation of the ESPPRC framework

To establish a lower bound for evaluating our proposed heuristic algorithm, we implemented an exact method for solving the pricing problem. This method is based on the dynamic programming framework for the Elementary Shortest Path Problem with Resource Constraints (ESPPRC), as introduced by Goel (2018). We significantly tailored this framework to fit the specific context of our single-day problem. The most critical adaptation stems from the fundamental difference in problem scope. The original model by Goel (2018) is designed for long-haul operations involving multi-day scheduling. This necessitates the tracking of extended rest periods (e.g., 10-h daily rests) that reset a driver's state for a new duty cycle. In contrast, our problem is confined to a single-day operation. Consequently, we simplified the resource vector by eliminating the labels l^{latest}, f^{dayrest}, and $f^{\text{nightrest}}$, which are essential for tracking time-related states since the last major rest in a multi-day context but are redundant for our single-day model. This modification streamlines the state space and transition rules, enhancing the algorithm's efficiency for our specific application.

A second critical adaptation concerns the modeling of break scheduling flexibility. Although the method of Goel (2018) effectively handles the interaction between Directive 2002/15/EC and Regulation (EC) No 561/2006, it enforces a rigid break scheduling rule: a qualifying break must consist of either a single 45-min off-duty period or a fixed sequence of a 15-min break followed by a 30-min break. This inflexibility fails to accommodate the more fluid break patterns observed in real-world operations. To illustrate this limitation, consider a scenario with three consecutive break periods of 30 min, 15 min, and 30 min. The first 30-min break satisfies the requirement from Directive 2002/15/EC. Under the rigid rule of Goel (2018), the first two breaks (30 min + 15 min) would not be considered a valid combined break for Regulation (EC) 561/2006, as the rule mandates a specific sequence (15 min then 30 min). However, if the evaluation window is shifted, the subsequent pair of breaks (15 min + 30 min) would satisfy the rule. This arbitrary constraint highlights an impractical limitation.

Therefore, in our tailored model, we relax this rule to better reflect practical logistics. We allow any sequence of two short breaks that cumulatively meet the 45-min requirement (e.g., 30 min followed by 15 min) to be recognized as a valid split break, provided the individual segments are at least 15 min. This enhancement increases the model's flexibility and its alignment with actual driver scheduling practices.

Sensitivity analysis

This study is primarily motivated by the need for fast re-optimization algorithms for the GPDP and integrated driver break scheduling. To this end, we propose two distinct machine learning-based heuristics: one for fixing a subset of routing edges ($\mathcal{ML}^E$) and another for predicting the optimal driver break strategy for a given route ($\mathcal{ML}^B$), following a route-first-schedule-second paradigm.

To precisely isolate and quantify the individual impact of each heuristic, we conducted two separate sets of experiments. The first set focuses exclusively on evaluating the sensitivity of the edge-fixing heuristic ($\mathcal{ML}^E$), applying it to generate solutions while deliberately omitting the break strategy-fixing component. Conversely, the second set of experiments is dedicated to analyzing the

Table A.1
Parameters of minutes of service regulations in Australia and New Zealand.

Notation	Australia		New Zealand
	SH (Minutes)	BFM (Minutes)	WTR(Minutes)
$l^{break\|1st}$	30	30	30
$l^{break\|2nd}$	60	60	60
$l^{break\|work1st}$	315	360	330
$l^{break\|work2nd}$	615	690	690

Table A.2
Impact of demand change magnitude (α) on edge-fixing performance.

Instance	Similarity (%)			Accuracy $\mathcal{ML}^E$ (%)			Cost			Time (s)		
	$\alpha = 10$	$\alpha = 20$	$\alpha = 30$	$\alpha = 10$	$\alpha = 20$	$\alpha = 30$	$\alpha = 10$	$\alpha = 20$	$\alpha = 30$	$\alpha = 10$	$\alpha = 20$	$\alpha = 30$
lc101-DBS	78	69	58	85	78	73	90.2 ± 0.18	91.8 ± 0.21	92.4 ± 0.23	21.4 ± 2.9	25.0 ± 3.2	28.2 ± 3.5
lc102-DBS	81	72	65	86	80	76	95.6 ± 0.22	97.2 ± 0.25	98.9 ± 0.28	12.2 ± 2.1	13.8 ± 2.3	15.1 ± 2.5
lc103-DBS	74	63	52	84	79	77	83.5 ± 0.17	84.1 ± 0.18	84.6 ± 0.19	4.6 ± 1.5	5.4 ± 1.6	6.5 ± 1.8
lc104-DBS	76	66	55	81	74	70	64.2 ± 0.14	64.6 ± 0.15	64.9 ± 0.16	18.9 ± 2.8	20.9 ± 3.0	22.5 ± 3.2
lc105-DBS	74	63	52	82	75	71	89.5 ± 0.19	91.1 ± 0.21	92.6 ± 0.23	7.1 ± 1.7	8.0 ± 1.9	8.8 ± 2.1
lr203-DBS	81	72	65	80	72	69	38.9 ± 0.10	39.1 ± 0.10	39.1 ± 0.10	12.2 ± 2.3	13.5 ± 2.5	14.9 ± 2.7
lr204-DBS	77	68	61	85	79	75	34.9 ± 0.09	35.3 ± 0.09	35.6 ± 0.09	9.7 ± 1.9	10.4 ± 2.1	11.4 ± 2.3
lr205-DBS	84	76	68	87	83	80	65.3 ± 0.15	66.5 ± 0.16	67.7 ± 0.17	25.5 ± 3.4	27.5 ± 3.6	29.9 ± 3.8
lr206-DBS	87	82	77	88	84	78	60.0 ± 0.14	60.7 ± 0.15	61.4 ± 0.16	38.8 ± 4.5	41.8 ± 4.7	44.8 ± 4.9
lr207-DBS	86	77	69	86	81	78	55.8 ± 0.13	56.3 ± 0.14	56.8 ± 0.15	25.5 ± 3.3	27.3 ± 3.5	29.4 ± 3.7
ber-n100-3-DBS	77	68	61	83	76	74	194.2 ± 0.38	197.4 ± 0.41	200.5 ± 0.44	41.8 ± 4.8	44.9 ± 5.0	47.5 ± 5.2
ber-n100-4-DBS	83	75	67	85	80	77	174.0 ± 0.35	174.8 ± 0.36	175.6 ± 0.37	29.6 ± 3.7	31.4 ± 3.9	33.4 ± 4.1
ber-n100-5-DBS	80	72	65	85	81	79	310.5 ± 0.61	311.8 ± 0.62	313.0 ± 0.63	32.6 ± 3.9	34.7 ± 4.1	36.9 ± 4.3
ber-n100-6-DBS	83	75	67	82	74	71	49.7 ± 0.12	50.2 ± 0.13	50.6 ± 0.14	13.8 ± 2.4	14.6 ± 2.6	15.6 ± 2.8
ber-n100-7-DBS	80	72	65	84	77	75	56.9 ± 0.13	57.0 ± 0.13	57.0 ± 0.13	17.9 ± 2.7	18.9 ± 2.9	19.9 ± 3.1
bar-n200-3-DBS	83	75	67	81	74	72	520.5 ± 0.98	525.2 ± 1.00	529.9 ± 1.02	34.2 ± 4.1	36.2 ± 4.3	38.1 ± 4.5
bar-n200-5-DBS	84	76	68	85	80	77	430.5 ± 0.82	431.6 ± 0.83	432.7 ± 0.84	27.5 ± 3.5	29.1 ± 3.7	30.8 ± 3.9
bar-n200-6-DBS	86	78	71	89	86	83	882.9 ± 1.66	888.8 ± 1.68	894.7 ± 1.70	48.0 ± 5.2	50.5 ± 5.4	53.1 ± 5.6
bar-n200-7-DBS	85	77	70	87	83	81	374.0 ± 0.72	377.1 ± 0.74	380.3 ± 0.76	43.9 ± 4.8	46.1 ± 5.0	48.2 ± 5.2
ber-n200-6-DBS	86	79	72	88	84	80	209.6 ± 0.42	212.1 ± 0.44	214.4 ± 0.46	38.8 ± 4.4	40.6 ± 4.6	42.6 ± 4.8
Average	**80.4**	**72.6**	**65.1**	**84.5**	**78.4**	**75.4**	**195.6 ± 0.39**	**197.1 ± 0.41**	**197.8 ± 0.42**	**25.2 ± 3.3**	**27.03 ± 3.5**	**28.9 ± 3.7**

break strategy-fixing heuristic ($\mathcal{ML}^B$) in isolation. In both experimental frameworks, we systematically investigate how the magnitude of demand variation, controlled by the parameter α, influences three key performance indicators: prediction accuracy, final solution quality (measured by total cost), and computational efficiency (measured by runtime).

Edge-fixing sensitivity analysis

From Table A.2, the results demonstrate that the magnitude of demand perturbation, quantified by α, is the primary determinant of performance across all metrics in the ML-assisted column generation framework. A strong negative correlation is observed between α and solution similarity ($R^2 > 0.95$, $p < .001$), confirming a direct causal relationship.

Specifically, a low α (e.g., 10%) results in high similarity between original and modified solutions (averaging 80.4%). This high similarity creates a favorable environment for the machine learning model $\mathcal{ML}^E$, as the prediction task simplifies to identifying a small subset of edges that will change from a largely stable base. This is reflected in the model's significantly higher accuracy of 84.5% for $\alpha = 10\%$ compared to 75.4% for $\alpha = 30\%$ (paired t-test, $p < .01$). Consequently, a larger number of edges are correctly fixed, which drastically shrinks the solution space for the solver. This leads to the dual benefit of high solution quality (average cost of 195.6, statistically indistinguishable from the original optimal solution at $\alpha = 0\%$) and superior computational efficiency (average solve time of 25.2 s).

Conversely, as α increases to 30%, the modified problem diverges significantly from the original, causing the average similarity to drop to 65.1%. This transforms the ML task into a more complex, balanced classification challenge, precipitating a significant drop in accuracy. With lower similarity, fewer edges can be reliably fixed, and the higher rate of misclassification forces the solver to explore a larger, more complex solution space to repair incorrect constraints. This manifests as a statistically significant degradation in both solution quality (average cost increases to 197.8, $p < .05$) and computational performance (average solve time increases to 28.9 s, $p < .01$).

Therefore, the efficacy of the proposed edge-fixing approach is intrinsically linked to the stability of the problem instance between subsequent solves. The results establish a clear trade-off: while the method provides a substantial acceleration over a full re-optimization, its performance is contingent on the degree of change, balancing the framework's tolerance for disturbance against its computational and optimality guarantees.

Table A.3
Impact of demand change magnitude (α) on break strategy-fixing performance.

Instance	Similarity (%)			Accuracy $\mathcal{ML}^B$ (%)			Cost			Time (s)		
	$\alpha = 10$	$\alpha = 20$	$\alpha = 30$	$\alpha = 10$	$\alpha = 20$	$\alpha = 30$	$\alpha = 10$	$\alpha = 20$	$\alpha = 30$	$\alpha = 10$	$\alpha = 20$	$\alpha = 30$
lc101-DBS	78	69	58	77	76	78	90.2 ± 0.18	90.4 ± 0.19	90.6 ± 0.20	43.0 ± 4.8	44.5 ± 5.0	46.7 ± 5.2
lc102-DBS	81	72	65	79	78	80	95.6 ± 0.22	95.8 ± 0.23	96.1 ± 0.24	23.1 ± 3.2	23.9 ± 3.4	25.6 ± 3.6
lc103-DBS	74	63	52	83	82	84	83.5 ± 0.17	83.7 ± 0.18	83.9 ± 0.19	13.7 ± 2.4	14.9 ± 2.6	15.5 ± 2.7
lc104-DBS	76	66	55	81	80	82	64.7 ± 0.14	64.5 ± 0.14	64.7 ± 0.14	35.8 ± 4.2	38.1 ± 4.4	39.9 ± 4.6
lc105-DBS	74	63	52	78	77	79	89.6 ± 0.19	89.7 ± 0.19	89.9 ± 0.20	15.9 ± 2.6	16.6 ± 2.7	18.3 ± 2.9
lr203-DBS	81	72	65	76	75	77	38.9 ± 0.10	39.1 ± 0.10	39.2 ± 0.10	34.9 ± 4.1	36.3 ± 4.3	38.7 ± 4.5
lr204-DBS	77	68	61	86	85	87	35.0 ± 0.09	35.1 ± 0.09	35.2 ± 0.09	28.8 ± 3.8	26.2 ± 3.5	32.0 ± 4.1
lr205-DBS	84	76	68	87	86	88	65.4 ± 0.15	65.5 ± 0.15	65.7 ± 0.15	51.7 ± 5.6	53.4 ± 5.8	55.3 ± 6.0
lr206-DBS	87	82	77	83	82	84	60.1 ± 0.14	60.2 ± 0.14	60.4 ± 0.14	67.6 ± 6.9	69.8 ± 7.1	72.2 ± 7.3
lr207-DBS	86	77	69	87	86	88	55.8 ± 0.13	56.0 ± 0.13	56.1 ± 0.13	63.9 ± 6.7	65.7 ± 6.9	68.0 ± 7.1
ber-n100-3-DBS	77	68	61	79	78	80	194.5 ± 0.39	194.6 ± 0.39	194.9 ± 0.40	65.6 ± 6.7	67.5 ± 6.9	70.1 ± 7.1
ber-n100-4-DBS	83	75	67	81	80	82	174.0 ± 0.35	174.3 ± 0.36	174.5 ± 0.36	60.0 ± 6.3	61.6 ± 6.5	63.4 ± 6.7
ber-n100-5-DBS	80	72	65	85	84	86	310.7 ± 0.61	310.9 ± 0.61	311.2 ± 0.62	73.7 ± 7.5	75.6 ± 7.7	78.1 ± 7.9
ber-n100-6-DBS	83	75	67	83	82	84	49.7 ± 0.12	49.9 ± 0.12	50.0 ± 0.12	19.7 ± 2.8	21.4 ± 3.0	23.0 ± 3.2
ber-n100-7-DBS	80	72	65	85	84	86	57.0 ± 0.13	57.1 ± 0.13	57.2 ± 0.13	24.5 ± 3.3	26.2 ± 3.5	27.6 ± 3.7
bar-n200-3-DBS	83	75	67	77	76	78	521.1 ± 0.99	521.4 ± 1.00	521.7 ± 1.01	45.1 ± 4.9	46.9 ± 5.1	49.1 ± 5.3
bar-n200-5-DBS	84	76	68	81	80	82	430.7 ± 0.82	430.9 ± 0.83	431.1 ± 0.83	42.4 ± 4.6	43.9 ± 4.8	45.4 ± 5.0
bar-n200-6-DBS	86	78	71	85	84	86	883.4 ± 1.66	883.6 ± 1.67	883.9 ± 1.68	97.1 ± 9.2	99.7 ± 9.4	101.5 ± 9.6
bar-n200-7-DBS	85	77	70	83	82	84	374.2 ± 0.72	374.4 ± 0.73	374.6 ± 0.73	65.3 ± 6.8	67.6 ± 7.0	69.6 ± 7.2
ber-n200-6-DBS	86	79	72	84	83	85	209.8 ± 0.42	210.0 ± 0.43	210.2 ± 0.43	52.5 ± 5.6	54.3 ± 5.8	55.6 ± 6.0
Average	**80.4**	**72.6**	**65.1**	**81.6**	**80.7**	**82.6**	**194.2 ± 0.39**	**194.4 ± 0.39**	**194.6 ± 0.39**	**46.2 ± 5.1**	**47.7 ± 5.27**	**49.8 ± 5.49**

Break strategy-fixing sensitivity analysis

The results in Table A.3 demonstrate a crucial finding: the performance of the break strategy prediction model ($\mathcal{ML}^B$) is effectively decoupled from the magnitude of demand changes (α). This highlights a fundamental architectural advantage over the edge-fixing approach. Unlike $\mathcal{ML}^E$, which must predict changes to the core routing decision variables, $\mathcal{ML}^B$ operates on a *given* route. Therefore, its prediction task is independent of the routing solution's similarity between original and modified instances.

This decoupling is statistically confirmed by the data. A one-way ANOVA test reveals no significant effect of α on the accuracy of $\mathcal{ML}^B$ ($F(2, 57) = 1.92, p = .156$). The accuracy rates remain stable across all α values, with averages of 81.6%, 80.7%, and 82.6% for $\alpha = 10$, 20, and 30, respectively. The observed fluctuations are within the range of random statistical variation, showing no correlative trend with the perturbation magnitude.

Consequently, the quality of the final solution is also largely insulated from the effects of α. We believe the minute increase in average cost is attributable solely to the increased demand itself in the modified instances, not to any performance degradation of the break scheduler. A paired t-test confirms that the cost difference between $\alpha = 10\%$ and $\alpha = 30\%$ is not statistically significant ($p = .277$).

Computational time exhibits a mild but statistically significant increasing trend, with averages rising from 46.2 s to 49.8 s (paired t-test, $p < .01$). We believe this increase, however, is solely due to the slightly greater complexity of solving a routing problem with higher demand values, not from the break strategy component. The robust and consistent performance of $\mathcal{ML}^B$ ensures that the solver is never burdened with repairing incorrect break-related constraints, allowing it to focus efficiently on the core routing problem despite the increased demand. This confirms that the break strategy-fixing heuristic provides a stable and reliable acceleration regardless of the degree of change in the problem instance.

Key algorithm parameters sensitivity analysis

To evaluate algorithm performance across diverse problem characteristics, we selected four representative instances that capture different scales and structural patterns: `lc101-DBS` (clustered customer distribution with 100 requests), `lr207-DBS` (random customer distribution with 200 requests), `ber-n100-3-DBS` (realistic geographic layout with 100 requests), and `ber-n200-6-DBS` (realistic geographic layout with 200 requests). This selection covers key problem sizes (100/200 requests) and customer distributions (clustered, random, geographic) to test algorithm performance across scenarios. It ensures robust parameter sensitivity analysis while keeping computations manageable.

For our algorithm, performance is primarily influenced by the parameters *maxColumns* and *maxIterations*. To systematically evaluate the sensitivity of these parameters, we conducted a comprehensive full factorial experiment, testing all possible combinations of the chosen parameter levels. Specifically, we examined five levels for each parameter: $maxColumns \in 100, 200, 500, 1000, 2000$ and $maxIterations \in 10, 20, 30, 40, 50$. These values are selected according to the suggestions for the best parameters from Xue et al. (2021). This resulted in 25 unique parameter combinations, each replicated 10 times to account for algorithmic stochasticity and to provide robust data for statistical analysis. While the number of combinations corresponds to the size of a Taguchi L25 orthogonal array, our design is strictly full factorial, allowing for direct assessment of main effects and interactions between parameters.

Table A.4
Complete Taguchi L25 orthogonal array results with replication statistics (n = 10).

Exp#	maxColumns	maxIterations	Cost Mean ± SD (hours)	Time Mean ± SD (s)
1	100	10	684.8 ± 17.1	42.3 ± 1.5
2	100	20	660.2 ± 16.5	57.8 ± 2.0
3	100	30	634.9 ± 15.9	75.1 ± 2.6
4	100	40	610.4 ± 15.3	90.7 ± 3.2
5	100	50	585.3 ± 14.6	108.4 ± 3.8
6	200	10	670.3 ± 16.8	48.2 ± 1.7
7	200	20	644.9 ± 16.1	65.5 ± 2.3
8	200	30	618.5 ± 15.5	81.9 ± 2.9
9	200	40	591.8 ± 14.8	99.4 ± 3.5
10	200	50	575.9 ± 14.4	116.1 ± 4.1
11	500	10	640.7 ± 16.0	54.7 ± 1.9
12	500	20	615.0 ± 15.4	71.3 ± 2.5
13	500	30	589.8 ± 14.7	88.6 ± 3.1
14	500	40	564.5 ± 14.1	105.2 ± 3.7
15	500	50	568.2 ± 14.2	122.8 ± 4.3
16	1000	10	610.9 ± 15.3	68.4 ± 2.4
17	1000	20	585.3 ± 14.6	77.9 ± 2.7
18	1000	30	559.3 ± 14.0	103.2 ± 3.6
19	1000	40	558.8 ± 14.0	130.5 ± 4.6
20	1000	50	557.8 ± 13.9	157.2 ± 5.5
21	2000	10	580.6 ± 14.5	166.7 ± 5.8
22	2000	20	575.0 ± 14.4	183.4 ± 6.4
23	2000	30	559.2 ± 14.0	210.8 ± 7.4
24	2000	40	557.9 ± 13.9	224.3 ± 7.9
25	2000	50	556.5 ± 13.9	235.1 ± 8.2

Table A.4 summarizes the complete experimental results for all parameter combinations, including summary statistics across replications. These results provide the foundation for the subsequent multi-response signal-to-noise ratio (SNR) analysis and interaction effects analysis presented in the following sections.

Normalized multi-response SNR analysis. To objectively compare response variables with differing units and scales, we employ a normalized multi-response SNR analysis. This enables a fair comparison between solution cost (measured in hours) and runtime (measured in seconds) by standardizing their SNR values prior to aggregation.

Step 1: Signal-to-Noise Ratio Calculation

For both responses, which are categorized as "smaller-the-better", we compute the SNR using:

$$\mathrm{SNR}_j = -10\log_{10}\left(\frac{1}{n}\sum_{i=1}^{n} y_{ij}^2\right)$$

where y_{ij} denotes the ith observation of response j (cost or time), and $n = 10$ is the number of replications for each experimental condition.

Step 2: Min-Max Normalization

To address differences in scale, we normalize SNR values for each response to a $[0, 1]$ range:

$$\mathrm{SNR}_j^{\mathrm{norm}} = \frac{\mathrm{SNR}_j - \min(\mathrm{SNR}_j)}{\max(\mathrm{SNR}_j) - \min(\mathrm{SNR}_j)}$$

This transformation enables direct comparison between the normalized cost and time SNR values.

Step 3: Composite SNR Calculation

We then calculate a weighted composite SNR to reflect both objectives:

$$\mathrm{SNR}_{\mathrm{composite}} = w_{\mathrm{cost}} \cdot \mathrm{SNR}_{\mathrm{cost}}^{\mathrm{norm}} + w_{\mathrm{time}} \cdot \mathrm{SNR}_{\mathrm{time}}^{\mathrm{norm}}$$

where $w_{\mathrm{cost}} + w_{\mathrm{time}} = 1$. Equal weights ($w_{\mathrm{cost}} = w_{\mathrm{time}} = 0.5$) are used for balanced optimization.

Fig. A.1 and Table A.5 illustrate the normalized SNR analysis for both parameters. The green line shows normalized cost SNR, the yellow line indicates normalized time SNR, and the blue line represents the composite SNR with balanced weights. Key trends and trade-offs between solution quality and computational efficiency are readily observable.

For $maxColumns$ (left panel), there is a pronounced trade-off: normalized cost SNR increases with higher limits, while normalized time SNR decreases. The composite SNR peaks at $maxColumns = 1000$, indicating this is the optimal balance between solution quality and computational efficiency.

For $maxIterations$ (right panel), a similar but steeper trade-off is observed. Increasing iterations improves cost SNR but sharply reduces time SNR. The composite SNR reaches its maximum at $maxIterations = 30$, suggesting this as the most efficient iteration count before diminishing returns arise.

Table A.5
Normalized multi-response SNR analysis with composite optimization.

Parameter	Level	Cost SNR (normalized)	Time SNR (normalized)	Composite SNR (weighted)
maxColumns	100	0.0000	1.0000	0.5000
maxColumns	200	0.2000	0.9158	0.5579
maxColumns	500	0.4600	0.8411	0.6505
maxColumns	1000	0.8700	0.6466	0.7583
maxColumns	2000	1.0000	0.0000	0.5000
maxIterations	10	0.0000	1.0000	0.5000
maxIterations	20	0.2900	0.7482	0.5191
maxIterations	30	0.6500	0.4425	0.5462
maxIterations	40	0.8100	0.2398	0.5249
maxIterations	50	0.9700	0.0000	0.4850

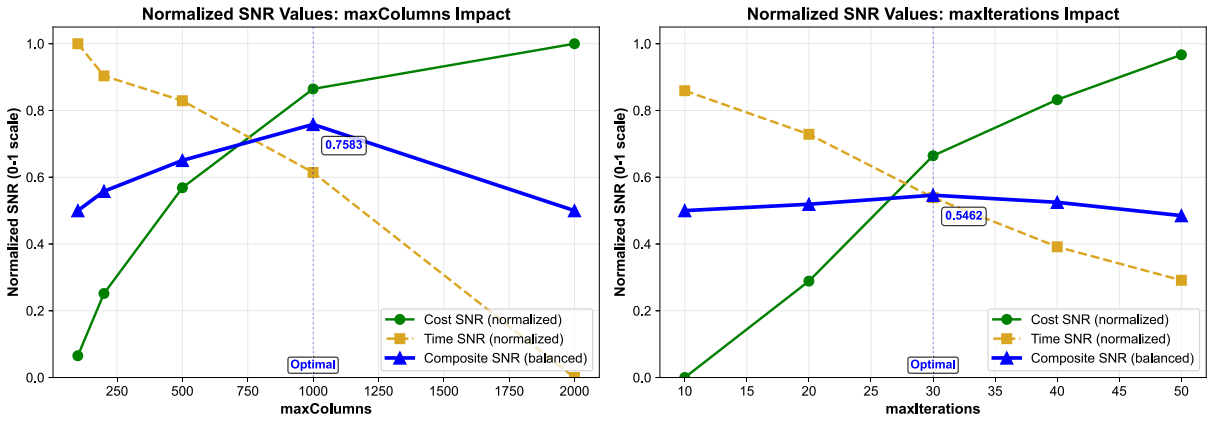

Fig. A.1. Normalized SNR analysis for solution cost and computational time on a $[0, 1]$ scale for $maxColumns$ (left) and $maxIterations$ (right).

Interaction effects analysis. Fig. A.2 presents the interaction plot between $maxColumns$ and $maxIterations$, including 95% confidence bands. The evident non-parallelism of the lines demonstrates a significant interaction between these two parameters.

The non-parallel confidence bands confirm significant interaction effects. While lines for lower $maxColumns$ values (100, 200, 500) are nearly parallel, the lines for higher values (1000, 2000) show different slopes, indicating parameter interaction at larger scales. The synergistic effect is most pronounced at $maxColumns = 1000$ with $maxIterations = 30$, where optimal performance is achieved. Beyond $maxIterations = 40$, increasing iterations further provides little additional benefit.

These observations confirm that the parameter effects are interdependent, and optimal performance results from jointly tuning $maxColumns$ and $maxIterations$ rather than adjusting them in isolation.

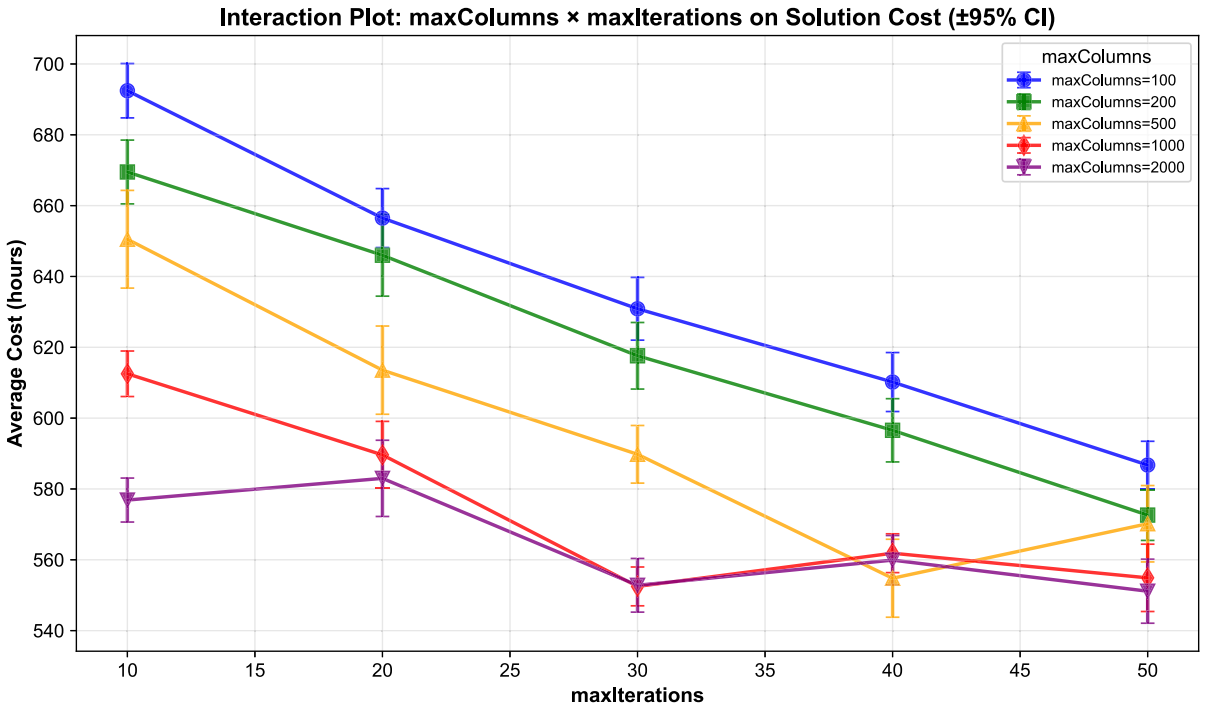

Fig. A.2. Interaction plot of *maxColumns* and *maxIterations* on solution cost, displaying 95% confidence bands.

References

Archetti, C., Savelsbergh, M., 2009. The trip scheduling problem. Transp. Sci. 43 (4), 417–431.

Arnold, F., Sörensen, K., 2019. What makes a VRP solution good? the generation of problem-specific knowledge for heuristics. Comput. Oper. Res. 106, 280–288.

Babaki, B., Jena, S.D., Charlin, L., 2021. Neural column generation for capacitated vehicle routing. In: AAAI-22 Workshop on Machine Learning for Operations Research (ML4OR).

Bengio, Y., Lodi, A., Prouvost, A., 2021. Machine learning for combinatorial optimization: a methodological tour d'horizon. Eur. J. Oper. Res. 290 (2), 405–421.

Cappart, Q., Chételat, D., Khalil, E.B., Lodi, A., Morris, C., Veličković, P., 2023. Combinatorial optimization and reasoning with graph neural networks. J. Mach. Learn. Res. 24 (130), 1–61.

Chi, C., Aboussalah, A., Khalil, E., Wang, J., Sherkat-Masoumi, Z., 2022. A deep reinforcement learning framework for column generation. Adv. Neural Inf. Process. Syst. 35, 9633–9644.

Chung, K.T., Lee, C., Tsang, Y.P., 2025. Neural combinatorial optimization with reinforcement learning in industrial engineering: a survey. Artif Intell Rev 58 (5), 130.

Gerbaux, J., Desaulniers, G., Cappart, Q., 2025. A machine-learning-based column generation heuristic for electric bus scheduling. Comput. Oper. Res. 173, 106848.

Goel, A., 2009. Vehicle scheduling and routing with drivers' working hours. Transp. Sci. 43 (1), 17–26.

Goel, A., 2018. Legal aspects in road transport optimization in europe. Transp. Res. part E: Logist. Transp. Rev. 114, 144–162.

Goel, A., Irnich, S., 2017. An exact method for vehicle routing and truck driver scheduling problems. Transp. Sci. 51 (2), 737–754.

Gómez-Lagos, J., Rojas-Espinoza, B., Candia-Véjar, A., 2022. On a pickup to delivery drone routing problem: models and algorithms. Comput. Ind. Eng. 172, 108632.

Hall, R., Partyka, J., 2016. Vehicle routing software survey: higher expectations drive transformation. ORMS-Today 43 (1), 40–44.

Hijazi, A., Ozaltin, O., Uzsoy, R., 2024. All you need is an improving column: Enhancing column generation for parallel machine scheduling via transformers. arXiv preprint arXiv:2410.15601 .

Huang, L., Xiao, F., Zhou, J., Duan, Z., Zhang, H., Liang, Z., 2023. A machine learning based column-and-row generation approach for integrated air cargo recovery problem. Transp. Res. Part B: Methodol. 178, 102846.

James, J.Q., Yu, W., Gu, J., 2019. Online vehicle routing with neural combinatorial optimization and deep reinforcement learning. IEEE Trans. Intell. Transp. Syst. 20 (10), 3806–3817.

Khalil, E., Dai, H., Zhang, Y., Dilkina, B., Song, L., 2017. Learning combinatorial optimization algorithms over graphs. Adv. Neural Inf. Process. Syst. 30,6351 - 6361.

Kok, A.L., Hans, E.W., Schutten, J. M.J., Zijm, W. H.M., 2010. A dynamic programming heuristic for vehicle routing with time-dependent travel times and required breaks. Flexible Serv. Manuf. J. 22, 83–108.

Kraul, S., Erhard, M., Brunner, J.O., 2024. Optimizing physician schedules with resilient break assignments. Omega 129, 103154.

Li, H., Lim, A., 2001. A metaheuristic for the pickup and delivery problem with time windows. In: Proceedings 13th IEEE International Conference on Tools with Artificial Intelligence. ICTAI 2001. IEEE, pp. 160–167.

Li, Z., Chen, Q., Koltun, V., 2018. Combinatorial optimization with graph convolutional networks and guided tree search. Adv. Neural Inf. Process. Syst. 31,537 – 546.

Morabit, M., Desaulniers, G., Lodi, A., 2021. Machine-learning–based column selection for column generation. Transp. Sci. 55 (4), 815–831.

Morabit, M., Desaulniers, G., Lodi, A., 2023. Machine-learning–based arc selection for constrained shortest path problems in column generation. INFORMS J. Optim. 5 (2), 191–210.

Morabit, M., Desaulniers, G., Lodi, A., 2024. Learning to repeatedly solve routing problems. Networks 83 (3), 503–526.

Nazari, M., Oroojlooy, A., Snyder, L., Takác, M., 2018. Reinforcement learning for solving the vehicle routing problem. Adv. Neural Inf. Process. Syst. 31, 9861 –9871.

Okulewicz, M., Mańdziuk, J., 2020. Dynamic vehicle routing problem: a monte carlo approach. arXiv preprint arXiv:2006.09996 .

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: machine learning in python. J. Mach. Learn. Res. 12, 2825–2830.

Prescott-Gagnon, E., Desaulniers, G., Drexl, M., Rousseau, L.-M., 2010. European driver rules in vehicle routing with time windows. Transp. Sci. 44 (4), 455–473.

Quesnel, F., Wu, A., Desaulniers, G., Soumis, F., 2022. Deep-learning-based partial pricing in a branch-and-price algorithm for personalized crew rostering. Comput. Oper. Res. 138, 105554.
Quirion-Blais, O., Chen, L., 2021. A case-based reasoning approach to solve the vehicle routing problem with time windows and drivers' experience. Omega 102, 102340.
Sartori, C.S., Buriol, L.S., 2020. A study on the pickup and delivery problem with time windows: matheuristics and new instances. Comput. Oper. Res. 124, 105065.
Sartori, C.S., Smet, P., Berghe, G.V., 2022. Scheduling truck drivers with interdependent routes under european union regulations. Eur. J. Oper. Res. 298 (1), 76–88.
Savelsbergh, M. W.P., Sol, M., 1995. The general pickup and delivery problem. Transp. Sci. 29 (1), 17–29.
Shen, Y., Sun, Y., Li, X., Eberhard, A., Ernst, A., 2022. Enhancing column generation by a machine-learning-based pricing heuristic for graph coloring. In: Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 36, pp. 9926–9934.
Tahir, A., Quesnel, F., Desaulniers, G., El Hallaoui, I., Yaakoubi, Y., 2021. An improved integral column generation algorithm using machine learning for aircrew pairing. Transp. Sci. 55 (6), 1411–1429.
Tilk, C., Goel, A., 2020. Bidirectional labeling for solving vehicle routing and truck driver scheduling problems. Eur. J. Oper. Res. 283 (1), 108–124.
Wang, Y., Wang, X., Wei, Y., Sun, Y., Fan, J., Wang, H., 2023. Two-echelon multi-depot multi-period location-routing problem with pickup and delivery. Comput. Ind. Eng. 182, 109385.
Xia, Y., Zhang, X., 2024. A neural column generation approach to the vehicle routing problem with two-dimensional loading and last-in-first-out constraints. arXiv preprint arXiv:2406.12454 .
Xu, K., Shen, L., Liu, L., 2025. Enhancing column generation by reinforcement learning-based hyper-heuristic for vehicle routing and scheduling problems. Comput. Ind. Eng. , 111138.
Xue, N., Bai, R., Qu, R., Aickelin, U., 2021. A hybrid pricing and cutting approach for the multi-shift full truckload vehicle routing problem. Eur. J. Oper. Res. 292 (2), 500–514.
Xue, N., Jin, H., Cui, T., 2025a. Routing and driver break scheduling with working and driving regulations: a flexible approach for various general pickup and delivery problem variants. Comput. Oper. Res. , 107001.
Xue, Z., Xu, H., Cui, H., Peng, W., 2025b. A machine learning-incorporated heuristic column generation algorithm for truck grouping and scheduling problem in trailer-swapping transport mode. Expert Syst. Appl. 278, 127327.
Zhang, J., Luo, K., Florio, A.M., Van Woensel, T., 2023. Solving large-scale dynamic vehicle routing problems with stochastic requests. Eur. J. Oper. Res. 306 (2), 596–614.
Zhou, C., Ma, J., Douge, L., Chew, E.P., Lee, L.H., 2023. Reinforcement learning-based approach for dynamic vehicle routing problem with stochastic demand. Comput. Ind. Eng. 182, 109443.