**Les Cahiers du GERAD** ISSN: 0711–2440

# Accelerated column generation: Application in real-time dial-a-ride problem

E. Amiri, A. Legrain, I. El Hallaoui

G–2024–72

November 2024



**GERAD** HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

**Tél. : 514 340-6053**
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

# Accelerated column generation: Application in real-time dial-a-ride problem

**Elahe Amiri** [a, b, c]

**Antoine Legrain** [a, b, c]

**Issmaïl El Hallaoui** [a, c]

[a] *Mathematics and Industrial Engineering Department, Polytechnique Montréal, Montréal (Québec), Canada, H3T 1J4*

[b] *CIRRELT, Montréal (Québec), Canada, H3T 1J4*

[c] *GERAD, Montréal (Québec), Canada, H3T 2A7*

elahe.amiri@polymtl.ca
antoine.legrain@polymtl.ca
issmail.el-hallaoui@polymtl.ca

**November 2024**
**Les Cahiers du GERAD**
**G–2024–72**

**Abstract :** This study explores accelerating strategies in column generation (CG) to effectively solve online dial-a-ride problems in large-scale ride-sharing systems. Traditional heuristics often lack guarantees on solution quality, and exact methods like CG can be computationally intensive for real-time applications. To address these challenges, we introduce and outline strategies aimed at speeding up the subproblem-solving phase during the solution process in real-time contexts. These subproblems are typically formulated as the Shortest Path Problem with Resource Constraints, in a CG approach, and are traditionally solved via dynamic programming. Our methods significantly enhance the scalability of CG in real-time contexts. We validate the approach using historical taxi trip data from New York City, handling up to 30,000 requests per hour. Computational experiments demonstrate significant reductions in processing times and the ability to produce high-quality solutions more efficiently.

**Keywords:** Dial-a-ride, column generation, real-time optimization, labeling algorithms, shortest path problem with resource constraints

**Résumé :** Cette étude explore les stratégies d'accélération de la génération de colonnes (CG) afin de résoudre efficacement les problèmes de demandes de transport en ligne dans les systèmes de covoiturage à grande échelle. Les heuristiques traditionnelles manquent souvent de garanties sur la qualité des solutions, et les méthodes exactes, comme la génération de colonnes, peuvent être très gourmandes en calcul pour les applications en temps réel. Pour relever ces défis, nous introduisons et décrivons des stratégies visant à accélérer la phase de résolution des sous-problèmes au cours du processus de solution dans des contextes en temps réel. Ces sous-problèmes sont généralement formulés comme le problème du plus court chemin avec contraintes de ressources, dans une approche CG, et sont traditionnellement résolus par programmation dynamique. Nos méthodes améliorent considérablement l'évolutivité de la CG dans des contextes en temps réel. Nous validons cette approche en utilisant des données historiques sur les trajets de taxi à New York, en traitant jusqu'à 30 000 demandes par heure. Les expériences informatiques démontrent des réductions significatives des temps de traitement et la capacité de produire des solutions de haute qualité plus efficacement.

**Mots clés :** Transport à la demand, génération de colonnes, optimisation en temps réel, algorithmes d'étiquetage, problème du plus court chemin avec contraintes de ressources

# 1 Introduction

In recent years, the emergence of ride-sourcing platforms such as Uber and Lyft has significantly transformed urban transportation by offering on-demand mobility services through smartphone applications. Despite the theoretical advantages of these services in mitigating congestion and pollution, their rapid expansion has introduced new challenges and worsened the situation. For instance, Erhardt et al. (2019) reported that between 2010 and 2016, traffic delays in San Francisco surged by 62%, whereas it was expected to be only 22% without the presence of ride-sourcing systems. Introducing shared alternatives like ride-sharing holds promise in addressing this problem by making more efficient use of vehicles, thereby contributing to a more sustainable transportation system. The potential impact of systematic ride-sharing was highlighted by Alonso-Mora et al. (2017), who showed that 98% of ride requests in New York City could be served with just 15% of the taxi fleet, maintaining an average waiting time of 2.8 minutes. However, implementing such measures in practice comes with a variety of challenges, notably *managing high levels of dynamism* and *handling a large volume of trip requests* in real time.

From a mathematical point of view, ride-sharing is modeled as a Dial-a-Ride Problem (DARP), where the objective is to design efficient vehicle routes to transport passengers from their origins to their destinations. In such systems, dispatch decisions are made in real-time, without prior knowledge of future trip requests. Most studies on dynamic DARP have focused on heuristic and metaheuristic methods to find quick solutions, as exact methods are often computationally intensive and considered impractical for large-scale, real-time applications. Against this backdrop, Bertsimas et al. (2019) examined the taxi routing problem (without ride-sharing) and discovered that optimal solutions to the off-line taxi routing problem markedly surpass the outcomes generated by conventional local improvement and greedy algorithms, especially under high demand conditions. This breakthrough suggests that optimization models, previously deemed too complex for practical application, can indeed be made manageable at scales necessary for real-world deployment.

Exact methods for addressing the DARPs primarily rely on Column Generation (CG) and branch and bound techniques, which have been successful in identifying optimal or near-optimal solutions for such problems. This method involves decomposing the problem into a master problem (MP) and subproblems (SPs). The SPs are typically modeled as the Shortest-path Problem with Resource Constraints (SPPRCs) and solved using dynamic programming methods. However, when applied to large-scale networks or under real-time constraints, this method may lack efficiency due to the exponential growth in the number of generated labels. Expanding the work of Amiri et al. (2024) which, introduced a primal-based method for solving the MPs, this paper shifts focus to the SPs, providing deeper insights into techniques critical to solve SPs in real-time applications. Recognizing that in real-time environments the objective shifts from solely achieving optimality to providing high-quality solutions within limited runtime, we introduce several acceleration strategies integrated into CG to enhance its scalability at large scales in real-time. The paper primarily concentrates on computational aspects, establishing a framework for assessing the impact of the proposed strategies. Through extensive experimentation on large-scale instances with up to 30,000 requests per hour, we demonstrate the robust scalability of the method, while surpassing prior studies in terms of average waiting time. The remainder of the paper is organized as follows. Section 2 reviews related work and situates our contributions, Section 3 describes the problem, Section 4 outlines our solution approach, Computational results are presented in Section 5, and Section 6 concludes the paper.

# 2 Related work

The DARP has been a key area of interest in operations research over the past few decades. It involves designing vehicle routes to transport people from their origins to their destinations. Cordeau (2006) introduced a three-index formulation for DARP, where binary variables are used to determine if a specific vehicle travels directly between two locations. This was further refined by Ropke et al. (2007),

who simplified the formulation by omitting the vehicle index. In the branch-and-price framework, variables are indexed by routes, and the problem is formulated as a set-partitioning problem (Røpke, 2006; Ropke and Cordeau, 2009).

Our study relies on this formulation, which necessitates the use of CG due to the large set of possible routes. For comprehensive reviews of different variants, solution strategies, and benchmark instances, readers are referred to Cordeau and Laporte (2007); Ho et al. (2018); Molenbruch et al. (2017). DARP can be classified into two operational modes: *static* or *dynamic*. In the first case ride requests are known in advance, while in the second one they are revealed gradually over time, and routes are adjusted accordingly in real-time. Our research delves into the dynamic DARP, which has received comparatively less attention than its static counterpart.

## 2.1 Heuristic and metaheuristic approaches to dynamic DARP

Most research on dynamic DARP has focused on heuristics to find fast solutions suitable for real-time application. This typically involves two phases: fast insertion heuristics assign new requests to vehicle routes, followed by a secondary heuristic or metaheuristic to optimize the existing solution during idle periods. Some insertion heuristics directly incorporate new requests into the dispatching plan without relocating prior assignments (Wong et al., 2014; Lois and Ziliaskopoulos, 2017), while others allow for the relocation of requests that have been scheduled but not yet served; for instance, Luo and Schonfeld (2011) employed a rejected-reinsertion heuristic, and Vallée et al. (2020) proposed and assessed three reinsertion heuristics aimed at rearranging previously accepted requests in situations where serving a new request is deemed impractical. To refine the current solution during idle times between request arrivals, Carotenuto and Martis (2017) proposed re-insertion of unserved requests, Attanasio et al. (2004) utilized parallel heuristics, combining random insertion with tabu search, and Lois and Ziliaskopoulos (2017) introduced a heuristic called *regret*, which computes a regret value for each request, assessing the benefit of transferring requests between vehicles. In a similar vein, De Oliveira et al. (2024) proposed a re-optimization heuristic combined with a tabu search in the context of patient transportation within a hospital. Daoud et al. (2020) proposed an auction-based mechanism for initial request assignment and introduces a decentralized protocol that enables vehicles to exchange requests and improve overall system efficiency. Another study by Souza et al. (2022) developed a bi-objective optimization model and two-stage heuristic for the heterogeneous dynamic DARP with no rejects. More recently, machine learning (ML) techniques have been integrated into DARP optimization. Tafreshian et al. (2021) integrated data-driven forecasting to anticipate future trips, proposing a proactive shuttle dispatching framework. Ackermann and Rieck (2021) developed a Markov decision process-based framework in which a reinforcement learning-trained agent enhances customer acceptance rates by evaluating multiple factors, rather than focusing solely on minimizing total distance. ML has also been used to enhance metaheuristics, such as Bongiovanni et al. (2020, 2022), who employed ML to learn evaluation functions within large neighborhood search algorithms.

## 2.2 Exact methods and optimization-based approaches

While the majority of research efforts on dynamic DARP have been directed towards heuristics and metaheuristics, a few studies have also devised exact solutions. These approaches primarily involve re-optimizing the static version of the problem within a rolling horizon framework. In this regard, Bertsimas et al. (2019) introduced a *backbone* algorithm and discussed methods to simplify optimization-based approaches for large-scale taxi routing problems without ride-sharing. Alonso-Mora et al. (2017) introduced a batch-based approach where trip requests are accumulated over 30 seconds. They employed a graph-based method to identify all possible matches between a clique of requests and a vehicle, subsequently solving an Integer Linear Program to find the best assignment. They also considered hard time windows to exclude the requests that couldn't be served quickly enough. Time windows are referred to as *hard* when any deviations from them are unacceptable. According to their evaluations on trip data from the New York City Taxi and Limousine Commission (2021) (NYCTLC), 98% of

the historical demand for taxi services in NYC could be fulfilled by a small proportion (15%) of the taxi fleet while keeping the wait time at a low average of 2.8 minutes. Following this pioneering work, Riley et al. (2019) considered a large-scale ride-sharing system modeled as a DARP, with the objective of minimizing the average waiting times. They implemented a rolling horizon strategy with epochs lasting 30 seconds and proposed a CG-based solution. A notable distinction of their research compared to Alonso-Mora et al. (2017), was the use of *soft time windows*, permitting late pickups by applying penalty costs, thereby guaranteeing service to all requests. In evaluations on the NYCTLC dataset, their approach outperformed the achievements of Alonso-Mora et al. (2017) in terms of waiting times. In a follow-up study, Riley et al. (2020) enhanced their approach with a model predictive control for idle vehicle relocation, resulting in a significant 30% improvement in average waiting time. Recently, Amiri et al. (2024) proposed a primal-based algorithm that integrates the integral primal simplex with CG. Experiments on the NYCTLC dataset demonstrated a 33% reduction in waiting times, surpassing the results of Riley et al. (2019). Table 1 summarizes recent studies on dynamic DARP.

**Table 1: Related studies on dynamic DARP, highlighting their objectives, solution methods, and datasets used**

| Reference | Objective | Solution Method | Dataset |
|---|---|---|---|
| Wong et al. (2014) | Min. used vehicles + Min. travel distance | Cheapest insertion | Synthetic (1,000/8 hours) |
| Lois and Ziliaskopoulos (2017) | Max. profit | Fast insertion + regret-based improvement | Philippi (1,619 /day) |
| Vallée et al. (2020) | Max. served requests | Online reinsertion heuristics based on local search | Padam (1,011/12 hours) |
| Carotenuto and Martis (2017) | Min. waiting time | Heuristic based on Reinsertion | Real case (800 requests) |
| Attanasio et al. (2004) | Max. served requests | Random insertion + tabu search | Synthetic |
| Santos and Xavier (2015) | Max. served requests + Min. cost paid | Greedy randomized adaptive search procedure | São Paulo (78,000/day) |
| Luo and Schonfeld (2011) | Min. used vehicles | Rejected-reinsertion heuristic | Synthetic (1,360/9 hours) |
| Souza et al. (2022) | Min. travel cost + Min. user inconvenience | Fast insertion + variable neighborhood search | Synthetic |
| Tafreshian et al. (2021) | Min. travel distance | Proactive optimization | NYC |
| Daoud et al. (2020) | Min. operational cost + Max. served requests | Decentralized exchange protocol + insertion heuristic | Synthetic |
| Ackermann and Rieck (2021) | Multi-objective | Markov decision process + reinforcement learning | Synthetic (200 requests) |
| Bongiovanni et al. (2020) | Max. served requests | Large neighborhood search + ML | San Francisco (6,000/day) |
| Bongiovanni et al. (2022) | Min. travel time + Min. user excess ride time | Greedy insertion + large neighborhood search with ML | San Francisco (6,000/day) |
| De Oliveira et al. (2024) | Min. weighted tardiness | Re-optimization heuristic + tabu search | Synthetic (100 requests) |
| Bertsimas et al. (2019) | Max. total profit | Backbone algorithm | NYC (26,109 /1.5 hour) |
| Alonso-Mora et al. (2017) | Max. requests served + Min. vehicle travel time | Graph-based matching | NYC (460,700 /day) |
| Riley et al. (2019) | Min. waiting time | Column Generation | NYC (32,869 /hour) |
| Riley et al. (2020) | Min. waiting time | Column Generation | NYC (59,820 /2 hours) |
| Amiri et al. (2024) | Min. waiting time | Integral Column Generation | NYC (59,820 /2 hours) |

## 2.3 Acceleration strategies in column generation

Within the CG framework for many routing and scheduling applications, the SP is typically defined as the SPPRC. These SPs are solved to generate shortest-path routes within a network starting from a source node and ending at a sink node while satisfying resource constraints. These generated routes provide potential columns to the MP until optimality is achieved. In the context of the CG, the standard approach for SPPRC is based on dynamic programming labeling algorithm (Irnich and Desaulniers, 2005). This algorithm iteratively extends partial paths from the source node, updating reduced costs and resource usage, and stops when no further extensions are possible. These methods are particularly effective for CG due to three key reasons. First, Dynamic Programming is able to supply the MP with several columns at a time. Second, it effectively handles complex constraints, including non-linear and

non-convex ones. And finally, the paths it identifies are inherently integer-valued. Nonetheless, when dynamic programming is applied to extensive large-scale problems, it struggles with the *curse of dimensionality*. During the resolution process, the number of generated labels may increase exponentially, while only a fraction of them are effective and this could significantly slow down the process (Himmich et al., 2018).

Significant research efforts have been dedicated to enhance the efficiency of this method. Dumitrescu and Boland (2003) leveraged data from the Lagrangian dual problem to calculate lower and upper bounds and illustrated the impact of their preprocessing methods on decreasing solution times. Another approach called *pulse algorithm*, was proposed by Lozano and Medaglia (2013). This method relies on systematically evaluating all potential paths while employing pruning to refine the search area. The study of Nagih and Soumis (2006) explored the impact of resource quantity on dominance rules by projecting the resource vector on a lower-dimensional subspace. Feillet et al. (2007) introduced refinements to accelerate the SP phase in CG. By incorporating Limited Discrepancy Search from Constraint Programming into the dynamic programming approach, they improved the exploration process, focusing on the most promising arcs during label extensions to enhance efficiency. Another strategy known as *bidirectional dynamic programming* relies on extending the labels in both directions, forward from the source and backward from the destination, merging them at common nodes to form complete paths (Righini and Salani, 2006).

### 2.3.1 Contributions

Despite numerous advancements in addressing the DARP, significant challenges persist, particularly in the domain of dynamic DARP and real-time optimization. Firstly, while heuristics and metaheuristics have been extensively explored for their computational efficiency, exact approaches like CG have not been thoroughly investigated for scalability in dynamic, real-time settings. Most existing studies on CG focus on static or small-scale instances, leaving an understanding gap regarding their performance under the stringent requirements of real-time applications. Secondly, although several acceleration strategies for CG have been proposed, their evaluation has been limited to static scenarios, without empirical validation on large-scale, real-world datasets. This raises questions about their practical applicability and scalability. Furthermore, there is a lack of specialized strategies designed to meet the computational demands of dynamic scenarios, where prompt solutions are critical. In response to these gaps, this paper introduces the following contributions aimed at improving the practical application of CG for dynamic DARP:

**Development of novel pruning strategies:** We introduce new pruning strategies that operate both statically during preprocessing and dynamically during the optimization process. These strategies significantly reduce computational overhead and accelerate the convergence of the CG.

**Evaluation and enhancement of acceleration strategies:** We implement and evaluate several existing acceleration techniques, particularly in the SP phase of CG, assessing their scalability and effectiveness for real-time, large-scale optimization. Building on these insights, we introduce new strategies specifically designed to improve computational efficiency and solution quality under the constraints of dynamic, real-world applications.

**Extensive empirical validation:** We conduct an extensive empirical study to validate the performance of the proposed techniques. Our experiments on large-scale instances demonstrate the robustness and scalability of our approach, showing its capability to handle up to 30,000 requests per hour while outperforming prior studies in terms of the average waiting time.

# 3 Problem statement

We use the formulation introduced by Riley et al. (2019) to model the DARP and apply their rolling horizon approach for batching requests. A brief outline of the model is provided here, with more

comprehensive details in Riley et al. (2019). The notations presented in Table 2 are used to formulate the problem.

**Table 2: Definition of the parameters**

| Symbol | Type | Description |
|---|---|---|
| $G = (N, A)$ | — | Graph of the request data, where $N$ represents the set of nodes and $A$ is the set of arcs |
| $\{o, s\}$ | — | The source and the sink node |
| $P$ | — | Set of pickup nodes |
| $D$ | — | Set of drop-off nodes |
| $V$ | — | Set of vehicles |
| $R^v$ | — | Set of feasible routes for vehicle $v$; $(R = \cup_{v \in V} R^v)$ |
| $I_v$ | — | Set of drop-off nodes for passengers already in vehicle $v$; $(I = \cup_{v \in V} I^v)$ |
| $\left[h_v^{start}, h_v^{end}\right]$ | — | Working hours of the vehicle $v$ |
| $q_v$ | integer | Capacity of vehicle $v$ |
| $t_{ij}$ | real | Shortest travel time from node $i$ to node $j$ |
| $t_i$ | real | Shortest travel time between pickup and drop-off of request $i$ |
| $t_i^{max}$ | real | Maximum ride duration for request $i$ defined as $t_i^{max} = \max\{\alpha t_i, \beta + t_i\}$ |
| $\alpha$, $\beta$ | real | Parameters to determine the permitted deviations from the shortest travel time of requests |
| $d_i$ | integer | Number of passengers for request $i$ |
| $e_i$ | real | Earliest possible pickup time (i.e., ready time) for request $i$ |
| $\Delta_i$ | real | Time required for pickup or drop-off for request $i$ |
| $c_r$ | real | Total waiting time of customers served along route $r$ |
| $p_i$ | real | Penalty for request $i$ if it is not planned to be served in the current solution |
| $a_i^r$ | binary | Parameter that equals 1 if customer $i$ is served by route $r$ |
| $y_r$ | binary | Variable that equals 1 if route $r$ is selected and 0 otherwise |
| $z_i$ | binary | Variable that equals 1 if request $i$ is not served and 0 otherwise |
| $x_{ij}$ | binary | Variable that equals 1 if the vehicle traverses the arc $(i, j)$ and 0 otherwise |
| $u_i$ | real | Variable representing the time at which node $i$ is visited by a vehicle |
| $w_i$ | real | Variable representing the vehicle load when departing from node $i$ |

As noted in Section 2, within the CG framework, variables are indexed by routes, and the MP is formulated as a set partitioning problem, as shown in model (1). In this formulation, $R$ denotes the set of all possible routes for the vehicles (i.e., $R = \cup_{v \in V} R_v$).

$$Z^*_{MP} = \min \sum_{r \in R} c_r y_r + \sum_{i \in P} p_i z_i \tag{1a}$$

$$\text{s.t.} \left( \sum_{r \in R} a_i^r y_r \right) + z_i = 1 \qquad \forall i \in P \qquad (\pi_i) \tag{1b}$$

$$\sum_{r \in R^v} y_r = 1 \qquad \forall v \in V \qquad (\sigma_v) \tag{1c}$$

$$z_i \in \mathcal{N} \qquad \forall i \in P \tag{1d}$$

$$y_r \in \{0, 1\} \qquad \forall r \in R \tag{1e}$$

The objective (1a) aims to minimize the waiting time of ride requests scheduled for service and the penalties incurred for those who remain unserved. Constraints (1b) ensure that variable $z_i$ is set to 1 if its corresponding request $i \in P$ is not served by any of the selected routes. Without loss of generality, we refer to a specific request $i$ by its pickup node, e.g., $i \in P$. Constraints (1c) ensure that exactly one route is assigned to each vehicle. And finally, constraints (1d) and (1e) restrict the domain of the decision variables. A route $r \in R^v$ starts from the departure stop of the vehicle $v$, and visits a sequence of pickup and drop-off nodes while satisfying the following constraints:

**Pairing and precedence:** Each request $i \in P$ should be picked up and dropped off by the same vehicle and its pickup node should be visited before its being dropped off.

**Time windows:** Each request $i \in P$ can only be visited after its ready time $e_i$.

**Ride time:** Travel time for the passengers of each request $i \in P$ should not deviate too much from the shortest possible time $t_i$. The upper limit $t_i^{max}$ for ride duration represent either a fixed deviation or a percentage of the shortest travel time using parameters $\alpha$ and $\beta$.

**Vehicle capacity:** Each vehicle $v \in V$ is constrained by its specified carrying capacity $q_v$.

To generate feasible routes, pricing SPs are used, each modeled as a SPPRC aiming to minimize the reduced cost. Let $\boldsymbol{\pi} = (\pi_i)_{i \in P}$ and $\boldsymbol{\sigma} = (\sigma_v)_{v \in V}$ denote the value of dual variables associated to constraints (1b) and (1c), respectively. The reduced cost associated with route $r \in R$ is calculated as $\bar{c}_r = c_r - \sum_{i \in P} a_i^r \pi_i - \sigma_v$. The formulation for the SP is provided in Appendix A.

# 4 Solution method

This section outlines our real-time column generation (RT-CG) approach for a large-scale DARP, which re-optimizes the static DARP at regular intervals by incorporating all known ride requests.

## 4.1 Online re-optimization architecture

Our re-optimization strategy is motivated by the work of Riley et al. (2019, 2020) who suggested using a rolling horizon scheme. This approach divides time into small epochs and periodically updates the dispatch plan. Let define time intervals as $[0, \ell), [\ell, 2\ell), \cdots$ where $[\tau\ell, (\tau+1)\ell)$ corresponds to epoch $\tau$ and $\ell$ denote the length of each epoch. During the re-optimization process at each epoch $\tau$, requests that arrived in the preceding epoch $\tau - 1$ are considered, along with any unserved requests from earlier epochs. At the beginning of each epoch, penalties for unserved requests must be updated, and the dual variables $\pi_i$ and $\sigma_v$ are initialized based on the solutions from the previous epoch $\tau - 1$. For newly arrived requests, $\pi_i$ is set to their corresponding penalties. Penalties are computed using the formula $p_i = \rho 2^{(\tau\ell - e_i)/10\ell}$, as proposed by Riley et al. (2019). In this equation, $\rho$ is a parameter designed to incentivize scheduling the request in the earliest possible epoch. The exponential nature of the formula ensures that the penalty increases significantly the longer the request remains unscheduled, doubling every ten periods. The vehicle's departure time and location are determined based on the solution from epoch $\tau - 1$. Specifically, we consider the first stop occurring within the interval $[(\tau+1)\ell, \infty)$, if such a stop exists, along with its departure time. If a vehicle remained idle during the current epoch, its departure location is its last visited stop (i.e., its current position), with a departure time of $(\tau+1)\ell$. After determining the inputs, the static DARP is re-optimized, and the system moves to the next epoch to repeat the process.

## 4.2 Real-time column generation

The stages of the RT-CG are schematically depicted in Figure 1. Similar to classical CG, the overall approach involves solving a linear relaxation of a Restricted Master Problem (RMP), which focuses on selecting routes and solving pricing SPs based on the dual values obtained from the RMP. The optimization process begins by constructing an initial feasible solution and initializing the dual variables. Next, we solve the SP for each vehicle to generate feasible routes (columns). These columns may be added to the RMP if they have negative reduced costs. Since solving SPs can be computationally intensive, the generated columns are stored in a column pool for future iterations of RMP resolution. After each iteration of solving the linear relaxation of the RMP, the reduced costs of the columns in the pool are updated based on the current dual values. Any columns with negative reduced costs are added to the RMP, and the process repeats until no usable columns with negative reduced costs remain in the pool. At this point, the SPs are resolved again to generate new columns. To enhance the convergence of the RT-CG, multiple columns are added per iteration. The maximum number of columns with negative reduced costs that can be added for each vehicle is denoted by $\lambda$. Finally, an MIP solver is used to solve the RMP exact with the established routes. The Algorithm 1 in Appendix B summarizes the process.

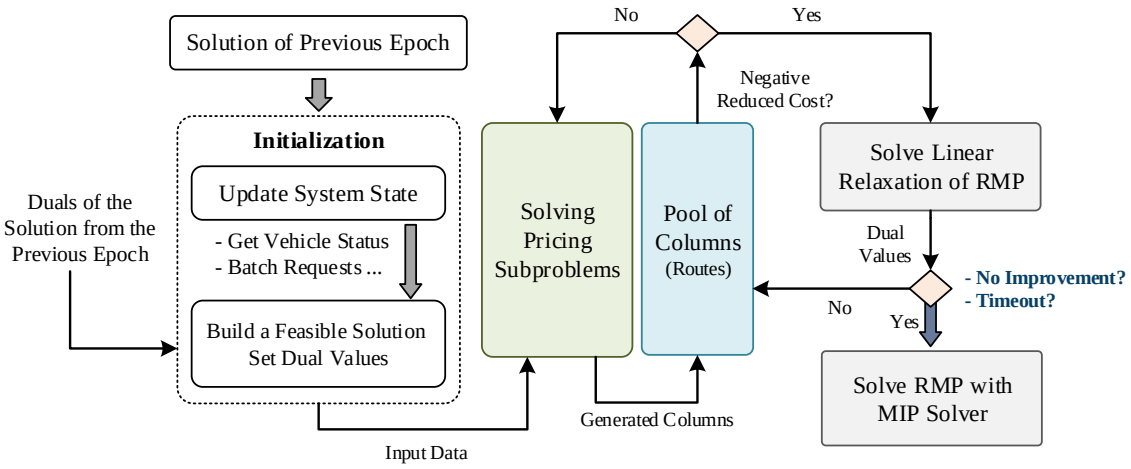



**Figure 1: Schematic representation of the stages in the overall approach**

## 4.3 Labeling algorithm for SPPRC

As previously stated, the SPs are formulated as SPPRC. This involves identifying the least reduced-cost paths in the network from the source to the sink while adhering to vehicle capacity and ride time constraints. To address this, a forward labeling algorithm is proposed. A labeling algorithm is a dynamic programming method that follows the following basic principles. The algorithm begins at a distinct source node and iteratively extends partial paths across the network until the path is completed by reaching the sink node. Each partial path is represented by a *label* that encapsulates key attributes such as the reduced cost and resource consumptions. To avoid the inefficiency of enumerating all possible paths, dominance rules are applied to eliminate useless labels that may not lead to a Pareto-optimal path. For a detailed overview of constrained shortest path problems and the relevant solution techniques, the reader is referred to Irnich and Desaulniers (2005). Next we explain the labeling procedure applicable in RT-CG to solve the online DARP.

### 4.3.1 Initialization.

The CG process begins by generating an initial feasible solution, which is essential for initializing the dual variables $\boldsymbol{\pi}$ and $\boldsymbol{\sigma}$ prior to solving the SPs. A straightforward approach is to use empty routes for the vehicles, including only the necessary stops for dropping off passengers already on board. In this scenario, since the vehicles are not serving any new requests, the initial dual value $\boldsymbol{\sigma}$ are set to zero. The dual multipliers $\boldsymbol{\pi}$ associated with the requests are set equal to their penalties $p_i$, reflecting the cost of leaving requests unserved. Alternatively, starting with well-informed dual values can substantially improve the efficiency of CG by enabling the early identification of beneficial columns, thereby reducing the number of iterations and accelerating convergence. A practical initialization strategy involves using dual values from the previous time period for unserved request from prior epochs. The intuition behind is that these values contain valuable information about the relative importance and opportunity cost of serving these request, based on the prior optimization results. For each newly arrived request $i$, which lacks prior dual information, we set $z_i = 1$ (indicating it is initially unserved) and assign its dual value equal to $p_i$.

### 4.3.2 Network structure.

The next step involves constructing the network graph for each specific vehicle to address its associated SP. The graph for vehicle $v \in V$ is denoted as $G_v = (N_v, A_v)$, where $N_v$ is the set of nodes, and $A_v$ represents the set of arcs. After constructing $G_v$, the arc costs are adjusted using the dual values $\boldsymbol{\pi}$ to

compute the reduced cost of the path. The dual value $\sigma_v$ associated with vehicle $v$ is applied once for the reduced cost of each path and is incorporated in the initialization of the label at the source node (as will be described in Section 4.3.5). For an arc $(i, j)$ the adjusted cost $\bar{c}_{ij}$ is define as Equation (2).

$$\bar{c}_{ij} = \begin{cases} (u_i - e_i) - \pi_i & \text{if } i \in P,\ j \in N_v \\ 0 & \text{if } i \in N_v \setminus P,\ j \in N_v \end{cases} \tag{2}$$

### 4.3.3 Definition of Labels.

A label $l$ that represents a partial path from the source $o$ to the node $\eta$ is defined with a tuple of the form:

$$l = \left(\eta, \bar{c}_l, \omega_l, u_l, \mathcal{O}_l, \mathcal{Q}_l, \left[T_l^{tMax_i}\right]_{i \in P}\right) \tag{3}$$

Where $\omega_l$ denotes the number of passengers in the vehicle after visiting the last node $\eta$, $u_l$ signifies the earliest time of starting the service at $\eta$, $\omega$ denotes the load onboard the vehicle after visiting the last node, $\mathcal{O}$ represents the set of open requests, $\mathcal{Q}$ denotes the set of both completed and open requests, and finally, $\left[T_l^{tMax_i}\right]_{i \in P}$ is the vector associated with ride time constraints. As defined by (Gschwind and Irnich, 2015), these constraints impose dynamic limitation on the duration a request remains onboard. The term *dynamic* refers to the fact that the time window for the delivery of each request is tied to its corresponding pickup time. To ensure that the path adheres to these ride time constraints, it is necessary to track the start time of each pickup and calculate the latest possible drop-off time. To simplify this process and achieve an efficient feasibility check, the resource window for $T_l^{tMax_i}$ for a request $i$ is defined based on the maximum allowable travel time $t_i^{max}$. This resource begins consumption at the point of picking up of request $i$, allowing for a fixed window rather than a dynamic one.

### 4.3.4 Pruning strategy.

To improve the computational efficiency of the problem, we implement a three-phase pruning process that help to narrow the search space. The first phase follows from the theorem outlined by Riley et al. (2019) and is based on the following corollary:

**Corollary 4.1.** For any request $i$, if the minimum possible waiting time to serve them directly from the vehicle $v$'s departure point exceeds their penalty $p_i$, then the request can be safely excluded from the network graph associated with that vehicle, as any assignment for this vehicle involving this request would be suboptimal.

In the first phase, the search space is pruned by removing the pickup and delivery node pairs associated with any request $i$ from the network $G_v$, which corresponds to the SP of vehicle $v$, if the request satisfies the condition specified in Corollary 4.1. We generalize this pruning strategy through the following theorem, which extends the principles established in Corollary 4.1 to encompass a broader set of conditions for limiting the number of generated labels throughout the process.

**Theorem 4.2.** In any feasible route $r$ assigned to vehicle $v$, if there exists a request $m \in P$ such that the waiting time for serving $m$ exceeds its associated penalty $p_m$ (i.e., $u_m^v - e_m > p_m$), then any feasible solution including this route is suboptimal.

**Proof.** Proof Let $r_1 = (o, \ldots, \eta, m, j, \ldots, m+n, \ldots, s)$ be a feasible route for vehicle $v \in V$, which includes serving request $m$ such that $u_m^v - e_m > p_m$. Suppose that a feasible solution with an objective value of $Z^{(1)}$ exists in which the route $r_1$ is assigned to vehicle $v$. Now, we construct an alternative solution by removing request $m$ from the route assigned to vehicle $v$. Let $r_2 = (o, \ldots, \eta, j, \ldots, s)$ be the new route obtained by deleting the pickup and delivery nodes of request $m$ from the original route. Denote the objective value of this new solution as $Z^{(2)}$.

Given that route $r_1$ is feasible, route $r_2$ remains feasible. Additionally, due to the triangle inequality, the travel times between remaining nodes do not increase, so the waiting times for other requests either decrease or stay the same. Therefore, the total waiting time for all other requests in $r_2$ is less than or equal to that in $r_1$. Considering the penalty for marking request $m$ as unserved, the difference in objective values between the two solutions is:

$$Z^{(1)} - Z^{(2)} = (u_m^v - e_m) - p_m + \delta \tag{4}$$

where $\delta$ represents any reduction in waiting times for other requests due to the shortened route. given that $u_m^v - e_m > p_m$, it follows that $(u_m^v - e_m) - p_m > 0$. Thus, $Z^{(2)} < Z^{(1)}$, proving that the original solution (1) is suboptimal. □

Theorem 4.2 is equivalent to enforcing a hard time window $[e_m, e_m + p_m]$ for each pickup node $m$. Building upon this equivalence, we introduce the following corollaries:

**Corollary 4.3.** For any request $i$, if serving it immediately after visiting node $j$, which is directly reached from the vehicle $v$'s departure point, results in a waiting time for request $i$ that exceeds its penalty $p_i$, then the arc connecting node $j$ to the pickup node of request $i$ can be excluded from the network graph associated with vehicle $v$.

**Corollary 4.4.** If, for a given label $l$, the last node $\eta$ is a pickup node and the service time $u_l$ falls outside the window $[e_\eta, e_\eta + p_\eta]$, the label is discarded as it cannot lead to a path that is a part of an optimal solution.

It should be noted that corollaries 4.1 and 4.3, are applied during the preprocessing phase to eliminate suboptimal requests and arcs from the network graph before solving the problem. However, Corollary 4.4, is used during the labeling process, where it discards non-promising labels dynamically as routes are explored. Figure 2 illustrates the pruning strategies on a a portion of a network graph, with different colors representing various pruning criteria.

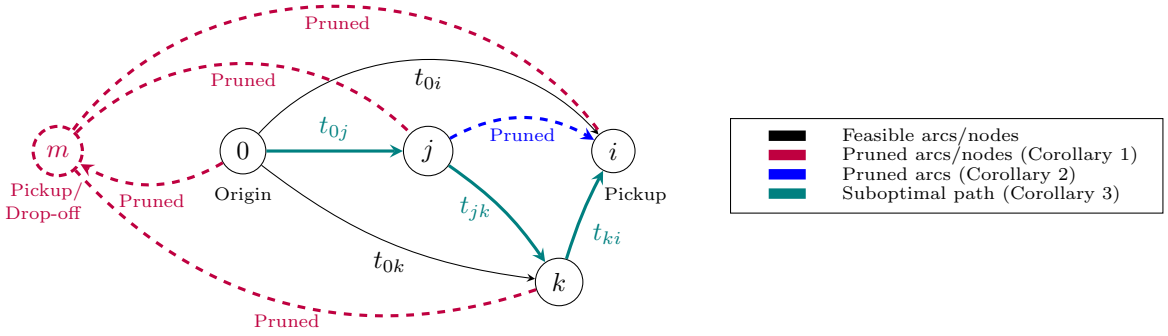



**Figure 2: Illustration of the pruning strategy in the network graph**

As observed in the figure, the decision to prune suboptimal paths depends on their size: single-node paths allow removal of pickup and delivery pair nodes, two-node paths result in edge pruning, and longer paths are handled during the labeling process rather than preprocessing.

#### 4.3.5 Extension and feasibility check.

At the source node, the resources are initialized as $l_0 = \left(o, 0, 0, h_v^{start}, \emptyset, \emptyset, [0]_{i \in P}\right)$. If the vehicle has onboard passengers (i.e., $I_v \neq \emptyset$), the resources of the initial label must be adjusted. Specifically, the load resource is set as $\omega_{l_0} = \sum_{i \in I_v} d_i$, and the sets $\mathcal{O}_{l_0}$ and $\mathcal{Q}_{l_0}$ are both initialized as $I_v$. For each open request $i \in I_v$, the initial consumption of the resource related to the travel time limitation

$T_{l_0}^{tMax_i}$ is calculated as the time elapsed since the request was picked up, defined as the difference between the vehicle's departure time $h_v^{start}$ and the request's pickup time $u_i^P$. Formally, this is: $T_{l_0}^{tMax_i} = h_v^{start} - u_i^P$.

Given a label $l$ as defined in (3) at node $\eta$, extending this label along an arc $(\eta, m)$ results in the generation of a new label $l'$. The cost and resource components of the new label are updated as:

$$\overline{c}_{l'} = \overline{c}_l + \overline{c}_{\eta m} \tag{5a}$$

$$\omega_{l'} = \omega_l + q_m \tag{5b}$$

$$u_{l'} = \max\{u_l + \Delta_\eta + t_{\eta m}, e_m\} \tag{5c}$$

$$\mathcal{O}_{l'} = \begin{cases} \mathcal{O}_l \cup \{m\} & \text{if } m \in P, \\ \mathcal{O}_l \setminus \{m\} & \text{if } m \in D \end{cases} \tag{5d}$$

$$\mathcal{Q}_{l'} = \begin{cases} \mathcal{Q}_l \cup \{m\} & \text{if } m \in P, \\ \mathcal{Q}_l & \text{otherwise} \end{cases} \tag{5e}$$

$$T_{l'}^{tMax_i} = \begin{cases} T_l^{tMax_i} + \Delta_\eta + t_{\eta m} & \forall i \in \mathcal{O}_l, \\ T_l^{tMax_i} & \text{otherwise} \end{cases} \tag{5f}$$

These equations, termed *resource extension functions* (REFs), are used to propagate labels towards successor nodes. The generated label $l' = \left(h, \overline{c}_{l'}, \omega_{l'}, u_{l'}, \mathcal{O}_{l'}, \mathcal{Q}_{l'}, \left[T_{l'}^{tMax_i}\right]_{i \in P}\right)$ is feasible if:

$$\omega_{l'} \leq q_v \qquad \text{(Vehicle Capacity)} \tag{6}$$

$$\forall i \in \mathcal{O}_{l'}, \quad T_{l'}^{tMax_i} \leq t_i^{max} \qquad \text{(Ride Time Limitation)} \tag{7}$$

$$\begin{cases} m \notin \mathcal{Q}_l & \text{if } m \in P, \\ m \in \mathcal{O}_l & \text{if } m \in D, \\ \mathcal{O} = \emptyset & \text{if } m = s \end{cases} \qquad \text{(Pairing and precedence)} \tag{8}$$

Expression (6) ensures the vehicle's capacity is not exceeded. Expression (7) enforces ride time limits, and finally, expression (8) guarantees proper pairing and precedence, ensuring that each request is served exactly once, drop-offs occur only if the corresponding pickups have been visited, and the vehicle can proceed to the sink only after all open requests are fulfilled. Additionally, any feasible extension of the label $l$ along the arc $(\eta, m)$, where $m \in P$ and $u_{l'} > e_m + p_m$, is deemed non-promising and is pruned based on Corollary 4.4.

#### 4.3.6 Dominance criteria.

The effectiveness of the dynamic programming algorithm is largely determined by its ability to discard feasible labels that do not contribute to the optimal solution. To this end, dominance rules are applied to decide whether a generated label should be retained or discarded. Given the reduced cost equation (2), when node $j$ is a drop-off node, the reduced cost $\overline{c}_{ij}$ satisfies the inequality $\overline{c}_{ij} + \overline{c}_{jk} \geq \overline{c}_{ik}$ for all $i, j \in N_v$. This property, referred to as the *delivery triangle inequality*, was introduced by Ropke and Cordeau (2009). This inequality is the basis for the following dominance rule, proposed by Gschwind and Irnich (2015) for DARP with Dynamic Time Windows. A feasible label $l_2$ is said to be dominated by another label $l_1$, if:

$$\eta_{l_1} = \eta_{l_2}, \quad u_{l_1} \leq u_{l_2}, \quad \overline{c}_{l_1} \leq \overline{c}_{l_2}, \quad \mathcal{O}_{l_1} \subseteq \mathcal{O}_{l_2}, \quad \mathcal{Q}_{l_2} \subseteq \mathcal{Q}_{l_1}, \quad \text{and} \tag{9}$$

$$T_{l_1}^{tMax_i} \leq T_{l_2}^{tMax_i} \quad \forall i \in \mathcal{O}_{l_1} \tag{10}$$

The conditions specified ensure that any feasible extension of $l_2$ will also be feasible for $l_1$, and the resource consumption in any extension of $l_2$ is equal to or exceeds the corresponding consumption in $l_1$. It is important to note that the expression (10) is critical for ensuring that label $l_1$ is at least as promising as label $l_2$ in terms of ride-time constraints.

#### 4.3.7 Label elimination.

We define label elimination rules using the sets $\mathcal{O}_l$, $u_l$, and $T_l^{tMax_i}$. To ensure feasibility, all open requests must be fulfilled by visiting their corresponding delivery nodes. Adopting the strategy proposed by Dumas et al. (1991), we evaluate each open request $i \in \mathcal{O}_l$. If it is impossible to deliver any open request directly from the current node $\eta$ within the permitted ride time, the corresponding label is eliminated.

### 4.4 Acceleration techniques

This section outlines various acceleration techniques aimed at accelerating the labeling process. Their impact on the Objective value and the run time will be evaluated later in Section 5.

#### 4.4.1 Relaxed domination rules.

The first acceleration strategy involves relaxing the last condition of the dominance rules specified in Equation (10). This modification allows more labels to be dominated. Moreover, since the last condition is computationally intensive to verify, omitting it reduces the computational effort required for comparisons. As a result, the labeling process better suited for real-time decision-making, where rapid response times are prioritized over strict optimality. We will show in Section 5.1.2 that this relaxation does not compromise solution quality.

#### 4.4.2 Limitation on the number of pickups.

Seconds acceleration method imposes a limitation on the number of requests that can be picked up during the path. This method incorporates a new resource, $\mathcal{P}$, into the label structure, which tracks the count of picked-up requests, constrained by a threshold $M^{Pick}$. To accommodate this limitation, the REF is modified as follows:

$$\mathcal{P}_{l'} = \begin{cases} \mathcal{P}_l + 1 & \text{if } m \in P, \\ \mathcal{P}_l & \text{otherwise} \end{cases} \tag{11}$$

Additionally, the dominance rules are updated to incorporate this new resource. A label $l_1$ now dominates another label $l_2$ if $\mathcal{P}_{l_1} \leq \mathcal{P}_{l_2}$, alongside the previously established conditions. three distinct sorting criteria to enhance its effectiveness.

#### 4.4.3 Truncated labeling.

This method, initially introduced by Dabia et al. (2017), accelerates the process by considering a limit on the number of labels stored at each node during possible extensions, a parameter denoted as $M^{Label}$. Labels are prioritized and retained based on a strategy that identifies the most promising ones, typically those with the lowest associated reduced costs. In our study, we have expanded upon the traditional truncated labeling method by exploring three distinct sorting criteria to enhance its effectiveness. The traditional approach sorts labels by their reduced costs, prioritizing paths with minimal costs. Another criterion we propose is to evaluate the *Normalized Reduced Cost*, calculated as: $\bar{c}_l/\mathcal{P}_l$.

The last criterion, inspired by lambda pricing from Bixby et al. (1992), utilizes a ratio that emphasizes cost efficiency relative to resource usage. The *Lambda Score* is calculated as:

$$\text{Lambda Score} = \frac{\sum_{i \in \mathcal{Q} \setminus I_v} (u_i - e_i)}{\sum_{i \in \mathcal{Q} \setminus I_v} \pi_i} \tag{12}$$

The lambda score favors paths that minimize total delays and serve a higher number of requests, as indicated by columns with a greater number of non-zero entries.

#### 4.4.4 Preventing the visit of pickups after drop-offs.

This acceleration strategy restricts the extension of labels towards pickup nodes once a drop-off node has been visited on the corresponding partial path. The primary objective of our problem is to minimize the total waiting time, which is directly influenced by the timing of pickups. Thus, it is advantageous to prioritize extending towards pickup nodes as much as possible before executing drops. Given that the epoch size is $\ell = 30$ seconds, there is sufficient flexibility to adjust decisions in subsequent epochs.

#### 4.4.5 Dynamic pricing.

To enhance CG efficiency, we implement a strategy termed *dynamic pricing* which systematically varies the maximum number of pickups allowed during the iterations of solving the SPs. Initially, we set a limit of one pickup, allowing quick generation of feasible solutions and providing well-informed dual values early in the process. In subsequent iterations, we gradually raise the limit to four pickups. This gradual increase not only maintains the solution's manageability but also enhances the quality of the dual information obtained in each phase.

### 4.5 Implementation strategies

In dynamic programming for label propagation, two main strategies are pulling and pushing. The pulling method extends labels from predecessors to a specific node, applying dominance rules once for all labels at that node. Conversely, the pushing method extends labels from a current node to each successor, applying dominance rules individually at each successor. While both have similar algorithmic complexities, pulling is generally more efficient in practice as it consolidates dominance checks into a single step per node, handling all incoming labels together (Desrochers and Soumis, 1988). The Algorithm 2 in Appendix C outlines our labeling algorithm with pulling strategy.

## 5 Results and discussion

This section presents the results of computational experiments conducted on real-world datasets. Two primary sets of instances from the NYCTLC dataset were employed. The first set, originally used in the work of Riley et al. (2019), includes 24 instances, each containing between 19,276 and 59,820 customers, with an average of approximately 48,100 customers. For sensitivity analysis, a second dataset with a higher average of around 50,570 customers was used to create reduced-size instances. These reduced instances, designed for manageability in testing, were generated by randomly selecting 30 epochs from a one-hour simulation window following a one-hour warm-start period. The resulting instances averaged about 520 customers.

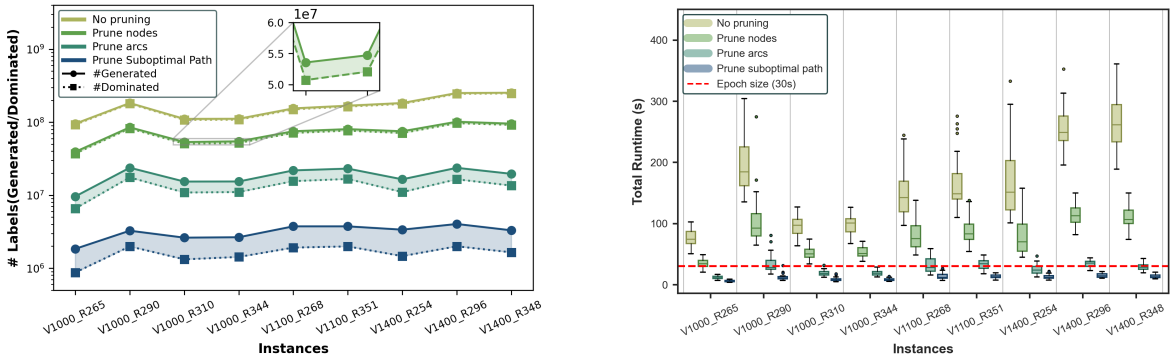
The algorithm was implemented in C++ and executed using IBM ILOG CPLEX Optimization Studio version 22.1.1 to solve the MPs. All experiments were conducted on a Linux machine with 16 CPU cores at 2.7 GHz and 16 GB of RAM. Unless otherwise specified, the common baseline settings across all experiments include the following parameters: $\alpha = 1.5$, $\beta = 240$ s, $\rho = 420$ s, $\lambda = 50$, $M^{Pick} = 2$, and $\ell = 30$. Additionally, no acceleration techniques were activated, all dominance rules were applied, SPs were solved for one iteration per epoch, and duals were initialized based on the prior solution. Specific variations to these settings are detailed within each section where relevant. For further details on the experimental instances and scenarios, refer to Appendix D.

### 5.1 Sensitivity analysis

This section presents a detailed sensitivity analysis examining the impact of various acceleration strategies on both solution quality and computational performance. The goal of this analysis is to identify the best configurations that balance minimizing waiting times with maintaining computational efficiency, ensuring the method's suitability for real-time applications.

### 5.1.1 Impact of pruning strategies.

Figures 3 present the impact of the pruning strategy as outlined in Section 4.3.4, on label generation and computational time. In these experiments, the baseline configuration involves no pruning, and pruning strategies are progressively activated: first pruning nodes, then adding pruning arcs, and finally applying all pruning strategies, including suboptimal paths. Figure a illustrates that as each pruning strategy is progressively activated, there is a substantial reduction in both the number of labels generated and dominated. When all pruning strategies are applied the number of generated and dominated labels is minimized across all instances. This progressive reduction indicates that each additional pruning strategy contributes to the efficiency of the label generation process by eliminating non-promising labels earlier, thereby reducing the computational burden, as fewer labels are carried forward for further processing.




(a) Impact on label generation

(b) Distribution of total runtime

**Figure 3: Impact of pruning strategies on the number of labels generated, dominated labels, and run time**

Figure b presents the total runtime distribution across instances for each pruning configuration. The runtime reduces notably as more pruning strategies are applied, with the lowest runtime observed when all pruning methods are utilized. The suboptimal path pruning strategy, in particular, demonstrates significant efficiency gains, keeping runtime consistently below the 30-second epoch size (as indicated by the red dashed line). This outcome highlights the substantial efficiency achieved through pruning, as it limits unnecessary label extensions and accelerates the overall process. For all subsequent experiments, except in Section 5.1.2, all pruning strategies are activated.

### 5.1.2 Impact of relaxed dominance rules.

Figure 4 illustrates the effect of applying relaxed dominance rules on computational performance across various instances for a single iteration. The bar chart represents the number of dominated labels (referencing the right axis), and the box plots illustrate the runtime (referencing the left axis). Accordingly, the number of dominated labels remains nearly similar between both settings, indicating that the frequency of dominance checks does not vary significantly. However, the runtime is substantially reduced when relaxed dominance rules are used, as shown by the consistently lower runtimes across all instances. This reduction is especially pronounced in larger instances, where the decreased complexity of each dominance check yields considerable time savings. Notably, this runtime improvement has minimal impact on solution quality, with an average objective increase of only 0.026% across all tests.

### 5.1.3 Impact of preventing pickups after drop-offs.

In these experiments, dual variables are initialized from empty routes (i.e. penalties) to remove warm-start effects, enabling a clearer assessment of impacts on average waiting time. Figure 5 illustrates the impact of the strategy that prevents visiting pickup nodes after a drop-off node has already been visited

along the path. As shown by the box plots, this strategy consistently leads to a noticeable reduction in runtime across all instances. The effect on solution quality is minimal, with an average increase in waiting time of only 0.035%. This result demonstrates the strategy's effectiveness in enhancing computational performance without compromising solution quality.

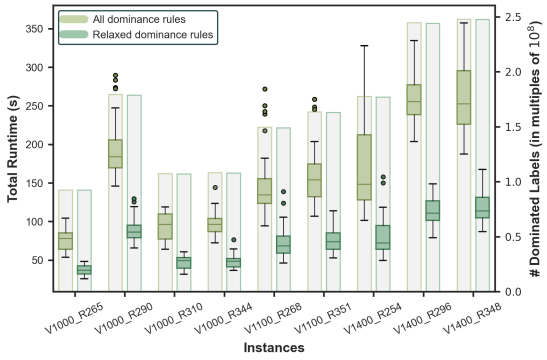



**Figure 4: Impact of applying relaxed dominance rules on run time**

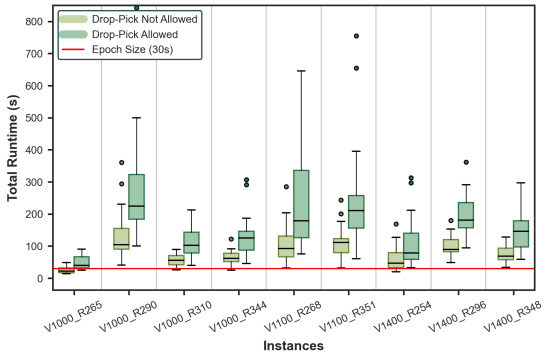
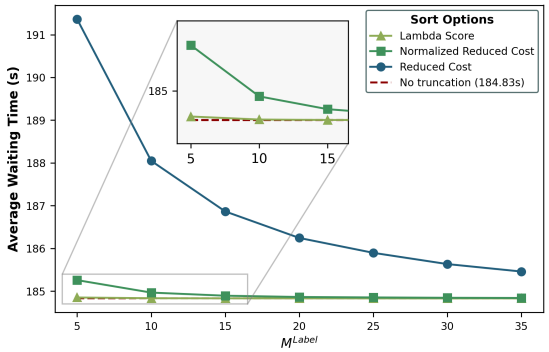


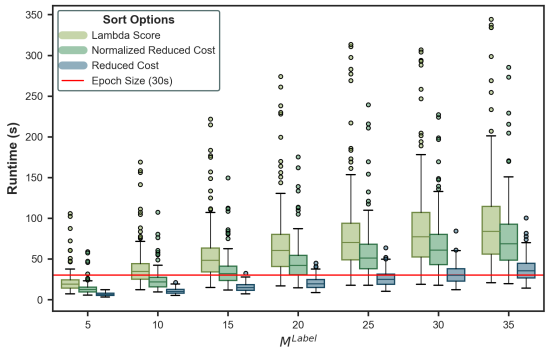

**Figure 5: Impact of preventing pickups after visiting drop-offs on run time**

### 5.1.4 Impact of truncated labeling.

Figure 6, highlights the effect of truncated labeling on runtime and the objective value, measured by average waiting time. In these experiments, we used the same baseline as in Section 5.1.3 for initializing dual variables. In Figure a the average waiting time is plotted against $M^{Label}$ for three sorting criteria: Reduced Cost, Normalized Reduced Cost, and Lambda Score. As $M^{Label}$ increases, the average waiting time generally decreases, especially for the Reduced Cost criterion, which shows a significant reduction at higher $M^{Label}$ values. However, even at higher $M^{Label}$, the Reduced Cost criterion remains less effective than the others. In contrast, the Normalized Reduced Cost and Lambda Score criteria maintain stable waiting times across all $M^{Label}$ values, closely aligning with the non-truncated baseline. Notably, the Lambda Score criterion consistently achieves the lowest waiting time across all values of $M^{Label}$.




**(a) Comparison of average waiting times**




**(b) Distribution of total runtime**

**Figure 6: Impact of truncated labeling on runtime and average waiting time. In figure (a), the red horizontal line represents the average waiting time when truncated labeling is disabled**

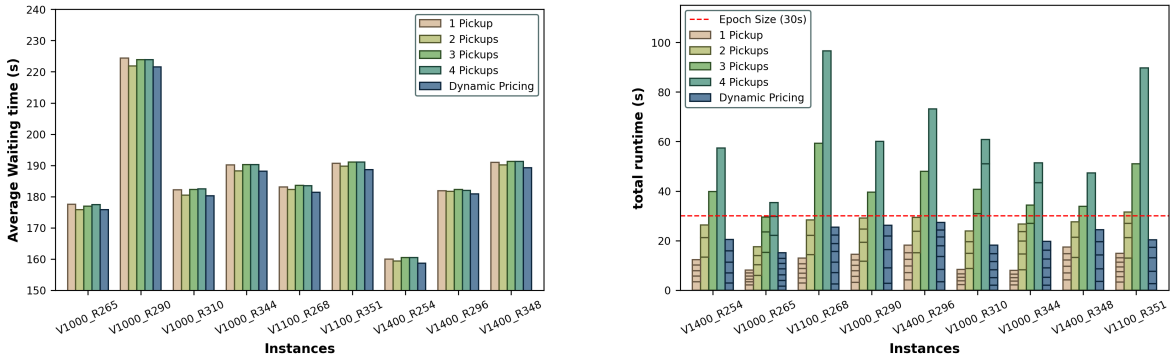
Figure b presents the distribution of total runtime for each sorting criterion across varying $M^{Label}$ values. Here, runtimes increase with higher values of $M^{Label}$ and exhibit greater variability. Specifically, the Reduced Cost criterion has the lowest runtime, while the Lambda Score criterion incurs the

highest. The Normalized Reduced Cost criterion strikes a balance, offering moderate runtimes while closely matching the solution quality of the non-truncated baseline.

The stability of the Normalized Reduced Cost and Lambda Score criteria arises from their ability to better balance path cost and the number of requests served. In contrast, the Reduced Cost criterion may prioritize paths with large dual values, potentially leading to suboptimal routes when fewer labels are retained. By factoring in the number of pickups, the Normalized Reduced Cost criterion favors paths with lower costs that serve more requests. The Lambda Score criterion further prioritizes solutions with smaller objective values and larger dual values, effectively optimizing both cost and service quantity. Overall, the analysis demonstrates that the *Normalized Reduced Cost* criterion offers the best balance between solution quality and computational efficiency. Although increasing $M^{Label}$ improves the solution, it also raises computational cost. Based on these findings, we selected $M^{Label} = 15$ with the *Normalized Reduced Cost* sorting criterion.

### 5.1.5 Impact of the number of pickups and dynamic pricing.

In this section, we analyze how varying the maximum number of pickups and implementing dynamic pricing affect system performance, as illustrated in Figure 7. In these evaluations, multiple iterations per epoch were allowed to observe the effects of these strategies not only on solution quality and runtime but also on the number of iterations that could be completed within the limited epoch duration. As Observed, increasing the number of pickups enhances the algorithm's flexibility in serving requests, improving objective values—this trend is evident when comparing 1 and 2 pickups in Figure a. However, further increasing the limit to 3 or 4 pickups does not yield additional gains; in fact, the objective values slightly deteriorate compared to the 2-pickup mode.




**(a) Comparison of average waiting time**

**(b) Impact on the total runtime**

**Figure 7: Impact of the number of pickups and dynamic pricing on runtime and waiting time across different instances. Stacked bars represent the average number of iterations, with each segment corresponding to the runtime of a specific iteration. The bottom segment is the first iteration, while the top segment is the final iteration**

Figure b provides insight into this phenomenon by showing that additional pickups increase computational complexity, leading to longer runtimes and fewer iterations per epoch. Specifically, with 3 or 4 pickups, most instances complete only one iteration per 30-second epoch (indicated by the red dashed line), limiting the algorithm's opportunity to refine solutions.

Table 3 further clarifies this effect with detailed results for instance V1400_R347, showing the evolution of the objective value and runtime per iteration across pickup strategies. With a single pickup, the algorithm completes four iterations within the epoch but achieves less competitive objective values due to limited flexibility. The 2-pickup strategy allows multiple iterations, improving the objective from 60,594 to 59,702 in the first two iterations. In contrast, with 3 or 4 pickups, only one iteration completes per epoch due to longer runtimes (35.9 and 49.0 seconds, respectively), leading to suboptimal objective values due to insufficient refinement opportunities. Dynamic pricing, however, maintains

**Table 3: Comparison of Elapsed Times and Objectives per Iteration for Different Pickup Strategies**

| Iterations | 1 | | 2 | | 3 | |
|---|---|---|---|---|---|---|
| | Objective | Runtime (s) | Objective | Runtime (s) | Objective | Runtime (s) |
| **Dynamic Pricing** | 60854 | 3.4 | 59709.5 | 7.9 | 59691.2 | 12.5 |
| **1 Pickup** | 60854 | 3.8 | 60141 | 5.9 | 60131 | 8.0 |
| **2 Pickup** | 60594 | 13.7 | 59702 | 23.7 | — | — |
| **3 Pickup** | 60594 | 35.9 | — | — | — | — |
| **4 Pickup** | 60594 | 49.0 | — | — | — | — |

computational efficiency by starting with a limited number of pickups and gradually increasing them. Notably, in the second iteration of dynamic pricing (with pickups limited to 2), the runtime drops to 4.5 seconds, compared to 13.7 seconds for the first iteration with the same pickup limit. This efficiency gain is due to the evolution of dual values after the initial iteration, which guide the search more effectively and reduce runtime in subsequent iterations. By leveraging this mechanism, dynamic pricing maintains runtimes within the epoch limit and achieves an objective of 59,691.2 by the third iteration. By balancing pickup flexibility with computational efficiency, dynamic pricing promotes faster convergence and improved solution quality across instances.

## 5.2 Comparing with prior studies.

This section compares our approach, RT-CG, with two prior methods: the M-RTRS proposed by Riley et al. (2019) and the F-ICG approach introduced by Amiri et al. (2024). M-RTRS employs a rolling horizon strategy to batch requests and iteratively applies the CG method to optimize the solution. An iterative algorithm that generates routes of increasing lengths is employed to address the pricing SPs. F-ICG, introduced by Amiri et al. (2024), proposes a primal-based algorithm that integrates the integral primal simplex with CG. It utilized label-setting methods for generating columns in the SPs. For a fair comparison, we used the same instances and tried our best to replicate the simulation environment. Table 4 presents the average waiting times for each approach across three instance size categories, grouped by the number of customers. The M-RTRS values presented in the table, are drawn from Riley et al. (2020)

**Table 4: Average waiting times in minutes by instance size**

| Method | < 40,000 | 40,000 - 50,000 | 50,000 < |
|---|---|---|---|
| **M-RTRS** | 2.33 | 3.83 | 3.78 |
| **F-ICG** | 1.69 | 2.43 | 2.51 |
| **RT-CG** | 1.70 | 2.42 | 2.48 |

RT-CG consistently outperforms M-RTRS, with the most significant improvement–around 35%– in instances with more than 40,000 customers. This performance is attributed to the label-setting method employed in solving the SPs, as opposed to M-RTRS's anytime approach. While RT-CG and F-ICG yield comparable results in terms of average waiting times, RT-CG offers a simpler and more straightforward implementation. Its ease of use makes it an accessible and efficient choice for practical applications, especially in large-scale scenarios. Overall, this comparison highlights RT-CG's capability to deliver substantial improvements over traditional methods, even in complex, large-scale scenarios.

# 6 Conclusion

This paper introduces a CG-based approach for large-scale online DARP. By integrating pruning strategies, dynamic pricing, and truncated labeling, the RT-CG method balances solution quality with computational efficiency, enabling real-time optimization in large-scale ride-sharing systems. Extensive experiments on real-world NYCTLC data demonstrate that RT-CG significantly reduces both runtime

and average waiting times compared to previous methods like M-RTRS. The combination of theoretical insights, such as penalty-based pruning, with practical implementation strategies provides a powerful framework for addressing large-scale DARP efficiently.

# A Pricing subproblem

Based on the notations presented in Tables 2, the SP is formulated as model (13).

$$Z^*_{\text{SP}} = \min \sum_{i\in P}(u_i - e_i) - \sum_{i\in P}\sum_{j\in N} x_{ij}\pi_i - \sigma_v \tag{13a}$$

$$\text{s.t.} \sum_{j\in N} x_{ij} = \sum_{j\in N} x_{ij} \qquad \forall i \in N \setminus \{o,s\} \tag{13b}$$

$$\sum_{j\in N} x_{oj} = 1 \tag{13c}$$

$$\sum_{j\in N} x_{js} = 1 \tag{13d}$$

$$\sum_{j\in N} x_{ij} - \sum_{j\in N} x_{n+i,j} = 0 \qquad \forall i \in P \tag{13e}$$

$$\sum_{j\in N} x_{ij} = 1 \qquad \forall j \in I \tag{13f}$$

$$u_j \geq (u_i + \Delta_i + t_{ij})\, x_{ij} \qquad \forall i,j \in N \tag{13g}$$

$$u_o \geq h_v^{start} \tag{13h}$$

$$u_s \leq h_v^{end} \tag{13i}$$

$$u_i \geq e_i \qquad \forall i \in P \tag{13j}$$

$$t_i \leq u_{n+i} - (u_i + \Delta_i) \leq t_i^{max} \qquad \forall i \in P \tag{13k}$$

$$t_i \leq u_i - \left(u_i^P + \Delta_i\right) \leq t_i^{max} \qquad \forall i \in I \tag{13l}$$

$$\omega_j \geq (\omega_i + d_j)\, x_{ij} \qquad \forall i,j \in N \tag{13m}$$

$$0 \leq \omega_i \leq Q \qquad \forall i \in N \tag{13n}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in N \tag{13o}$$

$$u_i \geq 0\ , \omega_i \geq 0 \qquad \forall i \in N \tag{13p}$$

The objective (13a) aims to minimize the reduced cost of the generated route. Constraints (13b), (13c) and (13d) ensure flow connectivity throughout the route and for the source and the sink. Constraints (13e) ensure that pick up and delivery of each request is handled by the same vehicle. Constraints (13f) guarantee the drop-off of on-board passengers. Constraints (13g) determine the arrival time of the vehicle at each node. Constraints (13h) and (13i) enforce vehicle working hour limitations. Constraints (13j) implies that each customer can be picked up after its earliest possible pickup time. Constraints (13k) and Constraints (13l) are ride-time constraints, imposing a dynamic time-window of $[0, t_i^{max}]$ on travel time to ensure passengers do not remain in the vehicle excessively long until drop-off. For each on-board request $i$, $u_i^P$ corresponds to the time the request was picked up. Constraints (13m) and (13n) ensure the vehicle capacity constraints.

# B Pseudocode of RT-CG

The Algorithm 1 summarizes the steps of the RT-CG method. In this pseudocode, the superscripts refer to the epoch or time step at which the corresponding notation is considered. For example, $\boldsymbol{\pi}^{\tau-1}, \boldsymbol{\sigma}^{\tau-1}$ indicate the dual values from the previous epoch $\tau - 1$, while $\boldsymbol{y}^{\tau}, \boldsymbol{z}^{\tau}$ represent the updated

decision variables for the current epoch $\tau$. This convention applies to all variables and sets. In our implementation of the RT-CG method, we allocate a maximum of $\boldsymbol{\ell - 5}$ **seconds** for solving the linear relaxation of the RMP and **5 seconds** for solving the RMP to integrality.

**Algorithm 1: Pseudocode of the RT-CG for epoch $\tau$**

**Input:** $\boldsymbol{\pi}^{\tau-1}, \boldsymbol{\sigma}^{\tau-1}$
**Output:** $\boldsymbol{\pi}^{\tau}, \boldsymbol{\sigma}^{\tau}, \boldsymbol{y}^{\tau}, \boldsymbol{z}^{\tau}, Z^*_{MP}$
$RunTime \leftarrow$ start the timer for the execution time;
$R^{\tau} \leftarrow \emptyset$ ; `// Clear the pool of columns for the current epoch`
$P^{\tau} \leftarrow$ Batch requests and update the set of unserved requests;
Build a feasible solution by setting $z_i = 1$ for all $i \in P^{\tau} \setminus P^{\tau-1}$;
Calculate the penalty $p_i$ for all $i \in P^{\tau}$;
**while** $RunTime \leq \ell - 5$ **do**
    $R' \leftarrow$ generate columns with negative reduced cost by solving SPs for all vehicles;
    **if** $R' = \emptyset$ **then**
        **break**;
    **else**
        $R^{\tau} \leftarrow R^{\tau} \cup R'$;
    **end**
    **while** $\exists_{r \in R^{\tau}}\ \bar{c}_r < 0$ *and* $RunTime \leq \ell - 5$ **do**
        Select columns with negative reduced cost from $R^{\tau}$ and add to RMP;
        $\boldsymbol{\pi}^{\tau}, \boldsymbol{\sigma}^{\tau}, Z_{RMP} \leftarrow$ Solve the linear relaxation of RMP;
        $R^{\tau} \leftarrow$ Update the reduced cost of columns based on current duals $\boldsymbol{\pi}^{\tau}, \boldsymbol{\sigma}^{\tau}$;
    **end**
**end**
$\boldsymbol{y}^{\tau}, \boldsymbol{z}^{\tau}, Z_{MP} \leftarrow$ Solve RMP with MIP solver to integrality;
**return** $\boldsymbol{\pi}^{\tau}, \boldsymbol{\sigma}^{\tau}$, and $\boldsymbol{y}^{\tau}, \boldsymbol{z}^{\tau}, Z^*_{MP}$;

# C Pseudocode of labeling algorithm

The Algorithm 2 outlines our labeling method. This algorithm begins by initializing the active and processed label sets. Labels are marked as *active* if they have not yet been fully extended or evaluated in all feasible directions.

In our implementation, we adopt a hybrid approach that combines both pulling and pushing strategies to enhance efficiency. Specifically, when selecting a node to pull *active* labels, we focus solely on the set of pickup nodes, $P_v$, rather than the entire set $N_v$ of all nodes, as is common in other studies. This targeted pulling reduces the computational load by narrowing down the nodes under consideration. After pulling labels to the selected pickup node (line 7 to 10), labels that have open requests are pushed towards their respective onboard nodes(line 11 to 19), further decreasing the computational effort avoiding unnecessary evaluations and extensions to drop nodes that are not related to open requests and therefore irrelevant to the current partial path. The main while loop continues as long as there are active labels across all nodes.

**Algorithm 2: Pseudocode of labeling algorithm**

**Input:** $G_v\,(N_v, A_v)$
**Output:** $\Gamma^s_{processed}$

$\Gamma^o_{active} \leftarrow \{l_0\}$, ;
$\Gamma^i_{active} \leftarrow \emptyset, \forall i \in N \setminus \{o\}$, ;
$\Gamma^i_{processed} \leftarrow \emptyset, \forall i \in N$;
**while** $\cup_{i\in N}\Gamma^i_{active} \neq \emptyset$ **do**
  $m \leftarrow$ select a node from $P_v$ as the target node for extending other labels;
  **for** *all label* $l \in \cup_{i\in N}\Gamma^i_{active}$ **do**
    $l' \leftarrow$ Extend label $l$ to node $m$;
    **if** *label* $l'$ *is feasible* **then**
      $\Gamma^h_{active} \leftarrow \Gamma^h_{active} \cup \{l\}$;
    **end**
    **if** *label* $l$ *has not been extended towards onboards* **then**
      **for** *all request* $i \in \mathcal{O}_l$ **do**
        $l'' \leftarrow$ Extend label $l$ to drop node $i + n$;
        **if** *label* $l''$ *is feasible* **then**
          $\Gamma^{i+n}_{active} \leftarrow \Gamma^{i+n}_{active} \cup \{l''\}$;
          $\Gamma^{i+n}_{active} \leftarrow$ remove dominated labels;
        **end**
      **end**
    **end**
    **if** *label* $l$ *is processed along all possible directions* **then**
      $\Gamma^h_{active} \leftarrow \Gamma^h_{active} \setminus \{l\}$;
      $\Gamma^h_{processed} \leftarrow \Gamma^h_{processed} \cup \{l\}$;
    **end**
  **end**
  $\Gamma^h_{active} \leftarrow$ remove dominated labels;
**end**
**return** $\Gamma^s_{processed}$;

# D Instance description, algorithmic setting and experimental scenarios

To evaluate the performance and scalability of our algorithm, we conducted computational experiments using real-world data from the NYCTLC dataset. The experiments are designed to assess the impact of various algorithmic strategies on solution quality and computational efficiency. In this section, we describe the instances used in our experiments and outline the baseline algorithmic settings and variations considered in our sensitivity analysis. We consider two sets of instances as described below:

**Set 1:** This set consists of 24 instances, each spanning two hours a day (7-9AM), for two days per month from July 2015 to June 2016. On average, each instance contains 48,100.5 customers, ranging from 19,276 to 59,820. These instances are based on the work of Riley et al. (2019, 2020), who partitioned Manhattan into a grid of 200-square-meter cells, with each cell serving as a potential pickup or drop-off location. The travel time matrix between these locations was precomputed using OpenStreetMap data (OpenStreetMap contributors, 2017). The fleet consists of 2,000 vehicles, each with a capacity of four passengers, initially distributed evenly across the grid. For further details on the instance creation, readers are referred to (Riley et al., 2019, 2020).

**Set 2:** The second set is also derived from the NYCTLC dataset and comprises instances covering the time window from 12:00 PM to 2:00 PM. After preprocessing the data to exclude requests originating or terminating outside the study area, six days were selected, similar to Set 1. A higher number of requests are received in these instances, with the number of customers ranging from 41,741 to 65,713, on average 50,570.7. Figure 8 illustrates the distribution of requests over a 24-hour period on September 26, 2015, using 30-second intervals for granularity.

The lower plot in Figure 8 presents the daily request pattern, where the average number of requests per epoch is approximately 124. Notably, this plot shows a steady increase in requests starting from the early morning, peaking later in the day. The top left and right panels provide zoomed-in views of the selected time periods for Set 1 (7:00 AM–9:00 AM) and Set 2 (12:00 PM–2:00 PM), respectively. Set 2 was chosen during the period 12:00 PM to 2:00 PM because the fleet size is assumed to be constant throughout the simulation, and despite fluctuations in the number of ride requests, the average request rate remains relatively stable during this time, at approximately 154.9 requests per epoch. Moreover, this average is close to the peak request rate observed later in the day, making it a suitable period for simulation. To facilitate sensitivity analysis on a smaller dataset, reduced-size instances were created by selecting 30 epochs at random from the time window 1:00 PM to 2:00 PM, following a warm-up period from 12:00 PM to 1:00 PM. In these reduced instances, vehicles may already have passengers onboard at the start. The number of customers in these smaller instances varies between 372 and 693, with an average of 520.7. In our plots throughout the paper, the reduced instances are labeled using the format `V{number}_R{number}`, where the first part indicates the number of vehicles used (e.g., `V1000`) and the second part represents the average number of requests across the 30 epochs selected (e.g., `R256`).

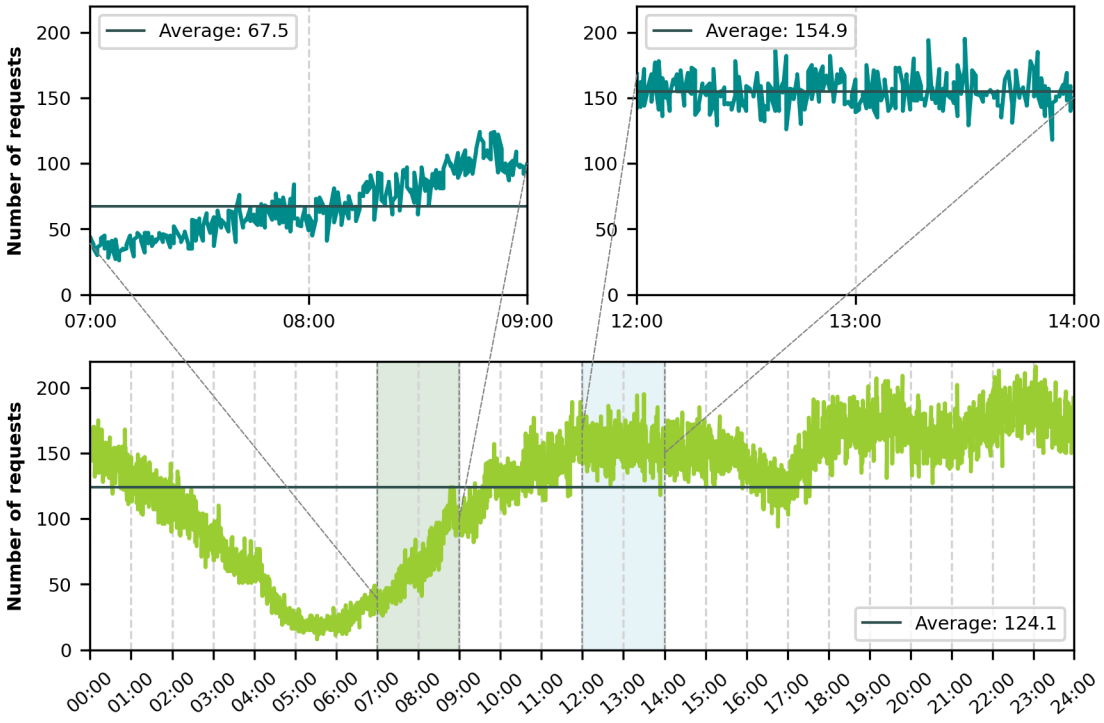



**Figure 8: Requests per epoch on Sep. 26, 2015. The figure shows the number of requests over a 24-hour period in the lower plot, with zoomed-in views for two specific intervals: 07:00-09:00 and 12:00-14:00**

In our sensitivity analysis, we investigate the impact of various strategies on solution quality and computational performance. Table 5 summarizes the experimental scenarios considered in our study. Each scenario is designed to evaluate the effect of specific algorithmic components while keeping other settings constant, either based on a baseline configuration or common settings. For Group 1, we initialized the dual variables based on penalties, effectively starting from empty routes. This approach eliminates the warm-start effect from previous epochs, allowing us to better assess the direct impact of these techniques on solution quality, particularly average waiting time. For Group 2, where we assess the impact of limiting the number of pickups per route and applying dynamic pricing, we did

not establish a separate baseline. Instead, we used common settings across all variations to directly compare the effects of these strategies relative to each other. Unlike the other experiments, we allowed multiple iterations per epoch in Group 2 to observe the effect of these strategies not only on solution quality and runtime but also on the number of iterations that can be executed within the limited epoch duration.

**Table 5: Summary of experimental scenarios**

| Description | Baseline/Common Settings | Variations |
|---|---|---|
| **Impact of Pruning** | • All dominance rules applied<br>• No pruning strategies<br>• No additional acceleration techniques<br>• SPs solved for one iteration per epoch<br>• Duals initialized based on prior epoch's solution<br>• Maximum number of pickups per route, $M^{Pick} = 2$ | 1. Applying pruning strategies:<br>  • Pruning nodes<br>  • Pruning nodes and arcs<br>  • Pruning nodes, arcs, and suboptimal paths<br>2. Applying relaxed dominance rules |
| **Impact of Accelerations** (Group 1) | • All dominance rules applied<br>• All pruning strategies applied<br>• No additional acceleration techniques<br>• SPs solved for one iteration per epoch<br>• Duals initialized from empty routes (penalties)<br>• Maximum number of pickups per route, $M^{Pick} = 2$ | 1. Preventing pickups after drop-offs<br>2. Applying truncated labeling:<br>  • *Sorting criteria*: Reduced Cost, Normalized Reduced Cost, Lambda Score<br>  • *Values of* $M^{Label}$: 5, 10, 15, ..., 35 |
| **Impact of Accelerations** (Group 2) | **Common Settings:**<br>• All dominance rules applied<br>• All pruning strategies applied<br>• No additional acceleration techniques<br>• SPs solved for multiple iterations per epoch<br>• Duals initialized based on prior epoch's solution | 1. Limiting the number of pickups per route:<br>  • *Values of* $M^{Pick}$: 1, 2, 3, 4<br>2. Applying dynamic pricing with $M^{Pick} = 4$ |

# References


Ackermann, C. and Rieck, J. (2021). New optimization guidance for dynamic dial-a-ride problems. In International Conference on Operations Research, pages 283–288. Springer.

Alonso-Mora, J., Samaranayake, S., Wallar, A., Frazzoli, E., and Rus, D. (2017). On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. Proceedings of the National Academy of Sciences, 114(3):462–467.

Amiri, E., Legrain, A., and El Hallaoui, I. (2024). Online optimization of a dial-a-ride problem with the integral primal simplex. In International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, pages 1–16. Springer.

Attanasio, A., Cordeau, J.-F., Ghiani, G., and Laporte, G. (2004). Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. Parallel Computing, 30(3):377–387.

Bertsimas, D., Jaillet, P., and Martin, S. (2019). Online vehicle routing: The edge of optimization in large-scale applications. Operations Research, 67(1):143–162.

Bixby, R. E., Gregory, J. W., Lustig, I. J., Marsten, R. E., and Shanno, D. F. (1992). Very large-scale linear programming: A case study in combining interior point and simplex methods. Operations research, 40(5):885–897.

Bongiovanni, C., Kaspi, M., Cordeau, J.-F., and Geroliminis, N. (2020). A predictive large neighborhood search for the dynamic electric autonomous dial-a-ride problem. In 9th Symposium of the European Association for Research in Transportation.

Bongiovanni, C., Kaspi, M., Cordeau, J.-F., and Geroliminis, N. (2022). A machine learning-driven two-phase metaheuristic for autonomous ridesharing operations. Transportation Research Part E: Logistics and Transportation Review, 165.

Carotenuto, P. and Martis, F. (2017). A double dynamic fast algorithm to solve multi-vehicle dial a ride problem. Transportation Research Procedia, 27:632–639.

Cordeau, J.-F. (2006). A branch-and-cut algorithm for the dial-a-ride problem. Operations research, 54(3):573–586.

Cordeau, J.-F. and Laporte, G. (2007). The dial-a-ride problem: models and algorithms. Annals of operations research, 153(1):29–46.

Dabia, S., Demir, E., and Woensel, T. V. (2017). An exact approach for a variant of the pollution-routing problem. Transportation Science, 51(2):607–628.

Daoud, A., Balbo, F., Gianessi, P., and Picard, G. (2020). Decentralized insertion heuristic with runtime optimization for on-demand transport scheduling. In 11th International Workshop on Agents in Traffic and Transportation (ATT 2020).

De Oliveira, R. M. M., Iori, M., Kramer, A., and Alves de Queiroz, T. (2024). A re-optimization heuristic for a dial-a-ride problem in the transportation of patients. In Metaheuristics International Conference, pages 145–157. Springer.

Desrochers, M. and Soumis, F. (1988). A reoptimization algorithm for the shortest path problem with time windows. European Journal of Operational Research, 35(2):242–254.

Dumas, Y., Desrosiers, J., and Soumis, F. (1991). The pickup and delivery problem with time windows. European journal of operational research, 54(1):7–22.

Dumitrescu, I. and Boland, N. (2003). Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path problem. Networks: An International Journal, 42(3):135–153.

Erhardt, G. D., Roy, S., Cooper, D., Sana, B., Chen, M., and Castiglione, J. (2019). Do transportation network companies decrease or increase congestion? Science advances, 5(5).

Feillet, D., Gendreau, M., and Rousseau, L.-M. (2007). New refinements for the solution of vehicle routing problems with branch and price. INFOR: Information Systems and Operational Research, 45(4):239–256.

Gschwind, T. and Irnich, S. (2015). Effective handling of dynamic time windows and its application to solving the dial-a-ride problem. Transportation Science, 49(2):335–354.

Himmich, I., El Hallaoui, I., and Soumis, F. (2018). A multidirectional dynamic programming algorithm for the shortest path problem with resource constraints. GERAD HEC Montréal.

Ho, S. C., Szeto, W. Y., Kuo, Y.-H., Leung, J. M., Petering, M., and Tou, T. W. (2018). A survey of dial-a-ride problems: Literature review and recent developments. Transportation Research Part B: Methodological, 111:395–421.

Irnich, S. and Desaulniers, G. (2005). Shortest path problems with resource constraints. In Column generation, pages 33–65. Springer.

Lois, A. and Ziliaskopoulos, A. (2017). Online algorithm for dynamic dial a ride problem and its metrics. Transportation research procedia, 24:377–384.

Lozano, L. and Medaglia, A. L. (2013). On an exact method for the constrained shortest path problem. Computers & Operations Research, 40(1):378–384.

Luo, Y. and Schonfeld, P. (2011). Online rejected-reinsertion heuristics for dynamic multivehicle dial-a-ride problem. Transportation research record, 2218(1):59–67.

Molenbruch, Y., Braekers, K., and Caris, A. (2017). Typology and literature review for dial-a-ride problems. Annals of Operations Research, 259:295–325.

Nagih, A. and Soumis, F. (2006). Nodal aggregation of resource constraints in a shortest path problem. European Journal of Operational Research, 172(2):500–514.

New York City Taxi and Limousine Commission (2021). trip record data, 2021.

OpenStreetMap contributors (2017). Openstreetmap contributors, planet dump retrieved from.

Righini, G. and Salani, M. (2006). Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. Discrete Optimization, 3(3):255–273.

Riley, C., Legrain, A., and Hentenryck, P. V. (2019). Column generation for real-time ride-sharing operations. In International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research, pages 472–487. Springer International Publishing.

Riley, C., Van Hentenryck, P., and Yuan, E. (2020). Real-time dispatching of large-scale ride-sharing systems: Integrating optimization, machine learning, and model predictive control. arXiv preprint arXiv:2003.10942.

Røpke, S. (2006). Heuristic and exact algorithms for vehicle routing problems. PhD thesis, University of Copenhagen, Copenhagen.

Ropke, S. and Cordeau, J.-F. (2009). Branch and cut and price for the pickup and delivery problem with time windows. Transportation Science, 43(3):267–286.

Ropke, S., Cordeau, J.-F., and Laporte, G. (2007). Models and branch-and-cut algorithms for pickup and delivery problems with time windows. Networks: An International Journal, 49(4):258–272.

Santos, D. O. and Xavier, E. C. (2015). Taxi and ride sharing: A dynamic dial-a-ride problem with money as an incentive. Expert Systems with Applications, 42(19):6728–6737.

Souza, A. L., Bernardo, M., Penna, P. H., Pannek, J., and Souza, M. J. (2022). Bi-objective optimization model for the heterogeneous dynamic dial-a-ride problem with no rejects. Optimization Letters, 16(1):355–374.

Tafreshian, A., Abdolmaleki, M., Masoud, N., and Wang, H. (2021). Proactive shuttle dispatching in large-scale dynamic dial-a-ride systems. Transportation Research Part B: Methodological, 150:227–259.

Vallée, S., Oulamara, A., and Cherif-Khettaf, W. R. (2020). New online reinsertion approaches for a dynamic dial-a-ride problem. Journal of computational science, 47:101199.

Wong, K.-I., Han, A., and Yuen, C. (2014). On dynamic demand responsive transport services with degree of dynamism. Transportmetrica A: Transport Science, 10(1):55–73.