Submitted to *Operations Research*
manuscript (Please, provide the manuscript number!)

Two-Stage Learning to Branch in Branch-Price-and-Cut Algorithms for Solving Vehicle Routing Problems Exactly

Zhengzhong You[1], Yu Yang[1]*, Xinshang Wang[2], Wotao Yin[2]
[1]Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL
[2]Alibaba Group US, Bellevue, WA
you.z@ufl.edu, yu.yang@ise.ufl.edu, xinshang.w@alibaba-inc.com, wotao.yin@alibaba-inc.com

Branching is one of the most important components in branch-price-and-cut (BPC) algorithms for solving vehicle routing problems (VRPs) exactly. However, learning to branch is much more challenging in BPC than in branch-and-cut algorithms that are used for solving general mixed integer programs because branching, in this case, is generally performed by adding a dense constraint to the restricted master problem (RMP), and meanwhile, the variables in the RMP change constantly. To address such challenges, we propose the first effective *learning-to-branch framework* in BPC algorithms, leading to a novel *two-stage learning-based branching* (2LBB) strategy. This serves as an innovative learning-based enhancement for the cutting-edge three-phase branching strategy for column-generation-based algorithms. In the 2LBB, the first stage focuses on narrowing down the list of promising candidates using computationally cheap features thereby lessening dependence on LP testing. The second stage, meanwhile, diminishes the burden on heuristic testing through an innovative partial testing approach. Moreover, we propose a novel theoretical model characterizing the fundamental trade-off between time spent making a single branching decision and the resulting branching quality. A formula, derived from the model, for dynamically adjusting the number of candidates to select for the second stage achieves consistently superior performance to ones obtained from trial-and-error tuning. The derivation easily generalizes to most branching strategies requiring time-consuming score computation. Through an extensive numerical study, we demonstrate that a dynamic version of the 2LBB, denoted by 2LBB-dy, achieves approximately 45% and 50% time reduction, respectively, compared to the state-of-the-art (SOTA) hand-crafted branching strategy in solving the capacitated vehicle routing problem (CVRP) and vehicle routing problem with time windows (VRPTW). In addition, our **RouteOpt** (an exact VRP solver available at `https://github.com/Zhengzhong-You/RouteOpt`), when equipped with the 2LBB-dy, achieves a 47% time reduction compared to the SOTA VRPSolver from Pessoa et al. (2020) for the CVRP.

Keywords: learning to branch; column generation; exact methods; vehicle routing

1. Introduction

The vehicle routing problem (VRP) is a well-known discrete optimization problem that has been extensively studied in the operations research and management science community since its inception in 1959 (Dantzig and Ramser 1959). It seeks routes of the minimum cost (usually measured

* Corresponding author

by the total travel distance) to serve a set of consumers, subject to restrictions such as vehicle capacity in the capacitated vehicle routing problem (CVRP) and time windows in the vehicle routing problem with time windows (VRPTW). Optimizing the routes can also decrease waiting time and improve customer satisfaction, leading to wide applications in the industry such as e-commerce (Vinsensius et al. 2020), healthcare (Heching et al. 2019), and emergency response (Zheng 2018) where the timely delivery of products and services is crucial.

Nonetheless, VRPs are generally challenging to solve optimally because of their combinatorial nature or, more intrinsically, the so-called NP-hardness. Approaches to solving VRPs exactly can generally be categorized into two types: *branch-and-cut* (B&C) algorithms solving compact (in terms of variables) formulations such as vehicle or commodity flow-based formulations (Bektaş et al. 2011), and *branch-price-and-cut* (BPC) algorithms solving the set-partitioning formulation (SPF; Costa et al. 2019). Branching is an indispensable component of these methods.

It is well-known that the choice of branching variable significantly impacts the effectiveness of B&C algorithms for solving general mixed integer programs (MIPs; Achterberg and Wunderling 2013). Expectedly, branching remains one of the most important decisions in BPC algorithms for solving VRPs. In particular, our numerical experiments show that selecting the branching variable randomly increases computation time by over 8 times compared with the three-phase branching (3PB) strategy employed in the state-of-the-art VRP solver, the VRPSolver developed in Pessoa et al. (2020). Moreover, in contrast to pricing and cutting planes, which are usually customized based on specific problem structures for acceleration (Lübbecke and Desrosiers 2005, Lysgaard et al. 2004), branching does not rely on such information as much and tends to bear better generalization across a wide range of VRP variants. Therefore, the study of branching strategy holds significant potential in enhancing BPC methods for various vehicle routing-related problems.

To enhance the readability of this paper, the meanings of some key abbreviations are summarized in Table 1, sorted in order of decreasing frequency of use.

Table 1 List of Abbreviations and Meanings

Abbreviation	**Meaning**	**Abbreviation**	**Meaning**
3PB	Three-Phase Branching	2LBB	Two-Stage Learning-Based Branching
BPC	Branch-Price-and-Cut	CG	Column Generation
SB	Strong Branching	BKF	Best K Formula
BBT	Branch-and-Bound Tree	RMP	Restricted Master Problem
BBN	Branch-and-Bound Node	B&C	Branch-and-Cut
SPF	Set-Partitioning Formulation	SOTA	State-of-the-Art

1.1. Motivation

Strong branching (SB; Applegate et al. 1995) is believed to be the most reliable way to obtain a small branch-and-bound tree (BBT; see Costa et al. 2019). It considers a set of variables required to be integral but currently taking fractional values, hereafter referred to as *candidates*, computes the gains from branching on each of them, selects one with the highest gains to branch on (see Section 3.2 for more details). Despite its high efficacy in producing small-sized BBTs, SB can be extremely time-consuming since column generation (CG) is required to solve the child linear programming (LP) problems for all the candidates. To alleviate the computational burden, a hierarchical branching scheme is often adopted. In particular, the VRPSolver employs a 3PB strategy modified from Pecin et al. (2017b).

The 3PB, recognized as state-of-the-art (SOTA) branching strategy in BPC algorithms for solving VRPs, progressively narrows branching candidates through three phases: initial screening using pseudocost and fractionality, subsequent LP testing shortening the list, and finally heuristic testing selecting the candidate with the highest heuristic score. By gradually applying more computationally intensive evaluations to fewer candidates, 3PB effectively balances computational cost and decision quality, making it the leading branching strategy for BPC algorithms prior to this work. Its well-structured phased approach inspires the design of our two-stage learning-based branching (2LBB) and lays the foundation for making learning-accelerated branching strategies practical.

While considered highly efficient, 3PB has two major weaknesses. Firstly, the screening process's *complete* reliance on pseudocost and fractionality before LP testing can potentially miss good candidates. Secondly, it requires performing *complete* time-consuming testing, while our newly proposed partial testing demonstrates that 80% of the time spent on complete heuristic testing is unnecessary. Overall, this research seeks to overcome these weaknesses to significantly reduce the computational effort while maintaining, if not improving, the branching decision quality.

Recently, considerable research effort has been dedicated to leveraging advanced machine learning (ML) techniques to mimic SB in the context of solving general MIPs and has successfully demonstrated that learned strategies can beat best-performing ones currently employed in solvers such as SCIP and CPLEX (e.g., Khalil et al. 2016, Alvarez et al. 2017, Yang et al. 2022). These ML models are trained with readily available historical and current-node information to solve compact formulations (i.e., formulations with a non-exponential and usually fixed number of variables). Once trained, the ML models can select high-quality branching variables with little extra computation cost, achieving significant acceleration overall. Nonetheless, models designed for branching selection in general MIPs mostly rely on three categories of features: static ones (e.g., initial LP matrix coefficients), dynamic ones from the current branch-and-bound node (BBN), and dynamic history-dependent ones (see Yang et al. 2022 and Alvarez et al. 2017 for details). These approaches fail for

BPC algorithms solving non-compact formulations, e.g., SPF. To the best of our knowledge, no successful attempt to improve branching in BPC via ML has been accomplished before this study. The unfortunate ineffectiveness of existing learning-to-branch paradigms and the considerable difficulty in making them work for BPC are mainly because of the following three reasons.

First, to balance the resultant BBT, branching is typically not performed on a nominal variable in the restricted master problem (RMP) but rather achieved by adding a dense constraint (see more explanation in Section 3.2), after which, repetitive CG is needed to compute the exact LP value of a child BBN. For clarity, we refer to the LP at the current BBN as the *parent LP*, the one at a child BBN obtained from the parent LP by adding the branching constraint as the *starting child LP*, and the one resulting from solving the starting child LP with CG as the *ending child LP*. This branching constraint, together with the variables that have yet to be generated in the repetitive CG, causes the ending child LP to significantly deviate from the parent LP. Worse still, such deviation is not known until the starting child LP is solved with CG, making it extremely difficult, if not impossible, to accurately predict the LP value changes based on conventional features of the current BBN. Second, generated by CG iteratively, variables are added to the RMP over time. As a result, minimal historical information about the recently generated variables is available to aid the branching variable selection, which, unfortunately, is essential for existing ML models in the literature to make accurate predictions. Third, the accumulated effect of the first reason leads to acute changes to the RMP, where the current RMP can be almost completely different from the initial RMP at the root. Thus, the standard static features lose predictive power. In summary, these three reasons bring about unique obstacles to using dynamic current-node, dynamic history-dependent, and static features, accordingly, for learning to branch in BPC algorithms.

To overcome these challenges, new features that capture the underlying problem structure, rather than focusing solely on the RMP, are needed. Although these new features can be useful, their effectiveness depends on appropriate learning methods and workflows. We believe the newly proposed two-stage scheme addresses these needs. In the first stage, graph-based features that are computationally cheap are used to narrow down the promising candidates where the learning target is the ranking of SB scores, thus reducing the dependence on LP testing. In the second stage, more computationally intensive features derived from LP testing are employed. At this stage, the learning target shifts from SB score ranking to the ratio of optimal value changes between the starting and ending child LPs relative to that of the parent LP. By applying a novel partial testing (see Section 4.4) that integrates predictions from the second stage ML model with LP scores, we efficiently determine which starting child LP should be further tested, enabling us to identify the best candidate with much less computational effort.

Moreover, SB, 3PB, and 2LBB necessitate testing multiple branching candidates before finalizing the decision. However, determining the optimal number of candidates to evaluate for minimizing overall solution time remains challenging as there is no established method to balance the trade-off between the time spent on making a single branching decision and the resulting decision quality. To shed light on this fundamental trade-off, we propose a novel theoretical model for deriving *Best K Formula* (BKF), which dynamically decides the best number of candidates to test based on the solution's progress. By embedding the distribution of local branching improvements, the BKF optimizes this trade-off, leading to additional improvement in computational efficiency. It is important to emphasize that 2LBB is specifically designed for algorithms that incorporate CG. In contrast, the BKF is broadly applicable to branching strategies involving computationally intensive testing, including both BPC algorithms (the focus of this study) and B&C algorithms that do not require CG, as supported by our experiments in Section EC.8.4 of the e-companion.

In this paper, we develop a novel learning-to-branch framework with a theory-based dynamic adjustment scheme that can overcome the weaknesses of the current state-of-the-art branching strategy in BPC, namely 3PB. Although this study focuses on VRPs, we believe that most ideas introduced herein are easily transferable to BPC for addressing operational challenges in supply chain operations, healthcare, and the airline industry, e.g., inventory routing problems (Desaulniers et al. 2016, Engineer et al. 2012), healthcare planning problems (Qiu et al. 2022, Trautsamwieser and Hirsch 2014), and crew rostering problems (Breugem et al. 2022, Quesnel et al. 2020).

1.2. Major Contributions and Outline

This paper concerns improving the branching decisions in BPC algorithms for solving VRPs exactly. It makes the following major contributions.

- *We propose the first effective learning-to-branch framework in BPC algorithms.* The first stage relies on graph-based features that are computationally cheap to narrow down the promising candidates, while the second stage finalizes the decision with substantially less computation effort by utilizing features obtained from LP testing. This design ensures that the framework makes branching decisions fast and effectively selects high-quality candidates to generate a small-sized BBT. Furthermore, by integrating richer information at each stage and defining appropriate learning tasks, it advances both efficiency and effectiveness beyond 3PB, making it a SOTA branching method for BPC algorithms.
- *We establish a novel theoretical model that gives rise to a formula for dynamically adjusting a key parameter in 2LBB and 3PB.* The model characterizes the trade-off between the testing time and branching decision quality, deciding the number of candidates to evaluate. It also generalizes to branching strategies reliant on computationally intensive scores and explains the performance differences between 2LBB and 3PB for CVRP and VRPTW instances.

- *We integrate the 2LBB method into RouteOpt and perform extensive numerical experiments to support the design of the two-stage learning workflow.* Our 2LBB reduces the average computation times by over 45% compared to the SOTA 3PB method and generalizes well across varying instance sizes by consistently lowering CPU time and BBT size. When embedded into RouteOpt together with other effective techniques (see Section EC.5 of the e-companion), 2LBB achieves a 47% time reduction on benchmark instances compared to the VRPSolver, and allows RouteOpt to prove optimality for an open CVRP instance.

The rest of the paper is organized as follows. In Section 2, we review related literature on branching in BPC algorithms and existing learning-to-branch methods. In Section 3, we include some preliminaries to facilitate readability. We describe our learning-to-branch framework in Section 4 and detail the learning methodology employed in the framework in Section 4.5. A theoretical model providing insights into the underlying trade-offs is presented in Section 5. The effectiveness of the proposed framework is demonstrated via an extensive numerical study in Section 6. Lastly, in Section 7, we make some concluding remarks and identify future research directions. Details of selected proofs and numerical results are included in the e-companion.

2. Literature Review

Numerous VRP variants have been extensively investigated in the past six and a half decades, leading to many effective exact algorithms. Laporte (2009) provides a thorough review of this topic. Currently, the best-performing ones are BPC algorithms, solving CVRP instances with up to 856 customers (Pessoa et al. 2020, Lima et al. 2014). This paper focuses on learning to branch in BPC algorithms. We first review branching strategies employed in BPC algorithms from two perspectives: branching variable type and selection. Then, we discuss some most related ML attempts.

2.1. Branching Variable Type

Branching on arcs (or edges in the undirected case) is typically the default approach in B&C methods for solving the two-index vehicle flow formulation (Laporte and Nobert 1983). This prevalence is because of the fact that the variables in such a flow formulation explicitly correspond to arcs/edges, and such branching selection leads to the most common branching on a single nominal variable in the formulation. Desrosiers et al. (1984) introduced branching on arcs in a BPC method for solving the VRPTW, which has become the primary choice in BPC algorithms (Irnich et al. 2010, Pessoa et al. 2020). Costa et al. (2019) underscored the popularity and highlighted its simplicity, ease of implementation, and robustness. In light of these considerations, we also adopt branching on edges in our learning-to-branch framework.

Despite the widespread application of branching on edges, Augerat et al. (1995) observed that this approach, for the CVRP, poses a relatively localized influence that might result in undesirably small changes in the dual bound after branching. Thus, they proposed to branch on sets, which entails selecting a set of vertices. If the total flow, induced by the current solution, passing through this set is fractional, then branching is performed on all edges incident to this vertex set. This branching strategy can be viewed as an aggregated version of branching on edges, thus, intuitively causing a broader impact. It has been applied in some BPC (Fukasawa et al. 2006, Jepsen et al. 2008, Pecin et al. 2017a, Yang et al. 2023) and B&C (Naddef and Rinaldi 2002, Lysgaard et al. 2004, Baldacci et al. 2004) algorithms. In a recent study, Silva and Uchoa (2022) proposed cluster branching, which can be viewed as a special case of branching on sets. It first divides the vertices into multiple clusters based on geographical location and then defines two sets of variables, i.e., intra- and inter-cluster variables. Their empirical study demonstrated that this strategy could substantially reduce the CPU time for instances with clustered customer locations. Notably, it proved optimality for multiple CVRP and VRPTW open instances.

2.2. Branching Variable Selection

Once the type of variables to branch on is determined, the question reduces to how to make high-quality selections efficiently, which is the major focus of studies on branching strategies. The most straightforward approach is to select randomly without referring to any additional information, which, not surprisingly, leads to a large-sized BBT and thus undesirable ineffectiveness. A slightly more effective alternative is the most fractional branching, which selects the variable with the largest fractionality (Augerat et al. 1995). The pseudocost branching, proposed by Bénichou et al. (1971) for solving general MIPs, has been widely adopted because of its high effectiveness and computational efficiency. It utilizes historical information to compute pseudocosts, which requires proper initialization. Applegate et al. (1995) introduced SB for solving the traveling salesman problem, which has been widely observed to produce small-sized BBTs. However, SB is generally not practical because of the time-consuming SB score computation. Linderoth and Savelsbergh (1999) combined the strengths of pseudocost branching and SB by using SB to initialize pseudocosts and improved the performance.

Pecin et al. (2017b) proposed to combine the most fractional branching, pseudocost branching, and SB into a 3PB in the context of solving VRPs by BPC algorithms. This approach comprises an initial screening phase and two estimation phases: one by solving LPs without CG and the other with heuristic CG. Such a phased strategy significantly reduces the number of branching candidates to consider, which, in turn, allows one to afford moderate extra computation while still selecting a branching variable of close-to-best quality with high chance. The 3PB, together with

limited memory subset row cuts (Pecin et al. 2017c) and enumeration (Baldacci et al. 2008), helped to optimally solve open benchmark instances with up to 199 customers. The 3PB has since been adopted by the VRPSolver, further underscoring its efficacy and applicability.

2.3. ML for Solving VRPs

ML has proven effective in enhancing various existing algorithms for solving large-scale discrete optimization problems, including VRPs. The application of ML to approaching VRPs can be roughly categorized into two types. The first type seeks to directly construct heuristic solutions (e.g., Khalil et al. 2017, Nazari et al. 2018, d O Costa et al. 2020) or predict the feasibility of a given problem (Mai and Navet 2021, van der Hagen et al. 2024). However, these heuristic-based ML methods, especially graph neural network-based methods, struggle with generalization, scale poorly with instance size, and lack quality guarantees. Their performance degrades significantly as the number of nodes increases, both in terms of solution quality and inference time (Cappart et al. 2023). In contrast, our ML models are significantly less complex, and therefore, they demonstrate strong generalization capability while maintaining the optimality guarantee.

The second type focuses on accelerating some specific components in a sophisticated exact algorithm. Morabit et al. (2021) applied graph neural networks to pick columns likely to improve the objective value of the current RMP. Such selection significantly reduced the size of the RMP, and thus, a 30% decrease in computing time for solving the root node was achieved. In another closely related work (Morabit et al. 2022), the authors applied supervised learning (logistic regression, random forest, and neural network) to identify arcs that are most likely to be used in an optimal solution. The unselected arcs were subsequently discarded, and the resulting residual graph was used for heuristic pricing. Readers can refer to Tahir et al. (2021) and Zhang et al. (2022) for additional works falling into this category.

Despite much success achieved in applying ML to enhance branching decisions for solving general MIPs (e.g., Khalil et al. 2016, Alvarez et al. 2017, Gasse et al. 2019, Yang et al. 2022), to the best of our knowledge, there is no known work on learning to branch in BPC algorithms for solving VRPs. A slightly related work is Pereira et al. (2022), where the authors used ML to assist in solving the crew pairing problem. Note that, instead of trying to solve it exactly, they aimed to compute a feasible solution via a branch-and-price heuristic. Building upon the reviewed pioneering efforts, our paper seeks to fill this research gap.

3. Preliminaries

Before jumping into the intricacies of the proposed framework, to facilitate readability, we present necessary preliminaries about a general BPC method for solving VRPs. We first review the SPF with

a focus on the associated dynamics in the solution process. Subsequently, we discuss the branching component, underscoring the differences between its usage in BPC and in B&C for solving MIPs. Lastly, we elaborate on an effective branching strategy widely adopted in BPC algorithms, which addresses the high computational overhead of SB while keeping the BBT size relatively small.

3.1. The Set-Partitioning Formulation

The standard set-partitioning formulation (SPF) for VRPs, including the classic CVRP and VRPTW, takes the following general form:

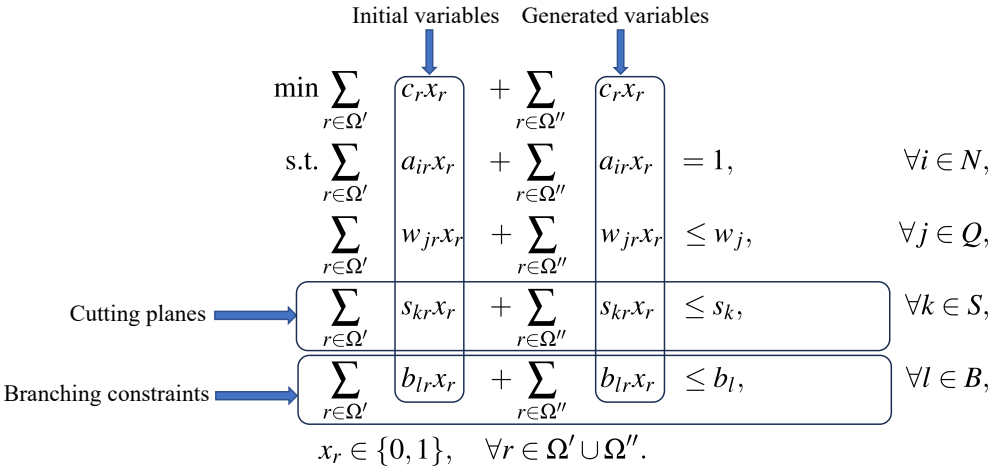
$$\begin{aligned} \min \ & \sum_{r\in\Omega} c_r x_r \\ \text{s.t.} \ & \sum_{r\in\Omega} a_{ir} x_r = 1, && \forall i \in N, \\ & \sum_{r\in\Omega} w_{jr} x_r \le w_j, && \forall j \in Q, \\ & x_r \in \{0,1\}, && \forall r \in \Omega, \end{aligned} \tag{1}$$

where N, Q, and Ω denote the sets of customers to serve, the index set for side constraints, and the set of vehicle routes, respectively. The coefficient c_r represents the cost of route r. Coefficient a_{ir} is a binary indicator, with $a_{ir} = 1$ if route r visits customer i, and 0 otherwise. The binary decision variable x_r equals 1 if route r is selected in the solution, and 0 otherwise. It is important to note that it suffices to include only elementary routes in the SPF, i.e., the customers of the route can only be visited exactly once. However, to expedite the CG process, ng-route relaxation (Baldacci et al. 2011) is often applied, which allows non-elementary routes. In this case, a_{ir} extends to a nonnegative integer representing the number of times route r visits customer i. Inequalities $\sum_{r\in\Omega} w_{jr} x_r \le w_j, \forall j \in Q$, represent the additional side constraints. For example, in the multi-depot VRP (Contardo and Martinelli 2014), constraints $\sum_{r\in\Omega_j} x_r \le m_j, \forall j \in D$, enforce the restriction on the fleet size, where D is the set of depots, m_j is the fleet size of depot j, and $\Omega_j \subseteq \Omega$ denotes the set of routes that start and end at depot $j \in D$.

Initially, only a small subset of routes $\Omega' \subseteq \Omega$ are included. Formulation (1) with Ω replaced by $\Omega' \cup \Omega''$ is referred to as the RMP, where Ω'' is the set of variables generated via CG. To solve the LP relaxation of the RMP, columns (variables) with negative reduced costs are generated iteratively via CG, also known as pricing. If the optimal solution to the RMP is fractional, branching is applied to help achieve integrality. We remark that the new variables complicate branching selection because historical data on their behaviors is lacking. Additionally, cutting planes increase the dimension of the dual problem and may even alter the structure of the pricing subproblem (Pecin et al. 2017b). Branching further complicates the situation by introducing branching constraints that accumulate as the tree grows deeper. Essentially, the BPC solution process involves alternating rounds of

column and row generation, resulting in highly dynamic changes (see Figure 1) to the RMP, which challenges existing learning-to-branch paradigms.

Figure 1 Illustration of the dynamic changes of the RMP in the BPC solution process



$$\begin{aligned}
\min \ & \sum_{r\in\Omega'} c_r x_r + \sum_{r\in\Omega''} c_r x_r \\
\text{s.t.} \ & \sum_{r\in\Omega'} a_{ir} x_r + \sum_{r\in\Omega''} a_{ir} x_r = 1, && \forall i \in N, \\
& \sum_{r\in\Omega'} w_{jr} x_r + \sum_{r\in\Omega''} w_{jr} x_r \le w_j, && \forall j \in Q, \\
& \sum_{r\in\Omega'} s_{kr} x_r + \sum_{r\in\Omega''} s_{kr} x_r \le s_k, && \forall k \in S, \\
& \sum_{r\in\Omega'} b_{lr} x_r + \sum_{r\in\Omega''} b_{lr} x_r \le b_l, && \forall l \in B, \\
& x_r \in \{0,1\}, \quad \forall r \in \Omega' \cup \Omega''.
\end{aligned}$$

3.2. The Branching Component

While branching on a nominal variable in MIPs is a popular choice, it is not the case in the BPC context because imposing $x_r = 0$ is not straightforward for CG and is often ineffective as it creates extremely unbalanced BBTs. In contrast, branching on an edge in the underlying graph requires no change in CG and leads to a more balanced tree, making it a mainstream choice in BPC algorithms (Costa et al. 2019). Branching on a given edge can be generally achieved by adding a dense constraint of the form $\sum_{r\in\Omega} b_r x_r \le 0$ or $\sum_{r\in\Omega} b_r x_r \ge 1$, where b_r counts the number of times route r traverses the edge. It is worth mentioning that such branching constraints also enforce restrictions on variables that have yet to be generated.

SB considers a set of branching candidates and performs tests by branching on each of them separately. Let z denote the optimal value of the parent LP. For a given candidate, let z_l and z_r be the LP values at the left and right child BBNs, respectively, when branched on. Then the SB score s of this particular branching choice is computed by the product rule (Achterberg 2007): $s = \max\{z_l - z,\ \epsilon\} \cdot \max\{z_r - z,\ \epsilon\}$, where $\epsilon > 0$ is typically set to 10^{-6}. SB then selects to branch on the one with the largest score. The calculation of pseudocost is similar to that of s by using historical information. Let $(\hat{x}_r)_{r\in\Omega'\cup\Omega''}$ be the solution to the parent LP. For a given edge e, let $\hat{x}(e)$ be the total flow on e computed by $\hat{x}(e) = \sum_{r\in\Omega'\cup\Omega''} a_{ir}\hat{x}_r$. Then the edge e can be considered as a branching candidate if $\hat{x}(e)$ is fractional. Let $\overline{\Delta z_l}$ and $\overline{\Delta z_r}$ denote the average value of $(z_l - z)$ and $(z_r - z)$ collected from historical branching, respectively. The pseudocost is computed as

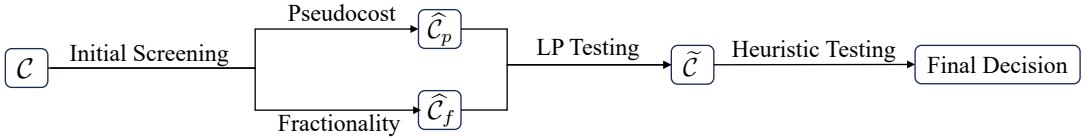
$\max\left\{(\hat{x}(e)-\lfloor\hat{x}(e)\rfloor)\cdot\overline{\Delta z_l},\ \epsilon\right\}\cdot\max\left\{(\lceil\hat{x}(e)\rceil-\hat{x}(e))\cdot\overline{\Delta z_r},\ \epsilon\right\}$. In this study, we adopt this way for our pseudocost calculation.

However, calculating such z_l and z_r values for each candidate can still need a large number of CG iterations, making this process time-consuming. To make SB faster, one can limit the number of candidates to test and apply heuristic CG in place of the much slower exact CG to solve the starting child LPs or even bypass CG completely. Without exact CG, the SB score computed is not precise, potentially compromising the quality of the branching decision. For the clarity of presentation, we thereafter refer to the procedure of branching on each candidate and solving the starting child LPs without any CG to compute an SB score as *LP testing* and refer to this SB score as *LP score*. The counterparts are called *exact testing* and *heuristic testing* when solving with exact CG and heuristic CG, respectively. The resultant SB scores are accordingly called *heuristic score* and *exact score*. Moreover, in this paper, unless stated otherwise, the term "pseudocost" always refers to the pseudocost computed using historical LP scores. Pseudocosts derived from heuristic and exact scores are termed *heuristic pseudocost* and *exact pseudocost*, respectively.

3.3. The Three Phase Branching Selection

The 3PB is considered the state-of-the-art branching strategy in BPC algorithms for solving VRPs, which combines the aforementioned two ideas for enhancing SB. In particular, it runs progressively time-consuming testing on a gradually reduced candidate list and totally avoids performing exact CG. Basically, it balances the trade-offs between the time consumed for branching decision-making and the decision quality. As suggested by its name, the 3PB (see Figure 2) comprises three phases—initial screening, LP testing, and heuristic testing—which gradually shorten the candidate list until a final decision is reached.

Figure 2 The workflow of the 3PB



In the initial screening, the complete set of branching candidates, denoted by $\mathcal{C}$, is first divided into two groups: one with initialized pseudocosts and the other without. For the former group, the top $\hat{\theta}_p$ candidates, denoted by $\widehat{\mathcal{C}}_p$, are selected based on pseudocosts, while for the other group, the top $\hat{\theta}_f$ candidates with the highest fractionality, denoted by $\widehat{\mathcal{C}}_f$, are chosen. The parameters $\hat{\theta}_p$ and $\hat{\theta}_f$ are typically set equal and let $\hat{\theta}_{pf} := \hat{\theta}_p + \hat{\theta}_f$. The $\hat{\theta}_{pf}$ candidates in $\widehat{\mathcal{C}}_p \cup \widehat{\mathcal{C}}_f$ then go through LP testing and the top $\tilde{\theta}$ candidates ranked by LP scores constitutes the set $\widetilde{\mathcal{C}}$. Lastly, heuristic testing is performed on candidates in $\widetilde{\mathcal{C}}$, and the one with the largest heuristic score is the final decision.

4. The Learning-to-Branch Framework

Similar to Khalil et al. (2016), Alvarez et al. (2017), Yang et al. (2022), and the various related studies cited therein, our goal is to apply ML to mimic SB in a computationally efficient manner. Unfortunately, learning-to-branch paradigms used in Yang et al. (2022) that have been proven to work well in the context of solving MIPs fail to achieve any prediction accuracy above 20% according to our preliminary experiments, confirming that learning to branch in BPC algorithms is much more difficult. Such difficulty stems from causes explained in Section 1.1, which are addressed by our novel learning-to-branch framework.

4.1. The High-Level Idea

As explained in Section 1.1, 3PB lacks the critical information necessary for efficiently making good branching decisions, and the three mentioned challenges prevent us from leveraging proven useful features from the literature for learning to optimize MIPs. In this section, we first identify new features to overcome the second and third challenges—the absence of historical information about recently generated (nominal) variables and the ineffectiveness of standard static features caused by the accumulated changes in the RMP. For the second challenge, we propose substituting the missing information with features of implicit variables. Since branching occurs on an edge in the graph—corresponding to a static variable in an equivalent compact formulation that remains unaffected by the CG process—historical data on these implicit variables can be collected throughout the solution process. To address the third challenge, our framework uses features derived from the graph (e.g., the depot and customer locations and the induced proximities), which offer consistent predictive power and require one-time computation, as a substitute for static RMP-derived features.

The first challenge—the unknown beforehand deviation of an ending child LP from its parent LP—is particularly difficult to overcome. Even with the features mentioned above, achieving accurate predictions for the deviation remains challenging. Performing exact CG, as is done in exact testing is the only reliable way to obtain this uncertain deviation. However, exact testing is often time-consuming, even when evaluating just two candidates. Our second stage aims to limit its use to necessary cases. Specifically, when the deviation between an ending child LP and its starting LP is expected to be minimal, exact testing for that starting child LP can be skipped without risk. However, directly predicting this deviation is as complex as predicting the exact SB score since knowing the deviation would allow for deriving the SB score. Drawing from the success of *ranking* methods in learning-to-branch tasks for MIPs, our second-stage model focuses on predicting the rank of the *deviation ratio*, ρ, for each starting child LP. Let $s := z_1 - z$ and $\hat{s} := z_2 - z$, where z, z_1, and z_2 are the optimal values of the corresponding parent, starting, and ending child LPs,

respectively. This deviation ratio, defined as $\rho := \frac{s}{\hat{s}} \geq 1$, reflects the similarity between the starting and ending child LPs and thus serves as a measure of the necessity for exact testing. By ranking the ρ values for all candidates—each with two starting child LPs (one from the left branch and the other from the right)—we can selectively limit exact testing to only those starting child LPs with high predicted deviation.

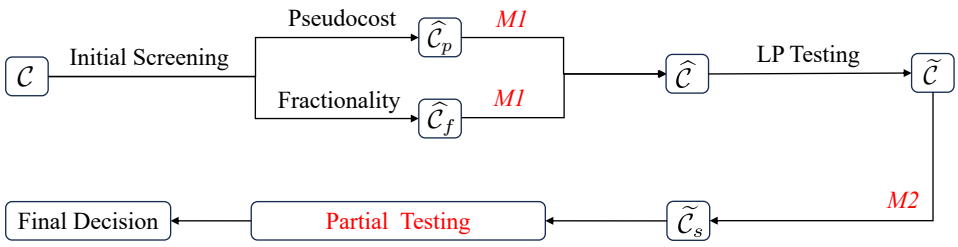
4.2. The Workflow

Our framework builds upon the cutting-edge hand-crafted 3PB detailed in Section 3.3. Recall that 3PB has two primary weaknesses. The first is the less effective initial screening, which leads to solving a large number of (often in the hundreds) starting child LPs without CG in the LP testing phase. The second is that it needs to perform several heuristic testing to determine the best candidate from those identified by LP testing. Although the number of heuristic testing may not be very large, each testing involves multiple iterations of heuristic CG and solving intermediate LPs, making it potentially over ten times slower than an LP testing.

To overcome these two weaknesses, our framework employs two ML models: one before the LP testing phase (*M1*) and one after LP testing (*M2*), as illustrated in Figure 3. *M1* further selects promising candidates from the set given by initial screening, thereby reducing the need to solve hundreds of LPs without CG in the LP testing phase. *M2* aids in selecting a subset of starting child LPs for testing while ensuring decisions match those from testing *all* starting child LPs. We term such testing as *partial testing*.

Specifically, in the first stage, *M1 mimics the behavior of performing exact testing on candidates that have passed the initial screening*, with the learning goal of predicting the ones with the highest exact scores. In the second stage, *M2 predicts whether the starting child LP value* (z_1) *accurately approximates the ending child LP value* (z_2) *on the candidates that have passed LP testing*. Generally speaking, starting child LPs expected to have a large deviation ratio will be subjected to partial testing for a more precise estimate of z_2. Details on the second stage are provided in Section 4.4.

Figure 3 The workflow of our learning-to-branch framework



4.3. The First Stage Selection

We propose a phased design with two stages of ML (see Figure 3) that define two sequential learning tasks. The first task utilizes easily computable features to filter out a large proportion of unpromising candidates from $\widehat{\mathcal{C}}_p$ and $\widehat{\mathcal{C}}_f$, producing a refined candidate set, $\widehat{\mathcal{C}}$. Notably, *M1* is applied separately to $\widehat{\mathcal{C}}_p$ and $\widehat{\mathcal{C}}_f$ to obtain $\widehat{\mathcal{C}}$, ensuring that some candidates from $\widehat{\mathcal{C}}_f$, if any, are retained. Without this treatment, our experiments indicate that *M1* tends to choose candidates almost exclusively from $\widehat{\mathcal{C}}_p$ when $\widehat{\mathcal{C}}_p \cup \widehat{\mathcal{C}}_f$ altogether is fed into *M1*, leading to unsatisfactory performance.

The reason is mainly twofold. First, the candidates in $\widehat{\mathcal{C}}_p$, i.e., those with pseudocosts initialized, are of higher quality, in an average sense, than those in $\widehat{\mathcal{C}}_f$ because the pseudocost of a variable is initialized when it has successfully passed the initial screening. In contrast, the pseudocosts of candidates in $\widehat{\mathcal{C}}_f$ have not been initialized so far, indicating that they failed such screening. Second, the pseudocosts get initialized fast as the BBT grows, causing $\widehat{\mathcal{C}}_f$ to vanish quickly. As a result, $\widehat{\mathcal{C}}_p$ constitute a vast majority of the training data, so *M1* naturally prioritizes members in $\widehat{\mathcal{C}}_p$. Nonetheless, some candidates from $\widehat{\mathcal{C}}_f$ can still be good branching choices, especially for branching decisions near the root of the BBT, where most of the pseudocosts have not yet been initialized. It is essential not to miss such candidates considering the decisions close to the root are generally of great importance (Pecin et al. 2017b, Pessoa et al. 2020).

4.4. The Second Stage Selection

The key idea of the second stage selection is to utilize the predicted ranking of ρ (output from *M2*) to avoid unnecessary testing. More specifically, this selection process relies on Conditions (2)-(4), proven to be sufficient in Proposition 1. It is based on the straightforward intuition: for two candidates $e_i, e_j \in \widetilde{\mathcal{C}}$, if the current testing score, not necessarily the exact score, of e_i exceeds that of e_j, and the deviation ratio of e_i is no larger than that of e_j, then e_j can be removed from $\widetilde{\mathcal{C}}$, as its exact score cannot surpass that of e_i. Further details are provided in Section 4.4.2.

It is rare, however, that these three conditions are satisfied for all candidate pairs (e_*, e_j) at the beginning, where $e_* \in \widetilde{\mathcal{C}}$ has the highest exact score. To address this, Algorithm 1 conducts exact testing on a subset of the starting child LPs in a dynamic manner to ensure these conditions hold across all pairs (e_*, e_j). Consequently, all such e_j will be removed from $\widetilde{\mathcal{C}}$, leaving only e_*, thus achieving the selection goal.

4.4.1. Assumptions on the behavior of *M2*. Let l_i and r_i denote the left and right starting child LPs, respectively, generated by adding the branching constraint associated with the i-th candidate $e_i \in \widetilde{\mathcal{C}}$ and define $\widetilde{\mathcal{C}}_s := \bigcup_{i=1}^{|\widetilde{C}|} \{l_i, r_i\}$ as the set of all starting child LPs. Given the input features, *M2* outputs a rank by the ρ values for each starting child LP in $\widetilde{\mathcal{C}}_s$. We formalize some

assumptions based on the observed behavior of *M2*, which relate the deviation ratio, ρ, to the output ranking of *M2*, denoted by d.

Assumptions:

(a) d is a function of the deviation ratio, i.e., $d = h(\rho)$, where $h : [1, \infty) \to \mathbb{Z}_+ \cup \{-1\}$.

(b) h is a monotonically increasing function such that $h(\rho) = -1$ if and only if $\rho = 1$.

(c) $h(\rho_1) = h(\rho_2) \leq \underline{d}$ implies that $\max\left\{\frac{\rho_1}{\rho_2}, \frac{\rho_2}{\rho_1}\right\} \leq \delta$, where $\underline{d}$ and δ are predefined parameters.

If *M2* achieves perfect accuracy, the assumptions can be justified as follows: (a) ρ should be the sole factor influencing the prediction d, thereby ensuring the existence of h; (b) d should increase monotonically as deviation ratio ρ increases, aligning with the learning target of *M2*; (c) Based on our observation, a small d indicates that s is a strong approximation of $\widehat{s}$, and thus, the corresponding ρ_1 and ρ_2 should be close to 1 when $h(\rho_1) = h(\rho_2) \leq \underline{d}$. While the same d value does not guarantee identical ρ values, a minor difference is expected since both ρ_1 and ρ_2 are close to 1 in this case. Therefore, it is reasonable to assume that $\max\left\{\frac{\rho_1}{\rho_2}, \frac{\rho_2}{\rho_1}\right\}$ is bounded by a constant δ.

4.4.2. Conditions to ensure a larger exact score. The conditions introduced here are fundamental to the proposed partial testing procedure. They establish sufficient criteria to prove that e_i has a larger exact score than e_j, even without knowing their exact scores. To proceed, we first clarify the notation. Let S_k denote the difference between the optimal value of the **current** child LP $p_k \in \widetilde{\mathcal{C}}_s$ and its parent LP value, and D_k represent the **current** deviation ratio. Note that "current" signifies dynamic determination based on the state of testing. If a child LP $p_k \in \widetilde{\mathcal{C}}_s$ has not yet been tested, S_k is set to s_k, and D_k is set to d_k. Otherwise, S_k is updated to $\widehat{s}_k$, and D_k is updated to $h(1) = -1$ to indicate that p_k has been tested. For clarity, when no superscript is attached, we do not distinguish the left or right child LP. When necessary, we employ the superscript l for the left child and r for the right, e.g., S_i^l and S_i^r denote the S values of the corresponding left and right child LPs associated with branching candidate e_i. The current score of e_i is represented as $S_i^{l\times r} := S_i^l \times S_i^r$ and its exact score is denoted by $\widehat{s}_i^{l\times r} := \widehat{s}_i^l \times \widehat{s}_i^r$.

Definition: We define e_i as a *better* branching candidate than e_j if the following conditions hold for the candidate pair (e_i, e_j):

$$S_i^{l\times r} > S_j^{l\times r} \cdot \delta^{\phi_1+\phi_2}, \tag{2}$$

$$\left(D_i^{\max} < D_j^{\max} \text{ and } \phi_1 = 0\right) \text{ or } \left(D_i^{\max} = D_j^{\max} \leq \underline{d} \text{ and } \phi_1 = \mathbb{1}_{D_i^{\max} \neq -1}\right), \tag{3}$$

$$\left(D_i^{\min} < D_j^{\min} \text{ and } \phi_2 = 0\right) \text{ or } \left(D_i^{\min} = D_j^{\min} \leq \underline{d} \text{ and } \phi_2 = \mathbb{1}_{D_i^{\min} \neq -1}\right), \tag{4}$$

where $D_i^{\max} := \max\{D_i^l, D_i^r\}$, $D_i^{\min} := \min\{D_i^l, D_i^r\}$, $D_j^{\max} := \max\{D_j^l, D_j^r\}$, and $D_j^{\min} := \min\{D_j^l, D_j^r\}$.

Condition (2) specifies that the score $S_i^{l\times r}$ for e_i must be greater than the score for e_j, scaled by a factor $\delta^{\phi_1+\phi_2}$. Conditions (3) and (4) further require that the $D_i^{\max}$ and $D_i^{\min}$ values of e_i are either strictly less than those of e_j or, if equal, must not exceed the predefined threshold $\underline{d}$. Together, these conditions ensure that e_i can be regarded as a better branching candidate than e_j, as they provide sufficient evidence that e_i achieves a larger exact score than e_j, as demonstrated in Proposition 1.

PROPOSITION 1. *Conditions* (2), (3), *and* (4) *are sufficient to prove that* e_i *has a larger exact score than* e_j, *i.e.,* $\widehat{s}_i^{l\times r} > \widehat{s}_j^{l\times r}$.

By verifying the conditions outlined in Proposition 1, it is possible to replicate the selection made by SB without requiring explicit computation of the exact scores, thereby significantly reducing computational complexity. Given a pair of candidates, the one with a larger current score $S^{l\times r}$ is referred to as the *Leader*, and the other is the *Competitor*. The candidate $e_* \in \widetilde{\mathcal{C}}$ that is better than the others is called the *Winner*, and the goal of our selection process is to identify it with the minimum necessary testing.

4.4.3. The partial testing procedure. In the ideal scenario, these conditions would hold for all candidate pairs (e_*, e_j) from the very beginning, making e_* the Winner, thereby eliminating the need for any additional testing. However, as indicated by our preliminary experiments, this rarely occurs in practice. Candidates with larger LP scores often tend to have larger ρ values, preventing the three conditions from holding simultaneously. To eventually satisfy these conditions, exact testing on some starting child LPs needs to be performed to update the $S^{l\times r}$ and D values.

The challenge lies in strategically selecting a subset of $\widetilde{\mathcal{C}}_s$ for testing to meet the conditions as efficiently as possible. For a given pair (e_i, e_j) with $S_i^{l\times r} > S_j^{l\times r}$, the strategy is to prioritize e_i, and perform testing on its starting LPs, i.e., l_i and r_i. If, after testing, the updated $S_i^{l\times r}$ remains larger, then the conditions are satisfied, eliminating the testing need for e_j. On the other hand, starting with l_j (or r_j) may lower both $S_j^{l\times r}$ and D_j^l (or D_j^r), and testing for e_i is still required.

The following Algorithm 1 outlines the aforementioned partial testing procedure. The procedure begins by initializing S_k and D_k to s_k and d_k, respectively, for all starting child LPs $p_k \in \widetilde{\mathcal{C}}_s$. In the subsequent sorting step, the candidates are sorted based on their current scores $S^{l\times r}$. The top two candidates, e_i and e_j, are selected to check if they satisfy Conditions (2), (3), and (4). If these conditions are satisfied, e_j will be removed from $\widetilde{\mathcal{C}}$. If not, exact CG will be performed on p_k, which is chosen as either the left starting child LP l_i or the right r_i depending on the relative values of D_i^l and D_i^r. Following this, S_k will be updated to $\widehat{s}_k$, and D_k will be updated to -1, after which the sorting step will be repeated.

Algorithm 1: The Partial Testing Procedure

Step 1. Sorting: If $|\widetilde{C}|=1$, **Stop**.
Sort candidates in $\widetilde{C}$ in descending order by $S^{l\times r}$, identify top candidates e_i, e_j.
Step 2. Checking: Set $p_k \leftarrow$ NULL.
If Condition (3) failed: if $D_i^l \geq D_i^r$, set $p_k \leftarrow l_i$, otherwise set $p_k \leftarrow r_i$.
If Condition (4) failed: if $D_i^l \leq D_i^r$, set $p_k \leftarrow l_i$, otherwise set $p_k \leftarrow r_i$.
Step 3. Testing:
If p_k *equals* l_i*:* Compute $\widehat{s}_i^l$, and update $S_i^l \leftarrow \widehat{s}_i^l$, $D_i^l \leftarrow -1$, $S_i^{l\times r} \leftarrow S_i^l \times S_i^r$;
If p_k *equals* r_i*:* Compute $\widehat{s}_i^r$, and update $S_i^r \leftarrow \widehat{s}_i^r$, $D_i^r \leftarrow -1$, $S_i^{l\times r} \leftarrow S_i^l \times S_i^r$;
Otherwise: Eliminate e_j from $\widetilde{C}$.
Go to Step 1.

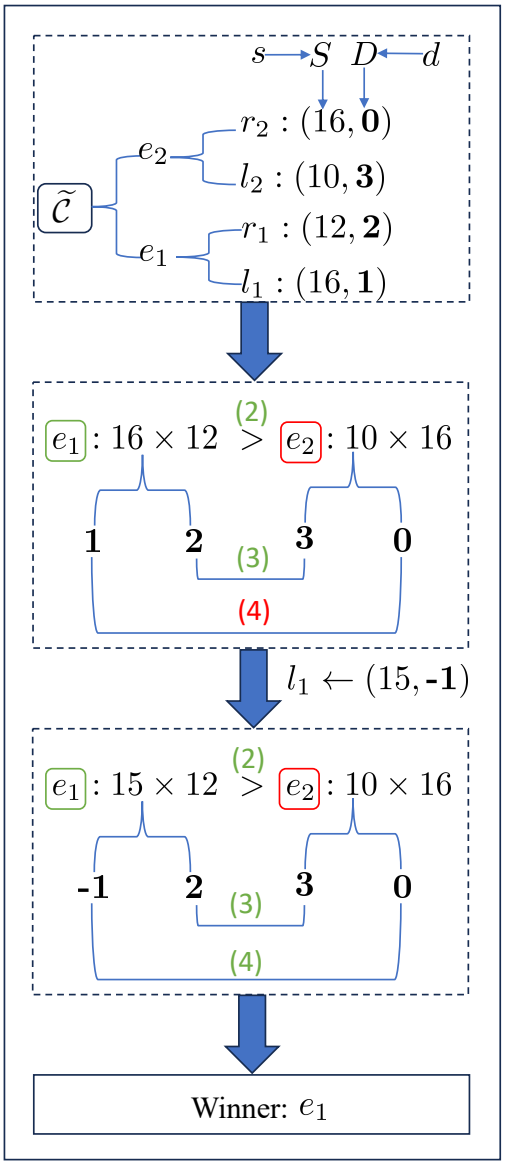


Figure 4: An illustrative example of the partial testing process in the proposed framework

An illustration example is provided in Figure 4, where $\underline{d}=3$, and the set of candidates $\widetilde{\mathcal{C}}=\{e_1,e_2\}$, with initial values $(S_1^l, D_1^l)=(16,\mathbf{1}), (S_1^r, D_1^r)=(12,\mathbf{2})$, $(S_2^l, D_2^l)=(10,\mathbf{3})$, and $(S_2^r, D_2^r)=(16,\mathbf{0})$. In Figure 4, the green square around e_1 indicates that e_1 is the Leader, while the red square around e_2 signifies that e_2 is the Competitor. Conditions (2), (3), and (4), are indexed as (2), (3), and (4), respectively, with the green color indicating a satisfied status and red for an unsatisfied status.

During the sorting step, e_1 is determined to be the Leader because $S_1^{l\times r}=16\times 12=192$ is larger than $S_2^{l\times r}=10\times 16=160$. Consequently, e_2 is designated as the Competitor. The algorithm proceeds to the checking step (*Step 2*), where Conditions (3) and (4) are evaluated. Since the Condition (4) is not satisfied, l_1 is chosen to compute $\widehat{s}_1^l$, which updates S_1^l from 16 to 15 and sets D_1^l to -1. The algorithm then returns to the sorting step (*Step 1*). Despite the update, e_1 remains the Leader because 15×12 is still greater than 10×16. The checking step is repeated, and this time both Conditions (3) and (4) are satisfied, so no further testing is required, and e_2 is removed from $\widetilde{\mathcal{C}}$. This leaves e_1 as the only candidate in $\widetilde{\mathcal{C}}$ and thus the Winner. The following Proposition 2 confirms that after a finite number of iterations, the Winner can be identified by Algorithm 1.

PROPOSITION 2. *Algorithm 1 terminates in a finite number of steps and identifies the Winner* e_* *that is better than any other candidate* $e_j \in \widetilde{\mathcal{C}}$.

4.4.4. An acceleration technique. When $|\widetilde{\mathcal{C}}|$ is large, Algorithm 1 can still require significant computation to identify e_*. We propose to address it by reusing information obtained from testing that has already been conducted. Given Assumptions (b) and (c), the following Proposition 3 estimates the exact score by approximating ρ based on other known deviation ratio values.

PROPOSITION 3. *Given a set of pairs* $\mathcal{S} := \{(\rho_\ell, h(\rho_\ell))\}_\ell \cup \{(M, M), (1, -1)\}$ *with* M *sufficiently large and a candidate pair* (e_i, e_j). *If* $D_i = -1$, *set* $\rho_1 = 1$, *otherwise, let* $\rho_1 = \min_{(\rho_k, h(\rho_k))\in\mathcal{S}}\{\rho_k : h(\rho_k) > D_i\}$. *Define* $\rho_2 := \max_{(\rho_k, h(\rho_k))\in\mathcal{S}}\{\rho_k : h(\rho_k) < D_j\}$. *It holds that* e_i *has a larger exact score than* e_j, *i.e.,* $\widehat{s}_i^{l\times r} > \widehat{s}_j^{l\times r}$, *if*

$$S_i^{l\times r} > S_j^{l\times r} \cdot \delta_a \tag{5}$$

is satisfied, where $\delta_a = \min\left\{a(D_i^l, D_j^l)\cdot a(D_i^r, D_j^r),\ a(D_i^l, D_j^r)\cdot a(D_i^r, D_j^l)\right\}$, *and*

$$a(D_1, D_2) = \begin{cases} \min\{\delta, \frac{\rho_1}{\rho_2}\}, & \text{if } D_1 = D_2 \le \underline{d}, \\ \min\{1, \frac{\rho_1}{\rho_2}\}, & \text{if } D_1 < D_2, \\ \frac{\rho_1}{\rho_2}, & \text{if } D_1 > D_2 \text{ or } D_1 = D_2 > \underline{d}. \end{cases}$$

Condition (5) is checked at the beginning of *Step 2* in Algorithm 1. If it is satisfied, *Step 2* terminates directly with p_k set to NULL and it moves directly to *Step 3*. In practice, testing four starting child LPs is often sufficient for Condition (5) to conclude that no further testing is necessary. This significantly accelerates the entire selection process, especially when $|\widetilde{\mathcal{C}}|$ is 10 or larger.

Remarks: Although the heuristic testing in 3PB and the proposed partial testing procedure in our 2LBB expedite the selection through different mechanisms, the best numerical performance is achieved when these approaches are combined. Specifically, we replace the exact CG with heuristic CG in Algorithm 1 in our implementation when performing the testing. This is motivated by the observation that heuristic CG is significantly faster than exact CG, and more importantly, the optimal value obtained from heuristic CG serves as a reliable way to approximate that of the ending child LP, leading to the same selection outcome as exact CG in most cases. While Propositions 1 and 3 may no longer guarantee sufficient conditions for certifying a larger exact score, the reliability of these approximations ensures that such cases are rare in practice.

4.5. Learning Methodology

The two ML models, *M1* and *M2*, in our ML module aim to predict the correct ranking with different learning goals. For *M1*, the goal is to rank candidates based on their exact scores, while for *M2*, the goal is to rank candidates based on the deviation ratio. *Learning to rank*, compared to more common supervised learning methods like regression or classification, tends to achieve higher accuracy in our task because the learning objective is relatively easier.

The input for a ranking task typically consists of multiple lists, each containing several items. For the k-th list with n_k items, let $\vec{f}_i^k \in \mathbb{R}^d$ denote the d-dimensional feature vector of item i, and $r_i^k \in \{1, \ldots, n_k\}$ its rank in descending order based on exact scores or the deviation ratio. Each list can be expressed as (x^k, y^k), where $x^k = \{\vec{f}_1^k, \ldots, \vec{f}_{n_k}^k\}$ and $y^k = \{r_1^k, \ldots, r_{n_k}^k\}$, forming training data for learning-to-branch tasks. However, directly using ranks as labels is unnecessary since candidates with similar scores typically have comparable performance. To reduce overfitting and enhance generalization, we recompute labels $\bar{y}_i^k$ following the approach in Yang et al. (2022):

$$\bar{y}_i^k = W - \max\left\{0, \left\lceil \frac{\tilde{s}^k - s_i^k}{\tilde{s}^k - s_{(n_k)}^k} \cdot W \right\rceil\right\}, \quad \text{where} \quad \tilde{s}^k = \alpha \cdot s_{(1)}^k, \quad \alpha \in (0,1). \tag{6}$$

Candidates with scores above the threshold $\tilde{s}^k$ (commonly set with $\alpha = 0.8$) receive the maximum label W, indicating equally optimal choices for branching.

We characterize each branching candidate using three types of features: $\mathcal{F}_E$, $\mathcal{F}_I$, and $\mathcal{F}_D$. The $\mathcal{F}_E$ features provide detailed LP-related information, including dual solution values and changes in LP objective values obtained from solving child LPs without using CG, which are computationally expensive. To the best of our knowledge, the $\mathcal{F}_E$ features are completely new in the literature. In contrast, $\mathcal{F}_I$ and $\mathcal{F}_D$ are inexpensive to compute. The $\mathcal{F}_I$ features capture the fundamental properties of candidates that are independent of previous LP results, whereas the $\mathcal{F}_D$ features are derived from previously computed $\mathcal{F}_E$ values. Together, these features support a two-stage learning approach (*M1* and *M2*), which effectively selects and ranks branching candidates. For detailed descriptions of feature definitions, computations, data collection, and training procedures, please refer to Section EC.3 of the e-companion.

5. A Theoretical Model for Determining the Number of Candidates

Branching strategies, such as the SB, 3PB, and 2LBB, require performing computationally intensive testing on a set of branching candidates to achieve small-sized BBTs. Nonetheless, determining the size of the set to test so that the overall solution time is minimized remains challenging because of a lack of a systematic way to balance the trade-off between the time consumed for making a single branching decision and the decision quality. The size is typically decided by a trial-and-error numerical approach and remains unchanged once decided. In Section EC.7.2 of the e-companion, we numerically decide a good static $\hat{\theta}$ for the 2LBB as a baseline. In this section, we propose a novel theoretical model that theoretically decides the best size based on the solution progress, which is the first to embed a distribution of the local branching improvements into a tree size estimation. More precisely, we derive the so-called *Best K Formula* (BKF) to dynamically adjust the size to achieve the minimum total computational time under our assumptions.

The BKF is broadly applicable to branching strategies that employ one-phase testing, such as SB, or multi-phase testing where either only one phase's duration can vary (e.g., 3PB) or one phase dominates the overall computation time (e.g., 2LBB). We refer to such a phase as the *key testing phase*. In 3PB and 2LBB, LP testing generally serves as the key phase. Notably, in 2LBB, this phase often accounts for over 80% of the total branching time. Therefore, BKF determines $\hat{\theta}_{pf}$ in 3PB and $\hat{\theta}$ in 2LBB. For multi-phase strategies, the applicability of BKF hinges on the assumption that the total branching variable selection time is approximately equal to the time spent on the key testing phase, plus $\tilde{c}$ representing the combined duration of other phases. If $\tilde{c}$ is negligible compared to the key phase, it can be treated as a constant. Unless stated otherwise, all proofs related to these derivations are included in Section EC.2 of the e-companion.

5.1. Derivation of the BKF

Given a set of n branching candidates, we seek to decide the number, k, of candidates randomly chosen from this set to go through the key testing phase such that the minimum possible time for solving the problem is achieved. Our theoretical model builds upon the following assumptions.

Assumptions

(d) The BBT is a binary tree and thus $n_{\text{inner}} = n_{\text{leaf}} - 1$, where n_{inner} is the number of inner BBNs, and n_{leaf} is the number of leaf BBNs. The time for solving each inner BBN consists of those consumed by branching, pricing, and adding cuts. For leaf BBNs, branching is not needed and the time only comprises the other two parts.

(e) It takes a constant time $2t$ to perform testing for a single branching candidate (two branches), a constant time $\tilde{c}$ for other phases in making a branching decision, and a constant time c' for pricing and adding cuts at a BBN. Moreover, $c \geq t$, where $c := c' + \frac{\tilde{c}}{2}$.

(f) The tree size estimation follows Le Bodic and Nemhauser (2017). For a given branching candidate, the corresponding dual bound improvements at the left and right child BBNs, denoted by o_1 and o_2, remain constant with $o_1 \geq o_2 \geq 1$. We define *testing score* r to be the geometric mean of o_1 and o_2, i.e., $r := \sqrt{o_1 \cdot o_2}$. The value of r varies for different branching candidates.

(g) The testing score for the n branching candidates take distinct values from $\left\{\frac{j}{n} \cdot R \colon\ j = 1, \cdots, n\right\}$, where R is maximum testing score. The value of r_i for candidate i is not known until it undergoes the testing.

Let $N(G)$ denote the size of the subtree rooted at the current BBN with a local optimality gap of G. In particular, for each leaf BBN, $N(0) = 1$. According to Le Bodic and Nemhauser (2017), $N(G)$ can be approximated by $N(G) = \varphi^{G-F} N(F)$, where F is no larger than G, and φ is the unique solution to the equation $x^{o_1} - x^{o_1 - o_2} - 1 = 0$. Given that φ does not admit a closed form, analyzing the properties of equations involving φ is complex. Therefore, we propose to approximate φ by $\varphi_g := 2^{\frac{1}{r}}$. The following Theorem 1 characterizes the quality of such approximation.

THEOREM 1. *The approximation error* $\left|\frac{\varphi_g-\varphi}{\varphi_g}\right| \le \frac{2^{-\frac{1}{o_1 v}}|2^{\frac{1}{v}}-2^{-v+\frac{1}{v}}-1|}{o_1 v^2}$, *where* $v := \sqrt{\frac{o_2}{o_1}} \in (0,1]$. *Moreover, the error is monotonically decreasing in terms of* v *when* $v \ge \frac{1}{3}$.

Theorem 1 bounds the approximation error. In particular, when $o_1 = 10$ and $v = \sqrt{\frac{1}{2}}$, which is typical for VRP instances, the error is no greater than 0.006. It is worth mentioning that this approximation only needs to be applied to the best one from the set of candidates considered and a good candidate usually has a value of o_2 close to that of o_1, making v close to 1 and thus the error bound close to 0. Consequently, φ_g serves as a good approximation and the tree size $N(G)$ is estimated to be $N(F)\cdot 2^{\frac{G-F}{r}}$ in our subsequent analysis.

The total computation time for the subtree is calculated as $T(k) = (c' + \tilde{c} + 2kt)n_{\text{inner}} + c' \cdot n_{\text{leaf}} = c'N(G) + (\frac{\tilde{c}}{2} + kt)(N(G)+1) \approx (c' + \frac{\tilde{c}}{2} + kt)N(G) = (c+kt)N(G)$. When the calculation is performed at the root, $T(k)$ estimates the total time it takes to solve the problem. Combining the above two equations leads to

$$T(k) \approx (c+kt)N(F)\cdot 2^{\frac{G-F}{r}}. \tag{7}$$

In BPC algorithms, enumeration tends to succeed when the local gap is small enough, helping close a BBN fast. Therefore, the subtree rooted at the current BBN, in this case, approximately has a constant size. In other words, when F is close to 0, $N(F)$ can be considered constant that is independent of k or r. If $N(F)$ can be estimated accurately, such information should be exploited. Otherwise, we can always use the case $F = 0$ with $N(0) = 1$ in Equation (7).

Intuitively speaking, increasing the number of candidates to test is likely to result in a higher testing score because of the increased probability of identifying the best candidate. The monotone relationship between r and k poses challenges in determining the value of k that minimizes $T(k)$. To address this, we first build a quantified connection between r and k. Let $j_1, \cdots, j_k$ be the k randomly selected from the n candidates. After testing, the candidate with the highest score among the k tested ones is selected. Thus, in view of Assumption (g), the final testing score $r = \max\{r_{j_1}, \ldots, r_{j_k}\}$ is a random variable. We proceed by examining the following two cases.

(i) Pure random selection. This case applies when there is no prior knowledge about the n candidates such as in the SB. The expected value of $T(k)$ is computed by

$$\mathbb{E}[T(k)] = \mathbb{E}\left[(c+kt)N(F)\cdot 2^{\frac{G-F}{r}}\right] = (c+kt)N(F)\sum_{i=k}^{n}\frac{\binom{i-1}{k-1}}{\binom{n}{k}}2^{\frac{G-F}{\frac{i}{n}\cdot R}}. \tag{8}$$

(ii) Conditional random selection. The selection is conditioned on that only candidates with a testing score larger than $\frac{(n-m)R}{n}$ will be selected.

$$T_k := \mathbb{E}[T(k)|m] = \mathbb{E}\left[(c+kt)N(F)\cdot 2^{\frac{G-F}{r}}\middle| m\right] = (c+kt)N(F)\sum_{i=k}^{m}\frac{\binom{i-1}{k-1}}{\binom{m}{k}}2^{\frac{G-F}{\frac{i+n-m}{n}\cdot R}}. \tag{9}$$

Case (ii) applies to the usage of *M1* because *M1* possesses some basic prediction power enabling it to effectively exclude $(n-m)$ unpromising ones. The output $k=\hat{\theta}$ candidates are considered as being randomly selected from the best m ones. When $m=n$, case (ii) reduces to (i), so case (i) can be considered as a special case. Therefore, we focus on case (ii) in the following discussion.

PROPOSITION 4. *Given two branching strategies with a respective constant number of testing k_1 and k_2 $(k_1>k_2)$ and respective m values of m_1 and m_2, the ratio of their expected solving time for the same instance $\tau := \frac{\mathbb{E}[T(k_1)|m_1]}{\mathbb{E}[T(k_2)|m_2]}$ decreases as ω increases, where $\omega := \frac{c}{t}$.*

The above proposition can be readily proved as follows. For the same instance, c and t remain unchanged across different strategies. In view of Equation (9), it follows that τ is proportional to $\frac{\omega+k_1}{\omega+k_2}$. Since $k_1>k_2$ and $\frac{\omega+k_1}{\omega+k_2}=1+\frac{k_1-k_2}{\omega+k_2}$, we have that τ decreases with ω. Proposition 4 suggests that for instances with a larger ω, overestimating the parameter k is less detrimental when $\tau>1$. This insight can provide guidance on tuning k for instances with distinct structures. Additionally, upon successful enumeration at a BBN, pricing becomes much faster, i.e., ω significantly decreases. In this case, a larger k is less desirable, and thus k should be reduced. Such a change in ω will be further studied numerically in Section 6.

PROPOSITION 5. *Given two branching strategies selecting the same number of candidates, i.e., $k_1=k_2=k$, but with $m_1>m_2$, it follows that $\tau>1$.*

Proposition 5 indicates that for two strategies using the same number of testing but differ in their m values, the one with a larger m will have a longer computational time than the other. Intuitively, a smaller m suggests a higher selection quality, leading to a more effective branching scheme. This correlation between m and τ is further supported by the comparison of the two case studies in Sections 6.1 and 6.2.

Let $k^*\in\arg\min_{1\le k\le m} T_k$, i.e., when k is set to k^*, the expected computational time is minimized. When m is relatively small, k^* can be obtained by computing T_k for $k=1,\cdots,m$, and select the smallest one. However, for large m, such computations become prohibitively expensive. In the next section, we provide an efficient way to compute k^*.

5.2. Asymptotic Study

In this section, we analyze the asymptotic behavior of the total computational time T_k as $n\to\infty$, which gives rise to a practical way of dynamically adjusting the key parameter $\hat{\theta}$ for *M1*. In this analysis, *M1* is considered to have a fixed prediction power in the sense that its m value grows proportionally with n. In other words, $\alpha := \frac{m}{n}$ is a constant with $\alpha\in(0,1)$. We demonstrate that k^* can be computed without explicitly computing T_k for every k in the range $\{1,\cdots,m\}$. More

specifically, we narrow down the possible values of k^* to just several options. Additionally, we establish that as $n \to \infty, T_k$ possesses a definite integral representation. This integral form facilitates a more efficient computation of T_k. Let $\mathbb{E}[r|m]$ be the conditional expectation of r given m. The following Lemma 1 gives a closed form for computing it.

LEMMA 1. $\mathbb{E}[r|m] = \sum_k^m \frac{\binom{i-1}{k-1}}{\binom{m}{k}} \frac{i+n-m}{n} R = \frac{(n+1)k+n-m}{n(k+1)} R$.

Define $\underline{T}_k := (c+kt)N(F)2^{\frac{G-F}{\mathbb{E}[r|m]}}$. In view of the convexity of the function $f(x) = 2^{1/x}$ and Jensen's inequality, $\underline{T}_k$ lower bounds T_k. For any given $k \in \{1, \cdots, m\}$, the following Theorem 2 establishes the existence of limits for T_k and $\underline{T}_k$ as $n \to \infty$, which enables us to focus on $\underline{T}_k$ when analyzing the asymptotic behavior of T_k.

THEOREM 2. *For $k \in \{1, \cdots, m\}$, the limits $\widehat{T}_k := \lim_{n\to\infty} T_k$ and $\widehat{\underline{T}}_k := \lim_{n\to\infty} \underline{T}_k$ exist. Moreover, $\widehat{T}_k = (c+kt)kN(F)\int_0^1 x^{k-1} B^{\frac{1}{\alpha x+1-\alpha}} \mathrm{d}x$ and $\widehat{\underline{T}}_k = (c+kt)N(F)B^{\frac{1}{1-\alpha/(k+1)}}$, where $B := 2^\beta$ and $\beta := \frac{G-F}{R}$.*

THEOREM 3. *The sequence $\widehat{U}_k := (c+kt)kN(F)B^{\frac{1}{1-\alpha}} \left(\frac{\alpha}{1-\alpha}\ln(B)\right)^{-k} \gamma\left(k, \frac{\alpha}{1-\alpha}\ln(B)\right)$ upper bounds $\widehat{T}_k$, where $\gamma(s,x) := \int_0^x t^{s-1}e^{-t}\mathrm{d}t$ is the lower incomplete gamma function.*

For any given $\widehat{k} \in \{1, \cdots, m\}$, we define $\Delta_{\widehat{k}} := \left\{1 \le k \le m : \widehat{\underline{T}}_k > \widehat{U}_{\widehat{k}}\right\}$. According to Theorems 2 and 3, $\widehat{T}_k$ is lower bounded by $\widehat{\underline{T}}_k$ and upper bounded by $\widehat{U}_k$. Consequently, any $k \in \Delta_{\widehat{k}}$ cannot achieve the smallest $\widehat{T}_k$. Furthermore, the following Theorem 4 describes the monotonicity property of $\widehat{\underline{T}}_k$ and suggests a convenient way to determine $\Delta_{\widehat{k}}$ without computing $\widehat{\underline{T}}_k$ for every $k \in \{1, \ldots, m\}$.

THEOREM 4. *$\widehat{\underline{T}}_k$ is monotonically decreasing for $k \in \{1, \ldots, \underline{k}^*\}$ and monotonically increasing for $k \in \{\underline{k}^*, \ldots, m\}$, where $\underline{k}^* = \max\left\{\arg\min_{1\le k\le m} \widehat{\underline{T}}_k\right\}$.*

Combining the lower and upper bounds of $\widehat{T}_k$ with its monotonicity, we can narrow down the range of k^* to a small number of options as shown in Theorem 5. In Proposition 6, we further provide an explicit expression for $\underline{k}^*$, enabling us to decide the value of k^* with even less computation.

THEOREM 5. *Let k'_l be the largest $k < \underline{k}^*$ and k'_r be the smallest $k > \underline{k}^*$, respectively, such that $\widehat{\underline{T}}_k \ge \widehat{U}_{\underline{k}^*}$. It then follows that k^* belongs to the set $\{k'_l + 1, \ldots, k'_r - 1\}$.*

In view of the monotonicity of $\widehat{\underline{T}}_k$, both k'_l and k'_r can be efficiently computed.

COROLLARY 1. *When $\widehat{\underline{T}}_{\underline{k}^*+1} \ge \widehat{U}_{\underline{k}^*}$ and $\widehat{\underline{T}}_{\underline{k}^*-1} \ge \widehat{U}_{\underline{k}^*}$, it holds that $k^* = \underline{k}^*$.*

Corollary 1 provides a sufficient condition for $k^* = \underline{k}^*$, which is a special case of Theorem 5 with $k'_l = \underline{k}^* - 1$ and $k'_r = \underline{k}^* + 1$. In this case, there is no need to compute any $\widehat{T}_k$. It is worth mentioning that the lower incomplete gamma function can be computed very efficiently (in microseconds) by many existing numerical libraries (e.g., Boost C++ library), rendering the evaluation of $\widehat{U}_k$ significantly faster than that of $\widehat{T}_k$.

COROLLARY 2. *Let k_l'' be the largest $k < \underline{k}^*$ and k_r'' be the smallest $k > \underline{k}^*$, respectively, such that $\widehat{\underline{T}}_k \geq \widehat{T}_{\underline{k}^*}$, it follows that $k_l'' \geq k_l'$ and $k_r'' \leq k_r'$. Moreover, $k^* \in \{k_l''+1, \cdots, k_r''-1\}$.*

Corollary 2 can be proved straightforwardly. According to Theorem 3, we have $\widehat{T}_{\underline{k}^*} \leq \widehat{U}_{\underline{k}^*}$. In view of the monotonicity of the $\widehat{\underline{T}}_k$ and the definitions of k_l'', k_r'', k_l', and k_r', we have $k_l'' \geq k_l'$ and $k_r'' \leq k_r'$. Therefore, Corollary 2 further narrows the range of k^*. For $k \leq k_l''$, it follows that $\widehat{T}_k \geq \widehat{\underline{T}}_k \geq \widehat{\underline{T}}_{k_l''} \geq \widehat{T}_{\underline{k}^*}$, which leads to $k^* \geq k_l''+1$. For $k \geq k_r''$, we also have the same inequality, proving the other direction, i.e., $k^* \leq k_r''-1$. Therefore, k^* belongs to the set $\{k_l''+1, \cdots, k_r''-1\}$. Let

$$k_1 = \sqrt{(\alpha\beta\ln\sqrt{2})(\alpha\beta\ln\sqrt{2}+2\alpha+2\omega-2)} + \alpha\beta\ln\sqrt{2} + \alpha - 1. \tag{10}$$

PROPOSITION 6. *If $k_1 > 1$, then $\underline{k}^* \in \arg\min_{k\in\{\lfloor k_1\rfloor, \lceil k_1\rceil\}} \widehat{\underline{T}}_k$.*

Proposition 6 computes $\underline{k}^*$ by evaluating the value of k_1 and suggests that when n is sufficiently large, $\underline{k}^*$ tends to be a constant.

The procedure for efficiently determining the value of k^* when n is large is outlined as follows.

(1) Compute k_1 by Equation (10). If $k_1 \leq 1$, then $\underline{k}^* = 1$, otherwise, pick $\underline{k}^* \in \arg\min_{k\in\{\lfloor k_1\rfloor, \lceil k_1\rceil\}} \widehat{T}_k$.
(2) Check if $\widehat{\underline{T}}_{\underline{k}^*+1} \geq \widehat{U}_{\underline{k}^*}$ and $\widehat{\underline{T}}_{\underline{k}^*-1} \geq \widehat{U}_{\underline{k}^*}$ hold. If so, then set $k^* = \underline{k}^*$. Otherwise, compute k_l'', k_r'' and pick $k^* \in \arg\min_{k\in\{k_l''+1, \cdots, k_r''-1\}} \widehat{T}_k$.

6. Numerical Study

Our extensive numerical study is performed using RouteOpt, an exact solver for the CVRP and VRPTWachieving state-of-the-art performance. Some implementation details of RouteOpt are included in Section EC.5 of the e-companion. While the default configuration of RouteOpt, which activates all optimization techniques, is most suitable for practical use, the simultaneous deployment of cutting planes, arc elimination, and enumeration at all BBNs can significantly interfere with the branching process. To clearly demonstrate the effectiveness of our methodology, we also evaluate a second configuration where cutting planes and arc elimination are applied only at the root node, with enumeration completely disabled. Restricting cutting planes to the root node during branching studies is standard practice (e.g., Alvarez et al. 2017, Khalil et al. 2016, Yang et al. 2022), and methods such as arc elimination and enumeration are not typically used in standard B&C algorithms. For convenience, we denote experiments conducted with the default (first) and second configurations of RouteOpt as Settings I and II, respectively (see Table 2). In Sections 6.1 and 6.2, we present two case studies on the CVRP and VRPTW, respectively, that demonstrate the effectiveness of the 2LBB derived from the proposed framework. Finally, Section 6.3 is dedicated to our attempt to solve open CVRP instances. Preliminary experiments performed to determine

and justify the design of some key components of our learning-to-branch framework are presented in Section EC.7 of the e-companion.

For the instances tested in this study, we adopt a systematic naming convention to categorize and clarify the characteristics of each batch. The naming format is *problem-size-replication-type*, where *problem* specifies the type of problem (e.g., VRPTW or CVRP), *size* denotes the number of customers, with *V* indicating a varied size within the batch, *replication* represents the number of instances in the batch, and *type* identifies the category of instances, including *0, 1, X*, and *H2*. Specifically, *0* and *1* correspond to newly generated instances, where *0* is used for training data generation and *1* for performance evaluation. The *X* category refers to the X-series benchmarks for CVRP (Uchoa et al. 2017), while *H2* denotes type 2 instances for VRPTW (Homberger and Gehring 2005). We used instance batches CVRP-120-90-1, CVRP-150-90-1, CVRP-180-90-1, and CVRP-V-42-X for CVRP, and VRPTW-160-60-1, VRPTW-180-60-1, VRPTW-200-60-1, and VRPTW-200-22-H2 for VRPTW to demonstrate the improvement achieved by the learned branching strategy when integrated into RouteOpt. Further details on the generation, attributes of these instances, and the rationale behind their selection can be found in the Section EC.4 of the e-companion.

In our experiments, two sets of computing resources were used: HiPerGator and Dell workstations. HiPerGator is a supercomputing cluster at the University of Florida. We used the hpg-default partition running Red Hat Enterprise Linux 8.8, where each node is equipped with two AMD EPYC 7702 64-core processors @2.0 GHz and 1003 GB RAM. On Hipergator, the code was compiled by g++ 12.2.0. Our group has access to 1080 CPU cores and 8640 GB RAM, which allows us to run hundreds of jobs simultaneously. Training data generation was efficiently conducted using this computing resource. For experiments necessitating precise time comparisons, we used four identical Dell workstations. Each workstation runs on Ubuntu 20.04 LTS and is equipped with the 12th Gen Intel(R) Core(TM) i9-12900K CPU @3.2 GHz and 128 GB RAM. On the workstations, g++ version 9.4.0 was used to compile the code. For all experiments in this section, Gurobi 10.0.0 (Gurobi Optimization, LLC 2023) was used as the LP and MIP solver inside our RouteOpt, except in Section 6.1.4 where CPLEX 22.1.0 (IBM Corporation 2023) was employed to ensure a fair comparison with VRPSolver. All code was run using a single thread. The source code of RouteOpt, together with all the ML code and CVRP and VRPTW instances, has been made publicly available on GitHub.

Table 2 Summary of the configurations used in the two experimental settings

Setting	Cutting Plane	Arc Elimination	Enumeration
I	all BBNs	all BBNs	activated
II	root node	root node	disabled

6.1. Case Study I: the CVRP

In this case study, we use the CVRP to demonstrate the performance of our framework. We integrate the trained ML module into RouteOpt and evaluate the performance improvement by solving instances in batches CVRP-120-90-1, CVRP-150-90-1, CVRP-180-90-1, and CVRP-V-42-X, aiming to highlight the superiority of 2LBB over the benchmark 3PB and emphasize its generalization capability. Based on preliminary experiments in Section EC.7 of the e-companion, we set $\hat{\theta}_{pf} = 100$, $\hat{\theta} = 10$, and $\tilde{\theta} = 3$ for 2LBB, with the only difference for 3PB being $\tilde{\theta} = 2$. For the dynamic version, we assign $\eta = 0.72$, while α is set to 0.60 for 3PB-dy and 0.37 for 2LBB-dy, respectively. For the parameters of the second stage selection, we set $\underline{d} = 2$ and $\delta = 1$. To assess the performance difference due solely to branching, experiments on batches CVRP-120-90-1, CVRP-150-90-1, and CVRP-180-90-1 are conducted under experimental setting II, using the optimal value u^* or $u^0 := (1+0.1\%)u^*$ as the upper bound. To further demonstrate its generalization across different settings and compatibility with other techniques, CVRP-V-42-X is tested under setting I. Details on training the ML models are included in Section EC.9.1 of the e-companion.

6.1.1. Superiority of the 2LBB. We compare the performance of RouteOpt using the 2LBB and the benchmark 3PB for instance batches CVRP-150-90-1. Additionally, we include two corresponding dynamic adjustments, i.e., 3PB-dy and 2LBB-dy, which leverage the BKF to dynamically determine the number of time-consuming testing conducted. We report the time for solving the root node (Root/s), the root gap (Root Gap/%), LP testing time (LP/s), heuristic testing time (Heuristic/s), the overall solution time (CPU/s), and the number of nodes explored (#Nodes). Tables 3 and 4 present the geometric means of these metrics when two upper bounds (UBs) are used, i.e., u^* and u^0, respectively. Detailed results for each instance are included in Sections EC.10.1 - 10.2 of the e-companion.

We observe that when u^* is used as the UB, employing the learning models is beneficial, regardless of whether BKF is applied. Specifically, when BKF is not used, the time reduction (2LBB versus 3PB) can reach 36%, while with BKF, the time reduction (2LBB-dy versus 3PB-dy) is approximately 27%. This lower ratio suggests that BKF may play a more significant role in the 3PB strategy. Indeed, in lower-level branching decisions, where limited historical information is available, the performance of the initial screening—on which 3PB heavily relies—is significantly deteriorated. However, when incorporating *M1*, 2LBB is less affected by the lack of historical information. Consequently, adopting a strategy of testing more at lower levels and less at higher levels can be expected to enhance the performance of 3PB more effectively.

From the perspective of the node metric, we observe less node reduction (25%) than time reduction (36%) when comparing 2LBB with 3PB. The additional time reduction arises from the second

stage, where fewer heuristic testing are required while maintaining a similar tree size, as shown in Section EC.7.3 of the e-companion. This time saving also causes 2LBB to require less time on average to solve a BBN (excluding LP testing), i.e., a smaller c, which further contributes to different behavior observed for BKF. Specifically, 3PB-dy processes 18% fewer nodes than 3PB, while 2LBB-dy processes 22% more nodes than 2LBB. Note that in BKF, a smaller c often leads to fewer testing being performed and, thus, a larger tree size. Despite these differences in tree size, both 3PB and 2LBB benefit from BKF, achieving time reductions of 23% and 13%, respectively. Overall, the 2LBB enhanced with BKF achieves the best performance, with a 44% time reduction compared to the baseline 3PB.

Table 3 Comparison of 3PB, 2LBB, 3PB-dy, and 2LBB-dy when u^* serves as the UB

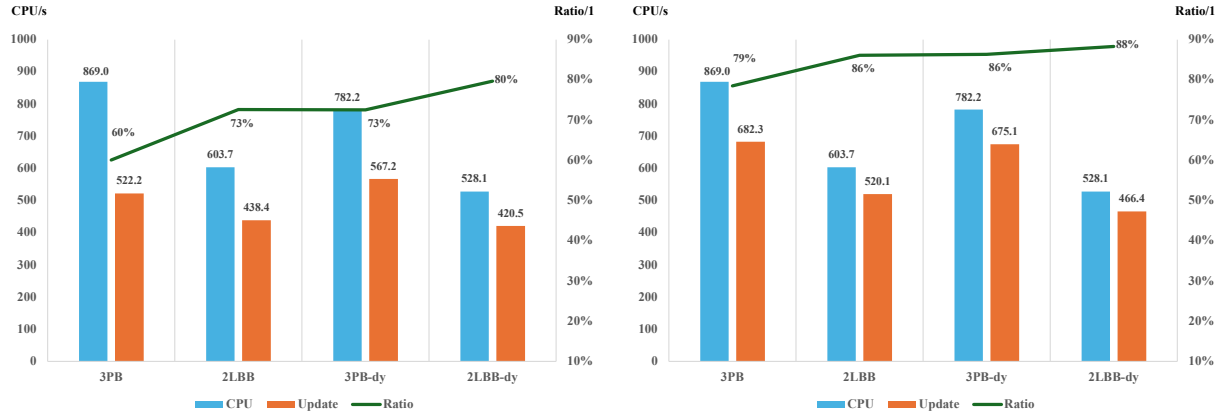
Version	Root/s	Root Gap/%	LP/s	Heuristic/s	CPU/s	#Nodes
3PB	24.5	0.13	218.1	139.8	570.2	297.9
2LBB			166.2	29.2	362.5	222.1
3PB-dy			152.0	108.9	436.8	245.5
2LBB-dy			102.9	21.8	317.2	270.1

For the case where u^0 is provided as the UB, similar to the conclusion drawn from Table 3, 2LBB still outperforms 3PB regardless of whether BKF is applied. However, the time reduction of 2LBB versus 3PB decreases to 31%. This reduction also applies to the versions using BKF. Further investigation reveals that although 2LBB finds better solutions or identifies the optimal solution more quickly than 3PB, the ratio of the time spent finding the solution to the overall solution time is lower for 3PB. Specifically, as shown in Figure 5, 3PB typically finds a better UB after spending 60% of the overall time, whereas 2LBB requires 73%. When a better solution is found, a large number of nodes can be pruned, accelerating the remaining solution process. We believe this explains why the time reduction diminishes. Another interesting observation is that 2LBB-dy explores 7% fewer nodes than 2LBB, which is the opposite of what we observed in Table 3. This is because, when given a larger UB, the tree size is larger, making BKF tend to run more LP testing to achieve the best possible trade-off between testing efforts and tree size.

Table 4 Comparison of 3PB, 2LBB, 3PB-dy, and 2LBB-dy when u^0 serves as the UB

Version	Root/s	Root Gap/%	LP/s	Heuristic/s	CPU/s	#Nodes
3PB	24.2	0.24	356.3	238.1	869.0	463.8
2LBB			317.8	71.9	603.7	398.3
3PB-dy			390.2	181.3	782.2	347.6
2LBB-dy			269.2	57.6	528.1	372.1

Figure 5 Comparison on time (nominal values & ratios) for updating the UB and finding the optimal solution

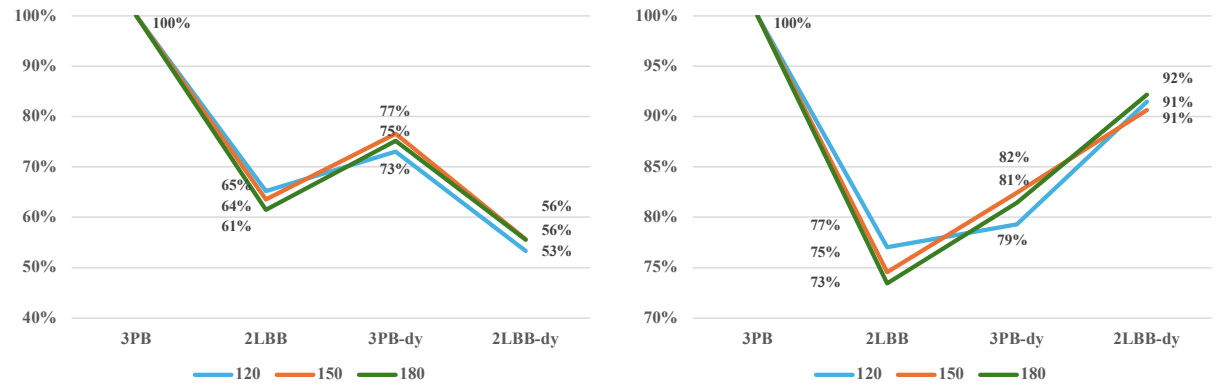


(a) Time (nominal values & ratios) to update UB

(b) Time (nominal values & ratios) to find the optimal solution

6.1.2. Generalization capability of the 2LBB. Generalization refers to the capability of a trained ML model to effectively adapt to new, previously unseen data drawn from the same distribution as the one used during model training. Generalization affects the practicability of a trained model. We investigate the generalization of our 2LBB by evaluating its performance in solving instances of different sizes. More specifically, we collect the training data by solving instance batch CVRP-150-270-0 and apply the resulting 2LBB to solve batches CVRP-120-90-1, CVRP-150-90-1, and CVRP-180-90-1. As illustrated by the ratios of tree size and CPU time with respect to the 3PB in Figure 6, the effectiveness of the 2LBB approach is consistent across instances with 120, 150, and 180 customers. These results confirm that our methodology generalizes well. Especially, the node ratio decreases as the size increases, indicating that 2LBB is potentially more powerful in solving larger instances.

Figure 6 Generalization capability of 2LBB across different instance sizes



(a) Ratio of CPU time to 3PB (CVRP)

(b) Ratio of tree size to 3PB (CVRP)

6.1.3. Performance on established CVRP benchmark instances. In this section, we report the computational results for the established CVRP benchmark instances, i.e., the batch CVRP-V-42-X. Most of these instances remain very challenging to prove optimality within several days, even with the best-known value as the initial UB. Some instances may still remain unsolved. Only reporting the results of instances that can be solved within a few hours is not enough to draw rigorous conclusions. Therefore, to include as many instances as possible, we decided to solve the more difficult instances (34 out of 42 instances) by setting an initial *cutoff* that is geometrically 0.14% lower than the best-known value, allowing the solver to stop within approximately 2 hours. It is important to note that the specified cutoff might not always represent a valid UB. The solver could terminate without finding any feasible solution. In this case, we conclude the given initial cutoff is a valid lower bound (LB). However, if at least one feasible solution is found, then the output solution is guaranteed to be optimal. For clarity, we consistently use the term *solve* to encompass both outcomes.

The key reason for not comparing the LB within the same time limit, as often done in other papers, is that BKF is not designed to maximize the LB under tight running time constraints but rather focuses on minimizing the total solution time, even though this necessitates spending significantly more time at the early stages of branching. In the dynamic version, fewer nodes are explored if the solver stops prematurely, potentially resulting in a worse LB and leading to the incorrect conclusion that the dynamic version underperforms. However, the dynamic version may significantly outperform the static version in later stages, as the initial intensive efforts lead to fewer nodes needing to be solved, each also requiring less testing. Ultimately, the dynamic version requires less time for the complete solution process. Additionally, it is both inconsistent and challenging to estimate total time reduction by comparing LB improvements.

Table 5 shows that 2LBB-dy still achieves the best performance among all four versions of RouteOpt, specifically achieving a 15% time reduction compared to 2LBB, and extending to a 38% time reduction for 3PB. Additionally, since cutting planes are enabled, which increases the average time excluding LP testing for solving a BBN, the dynamic version opts for more LP testing, which results in a node reduction of 31% and 16% for 3PB-dy versus 3PB and 2LBB-dy versus 2LBB, respectively. What is noteworthy is that 3PB-dy achieves a 28% time reduction compared to 3PB, demonstrating again the potential of the generalized and independent learning mechanism of BKF, which can significantly enhance its impact in learning-free branching strategies including the 3PB.

From the detailed results in Section EC.10.3 of the e-companion, we observed that for instances requiring fewer branchings—specifically, fewer than 64 using 2LBB—the time reduction of 2LBB-dy compared to 2LBB and 3PB is 9% and 32%, respectively. Conversely, for instances with more than 64 branchings, we observe time reductions of 22% and 43%, respectively. This larger time reduction

Table 5 Comparison of 3PB, 2LBB, 3PB-dy, and 2LBB-dy on batch CVRP-V-42-X

Version	Root/s	Root Gap/%	LP/s	Heuristic/s	CPU/s	#Nodes
3PB	366.8	0.11	389.2	329.2	4324.3	100.3
2LBB			321.9	91.2	3184.3	74.9
3PB-dy			410.6	256.4	3120.7	69.3
2LBB-dy			329.0	86.1	2691.5	62.7

for instances requiring extensive branching indicates that when more branchings are needed, a static $\hat{\theta}$ is less effective. Consequently, the application of BKF in solving more difficult instances gives rise to even more pronounced acceleration.

We also noticed that the time reduction brought by BKF decreases for CVRP as the average route length increases. Specifically, when using 8 as the threshold to categorize instances into two groups—short routes (< 8) and long routes (≥ 8)—we observed that for the short group, the time reduction of 2LBB-dy compared to 2LBB is 22%, whereas it significantly decreases to 10% for the long group. A similar observation can be made for 3PB-dy. The reason is that instances with longer routes incur significantly higher pricing time for each node, resulting in a smaller ratio of branching selection time versus overall time. Even though BKF helps greatly reduce computational time for later branching selection, the overall time reduction remains relatively minor. Additionally, the long group instances have smaller BBT sizes than the short group, which further exacerbates this effect.

6.1.4. Comparison with the VRPSolver. The VRPSolver, developed in Pessoa et al. (2020), is recognized as the best-performing tool for solving the CVRP exactly. We compare its performance with the RouteOpt version equipped with 2LBB-dy, which achieves the best results among the four tested versions. For testing, we select the CVRP-V-42-X instance batch. To ensure a fair comparison, we use the same cutoff, the same LP and MIP solver, CPLEX (version 22.1.0 in our experiments; IBM Corporation 2023) with identical parameters. VRPSolver with extension version v0.5.24 and BaPCod version v0.74, both using default parameters, are employed to solve the instances.

From Table 6, we observe that 2LBB-dy processes the root node much faster, largely because of RouteOpt's less aggressive cutting plane addition compared to VRPSolver, which we refer to as the branching-focused strategy, in contrast to VRPSolver's cutting-focused strategy. We adopt the branching-focused strategy for two main reasons. First, in 2LBB-dy, good branching decisions can be made without excessive computational efforts, making it more efficient to focus on branching rather than performing numerous rounds of cutting plane addition that offer minimal improvement to the lower bound. Second, RouteOpt's current heuristics for separating rank-1 cuts are not as effective as VRPSolver's. RouteOpt typically has a larger root gap, approximately 0.03% larger in this case.

Despite such a larger root gap, RouteOpt still reduces the overall solution time by 47% compared to VRPSolver. RouteOpt requires significantly less time at each node for both the pricing and cutting phases thanks to the less aggressive cutting strategy and the 2LBB-dy scheme that efficiently makes good branching decisions. This substantial time reduction per node compensates for the significantly increased number of nodes explored. We emphasize that this acceleration is not simply the result of a direct switch to a branching-focused strategy. Rather, it is the effectiveness brought by 2LBB-dy that ensures each branching decision requires less computational effort without little sacrifice in quality, ultimately leading to the success of our RouteOpt.

Table 6 Comparison of VRPSolver and 2LBB-dy on batch CVRP-V-42-X

Version	Root/s	Root Gap/%	LP/s	Heuristic/s	CPU/s	#Nodes
VRPSolver	1178.8	0.08	336.1	116.4	5773.3	21.8
2LBB-dy	429.1	0.11	316.6	146.2	3055.1	64.0

6.2. Case Study II: the VRPTW

We further demonstrate the effectiveness of 2LBB and BKF by applying them to various VRPTW instance batches: VRPTW-180-60-1, VRPTW-160-60-1, VRPTW-200-60-1, and VRPTW-200-22-H2. The critical parameters used, including those for the dynamic version (α and η), are consistent with those in the CVRP case study, with the exception of VRPTW-200-22-H2. For this instance batch, we observed a decay in the effectiveness of *M1* and selected a larger $\alpha = 0.45$ to address it. Additionally, we chose a smaller $\eta = 0.5$ in response to the decreased testing scores observed as branching progresses. All these instances are tested under setting II, even for VRPTW-200-22-H2, since most of them require very few branchings or are very hard to solve even after hours of effort using setting I. Therefore, no comparison with VRPSolver is made in this case study, as we do not enable RouteOpt to utilize all of its techniques. Readers can refer to Section EC.9.2 of the e-companion for training details.

6.2.1. Superior of the 2LBB. In this section, we compare the performance of four versions of RouteOpt for the instance batch VRPTW-180-60-1 and report the same metrics used in the CVRP case study. Detailed results for each instance are provided in Sections EC.10.5 - 10.6 of the e-companion.

As shown in Table 7, the results are consistent with those from the CVRP case study. Specifically, the time reduction for 2LBB compared to 3PB is 39%, and for 2LBB-dy compared to 3PB-dy, it is 36%. 2LBB-dy exhibits the best performance, achieving a 51% time reduction compared to 3PB. Furthermore, by incorporating BKF, 3PB-dy reduces computational time by 23% compared

Table 7 Comparison of 3PB, 2LBB, 3PB-dy, and 2LBB-dy with u^* serving as the UB

Version	Root/s	Root Gap/%	LP/s	Heuristic/s	CPU/s	#Nodes
3PB	24.0	0.17	175.3	124.0	414.0	176.0
2LBB			141.9	24.1	254.1	138.8
3PB-dy			119.2	92.2	317.0	157.6
2LBB-dy			79.1	17.1	201.5	164.5

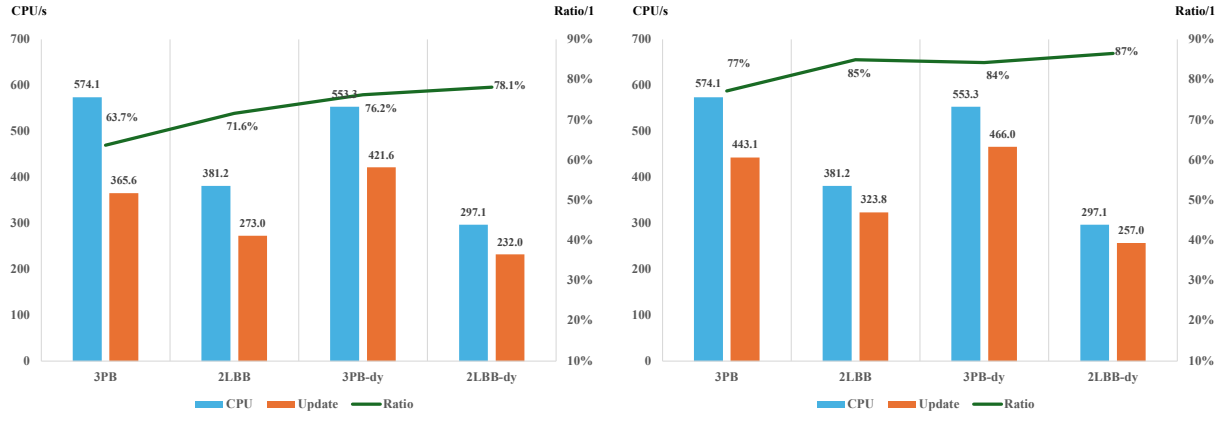
to 3PB. This once again demonstrates the effectiveness of BKF over a static scheme, even in a learning-free branching strategy.

From Table 8, a similar reduction in time reduction is observed. Specifically, the time reduction of 2LBB compared to 3PB diminishes to 34%, and for 2LBB-dy compared to 3PB, the reduction is 48%. As previously explained in the CVRP case study, this diminishing effect can be attributed to the relatively early updating of the UB or the early finding of the optimal solution, as illustrated in Figure 7. Moreover, because the UB is larger, BKF tends to prioritize decreasing the tree size. Consequently, the node ratios for both 3PB-dy versus 3PB and 2LBB-dy versus 2LBB have decreased compared to the results when the optimal value u^* is used as the UB.

Table 8 Comparison of 3PB, 2LBB, 3PB-dy, and 2LBB-dy when u^0 serving as the UB

Version	Root/s	Root Gap/%	LP/s	Heuristic/s	CPU/s	#Nodes
3PB	21.8	0.28	258.0	187.8	574.1	245.9
2LBB			237.3	52.0	381.2	218.1
3PB-dy			269.4	164.9	553.3	220.1
2LBB-dy			158.0	39.8	297.1	219.9

Figure 7 Comparison on time (nominal values & ratios) for updating the UB and finding the optimal solution



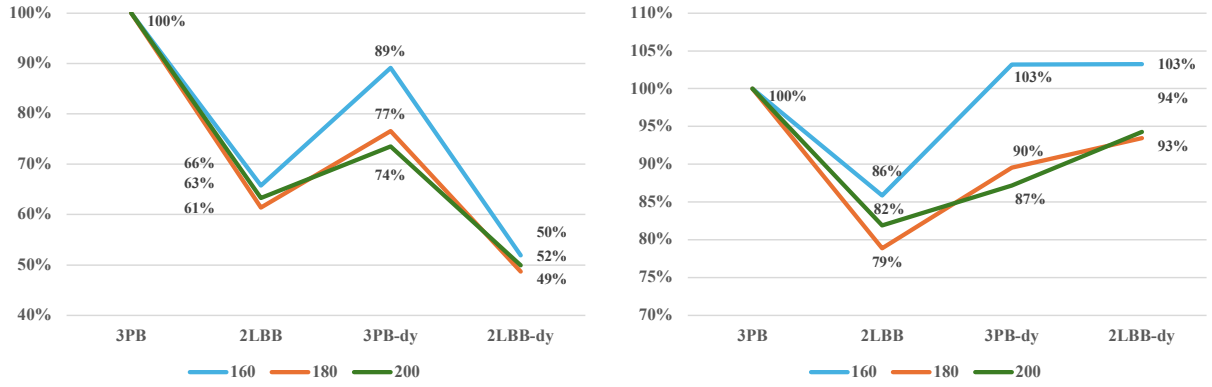
(a) Time (nominal values & ratios) to update UB

(b) Time (nominal values & ratios) to find the optimal solution

6.2.2. Generalization capability of the 2LBB. In this section, we collect training data by solving the instance batch VRPTW-180-600-0 and subsequently apply the trained *M1* and *M2* to solve instance batches VRPTW-160-60-1, VRPTW-180-60-1, and VRPTW-200-60-1. As illustrated

by the ratios of tree size and CPU time with respect to 3PB in Figure 8, the effectiveness of the 2LBB approach is consistent across instances with 160, 180, and 200 customers, achieving approximately a 40% time reduction. This reduction increases to 50% after incorporating BKF. Even without introducing the learning models to RoutOpt, applying just BKF to 3PB results in a 10% to 26% time reduction, with the reduction increasing as the instance size grows. This again indicates that BKF becomes even more effective when dealing with larger instances.

Figure 8 Generalization capability of 2LBB across different instance sizes



(a) Ratio of CPU time to 3PB (VRPTW)

(b) Ratio of tree size to 3PB (VRPTW)

6.2.3. Performance on established VRPTW benchmark instances. We report the computational results for the established VRPTW benchmark instances: the batch VRPTW-200-22-H2. Our preliminary results suggest that these instances tend to favor an aggressive cutting strategy that achieves a very small root gap and thus requires minimal need for branching. This characteristic, unfortunately, is not ideal for demonstrating the effectiveness of our learning-to-branch scheme. To increase the reliance on branching decisions, we continue employing setting II in this set of experiments. Furthermore, we stop adding cuts when the current gap between the LB and the cutoff is within 0.50%. This adjustment necessitates significantly more branching decisions, thereby suitable for demonstrating the differences among the four branching strategies. The cutoff is chosen to be 0.64% (on a geometric mean basis) smaller than the optimal value for 8 out of the 22 instances to ensure the solver stops within a 4-hour limit when using 2LBB-dy. For the remaining 14 relatively easy instances, the optimal value serves as the cutoff.

As shown in Table 9, the instance batch VRPTW-200-22-H2 exhibits significantly different characteristics compared to other instances. Specifically, for the 3PB version, where each branching decision performs LP testing for ten candidates and heuristic testing for two, the time per LP testing calculates to approximately $\frac{253.4}{10b} \approx \frac{25.3}{b}$ seconds and the time per heuristic test is $\frac{1284.8}{2b} \approx \frac{642.4}{b}$ seconds, where b is the number of branchings. This indicates that heuristic testing is over 25 times

as time-consuming as LP testing, suggesting that applying BKF to the heuristic phase could offer substantial time reduction. After incorporating BKF into the heuristic phase as described in Section EC.6.1.2 of the e-companion, the dynamic versions show significant improvements. Comparing 3PB-dy to 3PB, a time reduction of 29% and a reduction in the number of nodes by 32% were achieved. Similarly, when comparing 2LBB-dy with 2LBB, a time reduction of 26% and a reduction in the number of nodes by 44% were achieved. Furthermore, when comparing 2LBB-dy to 3PB, a CPU time reduction of 51% was achieved.

Table 9 Comparison of 3PB, 2LBB, 3PB-dy, and 2LBB-dy on batch VRPTW-200-22-H2

Version	Root/s	Root Gap/%	LP/s	Heuristic/s	CPU/s	#Nodes
3PB	169.5	0.38	253.4	1284.8	2959.2	114.7
2LBB			245.4	324.7	1944.8	122.7
3PB-dy			253.7	743.9	2112.5	78.2
2LBB-dy			147.8	351.8	1442.6	68.6

6.3. Open CVRP Instances

In this section, we investigate the performance of RoutOpt across four versions to tackle open instances for which optimal solutions are unknown. We picked five instances from the CVRPLIB X series: X-n294-k50, X-n313-k71, X-n336-k84, X-n344-k43, and X-n384-k52. These instances seem relatively easier to solve because they possess short average route lengths. While preparing this paper, we became aware that X-n344-k43 was solved optimally on September 14, 2023, by the VRPSolver with the cluster branching strategy in Silva and Uchoa (2022). Since no details about their results were reported, the computational effort they spent to solve it is unclear. We decided to still report the relevant results for completeness. It is worth mentioning that using one thread can take an excessive amount of time (several months) to complete the computation, which necessitates taking advantage of the rich computational resources we can access.

More specifically, we leverage the supercomputing resources on HiPerGator following the two-step procedure elaborated below. In the solution process, enumeration starts to succeed at some BBNs when the gap becomes small, and such BBNs are referred to as *enumerated BBNs*. We write the information of each enumerated BBN into a separate file and prune it from the BBT. This process continues until there are no open BBNs in the BBT and is referred to as the first step. This step is much more time-consuming, taking around a week since no parallelism is implemented. In the second step, each enumerated BBN is solved as a standalone job using one thread and 16 GB of memory, which usually finishes within three hours. HiPerGator enables us to process 500 jobs simultaneously, and the second step typically takes less than a day.

The 3PB is configured to handle such extremely difficult instances with $\hat{\theta}_{pf} = 30$ and $\tilde{\theta} = 3$. For a fair comparison, $\hat{\theta}_{pf} = 100$, $\hat{\theta} = 30$, and $\tilde{\theta} = 3$ are used for 2LBB. We managed to prove optimality for two instances: X-n294-k50 and X-n344-k43. Table 10 lists the time for the root node (Root/s), the lower bound for the root node (LB), and the upper bound as input (UB). In addition to the same time metrics, we add one more column indicating the number of enumerated BBNs (#E-nodes).

Table 10: Comparison of 3PB, 2LBB, 3PB-dy and 2LBB-dy on instances X-n294-k50 and X-n344-k43

Instance	Version	Root/s	LB	UB	LP/h	Heuristic/h	CPU/h	#Nodes	#E-nodes
X-n294-k50	3PB	79	46928	47161	28.3	25.4	170.5	23355	11678
	2LBB				24.0	10.0	135.1	20123	10062
	3PB-dy				9.0	15.7	111.6	16353	8176
	2LBB-dy				9.4	8.3	121.6	21297	10649
X-n344-k43	3PB	172	41922	42050	1.5	2.6	13.7	891	444
	2LBB				1.2	0.6	8.8	731	363
	3PB-dy				1.0	1.5	9.6	707	349
	2LBB-dy				0.7	0.5	7.8	687	342

We observe that the 2LBB consistently outperforms the 3PB for both instances. Specifically, for X-n294-k50, the time reduction is 21%, and for X-n344-k43, the time reduction is 36%. As expected, both dynamic versions achieve better performance than their static counterparts, especially in the comparison of 3PB-dy and 3PB for instance X-n294-k50. While 3PB devotes much more time to branching selection, the number of nodes is 1.4 times that of 3PB-dy, showing the effectiveness of the dynamic version. It is worth noting that for the instance X-n294-k50, 3PB-dy achieves better performance, a time reduction of 8%, than 2LBB-dy. Considering that the difference in time spent on LP testing is minimal, the remaining factor could be inaccuracies in the second stage selection procedure. We remark that in this experiment, the selection procedure uses the same $\underline{d}$ and δ throughout the paper, which may not be an ideal choice for instances with more than 20,000 BBNs. To elaborate, a large $\underline{d}$ and a small δ may cause the second stage to skip necessary testing, leading to suboptimal branching decisions. This negative impact can be amplified for instances with a large BBT size. It also suggests that a dynamic version adjusting this parameter should be beneficial.

7. Concluding Remarks

We propose the first effective learning framework that significantly enhances the SOTA 3PB strategy for exact BPC algorithms, along with a novel theoretical model that establishes the foundations for balancing branching decision time and quality. The 2LBB-dy, derived from this framework, demonstrates exceptional performance, achieving approximately 45% and 50% time reduction over the SOTA 3PB for CVRP and VRPTW, respectively, and enabling our RouteOpt to significantly outperform the best existing competitor, VRPSolver. Through a comprehensive numerical study, we further demonstrate that the learned strategy generalizes well to large unseen problems with similar structures and thus is especially suitable for scenarios where similar problems need to be solved on a regular basis, such as last-mile delivery in a given region.

2LBB-dy can be readily extended to problems beyond VRP, such as job scheduling and network optimization, especially those depending on column generation. While the specific features used to train the

ML models may differ across problem domains, a key insight from this work is the importance of focusing on features that capture the problem's intrinsic structure, such as the distribution of node locations and demands in the CVRP, rather than solely relying on the nominal mathematical formulation. That said, the effectiveness of 2LBB depends on the predictive accuracy of *M1* and *M2*. In more complex problems than CVRP or VRPTW, such as inventory routing problems (Desaulniers et al. 2016, Engineer et al. 2012), healthcare planning problems (Qiu et al. 2022, Trautsamwieser and Hirsch 2014), and crew rostering problems (Breugem et al. 2022, Quesnel et al. 2020), hierarchical branching is often employed, involving multiple types of branching variables. Using a single model to rank candidates across all variable types may reduce prediction accuracy and degrade 2LBB-dy's performance. A more practical solution is to train separate models for each variable type, rank candidates within their respective groups, and then use an additional stage to select the overall best candidate for branching.

Another prospective research direction concerns adapting the current method to other branching variable types or multiple variables at a time (Yang et al. 2021). For example, the recently proposed clustering branching (Silva and Uchoa 2022) has helped to solve over 20 open instances, suggesting its potential to tackle instances with certain structures. Nevertheless, the clustering branching gives rise to an exponentially large number of branching candidates (possible clusters), posing an evident challenge in the learning process. Moreover, incorporating new features characterizing such cluster structures is anticipated to be non-trivial. Finally, despite the wide applicability of our theoretical model, some branching variable types might deviate far from Assumption (g), suggesting a need for alternative assumptions for such cases. Most ideas proposed in deriving the BKF are expected to still apply therein.

Acknowledgments

This work is partially supported by the National Science Foundation [Grant CMMI-2309667] and Alibaba Group US [Grant AGR00022274]. We would like to express our gratitude to the area editor, associate editor, and three anonymous referees for providing valuable comments and suggestions.

References

Achterberg T (2007) *Constraint integer programming*. Ph.D. thesis, Technische Universitat Berlin.

Achterberg T, Wunderling R (2013) Mixed integer programming: Analyzing 12 years of progress. *Facets of Combinatorial Optimization: Festschrift for Martin Grötschel* 449–481.

Alvarez AM, Louveaux Q, Wehenkel L (2017) A machine learning-based approximation of strong branching. *INFORMS Journal on Computing* 29(1):185–195.

Applegate D, Bixby R, Chvátal V, Cook W (1995) Finding cuts in the tsp (a preliminary report). Technical report, Citeseer.

Augerat P, Naddef D, Belenguer J, Benavent E, Corberan A, Rinaldi G (1995) Computational results with a branch and cut code for the capacitated vehicle routing problem. Technical Report 949-M, Université Joseph Fourier, Grenoble, France.

Baldacci R, Christofides N, Mingozzi A (2008) An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming* 115:351–385.

Baldacci R, Hadjiconstantinou E, Mingozzi A (2004) An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. *Operations Research* 52(5):723–738.

Baldacci R, Mingozzi A, Roberti R (2011) New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research* 59(5):1269–1283.

Bektaş T, Erdoğan G, Røpke S (2011) Formulations and branch-and-cut algorithms for the generalized vehicle routing problem. *Transportation Science* 45(3):299–316.

Bénichou M, Gauthier JM, Girodet P, Hentges G, Ribière G, Vincent O (1971) Experiments in mixed-integer linear programming. *Mathematical Programming* 1:76–94.

Breugem T, Dollevoet T, Huisman D (2022) Is equality always desirable? Analyzing the trade-off between fairness and attractiveness in crew rostering. *Management Science* 68(4):2619–2641.

Cappart Q, Chételat D, Khalil EB, Lodi A, Morris C, Veličković P (2023) Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research* 24(130):1–61.

Contardo C, Martinelli R (2014) A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. *Discrete Optimization* 12:129–146.

Costa L, Contardo C, Desaulniers G (2019) Exact branch-price-and-cut algorithms for vehicle routing. *Transportation Science* 53(4):946–985.

d O Costa PR, Rhuggenaath J, Zhang Y, Akcay A (2020) Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. *Asian conference on machine learning*, 465–480 (PMLR).

Dantzig GB, Ramser JH (1959) The truck dispatching problem. *Management Science* 6(1):80–91.

Desaulniers G, Rakke JG, Coelho LC (2016) A branch-price-and-cut algorithm for the inventory-routing problem. *Transportation Science* 50(3):1060–1076.

Desrosiers J, Soumis F, Desrochers M (1984) Routing with time windows by column generation. *Networks* 14(4):545–565.

Engineer FG, Furman KC, Nemhauser GL, Savelsbergh MW, Song JH (2012) A branch-price-and-cut algorithm for single-product maritime inventory routing. *Operations Research* 60(1):106–122.

Fukasawa R, Longo H, Lysgaard J, Aragão MPd, Reis M, Uchoa E, Werneck RF (2006) Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming* 106:491–511.

Gasse M, Chételat D, Ferroni N, Charlin L, Lodi A (2019) Exact combinatorial optimization with graph convolutional neural networks. *Advances in neural information processing systems* 32.

Gurobi Optimization, LLC (2023) Gurobi Optimizer Reference Manual. URL https://www.gurobi.com/documentation/10.0/refman/index.html.

Heching A, Hooker JN, Kimura R (2019) A logic-based benders approach to home healthcare delivery. *Transportation Science* 53(2):510–522.

Homberger J, Gehring H (2005) A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. *European Journal of Operational Research* 162(1):220–238.

IBM Corporation (2023) IBM ILOG CPLEX Optimization Studio CPLEX User's Manual. URL https://www.ibm.com/docs/en/icos/22.1.0?topic=manuals-cplex-users-manual.

Irnich S, Desaulniers G, Desrosiers J, Hadjar A (2010) Path-reduced costs for eliminating arcs in routing and scheduling. *INFORMS Journal on Computing* 22(2):297–313.

Jepsen M, Petersen B, Spoorendonk S, Pisinger D (2008) Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research* 56(2):497–511.

Khalil E, Dai H, Zhang Y, Dilkina B, Song L (2017) Learning combinatorial optimization algorithms over graphs. *Advances in Neural Information Processing Systems* 30.

Khalil E, Le Bodic P, Song L, Nemhauser G, Dilkina B (2016) Learning to branch in mixed integer programming. *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

Laporte G (2009) Fifty years of vehicle routing. *Transportation Science* 43(4):408–416.

Laporte G, Nobert Y (1983) A branch and bound algorithm for the capacitated vehicle routing problem. *Operations-Research-Spektrum* 5:77–85.

Le Bodic P, Nemhauser G (2017) An abstract model for branching and its application to mixed integer programming. *Mathematical Programming* 166(1-2):369–405.

Lima I, Uchoa E, Pecin D, Pessoa A, Poggi M, Vidal T, Subramanian A, Oliveira D, Queiroga E (2014) CVRPLIB: Capacitated vehicle routing problem library. URL http://vrp.galgos.inf.puc-rio.br/index.php/en/updates.

Linderoth JT, Savelsbergh MW (1999) A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing* 11(2):173–187.

Lübbecke ME, Desrosiers J (2005) Selected topics in column generation. *Operations Research* 53(6):1007–1023.

Lysgaard J, Letchford AN, Eglese RW (2004) A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming* 100:423–445.

Mai TL, Navet N (2021) Deep learning to predict the feasibility of priority-based ethernet network configurations. *ACM Transactions on Cyber-Physical Systems (TCPS)* 5(4):1–26.

Morabit M, Desaulniers G, Lodi A (2021) Machine-learning–based column selection for column generation. *Transportation Science* 55:815–831.

Morabit M, Desaulniers G, Lodi A (2022) Machine-learning-based arc selection for constrained shortest path problems in column generation. *INFORMS Journal on Optimization* .

Naddef D, Rinaldi G (2002) Branch-and-cut algorithms for the capacitated vrp. *The vehicle routing problem*, 53–84 (SIAM).

Nazari M, Oroojlooy A, Snyder L, Takác M (2018) Reinforcement learning for solving the vehicle routing problem. *Advances in Neural Information Processing Systems* 31.

Pecin D, Contardo C, Desaulniers G, Uchoa E (2017a) New enhancements for the exact solution of the vehicle routing problem with time windows. *INFORMS Journal on Computing* 29(3):489–502.

Pecin D, Pessoa A, Poggi M, Uchoa E (2017b) Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation* 9:61–100.

Pecin D, Pessoa A, Poggi M, Uchoa E, Santos H (2017c) Limited memory rank-1 cuts for vehicle routing problems. *Operations Research Letters* 45(3):206–209.

Pereira P, Courtade E, Aloise D, Quesnel F, Soumis F, Yaakoubi Y (2022) Learning to branch for the crew pairing problem. *Les Cahiers du GERAD ISSN* 711:2440.

Pessoa A, Sadykov R, Uchoa E, Vanderbeck F (2020) A generic exact solver for vehicle routing and related problems. *Mathematical Programming* 183:483–523.

Qiu H, Wang D, Yin Y, Cheng TE, Wang Y (2022) An exact solution method for home health care scheduling with synchronized services. *Naval Research Logistics (NRL)* 69(5):715–733.

Quesnel F, Desaulniers G, Soumis F (2020) Improving air crew rostering by considering crew preferences in the crew pairing problem. *Transportation Science* 54(1):97–114.

Silva JMP, Uchoa E (2022) Branching on clustered vrp instances. *EURO 2022, Espoo - Finland*.

Tahir A, Quesnel F, Desaulniers G, El Hallaoui I, Yaakoubi Y (2021) An improved integral column generation algorithm using machine learning for aircrew pairing. *Transportation Science* 55(6):1411–1429.

Trautsamwieser A, Hirsch P (2014) A branch-price-and-cut approach for solving the medium-term home health care planning problem. *Networks* 64(3):143–159.

Uchoa E, Pecin D, Pessoa A, Poggi M, Vidal T, Subramanian A (2017) New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research* 257(3):845–858.

van der Hagen L, Agatz N, Spliet R, Visser TR, Kok L (2024) Machine learning–based feasibility checks for dynamic time slot management. *Transportation Science* 58(1):94–109.

Vinsensius A, Wang Y, Chew EP, Lee LH (2020) Dynamic incentive mechanism for delivery slot management in e-commerce attended home delivery. *Transportation Science* 54(3):567–587.

Yang Y, Boland N, Dilkina B, Savelsbergh M (2022) Learning generalized strong branching for set covering, set packing, and 0–1 knapsack problems. *European Journal of Operational Research* 301(3):828–840.

Yang Y, Boland N, Savelsbergh M (2021) Multivariable branching: A 0-1 knapsack problem case study. *INFORMS Journal on Computing* 33(4):1354–1367.

Yang Y, Yan C, Cao Y, Roberti R (2023) Planning robust drone-truck delivery routes under road traffic uncertainty. *European Journal of Operational Research* 309(3):1145–1160.

Zhang X, Chen L, Gendreau M, Langevin A (2022) Learning-based branch-and-price algorithms for the vehicle routing problem with time windows and two-dimensional loading constraints. *INFORMS Journal on Computing* 34(3):1419–1436.

Zheng YJ (2018) Emergency train scheduling on chinese high-speed railways. *Transportation Science* 52(5):1077–1091.

Online Supplement

EC.1. Proofs of Section 4.4

EC.1.1. Proof of Proposition 1

Without loss of generality, we assume $D_i^l \le D_i^r$ and $D_j^l \le D_j^r$. We first establish that Condition (3) implies $\frac{S_i^r}{\widehat{s}_i^r} \le \frac{S_j^r}{\widehat{s}_j^r} \cdot \delta^{\phi_1}$. We consider the following cases.

Case 1: If $D_i^r = -1 \le D_j^r$ and $\phi_1 = 0$, then $\frac{S_i^r}{\widehat{s}_i^r} = 1 \le \frac{S_j^r}{\widehat{s}_j^r}$.

Case 2: If $-1 < D_i^r < D_j^r$ and $\phi_1 = 0$, then $\frac{S_i^r}{\widehat{s}_i^r} = \frac{s_i^r}{\widehat{s}_i^r}$ and $\frac{S_j^r}{\widehat{s}_j^r} = \frac{s_j^r}{\widehat{s}_j^r}$. Since $D_i^r < D_j^r$ and considering h as an increasing function, it follows that $\frac{s_i^r}{\widehat{s}_i^r} \le \frac{s_j^r}{\widehat{s}_j^r}$, thus $\frac{S_i^r}{\widehat{s}_i^r} \le \frac{S_j^r}{\widehat{s}_j^r}$.

Case 3: If $-1 < D_i^r = D_j^r \le \underline{d}$ and $\phi_1 = 1$, then both $\frac{S_i^r}{\widehat{s}_i^r}$ and $\frac{S_j^r}{\widehat{s}_j^r}$ equate to $\frac{s_i^r}{\widehat{s}_i^r}$ and $\frac{s_j^r}{\widehat{s}_j^r}$ respectively. The ratio $\frac{s_i^r}{\widehat{s}_i^r} / \frac{s_j^r}{\widehat{s}_j^r} \le \delta$ implies $\frac{S_i^r}{\widehat{s}_i^r} \le \frac{S_j^r}{\widehat{s}_j^r} \cdot \delta$.

Similarly, given $D_i^l \le D_j^l$, it follows that $\frac{S_i^l}{\widehat{s}_i^l} \le \frac{S_j^l}{\widehat{s}_j^l} \cdot \delta^{\phi_2}$. Therefore, combining these inequalities leads to:

$$\widehat{s}_i^{l\times r} = \frac{\widehat{s}_i^l}{S_i^l} \times \frac{\widehat{s}_i^r}{S_i^r} \times S_i^{l\times r} \ge \frac{\widehat{s}_j^l}{S_j^l} \times \frac{\widehat{s}_j^r}{S_j^r} \times \delta^{-(\phi_1+\phi_2)} \times S_i^{l\times r} > \frac{\widehat{s}_j^l}{S_j^l} \times \frac{\widehat{s}_j^r}{S_j^r} \times S_j^{l\times r} = \widehat{s}_j^{l\times r}.$$

EC.1.2. Proof of Proposition 2

We begin by demonstrating that each starting child LP, p_k, undergoes the exact CG at most once. As described in Algorithm 1, whenever exact CG is performed on a starting child LP p_k, we adjust its D_k to -1. This update ensures that Conditions (3) and (4) can no longer be violated for p_k since D_k has reached its minimum possible value. Consequently, p_k will not be selected again in the checking step (*Step 2*) in Algorithm 1, which leads to a maximum of $2|\widetilde{\mathcal{C}}|$ exact CGs. Once no starting child LP is output by the checking step, the algorithm removes one candidate each time from the set until only one remains, guaranteeing a finite termination.

We then establish that the final output candidate is indeed the Winner. Since the remaining candidate satisfies Conditions (2), (3), and (4) compared to any removed candidates and considering the conclusion in Proposition 1, it is evident that this candidate holds a higher exact score than all others. Hence, the final candidate is conclusively the Winner.

EC.1.3. Proof of Proposition 3

Similar to the proof steps in EC.1.1, we assume, without loss of generality, that $a(D_i^l, D_j^l) \times a(D_i^r, D_j^r)$ is no greater than $a(D_i^l, D_j^r) \times a(D_i^r, D_j^l)$. Thus, $\delta_a := \min\left\{a(D_i^l, D_j^l) \cdot a(D_i^r, D_j^r),\, a(D_i^l, D_j^r) \cdot a(D_i^r, D_j^l)\right\} = a(D_i^l, D_j^l) \times a(D_i^r, D_j^r)$.

Given the increasing nature of the function h and by the definitions of ρ_1 and ρ_2, consider the case where $S_i^r \cdot b > S_j^r \cdot \frac{\rho_1}{\rho_2}$ for some positive constant b. It follows that $\widehat{s}_i^r \ge \frac{S_i^r}{\rho_1} > \frac{S_j^r}{b\cdot\rho_2} \ge \frac{\widehat{s}_j^r}{b}$. We

further show that if $S_i^r \cdot b > S_j^r \cdot a(D_i^r, D_j^r)$, then $\widehat{s}_i^r > \frac{\widehat{s}_j^r}{b}$ also holds. This is proved by considering two additional cases.

Case 1: If $D_i^r = D_j^r \leq \underline{d}$ and $a(D_i^r, D_j^r) = \delta$, then for $S_i^r \cdot b > S_j^r \cdot \delta$, we have

$$\widehat{s}_i^r = \frac{S_i^r}{\rho_i^r} > \frac{S_j^r \cdot \delta}{\rho_i^r \cdot b} \geq \frac{S_j^r}{\rho_j^r \cdot b} = \frac{\widehat{s}_j^r}{b}.$$

Case 2: If $D_i^r < D_j^r$ and $a(D_i^r, D_j^r) = 1$, then for $S_i^r \cdot b > S_j^r$, we obtain

$$\widehat{s}_i^r = \frac{S_i^r}{\rho_i^r} > \frac{S_j^r}{\rho_i^r \cdot b} \geq \frac{S_j^r}{\rho_j^r \cdot b} = \frac{\widehat{s}_j^r}{b}.$$

Thus, we have shown that if $S_i^r \cdot b > S_j^r \cdot a(D_i^r, D_j^r)$, then $\widehat{s}_i^r > \widehat{s}_j^r / b$ also holds. Similarly, if $S_i^l \cdot c > S_j^l \cdot a(D_i^l, D_j^l)$, then $\widehat{s}_i^l > \widehat{s}_j^l / c$, where c is also a positive constant.

Considering $S_i^{l\times r} := S_i^l \times S_i^r > S_j^{l\times r} \cdot \delta_a = S_j^l \cdot a(D_i^l, D_j^l) \times S_j^r \cdot a(D_i^r, D_j^r)$, it follows that $\frac{S_i^l}{S_j^l \cdot a(D_i^l, D_j^l)} > \frac{S_j^r \cdot a(D_i^r, D_j^r)}{S_i^r}$. By selecting b such that $\frac{S_i^l}{S_j^l \cdot a(D_i^l, D_j^l)} > b > \frac{S_j^r \cdot a(D_i^r, D_j^r)}{S_i^r}$, it holds that $S_i^l \cdot \frac{1}{b} > S_j^l \cdot a(D_i^l, D_j^l)$ and $S_i^r \cdot b > S_j^r \cdot a(D_i^r, D_j^r)$. Therefore, by the arguments above, we obtain $\widehat{s}_i^l > \widehat{s}_j^l \cdot b$ and $\widehat{s}_i^r > \widehat{s}_j^r \cdot \frac{1}{b}$, leading to the conclusion that $\widehat{s}_i^{l\times r} > \widehat{s}_j^{l\times r}$.

EC.2. Proofs of Section 6

The following two equations will be frequently used in the proofs. Since they are well-known in combinatorics, for ease of presentation, we list them below without proof.

$$\text{For } k, m \in \mathbb{Z}_+ \text{ and } k \leq m, \qquad \sum_{i=k}^{m} \binom{i}{k} = \binom{m+1}{k+1}. \tag{EC.1}$$

$$\text{For } k, m \in \mathbb{Z}_+ \text{ and } 2 \leq k \leq m+1, \qquad \sum_{i=k}^{m} \binom{i-1}{k-2} = \binom{m}{k-1} - 1. \tag{EC.2}$$

EC.2.1. Proof of Theorem 1

Define the function $h(x) := x^{o_1} - x^{o_1-o_2} - 1$ for $x \in [1, +\infty)$. By the Taylor expansion at $x = \varphi_g$, we have $h(x) = h(\varphi_g) + h'(\theta)(x - \varphi_g)$ with θ between φ_g and x. The derivative function $h'(x) = o_1 x^{o_1-1} - (o_1 - o_2)x^{o_1-o_2-1}$. Recall that φ is the root of $h(x)$, i.e., $h(\varphi) = 0$. When $h'(\theta) \neq 0$, it follows that $|\varphi_g - \varphi| = \left|\frac{h(\varphi_g)}{h'(\theta)}\right|$. Actually, we can shown that $h'(\theta) \geq h'(1) \geq 1$ as follows.

We first prove that $h'(x)$ monotonically increases with x for $x \geq 1$. Note that the second derivative is $h''(x) = o_1(o_1-1)x^{o_1-2} - (o_1-o_2)(o_1-o_2-1)x^{o_1-o_2-2}$. Given $o_1 \geq o_2 \geq 1$, we are left with two cases. When $o_1 \leq o_2 + 1$, we have $h''(x) = o_1(o_1-1)x^{o_1-2} + (o_1-o_2)(o_2+1-o_1)x^{o_1-o_2-2} \geq 0$. When $o_1 > o_2 + 1$, we have $(o_1-o_2)(o_1-o_2-1) = o_1^2 - o_1 + o_2^2 + o_2 - 2o_1o_2 \leq o_1^2 - o_1 \Rightarrow h''(x) = o_1(o_1-1)x^{o_1-2} - (o_1-o_2)(o_1-o_2-1)x^{o_1-o_2-2} \geq o_1(o_1-1)x^{o_1-2}(1-x^{-o_2}) \geq 0$. We next prove that $\theta > 1$. Note that $h(1) = -1 < 0$ and $\lim_{x\to\infty} h(x) \to +\infty$. In view of the fact that $h'(x)$ monotonically

increases with x for $x \geq 1$, we conclude that φ must be greater than 1. Moreover, we have $\varphi_g = 2^{\frac{1}{r}} > 1$, which leads to $\theta > 1$ since θ lies between φ and φ_g. Therefore, $h'(\theta) \geq h'(1) = o_2 \geq 1$ and we conclude that $|\varphi_g - \varphi| = \left|\frac{h(\varphi_g)}{h'(\theta)}\right| = \frac{|h(\varphi_g)|}{h'(\theta)} \leq \frac{|h(\varphi_g)|}{h'(1)} = \frac{\left|2^{\frac{1}{v}} - 2^{-v+\frac{1}{v}} - 1\right|}{o_1 v^2} \Rightarrow \left|\frac{\varphi_g - \varphi}{\varphi_g}\right| \leq \frac{2^{-\frac{1}{o_1 v}}\left|2^{\frac{1}{v}} - 2^{-v+\frac{1}{v}} - 1\right|}{o_1 v^2}$.

To show the monotonicity of the error bound with respect to v when $v \geq \frac{1}{3}$, we consider the function $f(v) = 1 - 2^{-v} - 2^{-\frac{1}{v}}$. It suffices to show that $f(v) \geq 0$ and it monotonically decreases with v since the other two terms, $2^{\frac{1}{v}(1-\frac{1}{o_1})}$ and $\frac{1}{o_1 v^2}$, are both non-negative and monotonically decrease with v. First, we consider the derivative $f'(v) = \ln(2)\left(2^{-v} - \frac{2^{-\frac{1}{v}}}{v^2}\right)$. By substituting v with $-\frac{1}{t}$, we notice that showing $2^t t - \frac{2^{\frac{1}{t}}}{t} \leq 0$ for $t \in [-3, -1]$ leads to $f'(v) \leq 0$ for $v \in [\frac{1}{3}, 1]$.

Let $g(t) = 2^t t$, then $2^t t - \frac{2^{\frac{1}{t}}}{t} = g(t) - g(\frac{1}{t})$ which has the same monotonicity as $g(t)$. We consider the derivative $g'(t) = (1 + \ln(2)t)2^t$. It is easy to see that there exists a unique solution $t^* = -\frac{1}{\ln(2)}$ such that $g'(t) < 0$ when $t \in [-3, t^*)$ and $g'(t) > 0$ when $t \in (t^*, -1]$. This implies that $2^t t - \frac{2^{\frac{1}{t}}}{t}$ reached its maximum at $t = -3$ or $t = -1$. Specifically, $g(-3) - g(-\frac{1}{3}) = 2^{-3} \cdot (-3) + 2^{-\frac{1}{3}} \cdot \frac{1}{3} \approx -0.11$ and $g(1) - g(-1) = 0$. It follows that $f'(v) \leq 0$ for $v \in [\frac{1}{3}, 1]$, and thus $f(v)$ monotonically decreases with v. Moreover, we have $f(v) \geq f(1) = 0$ for all $v \in [\frac{1}{3}, 1]$, which completes the proof. ■

EC.2.2. Proof of Proposition 2

It suffices to show that $\mathbb{E}[T_k|m]$ strictly increases with m. For convenience, we define three sequences. $a_i = \frac{\binom{i-1}{k-1}}{\binom{m+1}{k}}$, $b_i = \frac{\binom{i-1}{k-1}}{\binom{m}{k}}$, and $c_i = 2^{\frac{G-F}{\frac{i+n-m}{n} \cdot R}}$. Then, $\frac{\mathbb{E}[T_k|m+1] - \mathbb{E}[T_k|m]}{(c+kt)N(F)} = \sum_{i=k}^{m+1} a_i c_{i-1} - \sum_{i=k}^{m} b_i c_i = a_k c_{k-1} + \sum_{i=k}^{m}(a_{i+1}c_i - b_i c_i)$. Note that $\frac{a_{i+1}}{b_i} = \frac{i(m-k+1)}{(m+1)(i-k+1)} = \frac{1}{1-\frac{k-1}{i}} \frac{m-k+1}{m+1}$, which strictly decreases with i. Moreover, $\frac{a_{m+1}}{b_m} = \frac{m}{m+1} < 1$. Thus, for a given $1 \leq k \leq m$, $\frac{a_{k+1}}{b_k} = k\left(1 - \frac{k}{m+1}\right)$ may or may not be larger than 1. We discuss the following two cases.

(1) When $k\left(1 - \frac{k}{m+1}\right) \leq 1$, we have $a_{i+1} \leq b_i$ for $i \in \{k, \cdots, m\}$. Since c_i strictly decreases with i, we have $\sum_{i=k}^{m}(a_{i+1}c_i - b_i c_i) \geq \sum_{i=k}^{m}(a_{i+1} - b_i)c_k$. Moreover, $a_k c_{k-1} > a_k c_k$ holds, and thus it follows that $\frac{\mathbb{E}[T_k|m+1] - \mathbb{E}[T_k|m]}{(c+kt)N(F)} > a_k c_k + \sum_{i=k}^{m}(a_{i+1} - b_i)c_k = \left(\sum_{i=k}^{m+1} a_i - \sum_{i=k}^{m} b_i\right)c_k$. According to Equation (EC.1), we have $\sum_{i=k}^{m+1} a_i = \sum_{i=k}^{m} b_i = 1 \Rightarrow \mathbb{E}[T_k|m+1] - \mathbb{E}[T_k|m] > 0$.

(2) When $k\left(1 - \frac{k}{m+1}\right) > 1$, there exists $j \in \{k, \cdots, m-1\}$ such that $\frac{a_{j+1}}{b_j} \geq 1$ and $\frac{a_{j+2}}{b_{j+1}} < 1$. Therefore, $\sum_{i=k}^{m}(a_{i+1}c_i - b_i c_i) = \sum_{i=k}^{j}(a_{i+1} - b_i)c_i + \sum_{i=j+1}^{m}(a_{i+1} - b_i)c_i \geq \sum_{i=k}^{j}(a_{i+1} - b_i)c_j + \sum_{i=j+1}^{m}(a_{i+1} - b_i)c_{j+1} > \sum_{i=k}^{j}(a_{i+1} - b_i)c_j + \sum_{i=j+1}^{m}(a_{i+1} - b_i)c_j = \sum_{i=k}^{m}(a_{i+1} - b_i)c_j$. In view of $a_k c_k \geq a_k c_j$, we have $\frac{\mathbb{E}[T_k|m+1] - \mathbb{E}[T_k|m]}{(c+kt)N(F)} > \sum_{i=k}^{m}(a_{i+1} - b_i)c_j + a_k c_j = \left(\sum_{i=k}^{m+1} a_i - \sum_{i=k}^{m} b_i\right)c_j = 0 \Rightarrow \mathbb{E}[T_k|m+1] - \mathbb{E}[T_k|m] > 0$, which completes the proof. ■

EC.2.3. Proof of Lemma 1

By definition, $\mathbb{E}[r|m] = \sum_{k}^{m} \frac{\binom{i-1}{k-1}}{\binom{m}{k}} \frac{i+n-m}{n} R = \sum_{i=k}^{m} \frac{\binom{i-1}{k-1}}{\binom{m}{k}} \left(\frac{m}{n}\right)\left(\frac{i}{m} + \frac{n-m}{m}\right) R$. According to Equation (EC.1), we have $\sum_{i=k}^{m} \frac{\binom{i-1}{k-1}}{\binom{m}{k}} \left(\frac{i}{m} R\right) = \frac{R}{m\binom{m}{k}} \sum_{i=k}^{m} \frac{i!}{(k-1)!(i-k)!} = \frac{kR}{m\binom{m}{k}} \sum_{i=k}^{m} \binom{i}{k} =$

$\frac{m+1}{m}\frac{k}{k+1}R$, and $\left(1-\frac{m}{n}\right)R\sum_{i=k}^{m}\frac{\binom{i-1}{k-1}}{\binom{m}{k}}=\left(1-\frac{m}{n}\right)R$. Using these two equations, it follows that $\sum_{i=k}^{m}\frac{\binom{i-1}{k-1}}{\binom{m}{k}}\left(\frac{i+n-m}{n}R\right)=\frac{m}{n}\sum_{i=k}^{m}\frac{\binom{i-1}{k-1}}{\binom{m}{k}}\left(\frac{i}{m}R\right)+\left(1-\frac{m}{n}\right)R\sum_{i=k}^{m}\frac{\binom{i-1}{k-1}}{\binom{m}{k}}=\frac{m+1}{n}\frac{k}{k+1}R+\left(1-\frac{m}{n}\right)R=\frac{(n+1)k+n-m}{n(k+1)}R$, which completes proof. ■

EC.2.4. Proof of Theorem 2

Recall that $\underline{T}_k=(c+kt)N(F)2^{\frac{G-F}{\mathbb{E}[r|m]}}=(c+kt)N(F)B^{\frac{n(k+1)}{(n+1)k+n-m}}$, where $B:=2^{\frac{G-F}{R}}$. Given $m=\alpha n$, we have $\frac{n(k+1)}{(n+1)k+n-m}=\frac{1}{1-\alpha/(k+1)+\frac{k}{(k+1)n}}$, which increases with n. Moreover, $\frac{1}{1-\alpha/(k+1)+\frac{k}{(k+1)n}}\le\frac{1}{1-\alpha/(k+1)}$. Thus, we conclude that $\lim_{n\to\infty}\underline{T}_k$ exists and $\widehat{\underline{T}}_k=\lim_{n\to\infty}\underline{T}_k=(c+kt)N(F)B^{\frac{1}{1-\alpha/(k+1)}}$.

We next prove that $\widehat{T}_k$ exists and $\widehat{T}_k=(c+kt)kN(F)\int_0^1 x^{k-1}B^{\frac{1}{\alpha x+1-\alpha}}\,\mathrm{d}x$. We start by showing

$$\lim_{m\to\infty}\sum_{i=k}^{\lfloor\sqrt{m}\rfloor}\frac{\binom{i-1}{k-1}}{\binom{m}{k}}B^{\frac{1}{1-\alpha+\alpha i/m}}=0. \tag{EC.3}$$

In view of $0\le\sum_{i=k}^{\lfloor\sqrt{m}\rfloor}\frac{\binom{i-1}{k-1}}{\binom{m}{k}}B^{\frac{1}{1-\alpha+\alpha i/m}}\le\sqrt{m}\frac{\binom{\lfloor\sqrt{m}\rfloor-1}{k-1}}{\binom{m}{k}}B^{\frac{1}{1-\alpha}}$ and $\lim_{m\to\infty}\sqrt{m}\frac{\binom{\lfloor\sqrt{m}\rfloor-1}{k-1}}{\binom{m}{k}}B^{\frac{1}{1-\alpha}}=0$, Equation (EC.3) directly follows from the sandwich theorem. Then we show Equation (EC.4) holds.

$$\lim_{m\to\infty}\sum_{i=\lfloor\sqrt{m}\rfloor+1}^{m}\frac{\binom{i-1}{k-1}}{\binom{m}{k}}B^{\frac{1}{1-\alpha+\alpha i/m}}=k\int_0^1 x^{k-1}B^{\frac{1}{\alpha x+1-\alpha}}\,\mathrm{d}x. \tag{EC.4}$$

For convenience, let $h_i=\frac{i}{m}$ and $a_p^i=\frac{i}{m}-\frac{i-p}{m-p}$. For a given set $S^i=\{a_1^i,a_2^i,\cdots,a_{k-1}^i\}$, define the combinatorial sum $c(S^i,q):=\sum_{1\le j_1<j_2<\cdots<j_q\le k-1}(a_{j_1}^i\cdot a_{j_2}^i\cdots a_{j_q}^i)$. Then the following binomial ratio can be represented as

$$\begin{aligned}\frac{\binom{i-1}{k-1}}{\binom{m}{k}}=&\frac{k}{m}\frac{(i-1)(i-2)\cdots(i-k+1)}{(m-1)(m-2)\cdots(m-k+1)}=\frac{k}{m}\left(h_i-a_1^i\right)\left(h_i-a_2^i\right)\cdots\left(h_i-a_{k-1}^i\right)\\=&\frac{k}{m}\left[h_i^{k-1}+(-1)^1c(S^i,1)h_i^{k-2}+\cdots+(-1)^{k-2}c(S^i,k-2)h+(-1)^{k-1}c(S^i,k-1)\right].\end{aligned}$$

We further rewrite the following sum into two terms

$$\sum_{i=\lfloor\sqrt{m}\rfloor+1}^{m}\frac{\binom{i-1}{k-1}}{\binom{m}{k}}B^{\frac{1}{1-\alpha+\alpha h_i}}=k\sum_{i=\lfloor\sqrt{m}\rfloor+1}^{m}h_i^{k-1}B^{\frac{1}{1-\alpha+\alpha h_i}}\frac{1}{m}+R(m), \tag{EC.5}$$

where $R(m)=\frac{k}{m}\sum_{i=\lfloor\sqrt{m}\rfloor+1}^{m}\left[(-1)^1c(S^i,1)\left(\frac{i}{m}\right)^{k-2}+\cdots+(-1)^{k-1}c(S^i,k-1)\right]B^{\frac{1}{1-\alpha+\alpha h_i}}$ is the aggregation of the remaining terms. In view of the integrability (due to continuity and boundness) of $f(x)=x^{k-1}B^{\frac{1}{\alpha x+1-\alpha}}$ within [0,1] for a given positive integer k and the properties of Riemann integration, we have $\lim_{m\to\infty}\sum_{i=1}^{m}\left(h_i^{k-1}B^{\frac{1}{1-\alpha+\alpha h_i}}\right)\frac{1}{m}=\int_0^1 x^{k-1}B^{\frac{1}{\alpha x+1-\alpha}}\,\mathrm{d}x$. Furthermore, $0\le\sum_{i=1}^{\lfloor\sqrt{m}\rfloor}\left(h_i^{k-1}B^{\frac{1}{1-\alpha+\alpha h_i}}\right)\frac{1}{m}\le\sum_{i=1}^{\lfloor\sqrt{m}\rfloor}\left[\left(\frac{\lfloor\sqrt{m}\rfloor}{m}\right)^{k-1}B^{\frac{1}{1-\alpha}}\right]\frac{1}{m}\le\frac{B^{\frac{1}{1-\alpha}}}{m}\Rightarrow\lim_{m\to\infty}\sum_{i=1}^{\lfloor\sqrt{m}\rfloor}\left(h_i^{k-1}B^{\frac{1}{1-\alpha+\alpha h_i}}\right)\frac{1}{m}=0$. Therefore, it holds that

$$\lim_{m\to\infty}\sum_{i=\lfloor\sqrt{m}\rfloor+1}^{m}h_i^{k-1}B^{\frac{1}{1-\alpha+\alpha h_i}}\frac{1}{m}=\int_0^1 x^{k-1}B^{\frac{1}{\alpha x+1-\alpha}}\,\mathrm{d}x. \tag{EC.6}$$

Then we show that

$$\lim_{m\to\infty} \frac{R(m)}{\sum_{i=\lfloor\sqrt{m}\rfloor+1}^{m} h_i^{k-1} B^{\frac{1}{1-\alpha+\alpha h_i}\frac{k}{m}}} = 0. \tag{EC.7}$$

Observe that $0 \le \frac{c(S^i,b)\left(\frac{i}{m}\right)^{k-b-1}}{\left(\frac{i}{m}\right)^{k-1}} = c(S^i,b)\left(\frac{i}{m}\right)^{-b} \le \binom{k-1}{b}\left(a_{k-1}^i\right)^b\left(\frac{i}{m}\right)^{-b} = \binom{k-1}{b}\left[\frac{(m-i)(k-1)}{i(m-k+1)}\right]^b$ for $b = 1,\cdots,k-1$, and for $i \ge \lfloor\sqrt{m}\rfloor+1$, $\lim_{m\to\infty}\binom{k-1}{b}\left[\frac{(m-i)(k-1)}{i(m-k+1)}\right]^b = 0$. Thus, it follows that

$$\lim_{m\to\infty} \frac{c(S^i,b)\left(\frac{i}{m}\right)^{k-b-1}}{\left(\frac{i}{m}\right)^{k-1}} = 0. \tag{EC.8}$$

Let $\nu = \max_{i\in\{\lfloor\sqrt{m}\rfloor+1,\cdots,m\}}\left\{\frac{c(S^i,b)\left(\frac{i}{m}\right)^{k-b-1}}{\left(\frac{i}{m}\right)^{k-1}}\right\}$, then $0 \le \frac{\sum_{i=\lfloor\sqrt{m}\rfloor+1}^{m} c(S^i,b)\left(\frac{i}{m}\right)^{k-b-1} B^{\frac{1}{1-\alpha+\alpha i/m}\frac{k}{m}}}{\sum_{i=\lfloor\sqrt{m}\rfloor+1}^{m}\left(\frac{i}{m}\right)^{k-1} B^{\frac{1}{1-\alpha+\alpha i/m}\frac{k}{m}}} \le \nu$, because all terms in the numerator and denominator of the ratio are strictly positive. According to Equation (EC.8), for $i \ge \lfloor\sqrt{m}\rfloor+1$, $\lim_{m\to\infty}\nu = 0$ holds. Therefore, for $i \ge \lfloor\sqrt{m}\rfloor+1$, we have $\lim_{m\to\infty}\frac{\sum_{i=\lfloor\sqrt{m}\rfloor+1}^{m} c(S^i,b)\left(\frac{i}{m}\right)^{k-b-1} B^{\frac{1}{1-\alpha+\alpha i/m}\frac{k}{m}}}{\sum_{i=\lfloor\sqrt{m}\rfloor+1}^{m}\left(\frac{i}{m}\right)^{k-1} B^{\frac{1}{1-\alpha+\alpha i/m}\frac{k}{m}}} = 0 \Rightarrow$ Equation (EC.7) holds.

Combining Equations (EC.5), (EC.6), and (EC.7), we prove that Equation (EC.4) holds, which together with Equation (EC.3), leads to $\widehat{T}_k = (c+kt)N(F)\lim_{m\to\infty}\sum_{i=k}^{m}\frac{\binom{i-1}{k-1}}{\binom{m}{k}} B^{\frac{1}{1-\alpha+\alpha i/m}} = (c+kt)kN(F)\int_0^1 x^{k-1}B^{\frac{1}{\alpha x+1-\alpha}}\,\mathrm{d}x$, and completes the proof. ■

EC.2.5. Proof of Theorem 3

Let $\widehat{U}_k = (c+kt)kN(F)B^{\frac{1}{1-\alpha}}\left(\frac{\alpha}{1-\alpha}\ln(B)\right)^{-k}\gamma\left(k, \frac{\alpha}{1-\alpha}\ln(B)\right)$, where $\gamma(s,x) := \int_0^x t^{s-1}e^{-t}\,\mathrm{d}t$ denotes the lower incomplete gamma function. Consider the function $g(x) = -\frac{\alpha}{1-\alpha}x - \frac{1}{\alpha x+1-\alpha} + \frac{1}{1-\alpha}$. Its first derivative is given by $g'(x) = \frac{\alpha}{(\alpha x+1-\alpha)^2} - \frac{\alpha}{1-\alpha}$. Since $g'(x)$ is a decreasing function for $x\in[0,1]$ and that $g'(0) = \frac{\alpha}{(1-\alpha)^2} - \frac{\alpha}{1-\alpha} > 0$ and $g'(1) = \alpha - \frac{\alpha}{1-\alpha} < 0$, there exists a unique $x_0 \in [0,1]$ such that $g(x)$ is increasing for $x\in[0,x_0)$ and decreasing for $x\in(x_0,1]$. Moreover, in view of $g(0) = g(1) = 0$, we deduce that $g(x) \ge 0$ for x in $[0,1]$, which leads to $\frac{1}{\alpha x+1-\alpha} \le -\frac{\alpha}{1-\alpha}x + \frac{1}{1-\alpha}$ for $x\in[0,1]$. Therefore, it holds that

$$\widehat{T}_k = k\int_0^1 x^{k-1}B^{\frac{1}{\alpha x+1-\alpha}}\,\mathrm{d}x \le k\int_0^1 x^{k-1}B^{-\frac{\alpha}{1-\alpha}x+\frac{1}{1-\alpha}}\,\mathrm{d}x. \tag{EC.9}$$

Let $I = \int_0^1 x^{k-1}B^{-\frac{\alpha}{1-\alpha}x+\frac{1}{1-\alpha}}\,\mathrm{d}x$ and consider the substitution $x = \left(\frac{\alpha}{1-\alpha}\ln(B)\right)^{-1}t$. Through this change of variables, I can be expressed as follows.

$$\begin{aligned}
I &= \int_0^1 x^{k-1}B^{-\frac{\alpha}{1-\alpha}x+\frac{1}{1-\alpha}}\,\mathrm{d}x \\
&= \int_0^{\frac{\alpha}{1-\alpha}\ln(B)}\left[\frac{\alpha}{1-\alpha}\ln(B)\right]^{-k+1} t^{k-1}B^{-\frac{t}{\ln(B)}+\frac{1}{1-\alpha}}\,\mathrm{d}\left[\frac{\alpha}{1-\alpha}\ln(B)\right]^{-1}t \\
&= B^{\frac{1}{1-\alpha}}\left[\frac{\alpha}{1-\alpha}\ln(B)\right]^{-k}\int_0^{\frac{\alpha}{1-\alpha}\ln(B)} t^{k-1}e^{-t}\,\mathrm{d}t \\
&= B^{\frac{1}{1-\alpha}}\left[\frac{\alpha}{1-\alpha}\ln(B)\right]^{-k}\gamma\left(k, \frac{\alpha}{1-\alpha}\ln(B)\right) = \frac{\widehat{U}_k}{k}.
\end{aligned} \tag{EC.10}$$

Combining Equations (EC.9) and (EC.10), we have $\widehat{T}_k \le \widehat{U}_k$, which completes the proof. ■

EC.2.6. Proof of Theorem 4

Let $g(k) = N(F)(c+kt)\cdot 2^{\frac{G-F}{R}\frac{n(k+1)}{(n+1)k+n-m}}$. Taking the derivative with respect to k yields $g'(k) = h(k)\cdot \frac{N(F)}{R[(n+1)k+n-m]^2}\cdot 2^{\frac{n(G-F)(k+1)}{R[(n+1)k+n-m]}}$, where $h(k) = -n(G-F)(c+kt)(m+1)\ln(2) + Rt[(n+1)k + n-m]^2$. When written in the quadratic form, $h(k) = \hat{a}k^2 + \hat{b}k + \hat{c}$, we have $\hat{a} = Rt(n+1)^2 > 0$, $\hat{b} = t[\ln(2)(F-G)n(m+1) + 2R(n-m)(n+1)]$, and $\hat{c} = c\ln(2)(F-G)n(m+1) + Rt(m-n)^2$.

$$
\begin{aligned}
&\hat{b}^2 - 4\hat{a}\hat{c}\\
&= t^2\left[2R(-m+n)(n+1) + n(F-G)(m+1)\ln(2)\right]^2 - 4Rt(n+1)^2\left[Rt(m-n)^2 + c\ln(2)n(F-G)(m+1)\right]\\
&\ge 4R^2t^2\left[(-m+n)(n+1) + \frac{\ln(2)}{2R}n(F-G)(m+1)\right]^2 - 4R^2t^2(n+1)^2\left[(m-n)^2 + \frac{\ln(2)}{R}n(F-G)(m+1)\right]\\
&> 4R^2t^2\left[(m-n)^2(n+1)^2 + \frac{\ln(2)}{R}n(F-G)(m+1)(n-m)(n+1)\right]\\
&\qquad\qquad - 4R^2t^2(n+1)^2\left[(m-n)^2 + \frac{\ln(2)}{R}n(F-G)(m+1)\right]\\
&= 4\ln(2)Rt^2n(n+1)(F-G)(m+1)\left[(n-m) - (n+1)\right] = 4\ln(2)Rt^2n(G-F)(m+1)^2(n+1) \ge 0,
\end{aligned}
$$

where the first inequality is due to $c \ge t$ and $F < G$ and the second inequality results from $\left[\frac{\ln(2)}{2R}n(F-G)(m+1)\right]^2 > 0$. Therefore, $h(k) = 0$ has two distinct solutions, k_1 and k_2, as follows:

$$
k_3 = \frac{-\hat{b} - \sqrt{\hat{b}^2 - 4\hat{a}\hat{c}}}{2\hat{a}}, \quad \text{and} \quad k_4 = \frac{-\hat{b} + \sqrt{\hat{b}^2 - 4\hat{a}\hat{c}}}{2\hat{a}}. \tag{EC.11}
$$

In view of that $h(k) > 0$ for $k \in (-\infty, k_3)\cup(k_4,\infty)$, $h(k) < 0$ for $k \in (k_3, k_4)$, and $\frac{N(F)}{R[(n+1)k+n-m]^2}\cdot 2^{\frac{n(G-F)(k+1)}{R[(n+1)k+n-m]}} > 0$, then $g(k)$ is monotonically decreasing in (k_3, k_4) and monotonically increasing in (k_4,∞). Let $\omega = \frac{c}{t} \ge 1$ and $\beta = \frac{G-F}{R}$, we obtain

$$
\lim_{n\to\infty} k_3 = -\sqrt{(\alpha\beta\ln\sqrt{2})(\alpha\beta\ln\sqrt{2} + 2\alpha + 2\omega - 2)} + \alpha\beta\ln\sqrt{2} + \alpha - 1 \le \alpha - 1 \le 0.
$$

Therefore, the two intervals (k_3, k_4) and (k_4,∞) cover all $k > 0$, which completes the proof. ■

EC.2.7. Proof of Theorem 5

According to Theorem 3, it follows that that $\underline{\widehat{T}}_{k'_l} \ge \widehat{U}_{\underline{k}^*} \ge \widehat{T}_{\underline{k}^*}$. Considering the monotonicity of $\underline{\widehat{T}}_k$ for $k \le k'_l < \underline{k}^*$ as established in Theorem 4, it is evident that,

$$
\widehat{T}_k \ge \underline{\widehat{T}}_k \ge \underline{\widehat{T}}_{k'_l} \ge \widehat{U}_{\underline{k}^*} \ge \widehat{T}_{\underline{k}^*}.
$$

From the above, we conclude that $k^* \ge k'_l + 1$. By similar arguments, we derive that $k^* \le k'_r - 1$. Consequently, k^* belongs to the set $\{k'_l + 1, \ldots, k'_r - 1\}$. ■

EC.2.8. Proof of Proposition 3

By Equation (EC.11), we have

$$k_1 = \lim_{n\to\infty} k_4 = \sqrt{(\alpha\beta\ln\sqrt{2})(\alpha\beta\ln\sqrt{2}+2\alpha+2\omega-2)} + \alpha\beta\ln\sqrt{2}+\alpha-1.$$

If $k_1 \geq 1$, based on the monotonicity property from Theorem 4 and the definition of $\underline{k}^*$, we conclude that $\underline{k}^* \in \arg\min_{k\in\{\lfloor k_1\rfloor,\lceil k_1\rceil\}} \widehat{T}_k$, which completes the proof. ■

EC.3. Details of Learning Methodology

This section presents the main elements of our learning method. We discuss our choice of XGBoost as the learning model for our learning-to-rank task, due to its efficiency, ease of use, and proven effectiveness in related applications. Then, we describe the three types of features used—$\mathcal{F}_E$, $\mathcal{F}_I$, and $\mathcal{F}_D$—explaining how they are defined, computed, and selected based on their cost and relevance. We then introduce the structure of the feature vectors for both *M1* and *M2*, and explain our stepwise, importance-based feature selection process. Finally, we describe the data generation procedure for the two-stage learning, emphasizing how we replicate the real decision-making flow of *M1* and *M2* during data collection to ensure consistency between training and deployment.

EC.3.1. Choice of Machine Learning Model

The popular XGBoost package (Chen and Guestrin 2016) offers a highly optimized implementation of a distributed gradient boosting method capable of training learning-to-rank models efficiently. Compared to various advanced deep learning techniques, e.g., graph convolutional neural networks (Gasse et al. 2019, Morabit et al. 2021), it is significantly less demanding in terms of prior experience and time on parameter tuning. Moreover, XGBoost has been successfully applied to learn branching strategies for solving general MIPs (Yang et al. 2022). Therefore, we also employ it to train all the ML models in this paper. It offers three objectives in ranking: rank:pairwise, rank:ndcg, and rank:map. According to our preliminary experiments, rank:pairwise slightly outperforms the others in terms of accuracy and is thus used as our objective in training.

EC.3.2. The Feature Sets

As stated in Section 4.5, we have three categories of features: $\mathcal{F}_E$, $\mathcal{F}_I$, and $\mathcal{F}_D$. Computing $\mathcal{F}_E$ involves solving the starting child LPs without CG to acquire information such as the dual value associated with the newly added branching constraint and the LP value change. Thus, not all candidates in $\mathcal{C}$ will have these features collected in one branching selection. As shown in Section 4.2, LP testing is applied to further validate candidates in $\widetilde{\mathcal{C}}$. In this phase, $\mathcal{F}_E$ can be easily obtained, and thus, collected for candidates in $\widetilde{\mathcal{C}}$. This information will be used by *M2* to predict the

ranking of deviation ratio of the starting child LPs in $\widetilde{\mathcal{C}}_s$. $\mathcal{F}_I$ and $\mathcal{F}_D$ contain cheaply computable features that are independent and dependent, respective, of the history of $\mathcal{F}_E$. For example, for a given candidate edge, $\mathcal{F}_I$ includes information about the associated vertex density, travel cost, and resource consumption (i.e., the demands of the two endpoints). Such information has been shown to be helpful in selecting branching candidates in Naddef and Rinaldi (2002). Note that $\mathcal{F}_I$ also includes features derived from historical information but not from $\mathcal{F}_E$. In contrast, $\mathcal{F}_D$ includes features derived from those in $\mathcal{F}_E$ by mostly taking averages of their historical values.

All feature sets $\mathcal{F}_I$, $\mathcal{F}_D$, and $\mathcal{F}_E$ can be further subdivided based on whether the information comes entirely from one of the starting child LPs. For example, $\mathcal{F}_E$ can be divided into $\mathcal{F}_{E(l)}$ and $\mathcal{F}_{E(r)}$, where $\mathcal{F}_{E(l)}$ is a subset of $\mathcal{F}_E$ that excludes features related only to the right starting child LP. Similarly, $\mathcal{F}_{E(r)}$ is a subset of $\mathcal{F}_E$ that excludes features related only to the left starting child LP. For convenience, given item i, let f_i^I, f_i^D, and f_i^E represent its values of features belonging to the categories $\mathcal{F}_I$, $\mathcal{F}_D$, and $\mathcal{F}_E$, respectively. Additionally, for a given set of items C, we use $f^I(C)$ to denote the set $\{f_i^I : i \in C\}$, and similarly for $f^D(C)$ and $f^E(C)$. This notation also applies to $\mathcal{F}_{I(l)}$, $\mathcal{F}_{I(r)}$, $\mathcal{F}_{D(l)}$, $\mathcal{F}_{D(r)}$, $\mathcal{F}_{E(l)}$, and $\mathcal{F}_{E(r)}$.

For each candidate $i \in \widehat{\mathcal{C}}_p \cup \widehat{\mathcal{C}}_f$, obtained from the initial screening, only f_i^I and f_i^D will be computed for training *M1*. Although f_i^E is not computed, f_i^D can still be calculated as long as i has been selected into the set $\widehat{\mathcal{C}}$ at some previous branching decisions. Then the corresponding f_i^E has been computed there and can be used as historical information, which, by definition, is what is needed for computing f_i^D. For training *M2*, values f_i^I, f_i^D, and f_i^E will be computed for each candidate $i \in \widehat{\mathcal{C}}$ that has been deemed promising by *M1*. Note that for *M1*, each candidate has one feature vector formed by combining f_i^E and f_i^D. In contrast, for *M2*, each starting child LP has its own feature vector. For the left starting child LP l_i of candidate e_i, its feature vector consists of $f_i^{I(l)}$, $f_i^{D(l)}$, and $f_i^{E(l)}$. Similarly, for the right starting child LP r_i, its feature vector includes $f_i^{I(r)}$, $f_i^{D(r)}$, and $f_i^{E(r)}$. The selected and complete sets of features considered in this study are detailed in Table EC.1, with the selected ones highlighted in **bold**. We denote the edge e as a specific branching candidate and $X(e)$ as the set of routes that traverse edge e. The vector $(\hat{x}_r)_{r\in\Omega'\cup\Omega''}$ represents the LP solution to the parent LP.

We applied a feature selection approach similar to that in Yang et al. (2022). Specifically, we began by removing the feature with the lowest importance scores as identified by XGBoost. Following each removal, the model was retrained and evaluated. This process was continued until the model's accuracy just started to drop below 99% of its initial level with all features included.

EC.3.3. Training Data Generation

The design of the training data generation process should take into account how a trained model will be used. Generally speaking, the process should try to ensure that the data input into a trained

Table EC.1 The complete features for a specific branching edge e.

Feature set $\mathcal{F}_I$	Description
Tree depth	tree depth for the current branching selection
Current gap	gap of RMP value and UB for the current branching selection
Clustering coefficient	reciprocal of the average travel cost between two customers in the same cluster
Route length	average number of customers visited per route of RMP solutions
Distance from depot to center	distance between the depot and the center of all customers
Density	proportion of edges with midpoints within r units away from e's midpoint
Travel cost	distance between the two endpoints of e
Resource cost	resource consumed when traversing e
Distance to depot	distance from the e's midpoint to the depot
Flow	total flow on e, i.e., $\hat{x}(e) = \sum_{r \in \Omega' \cup \Omega''} a_{ir} \hat{x}_r$
Average flow	average of the historical total flow on e
Variable statistics	average, minimum, maximum of $\hat{x}_r$ for routes r traversing e with $\hat{x}_r > 0$
Exact pseudocost & its indicator	exact pseudocost of e & its indicator of whether this value has been initialized
# Branch times	number of times e has been branched on
Average up/down exact value change & its indicator	average differences between the optimal value of the up/down ending child LP and that of the parent LP & its indicator of whether this value has been initialized
Column sparsity statistics	average, minimum, maximum of column sparsity for $x_r \in X(e)$
Constraint sparsity statistics	average, minimum, maximum of constraint sparsity for $x_r \in X(e)$, where the constraint sparsity of x_r is defined as the average of the constraint sparsity over ones where x_r has a nonzero coefficient
Binding constraint statistics	average, minimum, maximum of the number of binding constraints where $x_r \in X(e)$ has a nonzero coefficient
Reduced cost statistics	average, minimum, maximum of the reduced cost of $x_r \in X(e)$
Objective coefficient statistics	average, minimum, maximum of the coefficient of $x_r \in X(e)$ in the objective function
Constraint interaction	number of constraints that all $x_r \in X(e)$ have nonzero coefficients
Historical LP features	historical values of column sparsity statistics, constraint sparsity statistics, binding constraint statistics, reduced cost statistics, objective coefficient statistics, and constraint interaction
Edge constraint sparsity	sparsity of the set-partitioning constraints of the two endpoints of e
Edge constraint dual	dual values of the set-partitioning constraints of the two endpoints of e
Edge reduced cost	travel cost of e minus half of the sum of dual values of the set-partitioning constraints for the two endpoints of e
Historical edge features	historical values of edge constraint sparsity, edge constraint dual, and edge reduced cost
Feature set $\mathcal{F}_D$	Description
Average up/down LP value change & its indicator	average differences between the optimal value of the up/down starting child LP and that of the parent LP & its indicator of whether this value has been initialized
Average up/down deviation ratio & its indicator	average deviation ratio of the left/right starting child LP & its indicator of whether this value has been initialized
Scaled average up/down LP value change	average up/down LP value difference multiplied by $[1-\hat{x}(e)]$ or $\hat{x}(e)$ (share indicator of *Average up/down LP value change*)
LP pseudocost & its indicator	LP pseudocost of e & its indicator of whether this value has been initialized
Feature set $\mathcal{F}_E$	Description
Up/down dual value	dual value associated with the constraint added for branching on e in the up/down starting child LP
Up/down RC on e	RC's change when doing extension through the edge e giving the duals of the up/down starting child LP and travel cost of e
Up/down LP value change	difference between the optimal value of the up/down starting child LP and that of the parent LP
Product	product of the up and down LP value changes
Reduced cost change	difference between the edge reduced cost of the up/down starting child LP and that of the parent LP
Up/down change in # of active columns	difference between $\lvert\{\hat{x}_r > 0 : r \in \Omega' \cup \Omega''\}\rvert$ of the up/down starting child LP and that of the parent LP
Up/down change in # of related active columns	difference between $\lvert\{\hat{x}_r > 0 : x_r \in X(e)\}\rvert$ of the up/down starting child LP and that of the parent LP

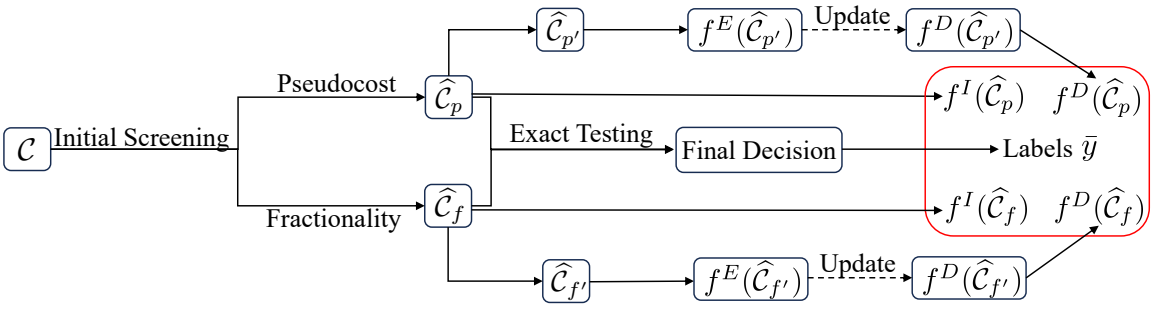
model when it is used follows a similar pattern to the training data. This is not much of a concern for conventional learning-to-branch settings, but it is not trivial for our case since we use a novel two-stage approach. Specifically, in real use, *M2* relies on *M1* to decide $\widehat{\mathcal{C}}$. This naturally motivates

us to first generate training data for *M1*, following the 3PB workflow where heuristic testing is substituted with exact testing. Once *M1* is trained, it is integrated into the 3PB, as shown in the Figure 3, to select a promising shortlist $\widehat{\mathcal{C}}$ from $\widehat{\mathcal{C}}_p$ and $\widehat{\mathcal{C}}_f$. Here again, heuristic testing is replaced by exact testing. The features of candidates in $\widehat{\mathcal{C}}$ are then collected to train *M2*.

There is another very subtle yet crucial issue with the training data generation for *M1*. Recall that training *M1* requires features in $\mathcal{F}_D$, which is computed based on expensive features $\mathcal{F}_E$ that is only calculated for candidates in $\widehat{\mathcal{C}}$. However, $\widehat{\mathcal{C}}$ is selected by *M1* from $\widehat{\mathcal{C}}_p$ and $\widehat{\mathcal{C}}_f$ while *M1* is not yet available when we are collecting data for training *M1*. To address this, we simulate a set $\widehat{\mathcal{C}}$ of size $\hat{\theta}_b$. Specifically, we proportionally select the top $\left\lfloor \hat{\theta}_b \cdot \frac{\hat{\theta}_p}{\hat{\theta}_{pf}} \right\rfloor$ and $\left(\hat{\theta}_b - \left\lfloor \hat{\theta}_b \cdot \frac{\hat{\theta}_p}{\hat{\theta}_{pf}} \right\rfloor\right)$ candidates from $\widehat{\mathcal{C}}_p$ and $\widehat{\mathcal{C}}_f$, based on pseudocost and fractionality, to form sets $\widehat{\mathcal{C}}_{p'}$ and $\widehat{\mathcal{C}}_{f'}$. In other words, $\widehat{\mathcal{C}}_{p'}$ and $\widehat{\mathcal{C}}_{f'}$ jointly serve as a substitute for the set $\widehat{\mathcal{C}}$. As illustrated in Figure EC.1, computationally expensive features $f^E(\widehat{\mathcal{C}}_{p'} \cup \widehat{\mathcal{C}}_{f'})$ are subsequently computed for these candidates to update features $f^D(\widehat{\mathcal{C}}_{p'} \cup \widehat{\mathcal{C}}_{f'})$, which together with $f^I(\widehat{\mathcal{C}}_{p'} \cup \widehat{\mathcal{C}}_{f'})$ and the labels computed by Equation (6)with respect to exact scores, constitute the training data for *M1*.

Ideally, the parameter $\hat{\theta}_b$ should be set to the minimum value such that $\widehat{\mathcal{C}}_{M1} \subseteq \widehat{\mathcal{C}}_{p'} \cup \widehat{\mathcal{C}}_{f'}$, where $\widehat{\mathcal{C}}_{M1}$ explicitly represents the candidate set selected by *M1*. It ensures that candidates in $\widehat{\mathcal{C}}_{M1}$ will have $\mathcal{F}_E$ computed during the data collection phase. Additionally, setting $\hat{\theta}_b$ too large is not desirable, as in the evaluation phase, *M1* avoids computing $\mathcal{F}_E$ for poor candidates. Therefore, in the data collection phase, we also want to avoid computing $\mathcal{F}_E$ for these poor candidates to maintain consistency between evaluation and data generation phases. Thus, when generating training data for *M1*, we choose $\hat{\theta}_b$ appropriately larger than $\hat{\theta}$. In this paper, we set $\hat{\theta}_b = 30$, while in the evaluation for *M1*, as shown in Section EC.7.2, $\hat{\theta}$ is set to 10.

Figure EC.1 Illustation of the data generation process for *M1*



EC.4. Instance Generation

In this study, we consider the two most important classic VRP variants, the CVRP and VRPTW. We evaluated CVRP instances generated by the Python script from the XML100 package (Lima et al. 2014) and those from the X benchmark series, following the attributes described by Uchoa

et al. (2017) (also summarized in Table EC.2). Depot locations varied among central, corner, and random positions within the grid. Customer positioning was either random, clustered, or random-clustered, with only half of the customers being clustered. There are seven demand patterns, defined by specific values or generated randomly with varying ranges and coefficients of variation (CV; ratio of the standard deviation to the mean). Finally, the average route size is defined as the ratio of the number of customers to the minimum number of vehicles used. In the Python script, the minimum number of vehicles is estimated by its lower bound, expressed as $\left\lceil \frac{\sum_i q_i}{Q} \right\rceil$, where q_i represents the demand of customer i, and Q is the vehicle capacity. This estimation avoids solving the corresponding bin-packing problem.

Table EC.2 Attribute values for generating the CVRP instances.

Attribute	Type
Depot positioning	1. random, 2. centered, 3. cornered
Customer positioning	1. random, 2. clustered, 3. random-clustered
Demand distribution	1. unitary, 2. small values with CV, 3. small values with small CV, 4. large values with large CV, 5. large values with small CV, 6. depending on quadrant, 7. many small values and few large values
Average route size	1. very short, 2. short, 3. medium, 4. long, 5. very long, 6. ultra long

We choose to consider 9 scenarios, which is the number of combinations of the first two attributes, i.e., depot positioning and customer positioning. Demand distribution and average route size are fixed at type 4 and type 2, respectively. The reason for choosing these 9 is that we found that the model trained by one of the nine scenarios cannot predict perfectly for the others, which suggests that the underlying distribution is notably different. For demand distribution and average route size, this performance variance is less obvious, and thus, we use only one type to represent the other types. We remark that choosing type 2 for average route size is particularly advantageous, as it effectively balances several factors. It ensures the pricing time is manageable, necessitates more branchings and the routes are not too short, thus maintaining the representativeness of the instances compared to other types. We use the python code to generate different instances of size 120, 150, and 180, and in each size batch, all 9 scenarios are included.

For training instances, each scenario has 30 instances, making a total of 270 instances, and for testing instances, each scenario has 10 instances, making a total of 90 instances. We remark that, for the original XML100 instances with an average route length of type 2, the gap is generally so small that only a few branchings are needed, thus not used in our study. As previously elaborated, the naming format of the instance batch is *problem-size-replication-type*. In this study, we use four batches of CVRP instances generated by the Python code: CVRP-150-270-0, CVRP-150-90-1, CVRP-120-90-1, and CVRP-180-90-1. For the established benchmark instances, specifically the X series, there are 100 instances with customer sizes ranging from 100 to 1000. After excluding

instances that are either too simple or exceedingly complex—where solving the root node alone demands hours of computing—42 instances remain. These instances vary in size from 119 to 915, and we name this batch as CVRP-V-42-X.

For the VRPTW, we follow the convention of Morabit et al. (2021) that generates instances from Homberger and Gehring (2005). They modified the instances with the aim of controlling the time spent on solving pricing subproblems. We are also motivated to keep it under control so that branching decisions play an important role in determining the overall BPC performance. Specifically, we reuse the same Python script provided by XML100 to generate CVRP instances and add **tight** time windows to the CVRP instances to generate VRPTW instances. The parameters used to generate VRPTW instances are included in Table EC.3, where $\mathcal{U}\{a,b\}$ denotes the discrete uniform distribution on $\{a,a+1,\cdots,b\}$ for any two given integers $a \leq b$. The established benchmark instances of VRPTW, such as those from Homberger and Gehring (2005), can be divided into three types: R, C, and RC, denoting random, clustered, and random-clustered customer distributions. We also consider these three types, using 20 instances of customer size 180 for each type, making a total of 60 instances for our testing set. For training purposes, we use 200 instances for each type, making a total of 600 instances of customer size 180. We generated four batches: VRPTW-180-600-0, VRPTW-180-60-1, VRPTW-160-60-1, and VRPTW-200-60-1. For the established benchmark instances, i.e., Homberger and Gehring (2005), we choose the type 2 series of size 200 as our testing bed. These instances are easy to solve by RouteOpt when using an aggressive cutting strategy (more time spent on adding cutting planes). Although doing so makes the RMP apparently harder, it pays off since the LB can be greatly enhanced, and thus, fewer branchings are needed. In this case, to highlight the differences brought by branching, we use setting II to test these instances with fewer cuts. There are 30 instances in total, and after removing those that are too easy or too hard under this setting, we are left with 22 instances, denoted as batch VRPTW-200-22-H2.

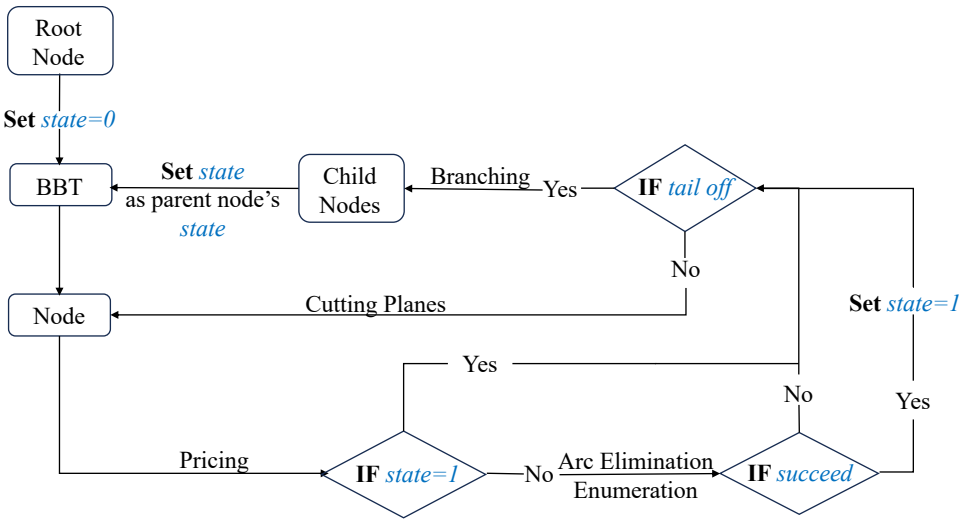
Table EC.3 Attribute values for generating the VRPTW instances.

Attribute	Type
Depot positioning	centered
Customer positioning	1. random, 2. clustered, 3. random-clustered
Demand distribution	large values with large CV
Vehicle capacity	1000
Maximum time resource (H)	12000
Ready time for type I customers	0
Due time for type I customers	$\mathcal{U}\{0.2H, 0.8H\}$
Ready time for type II customers (a_i)	$\mathcal{U}\{0.2H, 0.8H\}$
Due time for type II customers	$\mathcal{U}\{a_i, a_i + 0.2H\}$
Ratio of type I v.s. type II	4
Service time	$0.08H$

EC.5. RouteOpt

The development of RouteOpt is motivated directly by the needs of this research and also partially by the lack of completely open-source implementation of a BPC algorithm in academia that can achieve comparable performance to the VRPSolver. RouteOpt is implemented following the description outline in Pessoa et al. (2020). It employs most of the effective techniques for accelerating the pricing, including the bi-directional labeling (Righini and Salani 2006), *ng*-route relaxation (Baldacci et al. 2011), decremental state space relaxation (Martinelli et al. 2014), dynamic *ng* augmentation (Roberti and Mingozzi 2014, Bulhoes et al. 2018b), bucket graph (Sadykov et al. 2021), and arc elimination (Irnich et al. 2010, Sadykov et al. 2021). In addition, it applies effective cutting planes, including the RCCs (Naddef and Rinaldi 2002, Lysgaard et al. 2004) and lm-SRCs Pecin et al. 2017, Bulhoes et al. 2018a). Moreover, it performs an enumeration procedure (Baldacci et al. 2008, Yang 2023, 2024) to help close a BBN fast when the gap is relatively small. Its workflow also resembles a general BPC procedure, as shown in Figure EC.2. The state of each node is represented as a binary value, where a state of *1* indicates that all necessary columns have been enumerated, and a state of *0* denotes otherwise. Initially, the root node is set to a state of 0 and placed into the BBT as the input for the algorithm. Subsequently, each node is retrieved from the BBT to conduct pricing to solve the relaxed RMP to optimality. If the node's state is 1, we apply cutting planes to enhance the lower bound. Otherwise, we proceed with arc elimination and attempt to enumerate the columns before the cutting step. If enumeration succeeds, the node's state is updated to 1. When cutting planes tail off and the LP solution still remains fractional, branching is initiated, creating child nodes that inherit the state of their parent node. These child nodes are then placed back into the BBT. The algorithm terminates when all nodes in the BBT have been exhausted or optimality has been proved.

Figure EC.2 The BPC procedure in RouteOpt



EC.6. Implementation Details of BKF

To enhance BKF's practical applicability, we propose heuristic adaptations that address its inherent limitations, enabling its broader use with both model-free selection methods and scenarios involving multiple testing phases. We further provide guidelines for effectively estimating the input parameters required by BKF to achieve strong practical performance.

EC.6.1. Heuristic Adaptations

The direct application of BKF is potentially restricted by two requirements: (1) it necessitates an explicit selection model, e.g., *M1* in 2LBB, to narrow down the candidate set prior to the key testing phase; (2) it is structured to accommodate only one key testing phase, whereas, in practice, multiple key testing phases may be necessary. To enhance the practical applicability and effectiveness of BKF, we propose the following intuitive heuristic adaptations.

EC.6.1.1. Implicit selection model. Intuitively, any handcrafted selection rule based on ranking can be viewed as an implicit selection model. It is reasonable to assume that the least promising candidates are often ranked at the very end, allowing them to be safely screened out. Therefore, this serves the same purpose as *M1* in 2LBB but with a potentially larger m, i.e., lower selection accuracy. For example, the initial screening can be thought of as such a model by definition. In this paper, for the combination of 3PB with BKF, denoted as 3PB-dy, we treat the initial screening itself as an implicit selection model for our experiments, setting the parameter $\alpha = 0.6$ obtained through fine-tuning. It is important to note that although the determination of k for 3PB could strictly follow the method of calculating $\mathbb{E}[T(k)]$ for each k within $[1, n]$ as outlined in Equation (8) and selecting the one that achieves the smallest $\mathbb{E}[T(k)]$, this process is considerably time-consuming due to the computation of combinatorial sums, and thus is avoided.

EC.6.1.2. Greedy determination. To address scenarios with multiple key testing phases, we propose a greedy algorithmic approach, wherein we sequentially determine the best k_i for each phase $i \in \mathbb{Z}^+$ using the fine-tuned parameter m_i. Originally, this section aimed to highlight that LP testing generally serves as the primary key testing phase for both 3PB and 2LBB. However, there are instances where heuristic testing is significantly more time-consuming, e.g., over 25 times as one LP testing, as evidenced by the VRPTW instances discussed in Section 6.2.3. In such scenarios, even reducing one unnecessary test in the heuristic phase can yield substantial time savings. Therefore, in cases where multiple phases are time-dominant in the branching selection, we employ this greedy determination strategy in our numerical design to achieve further acceleration.

EC.6.2. Practical Usage

The above procedure decides the value of k that helps achieve the smallest computational time in practice. It involves seven parameters, G, F, c, t, n, m, and R. However, as we elaborated in Section EC.8, when Assumption (f) about the tree size significantly deviates from reality, discrepancies between our predicted objective function, i.e., $\widehat{T}_k$, and the actual solution time can be substantial. Therefore, we introduce an additional parameter, η, to correct R. The way of computing the parameters is summarized in Table EC.4. Let U_1 be the set of all explored nodes so far, and U_2 be the set of all explored nodes that have branched at least as many times as the current node. It is important to note that the maximum local optimality gap at a leaf node does not necessarily have to be zero. Consider the case where all columns at a node have been enumerated, allowing the node to be solved directly by an integer programming solver. Although the local optimality gap at this node is not zero, it indeed qualifies the node as a leaf node.

Table EC.4 Parameter Calculation for the BKF

Parameter	Calculation
m	Needs fine-tuning; the less effective the selection model is, the greater m should be.
η	Needs fine-tuning; the smaller testing score as the branching level increases, the smaller η should be.
n	Set to the number of candidates input in this testing phase.
c	Set to the mean of c_u, for all $u \in U_1$.
t	Set to the mean of t_u, for all $u \in U_1$.
G	Set to the local optimality gap at the current node.
F	Set to the current maximum local optimality gap at the leaf node, if not known, then set to 0 with $N(F) = 1$.
R	Set to $\eta R'$, where, based on the equation in Lemma 1 as $n \to \infty$, R' is set to $\frac{r}{1-\alpha/(k+1)}$, and r is the geometric mean of the testing score of the last branching decision for all $u \in U_2$.

EC.7. Preliminary Experiments

In this section, we present a series of experiments under experimental setting II to determine and justify the design of multiple key components in our framework. We compare the training of *M1* using LP-based features versus graph-based features, demonstrating that graph-based features perform significantly better. Through three carefully designed sets of experiments, we effectively show the performance of *M1* and *M2*, as well as the superiority of our dynamic adjustment compared to existing methods. Moreover, we highlight the substantial gap in computational time between our BKF and the other naive dynamic scheme and validate the predicted solution time, i.e., $\widehat{T}_k$ and Proposition 4. These aspects are particularly emphasized as they either form the foundation of our dynamic adjustment scheme or provide compelling evidence of its practical efficacy. These experiments were conducted using the instance batches CVRP-150-270-0 for training purposes and CVRP-150-90-1 for evaluating performance when integrated into RouteOpt. The numerical

results reported are geometric means of values across all instances within the respective batches. To quantify the success of a branching strategy in selecting promising candidates, we first introduce the following definition to establish a clear metric for evaluation.

Definition of p-accuracy: A selection of n' out of n items is considered p-accurate if, among the selected n', there exists at least one whose value (e.g., exact score) is no smaller than p (e.g., 0.8) times the highest values of all n candidates. For a branching procedure performing such selection L times (i.e., branching L times), its p-accuracy is defined as the number of p-accurate selections divided by L.

EC.7.1. LP-based Features versus Graph-Based Features for Training *M1*

Recall that $\hat{\theta}$ is the number of candidates selected by *M1* from the initial screening of $\hat{\theta}_{pf} = 100$ candidates. The prediction accuracy of *M1* is crucial as it enables us to use a smaller $\hat{\theta}$ while still maintaining a high probability of having at least one good candidate in $\widehat{\mathcal{C}}$. Consequently, the computational effort required for LP testing can be reduced.

We trained two versions of *M1*: one using LP-based features from Yang et al. (2022) that are commonly used in learning branching strategies for solving MIPs, and the other using selected graph-based features, $\mathcal{F}_I$ and $\mathcal{F}_D$, as highlighted in bold in Table EC.1. We calculate their p-accuracy under different choices of $\hat{\theta}$ values and summarize the results in Table EC.5. The *M1* trained with traditional LP-based features, not surprisingly, shows a low p-accuracy for $p = 1.0$, 0.9, and 0.8, which confirms the difficulty and the challenges explained in Section 1.1. In contrast, the *M1* trained with graph-based features consistently improves the accuracy by approximately 20%, making 1.0-accuracy close to 70% when $\hat{\theta} = 10$. Therefore, we adopt graph-based features for training *M1* in our framework.

Table EC.5 Summary of p-accuracy for the two versions of the model under different $\hat{\theta}$ values.

Version	$\hat{\theta}=5$			$\hat{\theta}=10$			$\hat{\theta}=15$			$\hat{\theta}=20$		
	$p=1.0$	$p=0.9$	$p=0.8$	$p=1.0$	$p=0.9$	$p=0.8$	$p=1.0$	$p=0.9$	$p=0.8$	$p=1.0$	$p=0.9$	$p=0.8$
LP-based	33.51%	41.06%	50.00%	46.09%	54.55%	63.55%	56.38%	64.61%	73.17%	64.57%	72.77%	79.91%
Graph-based	52.74%	61.82%	71.05%	69.87%	77.52%	84.00%	80.54%	85.87%	90.58%	87.68%	91.14%	94.38%

EC.7.2. Effectiveness of *M1*.

In this section, we demonstrate the effectiveness of *M1* in narrowing down the population of promising candidates by incorporating it directly into the 3PB. Specifically, we use *M1* to help downsize the branching candidate set $\widehat{\mathcal{C}}_p \cup \widehat{\mathcal{C}}_f$ obtained from the initial screening into a much smaller $\widehat{\mathcal{C}}$, which subsequently undergoes LP testing to be turned into $\widetilde{\mathcal{C}}$. The 3PB with *M1* incorporated is referred to as WithM1. For consistency, we set $\hat{\theta}_{pf} = 100$ and $\tilde{\theta} = 1$ in both the 3PB and WithM1. We vary the values of $\hat{\theta}$ and consider three choices: $\hat{\theta} = 5, 10$, and 15. The results of average LP testing time

per branching selection (LP/s), average time for solving a BBN excluding LP testing (Other/s), computational time (CPU/s) and tree size (Node) are summarized in Table EC.6. As it shows, there is a modest increase in LP testing time and solving a BBN time when *M1* is used. We observed that the branching candidates selected by *M1* usually correspond to denser branching constraints, requiring slightly more time for LP testing. We also observed that although the best candidate in $\widehat{\mathcal{C}}$ decided by *M1* often has a higher LP testing score, the gap between LP testing score and exact score is also larger, thus requiring more iterations of column generation to eliminate this gap when solving a BBN. Fortunately, such trivial extra computational efforts paid are completely worth the impressive gain in nodes reduction of 40% ($\hat{\theta}=5$), leading to an overall time reduction of 38%.

Additionally, as $\hat{\theta}$ increases, the node reduction diminishes from 40% to 39%, and finally to 35%. This trend can be attributed to the marginal effects inherent in the testing process; notably, the node reduction achieved is not linear but exhibits a significant decay as the number of testing increases. This behavior can not only be predicted by BKF but also observed in the progressively smaller decrements in tree size when $\hat{\theta}$ is increased from 5 to 10, and from 10 to 15—yielding reductions from 47.4 to 1.4 (WithM1) and from 91.3 to 29.5 (3PB). Since *M1* reaches its effectiveness limit much earlier than 3PB, the node reductions correspondingly diminish. Although $\hat{\theta}=5$ gives the minimum overall time for CVRP-150-90-1 under setting II, to make $\hat{\theta}$ more robust to different instances and settings, we decide to set $\hat{\theta}$ to 10 in the 2LBB since the best balance of tree size and computational time is achieved.

Table EC.6 Summary of performance metrics with and without *M1* under different $\hat{\theta}$ values.

Version	$\hat{\theta}=5$				$\hat{\theta}=10$				$\hat{\theta}=15$			
	LP/s	Other/s	CPU/s	Node	LP/s	Other/s	CPU/s	Node	LP/s	Other/s	CPU/s	Node
WithM1	0.75	0.76	403.6	346.2	1.41	0.78	458.2	298.8	2.06	0.78	556.5	297.4
3PB	0.71	0.73	647.8	578.6	1.35	0.75	714.8	487.3	2.01	0.75	829.4	457.8

EC.7.3. Effectiveness of *M2*

We demonstrate its effectiveness by comparing the 2LBB derived following the workflow in Figure 3, denoted as Control, and the Control without using *M2*, denoted as NoM2. In NoM2, the candidate set $\widehat{\mathcal{C}}$ selected by *M1* undergoes LP testing, and the $\tilde{\theta}$ candidates with the highest LP scores are directly selected for heuristic testing. We test three cases with $\tilde{\theta}=1,2$, and 3. In Table EC.7, we include the column representing the average heuristic testing time per branching selection (Heur/s) for $\tilde{\theta}=2,3$. When $\tilde{\theta}=1$, the candidate with the highest LP testing score is directly selected as the final branching decision, and thus the heuristic testing time is not included. As expected, the time spent on LP testing and other components for solving a BBN remains nearly the same if considering the computational noise. What is worth noting is that *M2* is highly effective in reducing the time for heuristic testing by over 80% ($\tilde{\theta}=3$), while the tree size increases only by 3%. Therefore, the

overall solution time is decreased by approximately 24%. Moreover, as $\tilde{\theta}$ grows from 2 to 3, only minor increase in heuristic testing time is observed for the **Control** version. Again, we choose $\tilde{\theta}=2$ for 3PB and $\tilde{\theta}=3$ for 2LBB by considering both the solution speed and the number of nodes, and we keep this setting in all remaining experiments except for those in solving open instances, which have super large gaps and require more consideration on the metric of the number of nodes.

Table EC.7 Summary of performance metrics with and without *M2* under different $\tilde{\theta}$ values.

Version	$\tilde{\theta}=1$				$\tilde{\theta}=2$					$\tilde{\theta}=3$				
	LP/s	Other/s	CPU/s	Node	LP/s	Heur/s	Other/s	CPU/s	Node	LP/s	Heur/s	Other/s	CPU/s	Node
NoM2	1.41	0.78	458.2	298.8	1.40	0.97	0.76	464.6	231.2	1.39	1.37	0.77	478.1	215.1
Control					1.40	0.21	0.76	365.4	227.9	1.39	0.27	0.76	362.5	222.1

EC.8. Superiority of Theory-Derived Dynamic Adjustment

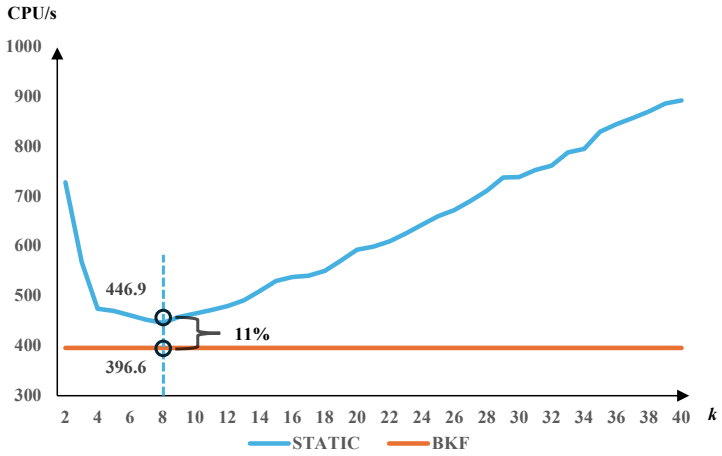
In this section, we conduct three sets of experiments to demonstrate the superiority of BKF. To begin with, we underscore the necessity of a dynamic mechanism by comparing the performance of RouteOpt using BKF against a fine-tuned static k. Subsequently, we illustrate that BKF outperforms a simpler dynamic scheme, highlighting that although its design is complex, it is justified by its effectiveness. Lastly, we conclusively demonstrate why BKF achieves superior performance in the aforementioned cases by showing that $\widehat{T}_k$ is able to accurately predict real solution times.

To exclude the noise brought by *M2*, all experiments in this section are conducted under the same setting as NoM2. Specifically, after LP testing, the two candidates with the highest LP testing scores are selected for heuristic testing directly, and the candidate with the highest heuristic testing score is chosen as the final branching decision. To align with the notation used in Section 6, we remark that k stands for the number of candidates that undergo LP testing, i.e., $\hat{\theta}$ in the 2LBB framework.

EC.8.1. BKF is Better than Static Scheme

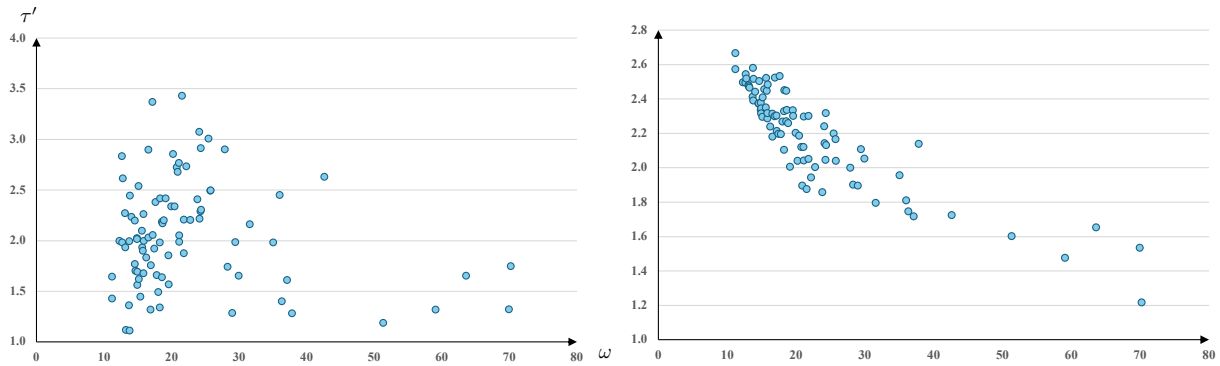
We demonstrate that the BKF outperforms the static scheme even when k is finely tuned. To find the best static k that gives the minimum solution time, we conducted 39 groups of experiments, denoted by STATIC-k, where k ranges from 2 to 40 and remains unchanged throughout the solution procedure. In Figure EC.3, the CPU time for k values from 2 to 40 is depicted by the blue curve, alongside an orange horizontal line representing the CPU time when using the BKF. The results show that the minimum solution time of 446.9 seconds is achieved when $k=8$. In contrast, our BKF requires only 396.6 seconds, which is 11% less than the best static k. This indicates that even with the most aggressive parameter tuning, the computational performance is inferior to that achieved with the proposed dynamic adjustment. In practical applications, such aggressive tuning is often impractical, further emphasizing the necessity of our parameter-free BKF .

Figure EC.3 CPU time for k values from 2 to 40 compared with the BKF.



Additionally, using the data at hand, we can provide empirical evidence for Proposition 4. We collected ω and $\tau := \frac{\text{CPU}_{k=40}}{\text{CPU}_{k=8}}$ for each instance. As shown in Figure EC.4a, we found that the monotonic relationship is not clear. Further study revealed that this is because of the fact that the ratio $\frac{\text{Node}_{k=40}}{\text{Node}_{k=8}}$ varies across different instances. In Proposition 4, the condition *same instance* implies that $\frac{\text{Node}_{k=40}}{\text{Node}_{k=8}}$ remains constant for ω. Therefore, we used the revised τ', i.e., $\tau' = \frac{\text{CPU}_{k=40}/\text{Node}_{k=40}}{\text{CPU}_{k=8}/\text{Node}_{k=8}}$ to discount the noise brought by the varying $\frac{\text{Node}_{k=40}}{\text{Node}_{k=8}}$ among different instances. As shown in Figure EC.4b, τ' indeed decreases as ω increases, providing strong empirical evidence for Proposition 4.

Figure EC.4 Comparison of τ and τ' for different instances.



(a) Original τ vs. ω

(b) Revised τ' vs. ω

EC.8.2. BKF is Better Than Other Dynamic Adjustment

As suggested by Pessoa et al. (2020), one can first estimate the subtree size rooted at the current node, and then the number of testing can be computed by $k = \min\{\zeta^{\text{num}}, \zeta^{\text{estim}} \cdot n^{bb}\}$ (hereafter referred to as linear dynamic adjustment), where ζ^{num}, ζ^{estim}, and n^{bb} represent the maximum number of candidates available to be tested, the estimation factor, and the estimated subtree size,

respectively. To investigate whether assuming the linear relationship between k and n^{bb} is appropriate, we conducted experiments using BKF and linear dynamic adjustment with three different parameters. These are denoted as BKF, LINEAR-01, LINEAR-02, and LINEAR-03, where the number in the name represents the value of ζ^{estim} used. For example, LINEAR-01 corresponds to $\zeta^{\text{estim}} = 0.1$. We reported the time for LP testing (LP/s), the overall solution time (CPU/s), the total number of candidates performing LP testing (#LP), and the number of nodes explored (Node). We found that BKF needed to perform only half the number of LP testing compared to the linear dynamic adjustment methods, yet achieved a smaller BBT size. Specifically, compared to LINEAR-01, the node reduction reached 13%. The results empirically prove that our BKF provides a better strategy for decreasing k than linear dynamic adjustments.

The reduction of overall solution time greatly benefited from the reduction in LP testing. However, the two reductions do not match exactly. As analyzed in Section EC.7.2, a better branching candidate usually increases the average pricing time for a node. Nevertheless, this cost is generally offset by the overall benefits. Additionally, we observed that as ζ^{estim} increased, the number of total LP testing decreased, which seems counterintuitive. This is because of the reduction in tree size, leading to fewer nodes requiring branching and testing.

Table EC.8 Comparison of performance metrics for BKF and linear dynamic adjustment with different parameters.

	LP/s	CPU/s	#LP	Node
BKF	98.3	396.6	742.2	258.3
LINEAR-01	181.4	467.5	1519.5	297.2
LINEAR-02	172.7	455.4	1418.5	282.2
LINEAR-03	180.6	455.7	1463.9	268.7

EC.8.3. $\widehat{T}_k$ is an Accurate Predictor

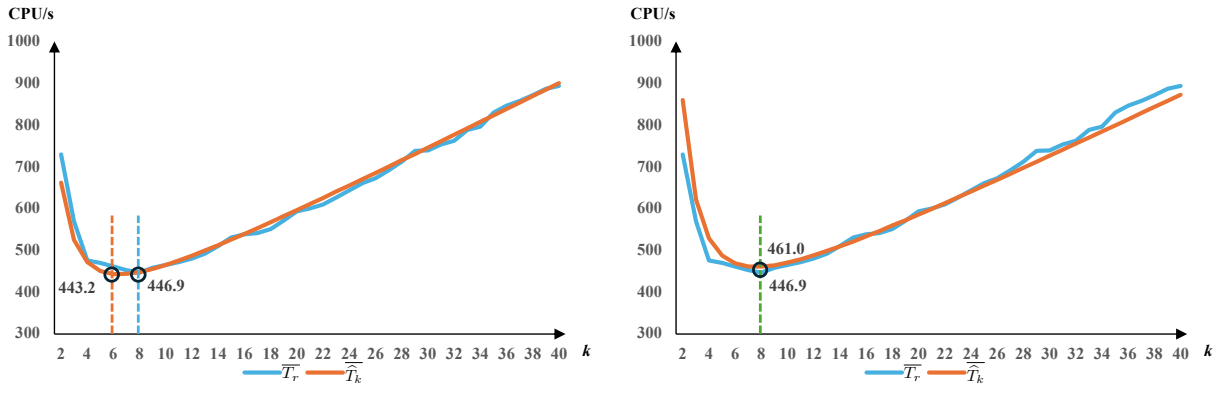
While achieving the best computational time is impressive, stronger empirical evidence would come from directly validating that the derived solution time, $\widehat{T}_k$, aligns with the actual solution time, T_r, ensuring that we are optimizing the correct function. To achieve this, we compare the geometric means of T_r and $\widehat{T}_k$, denoted respectively as $\overline{T_r}$ and $\overline{\widehat{T}_k}$, across the same batch of instances. We collected $\overline{T_r}$ from 39 experimental groups detailed in Section EC.8.1. For each group, we calculated $\overline{\widehat{T}_k}$ using data extracted from the output log files. Specifically, we analyzed the log for the group STATIC-40 and obtained the statistical data as outlined in Section EC.6.2. Using this statistical data, we calculated $\widehat{T}_k$ for each instance and then computed its geometric mean, $\overline{\widehat{T}_k}$. This group was selected because the parameter 40 is presumed to be no smaller than m, ensuring that the best decision at the root node achieves the highest testing score, which facilitates the evaluation of R.

We study two cases with different (η, α) combinations, i.e., $(1.00, 0.40)$ and $(0.72, 0.37)$. Their curves are depicted in orange in Figures EC.5a and EC.5b, respectively, while $\overline{T_r}$ is shown in blue. To

better visualize Figure EC.5a, we scaled the data points of $\overline{\widehat{T}_k}$ by a factor of the sum of $\overline{T_r}$ divided by the sum of $\overline{\widehat{T}_k}$, which is 5.5, ensuring that the two curves are in the same region. For Figure EC.5b, no scaling was necessary. As seen in both Figures EC.5a and EC.5b, the shape of the curves for the predicted solution time and the ground truth solution time match quite well. Given the definition of the *square root mean square normalized error* (SQMSNE), i.e., $\text{SQMSNE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{\overline{T_r}-\overline{\widehat{T}_k}}{\overline{T_r}}\right)^2}$, where i represents the i-th data point, and n is the number of all data points, we find that for the combination $(0.72, 0.37)$, the SQMSNE is only 0.042. In other words, for each data point, the average deviation is only 4.2% from the ground truth value. Moreover, both curves achieve their lowest data points when $k = 8$. It is an important observation since we care more about finding the k^* that minimizes T_r rather than predicting the precise value of T_r in the dynamic adjustment application. This further demonstrates $\overline{\widehat{T}_k}$ accurately approximates $\overline{\widehat{T}_k}$.

However, for the combination $(1.00, 0.40)$, the SQMSNE is 0.818, indicating a significant deviation from the ground truth. We attribute the error mainly to the inaccurate estimation of R. As the branching level deepens, there is a noticeable decreasing trend in testing scores. Consequently, using the testing score of the root node to represent R throughout the entire solution procedure can lead to an underestimated tree size, since R is overestimated in this case. Such overestimation further leads to underestimating the BBT size and the need for testing. As a result, the k^* values of the two curves differ, with the k^* given by $\overline{\widehat{T}_k}$ smaller than that of $\overline{T_r}$. Therefore, for most experiments, if applicable, we opt for $\eta = 0.72$ to adjust the estimation of R as indicated in Section EC.6.2. For instances with a more pronounced decay in the testing score at deeper levels of branching, we employ an even smaller $\eta = 0.50$.

Figure EC.5 Comparison of $\overline{T_r}$ and $\overline{\widehat{T}_k}$ for different (η, α) combinations.



(a) $(1.00, 0.40)$ (b) $(0.72, 0.37)$

EC.8.4. BKF Also Works for B&C Algorithms

Although this study primarily focuses on BPC, the derivation of BKF extends beyond this context. To validate its broader applicability, we assess the effectiveness of BKF in a B&C algorithm. We conduct the case study on the CVRP, employing the two-commodity formulation proposed in Baldacci et al. (2004). In this formulation, the number of vehicles in use needs to be specified, and we set it to $\left\lceil \frac{\sum_i q_i}{Q} \right\rceil$ in our experiments, where q_i denotes the demand of each customer and Q represents the vehicle capacity.

We construct our testbed, denoted as *X50*, using the X-series (Uchoa et al. 2017), which consists of 100 instances. For each instance, we use the first 50 nodes, comprising the depot and 49 customers. The optimal value of each instance, readily obtained using RouteOpt, serves as the cutoff value. Our B&C algorithm is implemented using CPLEX (version 22.1.0; IBM Corporation 2023) with default settings and applies RCCs separated by the CVRPSEP package (Lysgaard 2003) as user-defined cuts. For branching, we continue to use 3PB; however, since the B&C framework does not involve CG, heuristic testing is omitted. Therefore, the number of LP testing conducted becomes the only critical parameter we aim to determine in this context. We consider eight strategies: STATIC-5, STATIC-10, STATIC-15, STATIC-20, LINEAR-01, LINEAR-02, LINEAR-03, and BKF. For each instance under each strategy, a time limit of 2 hours is enforced.

There are 29 instances for which B&C explores fewer than 100 nodes under all eight strategies. To ensure statistical reliability, we exclude them in our analysis, which results in 71 instances being considered. When reporting the performance metrics, we divide the instances into two sets. The first set includes 41 instances that can be solved to optimality within the time limit in all eight strategies. The second set includes the remaining ones (30 instances). For the first set, we report the number of nodes explored (#Nodes) and the computation time (CPU/s). For the second set, we report the final optimality gap (Gap/%) and #Nodes.

As shown in Table EC.9, BKF achieves the best performance in terms of computational time, outperforming all linear dynamic and static strategies. In the first set, BKF reduces runtime by 34% compared to LINEAR-02, the best linear dynamic strategy, and by 41% compared to STATIC-5, the best static one. In the second set, BKF also obtains the smallest optimality gap among all strategies. These results further demonstrate that our proposed BKF is also an effective dynamic branching scheme for B&C algorithms.

EC.9. Training Details

For training purposes, we used GPU-accelerated computations with NVIDIA GeForce RTX 2080 graphics cards. These GPUs are managed by driver version 535.154.05 and operate with CUDA version 12.2, enabling efficient execution of parallel processes essential for our experiments. Each GPU is equipped with 8 GB of memory.

Table EC.9 Comparison of the eight strategies for CVRP batch: X50

Instance	STATIC-5		STATIC-10		STATIC-15		STATIC-20		LINEAR-01		LINEAR-02		LINEAR-03		BKF	
	CPU	Node	CPU	Node	CPU	Node	CPU	Node	CPU	Node	CPU	Node	CPU	Node	CPU	Node
X-n110-k13	7.9	52	7.2	30	7.0	20	9.8	34	16.3	179	14.9	169	12.6	107	6.1	40
X-n134-k13	4.1	40	3.8	29	3.0	14	3.4	20	7.1	96	7.4	78	11.4	153	3.0	22
X-n143-k7	8.6	138	10.4	179	11.6	123	10.7	90	7.1	185	7.1	186	8.2	234	7.9	232
X-n148-k46	2369.2	2620	1061.1	812	496.6	329	690.7	364	922.4	960	1079.3	1141	1062.8	1116	456.7	590
X-n167-k10	2146.7	10840	2258.9	7475	2642.9	5711	2298.1	4219	633.5	4739	605.8	4312	782.9	5550	666.1	5776
X-n172-k51	340.3	526	719.0	633	662.3	446	559.9	314	731.2	1132	721.9	1018	488.0	750	338.2	610
X-n181-k23	65.5	470	67.9	293	32.2	112	39.1	147	48.2	353	39.3	258	44.3	314	41.3	393
X-n186-k15	48.3	282	30.9	107	37.0	88	58.3	127	54.9	404	48.7	340	47.3	463	33.9	291
X-n200-k36	687.4	1839	731.4	1676	572.3	1018	784.2	1084	718.3	2076	484.9	1534	597.3	1645	309.1	1030
X-n204-k19	7.9	148	4.0	35	6.0	36	4.6	25	6.6	127	6.1	86	7.1	125	5.8	89
X-n209-k16	51.8	350	35.1	128	49.7	163	68.7	182	47.2	328	59.7	403	60.8	452	40.1	317
X-n214-k11	40.6	517	35.9	351	44.5	334	45.7	280	32.5	628	37.9	681	37.2	523	24.4	394
X-n233-k16	13.3	223	13.8	160	13.0	115	18.3	137	20.9	382	17.6	317	17.8	306	11.8	194
X-n237-k14	6.6	54	5.8	26	9.8	50	7.2	24	6.8	136	11.7	160	12.0	146	4.2	26
X-n242-k48	4083.1	4285	4060.1	3571	1539.8	1056	1594.8	967	1283.2	2549	1483.7	2268	1093.7	1737	1508.5	2379
X-n251-k28	842.3	3107	1154.0	3211	646.7	1435	620.0	1257	486.4	2047	458.2	1816	359.9	1492	495.4	2761
X-n261-k13	11.4	197	7.9	80	11.1	89	12.2	66	15.4	290	13.9	207	13.1	182	7.0	89
X-n270-k35	2767.2	11140	3434.7	9699	2348.6	5729	2661.2	4621	1084.9	5298	1030.4	6336	1103.1	5092	915.5	6048
X-n298-k31	76.3	230	105.9	177	106.2	139	114.3	128	170.4	630	141.7	566	145.7	486	95.6	410
X-n308-k13	2662.0	3477	1949.3	1904	1917.4	1735	2407.4	1558	1536.2	2873	1347.8	2520	1199.8	2328	728.1	1616
X-n317-k53	167.6	287	438.6	573	333.3	303	308.0	212	381.2	665	345.9	687	433.4	675	251.2	482
X-n331-k15	7.2	217	7.5	177	7.7	109	9.8	113	9.4	485	9.5	356	9.1	257	5.0	177
X-n344-k43	402.8	2294	598.5	2376	591.2	1959	643.9	1556	567.6	3759	478.1	3014	542.6	3006	254.3	1757
X-n351-k40	32.6	192	47.6	184	31.8	129	37.5	101	54.9	284	57.0	289	39.9	223	32.2	228
X-n359-k29	536.9	2435	570.6	1669	518.2	1066	520.3	866	619.7	3746	386.0	2265	397.6	1966	442.0	2430
X-n367-k17	545.8	972	465.7	604	345.3	335	608.2	409	386.4	790	303.6	618	389.9	632	219.5	485
X-n401-k29	1855.1	4332	4727.6	7266	4211.5	4633	4456.9	4544	1431.4	2422	1603.0	2428	1218.6	2280	608.6	1602
X-n429-k61	4457.9	6462	6254.8	5894	6909.3	4521	4360.4	1980	2107.4	4173	2193.5	4155	2040.3	2479	1484.5	2671
X-n449-k29	795.4	4555	713.3	3604	1223.4	4415	1337.0	4365	326.4	3034	347.0	3130	390.5	3050	232.1	2192
X-n480-k70	405.2	1522	417.4	1110	301.6	655	486.7	753	495.3	2004	422.6	1577	420.7	1580	294.5	1317
X-n599-k92	1561.5	4125	2050.4	3291	2527.8	2905	3437.3	3172	1216.0	3038	1457.1	3265	988.1	2667	1195.0	3323
X-n627-k43	15.4	226	10.4	96	8.8	67	10.4	50	6.0	69	6.5	81	6.5	81	5.6	35
X-n641-k35	21.2	273	17.2	103	26.9	154	21.9	91	35.6	362	20.2	200	29.7	257	12.7	127
X-n655-k131	89.9	447	222.1	764	93.0	255	80.2	145	95.5	491	112.4	536	125.5	592	55.4	297
X-n685-k75	1421.3	3761	1486.2	2584	2396.3	3235	2240.8	2045	753.9	2452	1037.2	3134	832.9	2611	667.0	2595
X-n701-k44	273.7	1879	129.4	704	268.4	1154	218.7	534	158.8	1139	141.6	1025	157.3	882	87.5	800
X-n783-k48	261.9	1869	333.0	1507	263.3	806	420.9	995	175.8	1409	199.7	1771	221.0	1575	134.7	1250
X-n801-k40	7.8	82	6.3	53	7.8	65	9.5	62	10.1	177	8.2	113	12.4	175	6.5	75
X-n819-k171	3.3	21	3.4	22	3.8	22	3.2	18	11.6	408	3.8	23	3.8	23	3.6	64
X-n876-k59	21.8	224	40.4	350	28.9	149	25.5	80	37.0	502	31.5	312	33.7	352	18.6	196
X-n957-k87	13.8	169	15.6	120	19.4	124	17.7	72	20.7	328	25.5	300	18.9	195	8.2	81
G.M.	119.5	637.6	125.8	460.6	120.2	340.6	130.1	291.1	113.2	756.0	106.9	636.2	107.9	613.4	70.1	441.8
	Gap	Node	Gap	Node	Gap	Node	Gap	Node	Gap	Node	Gap	Node	Gap	Node	Gap	Node
X-n101-k25	1.81	1277	1.97	707	1.93	536	1.91	483	1.75	829	1.75	710	1.95	581	1.87	945
X-n106-k14	0.32	3732	0.32	1935	0.30	1643	0.29	1386	0.17	6788	0.18	5554	0.18	4803	0.19	5117
X-n115-k10	3.28	3879	3.01	2022	3.12	1571	3.26	1149	3.22	1172	3.06	1280	3.08	1303	2.85	2118
X-n125-k30	0.29	2349	0.28	1431	0.24	1233	0.21	995	0.19	3489	0.16	3223	0.12	3922	0.18	3171
X-n129-k18	0.98	3049	0.75	2032	0.85	1847	1.03	1130	0.57	4744	0.60	4025	0.66	3730	0.55	4550
X-n153-k22	12.47	932	12.81	488	12.81	444	12.74	397	12.81	539	12.81	534	12.81	534	12.81	515
X-n176-k26	12.24	516	12.12	348	12.12	333	12.12	245	12.12	381	12.12	378	12.12	353	12.12	362
X-n195-k51	1.54	2026	1.36	1459	1.51	1060	1.61	733	1.14	3189	1.16	2722	1.31	1684	1.02	2950
X-n219-k73	0.36	3078	0.28	2237	0.27	1545	0.22	1624	0.22	3454	0.10	4353	0.30	2085	0.33	2079
X-n228-k23	2.02	747	1.62	543	1.80	339	1.35	301	1.59	448	1.32	444	1.59	432	1.43	570
X-n266-k58	3.11	1389	2.85	815	2.90	759	2.95	596	3.16	956	3.17	1005	3.07	1133	2.31	1459
X-n280-k17	2.02	576	1.82	308	2.19	242	2.30	199	1.69	353	1.90	268	2.17	261	1.55	495
X-n289-k60	2.27	1047	2.48	687	2.12	535	2.29	417	2.17	431	2.17	432	2.06	542	1.95	848
X-n294-k50	0.64	8449	0.66	4972	0.76	3898	0.76	2906	0.51	15067	0.42	15594	0.50	14303	0.48	12441
X-n313-k71	1.15	1074	1.26	700	1.16	387	1.14	417	0.99	1008	0.97	826	0.92	970	0.63	1867
X-n336-k84	2.28	1136	2.39	711	2.38	627	2.72	441	2.26	544	2.26	569	2.32	512	2.06	952
X-n376-k94	0.84	1218	0.74	938	0.68	856	0.59	864	0.50	1869	0.58	1411	0.54	1574	0.46	2272
X-n384-k52	1.55	1967	1.35	2149	1.53	1068	2.01	623	1.26	2138	1.40	1879	1.32	2087	1.09	3258
X-n411-k19	0.51	2021	0.60	949	0.56	597	0.75	362	0.48	1499	0.37	2003	0.46	1137	0.47	1856
X-n420-k130	0.71	2060	0.55	1628	0.37	1491	0.32	1295	0.00	6386	0.00	5283	0.00	5555	0.00	5855
X-n469-k138	4.78	510	5.03	368	5.27	239	5.27	195	4.66	386	4.71	396	4.72	367	4.80	415
X-n491-k59	1.32	3958	1.22	2979	1.67	1324	1.71	1544	1.47	3376	0.85	5530	0.95	3987	0.73	5284
X-n536-k96	0.00	10060	0.00	19333	0.00	17155	0.01	12563	0.00	4127	0.00	5009	0.00	4339	0.00	8945
X-n548-k50	0.07	12000	0.00	12034	0.08	6439	0.11	4331	0.00	14907	0.00	13352	0.00	13712	0.00	14714
X-n573-k30	2.22	620	2.00	348	2.44	282	2.38	228	2.38	358	2.38	345	2.38	344	2.38	357
X-n586-k159	2.53	1026	2.65	765	2.84	506	2.80	445	2.63	577	2.55	630	2.55	618	2.55	750
X-n749-k98	0.99	2783	1.01	1524	1.08	917	1.09	719	0.89	2113	0.79	2002	0.96	1306	0.77	2724
X-n766-k71	4.29	444	4.78	274	4.41	276	4.62	270	4.22	260	4.12	217	4.42	286	4.06	252
X-n837-k142	1.30	3891	0.72	3515	0.88	1929	1.16	1333	1.12	3353	0.80	5498	1.15	2811	0.91	5789
X-n916-k207	2.49	1249	2.38	898	2.65	497	2.58	464	2.31	674	2.34	615	2.29	616	2.32	804
Average (Gap)	2.35	-	2.30	-	2.36	-	2.41	-	2.22	-	2.17	-	2.23	-	2.10	-
G.M. (Node)	-	1761.5	-	1207.7	-	882.7	-	713.6	-	1470.0	-	1455.6	-	1321.7	-	1766.5

EC.9.1. Training Details for CVRP

The training data is generated by solving instances in batch CVRP-150-270-0, conducted on HiPer-Gator, which can run all 270 instances simultaneously. We set the time limit to 24 hours. In order to generate more training data with less computational effort, we set the RouteOpt solution mode to setting II, and excluded nodes with depth exceeding 5. Doing so not only reduces data generation time significantly but also helps the machine learning model focus on branching decisions made close to the root, which are more important than those made deeper in the tree. The training data generation process, as illustrated in Section 4.2, is also a two-stage process.

In the first stage, data is generated for *M1*, where $\hat{\theta}_{pf} = 100$ branching candidates are selected for exact testing to obtain the exact score for each candidate. The generation process for most instances can be completed within 6 hours, although some instances with long route sizes may take over 24 hours because of long pricing times. After getting the raw data, one necessary step is to convert the exact score to the label using Equation (6), where $\alpha = 0.8$ and $W = 5$ are used in training *M1* for the CVRP. After preprocessing the raw data obtained from the generation process, it is input into the training process. Our preliminary results suggested that tuning hyperparameters, except for *max_depth* in XGBoost, did not significantly improve accuracy, and thus, for training *M1* for CVRP, we set the *max_depth* parameter to 7 (default: 6), while other hyperparameters remained unchanged. Under such a choice of hyperparameters, the training time was only 16.2 seconds.

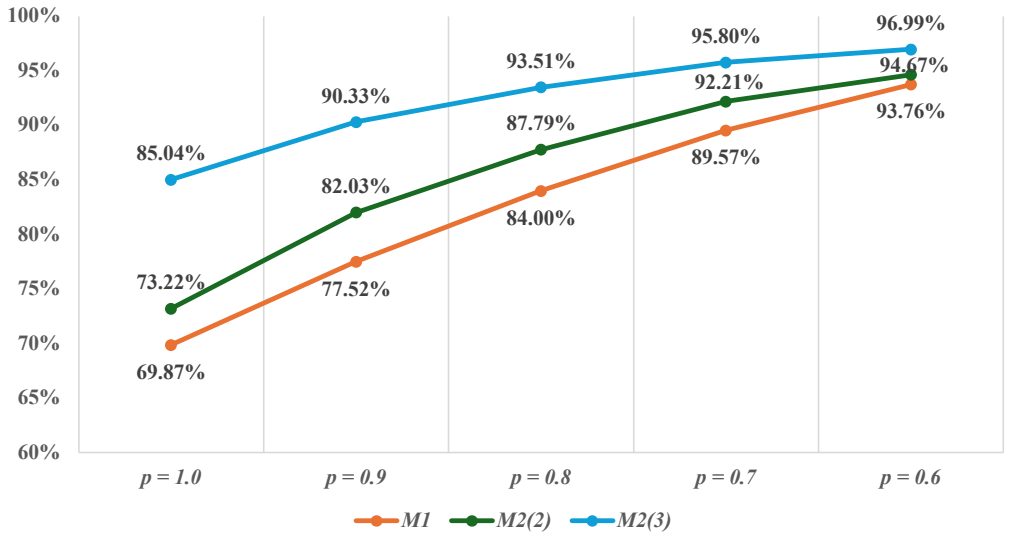
In the second stage, we need to incorporate the trained *M1* into RouteOpt. When branching, we obtain the set $\widetilde{C}$ ($|\widetilde{C}| = 3$) based on the output of *M1*. Each starting child LP of $e \in \widetilde{C}$ then undergoes exact testing, and the corresponding feature vector is obtained for each starting child LP. Since only 6 LPs are solved by exact CG in each branching decision, data generation for most instances is completed within 2 hours, and for all instances, it is finished within 6 hours. As each group only has 6 feature vectors, we use a smaller W in Equation (6), choosing 2. We apply the same hyperparameters used in training for *M1*, and the training time was only 2.8 seconds because of the much smaller dataset compared to the one used in the first stage.

The p-accuracy is plotted in Figure EC.6. For *M1*, the reported p-accuracy corresponds to the scenario where 10 candidates are selected out of 100, i.e., $n' = 10$ and $n = 100$ in the definition of p-accuracy. As observed, the accuracy of directly predicting the best candidate reaches nearly 70%, and increases to 84% when the threshold is lowered to 0.8 of the highest score. This underscores *M1*'s efficacy in significantly reducing the size of the candidate pool obtained from initial screening.

Additionally, we report two p-accuracies for *M2*, differing in their number of selections with $n' = 2$ and $n' = 3$, while maintaining $n = 6$ for both cases. The choice of 2 is because each candidate has 2 starting child LPs, and 3 serves as an extension to assess how the model performs with more

tolerance. As depicted in the figure, the 1.0-accuracies for these settings exceed that of *M1*, reaching 73% and 85%, respectively, and increase to 88% and 94% when $p = 0.8$. These high accuracies strongly support the assumptions outlined in Section 4.4.

Figure EC.6 Summary of p-accuracy of *M1* and *M2*.

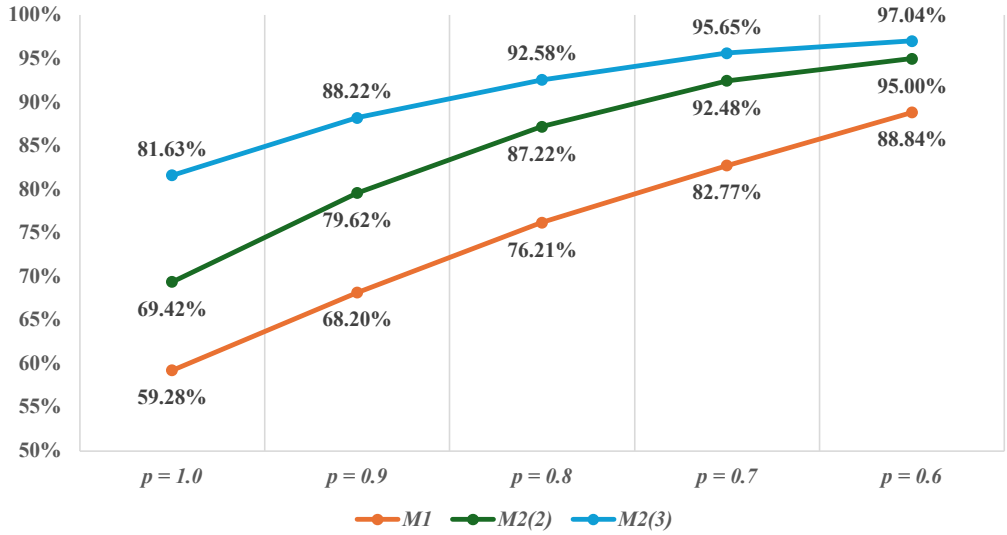


EC.9.2. Training Details for VRPTW

The training data is generated by solving instances in batch VRPTW-180-600-0, conducted on HiPerGator. The settings for generating the data, including time limits, parameters, workflow, and training hyperparameters, are identical to those used in CVRP to ensure the consistency of our learning methodology. The training time also remains consistent with CVRP, with 21.2 seconds for the first stage and 2.5 seconds for the second stage.

According to Figure EC.7, we obtained similar accuracy curves in CVRP for *M1* and *M2*. Specifically, the 1.0-accuracy reaches nearly 60%, and it increases to 76% when the threshold is lowered to 0.8 of the highest score. We note that there is an obvious 10% accuracy gap between VRPTW and CVRP, which indicates that the features employed may not be enough for VRPTW, and more features highlighting the differences between VRPTW and CVRP may be needed in future studies. Nevertheless, this accuracy gap does not lead to a significant performance gap as demonstrated by results in Section 6.1.2 and Section 6.2.2, where we found that 2LBB consistently achieves 35% to 40% time reduction for both CVRP and VRPTW case studies. This suggests that an 0.8-accuracy around 75% is good enough for the first stage. As for the second stage, while the 1.0-accuracy is somewhat lower compared with CVRP, for other p values, the accuracy remains consistent, with an 0.8-accuracy of around 87% and 93%.

Figure EC.7 Summary of p-accuracy of *M1* and *M2*.



EC.10. Detailed Experimental Results

The comprehensive experimental results for the two case studies are included in Tables EC.10 - EC.16. For the 3PB, 2LBB, 3PB-dy, and 2LBB-dy, we list their time for solving the root node (Root/s), the rounded-up root node lower bound (LB), the input upper bound (or cutoff, if not valid) (UB), LP testing time (LP/s), heuristic testing time (Heur/s), the overall solution time (CPU/s), and the number of nodes explored (Node). Furthermore, for the established benchmark instances, the best known solution (BKS) is reported and if the UB matches the BKS, it is highlighted in bold. The results of the VRPSolver are presented in Tables EC.13. Additionally, the geometric mean, denoted as G.M., for each group of results is provided in the last row of the respective tables (all zero values are excluded from the geometric mean calculation).

EC.10.1. Detailed Results of CVRP-150-90-1 (UB=u^*)

Table EC.10: Comparison of 3PB, 2LBB, 3PB-dy and 2LBB-dy for CVRP batch: CVRP-150-90-1 (UB=u^*)

Instance	Root	LB	UB	3PB				2LBB				3PB-dy				2LBB-dy			
				LP	Heur	CPU	Node	LP	Heur	CPU	Node	LP	Heur	CPU	Node	LP	Heur	CPU	Node
XML150_1142_01	11.0	28320	28385	467.7	247.1	928.4	989	416.7	56.0	649.5	891	353.1	252.2	840.1	1185	167.0	30.3	443.2	1065
XML150_1142_02	15.5	34700	34756	438.0	261.9	1096.8	1099	387.5	58.4	786.9	925	314.4	215.1	868.1	985	176.2	38.7	629.9	1137
XML150_1142_03	24.7	28454	28480	29.7	17.3	84.8	43	18.9	4.5	55.7	27	26.4	10.5	69.9	27	22.9	3.5	60.2	31
XML150_1142_04	11.4	28015	28089	648.3	376.2	1310.2	1399	520.9	85.1	841.1	1147	433.4	314.4	1023.1	1337	195.1	38.4	589.4	1637
XML150_1142_05	23.3	28639	28695	184.9	120.5	444.0	249	140.0	22.6	275.5	193	94.7	82.2	287.1	193	64.5	18.2	202.2	207
XML150_1142_06	18.4	23511	23566	289.7	199.8	703.2	561	162.6	27.7	305.0	301	152.5	143.2	459.5	439	83.9	20.5	257.1	399
XML150_1142_07	22.6	33799	33871	589.5	406.2	1403.6	707	402.2	75.8	779.2	469	494.0	422.2	1377.3	795	236.7	65.7	754.3	751
XML150_1142_08	37.1	34875	34925	279.7	176.6	650.4	239	148.3	22.5	289.8	129	123.4	88.8	339.8	141	84.6	18.2	244.6	161
XML150_1142_09	23.5	32595	32633	80.8	53.5	251.7	123	53.7	13.6	151.3	83	66.0	44.2	210.3	103	53.3	13.3	195.4	141
XML150_1142_10	26.8	25677	25752	840.0	595.3	1981.7	1027	882.4	161.1	1584.2	1017	579.3	549.3	1678.4	997	279.9	72.0	998.9	1139
XML150_1242_01	21.6	14421	14447	985.9	664.2	2367.4	1419	472.6	84.5	922.7	757	399.7	389.7	1241.3	913	267.7	53.7	936.6	1199
XML150_1242_02	46.7	34548	34564	171.4	95.3	384.4	119	155.1	27.1	291.7	107	112.5	58.7	268.8	77	91.0	18.9	207.8	81
XML150_1242_03	12.8	23199	23236	205.0	118.9	413.8	411	105.6	18.2	174.8	207	77.5	59.3	194.9	243	53.2	13.0	139.6	293
XML150_1242_04	14.2	25096	25157	998.1	679.7	2275.8	2587	798.3	156.3	1439.0	2107	586.1	563.2	1719.0	2463	362.9	96.7	1149.1	2873
XML150_1242_05	23.6	20973	21003	144.2	85.8	340.1	171	108.2	16.0	208.6	133	104.3	72.6	278.5	163	71.1	14.5	197.1	179
XML150_1242_06	12.2	26771	26783	31.4	16.8	68.3	61	57.6	10.0	95.3	105	42.0	22.6	88.2	79	29.0	5.1	60.1	89

Continued on next page

Table EC.10 – *Continued from previous page*

Instance	Root	LB	UB	3PB				2LBB				3PB-dy				2LBB-dy			
				LP	Heur	CPU	Node	LP	Heur	CPU	Node	LP	Heur	CPU	Node	LP	Heur	CPU	Node
XML150_1242_07	17.4	21098	21112	43.4	22.9	98.9	79	38.3	6.5	72.8	79	28.3	14.8	67.4	59	22.0	3.9	51.0	59
XML150_1242_08	10.3	17842	17861	54.4	27.6	107.1	133	37.2	5.3	62.7	87	25.8	13.2	56.5	67	22.0	3.6	45.9	85
XML150_1242_09	13.0	29991	30040	976.6	535.2	1882.6	1635	815.7	116.0	1219.4	1387	421.4	298.8	955.3	1081	289.9	60.7	855.1	2089
XML150_1242_10	22.3	48991	49031	238.9	160.0	595.3	355	151.5	23.6	313.7	225	136.8	115.7	422.8	285	118.1	30.1	395.9	405
XML150_1342_01	15.1	32111	32184	745.9	502.5	1646.4	1353	498.4	87.9	822.2	845	420.3	368.6	1088.9	1067	200.7	59.2	622.5	1201
XML150_1342_02	15.6	39413	39454	142.7	95.6	321.9	259	139.7	26.8	248.3	251	100.9	86.5	266.4	239	56.9	16.0	153.7	235
XML150_1342_03	41.2	22667	22691	54.8	38.1	189.4	59	56.6	13.8	161.4	53	49.5	22.1	146.4	35	37.2	6.6	108.5	25
XML150_1342_04	14.5	31654	31707	80.8	49.8	188.0	155	51.4	8.2	100.0	97	49.1	34.2	130.8	123	33.2	7.3	87.3	119
XML150_1342_05	24.7	24258	24294	259.5	162.6	597.8	333	193.3	32.6	346.7	253	167.7	135.8	451.2	271	111.8	28.6	302.4	333
XML150_1342_06	12.8	23825	23881	339.2	186.2	683.2	617	213.6	29.8	345.6	379	154.6	121.8	397.8	467	90.3	18.0	254.2	541
XML150_1342_07	35.5	34050	34075	86.7	51.2	291.7	107	56.9	11.4	186.5	69	52.4	29.3	186.6	59	42.6	8.8	165.2	59
XML150_1342_08	15.9	38752	38828	569.8	379.8	1309.2	1019	402.6	76.4	723.5	683	459.6	375.7	1212.9	1097	161.4	49.3	580.9	989
XML150_1342_09	17.7	34435	34521	1207.2	814.3	2750.8	1997	1076.9	208.4	1857.5	1615	711.8	597.3	1880.6	1611	258.8	61.3	880.5	1465
XML150_1342_10	11.8	35572	35594	62.2	44.0	149.6	143	34.3	4.8	68.4	83	43.9	37.3	120.0	125	27.2	5.8	62.6	83
XML150_2142_01	34.0	22653	22716	1382.9	784.9	3398.1	1457	1241.0	195.2	2471.9	1225	881.0	760.4	2820.7	1441	568.6	144.5	2208.8	1713
XML150_2142_02	68.4	20074	20104	149.4	112.0	600.4	93	136.0	29.6	458.6	79	156.8	123.8	592.9	87	123.6	29.0	487.4	103
XML150_2142_03	16.9	24818	24864	171.5	105.3	370.8	247	128.7	20.4	221.4	183	88.3	68.1	229.7	195	67.7	13.6	189.8	271
XML150_2142_04	151.2	20838	20855	89.5	65.5	484.4	35	83.2	24.8	452.3	31	108.4	45.6	445.8	25	93.1	11.1	392.5	25
XML150_2142_05	27.3	24428	24490	271.6	182.1	645.4	209	302.5	58.2	564.8	223	168.8	165.5	526.9	205	120.0	33.9	366.6	235
XML150_2142_06	21.0	23494	23529	109.4	61.4	239.1	149	85.5	15.2	156.9	115	74.0	45.4	177.6	119	43.9	7.6	109.5	109
XML150_2142_07	21.8	24460	24496	99.2	61.1	239.7	151	90.6	15.0	181.3	139	80.9	58.9	221.1	151	42.3	7.8	112.5	111
XML150_2142_08	10.9	29058	29102	161.1	85.8	312.5	371	127.8	15.0	196.6	281	69.2	47.1	162.7	237	56.1	8.8	125.0	313
XML150_2142_09	35.3	23745	23772	27.8	19.4	111.0	35	23.7	7.0	83.6	27	48.3	19.9	134.4	35	31.3	5.9	98.3	31
XML150_2142_10	35.0	20589	20620	119.4	70.0	308.2	115	78.8	13.5	183.9	79	84.4	52.8	243.7	97	61.1	11.1	178.0	93
XML150_2242_01	18.7	17103	17126	186.5	120.0	412.5	271	118.0	19.5	206.1	167	101.4	87.1	271.9	213	81.6	14.4	189.6	215
XML150_2242_02	24.6	16487	16499	51.0	30.0	133.6	83	39.9	5.4	94.4	65	48.5	25.9	126.7	73	39.0	7.0	99.9	75
XML150_2242_03	11.2	23788	23835	631.6	350.2	1206.7	1341	558.0	86.3	834.6	1181	324.4	228.2	720.7	1037	250.3	49.7	619.4	1737
XML150_2242_04	29.6	17585	17618	1335.9	933.4	3217.8	1681	803.9	141.7	1496.8	963	769.8	653.1	2084.9	1243	405.2	86.2	1302.1	1401
XML150_2242_05	17.4	22916	22929	44.0	32.6	118.3	101	30.5	6.8	68.7	69	39.8	24.9	101.0	81	27.9	7.9	69.2	67
XML150_2242_06	23.3	16225	16240	74.6	47.6	193.2	119	55.0	13.1	123.7	85	54.1	30.3	137.0	77	44.4	11.9	116.5	91
XML150_2242_07	20.3	22465	22501	441.5	256.7	942.3	613	364.0	51.7	630.4	531	352.5	274.7	921.6	689	179.2	38.4	501.2	673
XML150_2242_08	22.2	20679	20694	49.2	31.3	129.4	75	30.5	4.7	71.9	43	38.7	21.2	98.8	53	30.3	7.0	75.7	47
XML150_2242_09	20.9	23305	23337	35.1	19.0	88.7	37	22.0	3.8	55.0	23	34.0	16.4	83.1	31	29.9	1.7	66.0	33
XML150_2242_10	30.9	16911	16927	55.3	35.4	177.2	51	27.4	2.8	87.4	27	37.8	16.7	109.4	25	51.6	8.6	139.4	51
XML150_2342_01	45.3	19714	19743	105.5	60.0	274.4	69	74.8	8.9	176.2	49	83.6	41.5	213.0	51	66.2	11.7	176.3	53
XML150_2342_02	116.6	21331	21374	680.5	592.4	2428.6	201	524.6	101.9	1459.9	157	422.4	365.1	1553.7	131	375.6	116.7	1469.8	191
XML150_2342_03	8.6	28395	28471	1346.9	794.1	2767.3	3187	1256.9	233.4	2063.4	3013	543.9	473.5	1457.0	2205	446.3	134.4	1350.1	3615
XML150_2342_04	11.8	28883	28948	154.1	85.7	324.3	367	94.3	14.7	156.3	227	68.2	50.3	178.5	259	43.3	8.2	118.1	301
XML150_2342_05	10.6	25545	25613	800.3	464.9	1588.0	1639	732.5	124.7	1167.2	1479	497.5	417.5	1256.8	1729	347.7	77.8	983.5	2473
XML150_2342_06	14.5	20893	20914	21.5	12.1	56.5	49	12.3	1.8	33.6	25	20.2	8.7	49.8	33	15.2	2.1	37.0	29
XML150_2342_07	74.8	19936	19978	678.2	487.7	2175.1	515	382.2	62.0	1049.5	283	354.3	311.3	1391.8	345	346.7	80.3	1337.2	467
XML150_2342_08	18.5	25060	25139	1044.2	650.1	2353.0	1373	1094.6	173.4	1981.7	1417	835.4	702.6	2417.0	1733	548.4	130.1	1933.0	2295
XML150_2342_09	33.2	21543	21584	211.9	136.1	571.9	183	243.9	40.5	506.8	177	252.3	227.6	829.6	297	194.8	49.0	711.4	373
XML150_2342_10	25.8	25658	25692	60.3	36.3	166.3	69	43.9	5.4	103.4	49	53.3	36.4	154.3	65	41.4	7.9	116.9	65
XML150_3142_01	33.0	39638	39696	152.5	94.6	427.1	201	127.2	23.1	302.3	163	103.2	78.6	335.7	175	83.1	16.5	298.0	223
XML150_3142_02	12.1	53175	53275	1443.3	947.4	3164.1	3903	1422.7	266.6	2476.8	3863	834.0	858.7	2533.3	4295	533.9	162.1	2185.6	6763
XML150_3142_03	123.3	34951	34988	152.8	97.2	706.5	89	81.9	16.1	473.0	51	130.5	72.0	607.3	57	87.7	15.6	516.4	47
XML150_3142_04	21.1	45656	45773	1192.2	827.6	2872.7	1929	1279.1	273.9	2431.2	1917	709.1	601.7	2015.6	1587	322.6	98.9	1306.6	1889
XML150_3142_05	30.4	41377	41425	330.5	261.5	933.8	387	284.1	58.1	641.2	309	254.5	249.5	861.9	367	246.1	79.3	882.6	557
XML150_3142_06	38.5	37305	37332	70.1	41.3	209.2	73	69.6	14.5	168.5	59	86.4	58.8	258.3	87	57.1	11.2	162.1	67
XML150_3142_07	19.8	42000	42049	82.8	52.0	205.9	141	66.2	13.5	136.4	107	61.8	45.2	175.7	131	38.3	8.1	106.4	111
XML150_3142_08	25.1	39124	39169	332.0	246.1	869.8	613	178.1	28.7	375.1	329	210.4	198.9	670.4	525	99.2	23.4	350.3	453
XML150_3142_09	18.9	47685	47754	840.3	548.4	1834.9	1565	716.1	121.5	1220.4	1323	581.0	486.8	1509.1	1527	323.2	77.5	972.2	1855
XML150_3142_10	33.9	40293	40330	67.3	45.6	208.1	65	57.6	12.2	148.7	53	63.7	40.9	186.4	59	41.7	6.9	120.9	45
XML150_3242_01	13.7	39478	39529	606.9	394.0	1316.7	1079	493.2	76.0	808.8	847	324.6	225.0	769.6	739	245.4	55.7	717.7	1333
XML150_3242_02	42.1	44790	44800	84.2	57.8	261.7	89	50.1	12.3	138.2	57	60.4	35.2	170.8	55	57.1	10.3	152.4	73
XML150_3242_03	60.3	37023	37078	617.2	455.7	1961.6	405	450.1	63.6	1158.6	289	360.0	296.5	1333.2	321	219.4	53.5	951.3	305
XML150_3242_04	26.2	39509	39539	186.3	119.0	503.0	211	129.6	19.3	256.9	135	95.3	70.1	277.8	133	66.8	16.1	199.2	135
XML150_3242_05	19.6	45528	45577	1427.8	863.8	2999.4	1579	1468.5	229.9	2414.5	1707	963.8	815.7	2517.6	1837	779.8	197.2	2349.2	2891
XML150_3242_06	23.9	44682	44702	578.0	365.8	1304.6	717	479.5	72.2	844.3	595	489.6	399.3	1268.2	789	262.2	50.0	731.8	867
XML150_3242_07	53.4	26367	26381	174.9	164.0	1011.3	265	136.9	30.6	687.0	209	223.0	177.8	1013.7	289	147.3	31.4	777.2	267
XML150_3242_08	29.0	29294	29318	88.8	62.9	247.4	111	76.1	14.7	174.5	97	89.6	73.1	262.4	123	65.7	15.5	194.5	139
XML150_3242_09	76.5	26996	27016	223.3	146.0	639.4	127	158.9	22.1	376.3	85	160.1	108.9	476.7	83	137.1	31.4	425.7	107
XML150_3242_10	17.2	43766	43796	407.7	269.4	944.4	741	275.7	47.0	506.9	491	178.3	167.3	521.7	497	106.3	24.1	346.1	553
XML150_3342_01	30.1	42318	42354	104.2	60.5	273.4	145	59.5	9.3	149.4	81	60.3	35.9	180.3	93	47.2	6.8	140.4	87
XML150_3342_02	13.4	52704	52806	428.0	280.1	938.6	891	389.5	66.5	667.1	803	340.4	262.3	840.5	931	149.3	37.9	475.7	1045
XML150_3342_03	83.1	32614	32637	124.0	101.6	652.6	107	108.2	25.1	460.5	73	159.8	131.7	636.8	95	93.8	14.5	392.1	69
XML150_3342_04	75.1	43903	43945	51.1	34.2	410.4	89	42.1	9.0	311.8	69	59.1	27.6	323.1	67	47.8	5.5	316.0	67
XML150_3342_05	24.9	44809	44874	1020.5	689.9	2402.1	1413	978.5	183.6	1813.9	1419	762.6	775.6	2348.8	1721	457.7	135.6	1462.5	1785
XML150_3342_06	23.3	40117	40139	111.6	70.6	280.5	153	72.5	15.2	155.9	95	77.0	54.5	213.6	125	57.6	10.3	142.1	103
XML150_3342_07	66.3	37225	37293	852.3	739.0	2937.3	683	676.2	130.6	1825.6	539	517.6	603.6	2170.3	575	327.7	99.3	1465.0	565
XML150_3342_08	15.4	45874	45939	814.5	548.1	1890.1	1745	772.8	151.2	1395.8	1603	564.6	483.1	1534.6	1725	323.2	91.4	1063.4	2185
XML150_3342_09	61.1	39240	39261	35.1	24.9	170.4	33	17.7	4.7	108.0	17	35.8	13.4	133.3	17	31.2	3.9	118.8	17
XML150_3342_10	13.4	45225	45311	964.8	619.2	2104.8	2241	563.1	98.5	969.3	1211	421.5	410.9	1222.8	1625	266.7	74.8	853.4	1935
G.M.	24.5	-	-	218.1	139.8	570.2	297.9	166.2	29.2	362.5	222.1	152.0	108.9	436.8	245.5	102.9	21.8	317.2	270.1

EC.10.2. Detailed Results of CVRP-150-90-1 (UB=u^0)

Table EC.11: Comparison of 3PB, 2LBB, 3PB-dy and 2LBB-dy for CVRP batch: CVRP-150-90-1 (UB=u^0)

Instance	Root	LB	UB	3PB				2LBB				3PB-dy				2LBB-dy			
				LP	Heur	CPU	Node	LP	Heur	CPU	Node	LP	Heur	CPU	Node	LP	Heur	CPU	Node
XML150_1142_01	10.2	28314	28413	878.0	488.2	1694.9	2097	686.5	103.4	1008.1	1565	493.0	254.9	926.4	1193	282.5	66.7	600.6	1603
XML150_1142_02	19.0	34702	34791	531.6	335.5	1357.7	1237	396.1	70.9	765.7	933	669.9	264.6	1268.7	993	581.3	119.6	1094.2	1509
XML150_1142_03	25.2	28453	28508	55.4	30.1	137.8	77	29.3	5.9	73.7	37	68.0	22.5	129.0	53	45.0	10.4	94.6	43
XML150_1142_04	11.7	28014	28117	745.9	453.7	1496.1	1511	910.7	165.1	1273.9	1853	782.2	439.3	1469.0	1451	462.2	135.3	851.9	1697
XML150_1142_05	24.8	28639	28724	414.7	316.7	961.2	515	422.7	86.6	659.8	499	323.7	219.7	684.5	375	185.2	43.9	353.1	331
XML150_1142_06	18.2	23509	23590	332.2	245.5	844.3	541	259.8	49.8	461.8	393	301.1	218.6	729.8	491	187.0	44.9	420.4	525
XML150_1142_07	21.6	33796	33905	1197.2	836.7	2480.7	1581	1139.0	313.8	1825.2	1447	974.0	637.9	1949.0	1281	548.8	201.7	1199.2	1105
XML150_1142_08	41.9	34872	34960	350.9	234.7	856.6	299	511.1	129.4	837.6	419	647.7	366.0	1242.3	425	321.6	81.5	596.6	323
XML150_1142_09	29.1	32593	32666	83.9	55.1	277.2	119	181.8	47.0	331.6	221	84.4	43.3	230.8	87	78.6	16.6	186.1	101
XML150_1142_10	30.2	25679	25778	1079.2	780.4	2363.7	1009	1557.6	343.7	2377.2	1429	997.8	686.2	2139.8	941	724.8	251.8	1387.6	1211
XML150_1242_01	19.8	14420	14461	1002.3	696.6	2401.5	1561	1308.5	328.2	2125.3	1915	1256.1	695.4	2367.8	1595	638.0	195.6	1295.3	1509
XML150_1242_02	34.4	34545	34599	196.2	112.1	434.3	161	201.4	46.1	348.0	157	415.6	171.5	711.1	185	308.6	41.1	444.6	155
XML150_1242_03	13.6	23199	23259	348.9	215.4	658.3	637	402.5	92.7	573.9	693	349.3	193.2	612.8	615	240.3	63.0	378.7	635
XML150_1242_04	15.2	25095	25182	1295.8	896.7	2764.4	3201	1617.6	429.3	2556.1	3935	1382.9	816.1	2770.4	2975	1074.6	360.0	1911.2	3541
XML150_1242_05	19.7	20971	21024	750.9	501.7	1506.4	925	637.2	147.9	923.6	763	825.0	354.0	1333.0	615	404.4	103.8	619.0	567
XML150_1242_06	13.4	26771	26810	82.2	49.3	165.4	113	104.7	18.0	155.7	137	260.7	66.0	365.5	125	243.2	29.7	305.1	167
XML150_1242_07	16.3	21096	21133	37.6	20.5	93.3	63	68.4	10.8	112.9	111	99.4	40.8	176.6	115	59.5	10.3	100.0	83
XML150_1242_08	10.5	17840	17879	85.7	45.8	166.4	169	85.9	16.4	127.1	163	87.7	34.4	148.1	131	59.9	10.1	96.8	139
XML150_1242_09	13.6	29990	30070	1747.1	1032.2	3108.2	3141	1196.8	271.3	1703.9	2035	1404.0	481.4	2089.7	1569	903.0	209.3	1330.8	2013
XML150_1242_10	22.3	48990	49080	372.2	271.1	896.1	551	217.0	56.0	476.5	303	704.1	220.6	1155.8	435	467.7	63.0	723.5	445
XML150_1342_01	16.1	32112	32216	1023.8	718.4	2168.3	1831	758.5	170.3	1216.5	1309	917.2	501.4	1715.2	1455	597.6	150.8	1020.8	1393
XML150_1342_02	13.9	39411	39493	149.3	102.0	324.0	261	134.3	33.0	252.8	223	198.3	94.3	365.2	219	148.1	36.9	256.0	305
XML150_1342_03	35.2	22664	22714	100.5	68.4	309.1	127	104.5	24.1	246.4	121	144.7	59.9	291.1	81	58.0	10.3	151.9	59
XML150_1342_04	14.9	31657	31739	82.9	46.9	174.9	147	65.6	13.6	118.8	117	80.4	40.9	167.3	127	44.4	11.0	94.1	93
XML150_1342_05	24.2	24258	24318	455.2	336.8	1037.7	555	260.6	48.5	498.1	319	475.3	236.7	901.9	343	279.8	60.9	518.9	409
XML150_1342_06	11.9	23824	23905	331.4	190.4	661.9	589	320.2	52.8	488.7	571	211.9	147.1	485.4	489	132.6	28.2	300.1	527
XML150_1342_07	34.2	34049	34109	230.3	185.0	687.5	285	251.6	66.1	566.8	291	474.6	182.0	966.1	267	270.0	63.3	637.6	241
XML150_1342_08	15.8	38751	38867	972.8	694.6	2053.0	1841	898.8	226.0	1420.7	1615	795.9	425.7	1510.2	1215	510.0	144.5	933.9	1411
XML150_1342_09	18.3	34433	34556	1971.8	1379.2	4371.7	3123	2924.0	756.0	4609.1	4343	1527.2	808.8	2864.4	1845	623.7	165.0	1718.5	2123
XML150_1342_10	13.9	35571	35630	115.0	76.4	248.4	213	63.6	17.6	116.5	117	223.7	63.1	332.9	161	184.5	25.6	252.8	187
XML150_2142_01	36.5	22656	22739	1399.2	889.7	3509.9	1393	1302.8	231.4	2643.7	1297	924.0	726.7	2847.9	1339	1655.8	492.6	3437.7	2717
XML150_2142_02	49.0	20071	20124	159.4	118.1	719.9	107	204.8	46.6	610.0	131	234.3	132.8	674.4	105	187.0	44.6	546.9	107
XML150_2142_03	16.1	24815	24889	303.4	197.6	619.6	425	400.5	78.2	589.6	535	391.1	217.3	707.8	501	243.0	57.1	408.1	581
XML150_2142_04	166.1	20836	20876	144.2	96.3	759.4	65	126.0	20.5	558.6	59	235.0	76.7	732.9	47	172.8	26.1	616.9	41
XML150_2142_05	22.6	24430	24514	581.5	406.2	1256.6	465	342.3	65.9	562.6	271	345.3	250.6	797.3	287	221.3	72.9	485.3	311
XML150_2142_06	20.6	23493	23553	207.5	126.1	440.1	269	195.7	40.4	299.0	231	106.2	61.1	235.5	139	100.6	22.7	195.5	163
XML150_2142_07	17.9	24458	24520	119.6	69.0	275.8	203	125.0	21.2	247.7	207	95.9	59.1	240.8	171	59.8	11.1	150.9	163
XML150_2142_08	9.4	29059	29131	234.6	124.9	421.5	557	250.1	44.7	356.4	571	329.8	126.2	507.7	545	132.0	24.2	217.9	439
XML150_2142_09	33.7	23744	23796	31.8	20.2	115.7	33	30.1	8.0	98.3	31	51.2	22.8	148.1	33	40.8	5.0	99.3	23
XML150_2142_10	35.9	20585	20641	262.8	182.2	652.1	215	105.3	17.0	241.2	87	158.2	82.0	374.0	105	150.2	29.2	308.5	139
XML150_2242_01	17.7	17102	17143	291.8	183.2	592.3	367	247.2	49.1	378.3	305	277.5	139.8	509.0	293	206.1	36.3	317.5	309
XML150_2242_02	23.3	16486	16515	116.4	72.1	257.0	161	107.7	21.0	205.5	143	147.4	68.7	284.8	141	83.1	11.5	157.3	95
XML150_2242_03	10.5	23788	23859	811.7	456.6	1530.4	1703	774.0	159.0	1096.8	1613	563.1	300.5	1050.8	1243	262.0	49.2	627.0	1617
XML150_2242_04	26.9	17585	17636	1606.7	1010.6	3382.2	1499	1087.5	226.0	1927.0	1065	1447.9	625.0	2587.2	1009	964.2	232.2	1829.5	1435
XML150_2242_05	15.5	22915	22952	90.4	66.6	209.3	187	73.8	21.9	136.3	135	221.0	65.8	336.7	161	140.5	24.5	208.3	167
XML150_2242_06	28.4	16225	16256	108.1	68.5	273.2	137	137.3	34.2	244.3	177	96.5	49.8	210.3	99	66.6	13.8	149.0	97
XML150_2242_07	18.7	22463	22524	838.1	462.5	1603.3	1003	1005.9	173.8	1456.0	1223	643.9	305.6	1191.8	779	465.0	85.1	880.6	1027
XML150_2242_08	20.8	20679	20715	74.4	45.8	161.9	125	31.0	5.5	78.7	49	64.5	20.4	120.2	51	59.2	11.2	106.0	63
XML150_2242_09	25.3	23304	23360	121.1	80.6	248.8	95	107.3	26.1	171.7	83	119.1	48.8	206.9	57	72.1	12.5	121.9	53
XML150_2242_10	26.8	16910	16944	48.2	31.4	165.5	47	25.6	4.5	84.2	25	52.7	24.0	142.9	33	74.2	17.6	162.8	65
XML150_2342_01	50.3	19713	19763	96.1	54.6	265.7	63	86.1	12.6	195.6	55	130.1	59.2	301.8	65	89.6	13.7	205.3	51
XML150_2342_02	109.7	21330	21395	725.0	602.7	2203.3	271	901.6	231.5	1869.6	291	601.7	426.3	1661.2	191	344.6	90.8	1080.8	171
XML150_2342_03	9.9	28398	28499	1241.5	761.9	2495.2	2741	1410.0	325.7	2177.6	3073	766.0	498.1	1608.6	1955	549.7	162.7	1124.0	2127
XML150_2342_04	14.4	28884	28977	232.1	125.0	446.7	455	174.8	27.6	264.1	329	113.8	57.6	230.8	219	112.5	26.8	198.2	343
XML150_2342_05	11.4	25543	25639	1626.0	951.6	3122.4	3227	1012.8	199.4	1610.5	1997	1131.0	627.6	2121.6	2311	599.4	165.9	1208.3	2615
XML150_2342_06	12.6	20890	20935	55.4	32.1	115.3	115	51.9	15.4	90.5	103	69.8	24.4	116.7	89	43.5	6.7	74.0	79
XML150_2342_07	56.3	19935	19998	616.5	452.7	1936.8	483	420.7	81.0	988.9	331	563.7	296.1	1349.1	299	285.3	60.5	747.7	275
XML150_2342_08	21.6	25065	25164	1603.7	985.8	3481.2	2087	869.8	154.2	1522.2	1077	1464.2	955.9	2920.5	2013	853.3	251.2	1591.3	2075
XML150_2342_09	32.1	21543	21606	288.1	199.4	814.7	219	314.4	66.7	560.6	225	620.4	329.9	1224.9	339	448.8	108.1	939.4	345
XML150_2342_10	20.8	25653	25718	126.3	81.6	282.4	161	145.0	36.6	240.1	175	171.9	101.0	360.4	169	85.7	22.7	177.0	119
XML150_3142_01	34.3	39636	39736	300.7	222.7	791.2	415	155.3	37.8	391.9	217	156.3	100.9	453.9	209	176.6	44.3	413.1	257
XML150_3142_02	12.1	53175	53328	3554.7	2666.4	7881.4	11639	3881.3	1095.0	6798.8	12507	2267.0	1111.7	4372.0	5357	2013.8	592.1	3762.6	7899
XML150_3142_03	123.9	34950	35023	402.8	302.5	1396.7	227	183.9	45.6	613.1	109	498.3	238.3	1189.9	143	285.7	54.5	771.1	129
XML150_3142_04	24.8	45661	45819	1427.4	1025.7	3265.2	1877	1042.1	299.2	1946.5	1371	1146.5	719.2	2506.6	1421	744.2	286.4	1663.8	1725
XML150_3142_05	33.3	41379	41466	518.9	408.2	1276.5	597	395.6	84.0	744.9	425	423.6	179.5	815.1	263	335.0	67.3	610.6	321
XML150_3142_06	38.1	37298	37369	177.7	131.8	542.0	157	129.9	36.9	317.1	93	161.9	94.3	428.4	123	90.4	17.4	257.0	79
XML150_3142_07	20.3	42000	42091	88.6	56.0	227.5	141	65.2	18.4	148.5	103	85.2	42.8	192.1	101	84.5	19.7	173.6	143
XML150_3142_08	22.9	39119	39208	535.2	442.3	1292.8	1057	493.3	127.9	914.6	947	379.2	214.7	865.1	555	363.1	104.9	755.1	793
XML150_3142_09	19.1	47683	47802	1223.9	862.4	2630.6	2195	906.7	231.3	1536.3	1557	1249.7	513.9	2106.4	1373	1051.8	303.9	1711.5	2107
XML150_3142_10	31.4	40292	40370	96.4	59.9	269.8	91	63.8	13.5	149.3	59	141.5	64.9	292.2	89	105.0	22.6	211.2	81
XML150_3242_01	15.3	39477	39569	909.3	590.6	1805.4	1365	1323.4	328.7	2048.7	1873	1008.6	292.2	1447.7	669	952.7	159.0	1272.6	941

Continued on next page

Table EC.11 – *Continued from previous page*

Instance	Root	LB	UB	3PB				2LBB				3PB-dy				2LBB-dy			
				LP	Heur	CPU	Node	LP	Heur	CPU	Node	LP	Heur	CPU	Node	LP	Heur	CPU	Node
XML150_3242_02	43.8	44789	44845	106.9	70.6	396.7	89	130.2	35.6	302.7	99	188.5	92.5	444.6	83	241.6	19.5	394.4	91
XML150_3242_03	44.4	37019	37115	1271.8	877.9	3073.9	761	1402.2	320.5	2546.3	783	1762.4	702.2	3221.3	603	979.7	236.1	1896.1	615
XML150_3242_04	27.8	39507	39579	278.0	185.5	780.7	333	275.1	64.8	529.7	291	335.9	119.3	621.2	181	270.2	55.1	474.7	209
XML150_3242_05	19.4	45527	45623	6055.1	3508.0	10930.2	5737	3914.8	880.3	5681.4	3685	4888.8	1967.5	7652.4	3537	2832.3	400.1	4160.8	2549
XML150_3242_06	23.5	44681	44747	2132.5	1410.5	4275.4	2539	2409.7	571.2	3717.9	2745	3901.6	679.9	5158.9	1181	3975.0	383.8	4800.2	1517
XML150_3242_07	46.3	26366	26407	372.4	401.3	1866.5	561	421.5	176.7	1628.3	639	598.3	290.1	1547.6	389	871.5	165.4	1706.7	491
XML150_3242_08	25.9	29292	29347	295.7	268.4	738.9	387	199.3	50.8	364.7	243	376.4	187.8	728.8	257	181.2	42.9	357.0	205
XML150_3242_09	75.1	26995	27043	351.4	195.6	911.3	159	653.5	150.0	1065.0	259	932.7	346.6	1575.5	221	441.1	82.5	765.6	191
XML150_3242_10	18.2	43764	43840	738.5	559.0	1771.1	1235	465.8	125.9	901.5	725	770.4	275.9	1330.0	649	373.2	63.2	779.0	753
XML150_3342_01	32.0	42317	42396	184.8	114.8	435.0	219	160.7	38.5	297.2	175	242.0	109.3	450.9	191	184.9	41.7	329.7	191
XML150_3342_02	14.3	52702	52859	728.2	571.0	1632.5	1643	731.1	193.2	1296.1	1541	795.5	398.1	1467.0	1207	507.3	146.1	976.8	1473
XML150_3342_03	106.6	32613	32670	209.0	176.1	990.5	169	318.4	94.1	845.6	305	731.1	332.6	1584.0	301	393.6	71.2	827.9	269
XML150_3342_04	84.1	43902	43989	152.2	107.9	648.3	215	56.2	16.6	303.6	77	167.0	51.4	479.2	99	107.5	16.0	379.4	87
XML150_3342_05	19.6	44801	44919	1705.3	1344.4	3814.6	2797	1380.8	402.3	2505.3	2177	3114.7	954.7	4740.5	2111	1160.7	264.6	2015.8	1711
XML150_3342_06	22.7	40110	40179	340.0	250.6	855.3	427	350.2	88.5	634.4	419	355.9	121.9	619.9	215	300.8	51.7	502.3	277
XML150_3342_07	66.7	37227	37330	935.3	806.3	3021.9	731	532.5	156.9	1401.2	403	945.2	610.7	2425.6	547	658.1	163.8	1694.9	587
XML150_3342_08	16.2	45874	45985	1345.6	871.5	2849.3	2719	1068.8	214.8	1893.5	2097	1478.8	552.9	2525.0	1903	740.2	154.2	1461.5	1877
XML150_3342_09	53.6	39238	39300	84.6	56.7	255.7	69	64.0	21.8	175.4	49	55.0	14.3	150.8	19	110.3	23.1	225.7	49
XML150_3342_10	11.5	45222	45356	1352.6	992.1	3061.5	3283	1438.0	422.2	2267.7	3373	1542.2	761.3	2765.8	2657	1069.2	385.9	1901.5	3455
G.M.	24.2	-	-	356.3	238.1	869.0	463.8	317.8	71.9	603.7	398.3	390.2	181.3	782.2	347.6	269.2	57.6	528.1	372.1

EC.10.3. Detailed Results of CVRP-V-42-X

Table EC.12: Comparison of 3PB, 2LBB, 3PB-dy and 2LBB-dy for CVRP batch: CVRP-V-42-X

Instance	Root	LB	UB	BKS	3PB				2LBB				3PB-dy				2LBB-dy			
					LP	Heur	CPU	Node	LP	Heur	CPU	Node	LP	Heur	CPU	Node	LP	Heur	CPU	Node
X-n120-k6	211.3	13281	**13332**	13332	74.4	33.8	888.7	25	119.1	20.1	1277.6	29	136.7	25.8	785.0	15	162.4	13.3	1097.7	27
X-n134-k13	318.3	10880	**10916**	10916	188.6	27.9	2026.9	25	84.7	9.0	1417.0	13	98.4	11.9	1146.2	9	129.0	11.7	1339.3	11
X-n143-k7	334.6	15637	**15700**	15700	287.4	109.2	4257.0	37	228.6	71.9	3336.4	31	920.8	336.6	8498.2	83	263.6	58.9	2755.4	25
X-n186-k15	339.1	24053	24130	24145	89.1	56.2	2585.8	37	86.0	25.0	2440.9	29	282.7	72.1	2931.9	39	165.1	39.9	2421.9	33
X-n190-k8	1152.8	16957	16962	16980	27.5	23.4	2427.3	7	28.5	16.5	2500.0	5	128.7	61.9	2774.9	7	96.9	19.9	2560.2	7
X-n200-k36	43.1	58530	**58578**	58578	55.7	30.4	428.0	79	66.0	10.1	406.5	87	64.1	25.9	383.6	71	51.1	8.5	386.2	79
X-n209-k16	302.5	30579	30644	30656	123.3	68.6	2840.2	27	75.1	8.9	2078.6	19	257.3	79.5	2110.6	21	181.9	23.3	1996.1	19
X-n214-k11	770.8	10812	10830	10856	154.4	65.7	3404.1	19	115.5	27.8	2776.3	15	231.7	77.0	2960.8	17	204.8	35.3	2925.1	15
X-n228-k23	484.5	25689	**25742**	25742	306.2	297.6	6697.1	121	351.5	155.7	7471.8	145	334.9	126.3	2602.3	57	368.4	90.9	4008.8	81
X-n233-k16	396.8	19124	19170	19230	576.1	281.8	8716.5	69	663.5	126.2	6978.1	61	357.0	55.1	3309.5	21	412.0	55.0	3748.4	27
X-n242-k48	119.9	82634	**82751**	82751	405.8	189.1	2023.8	459	343.6	41.9	1482.9	331	256.0	149.4	1675.6	351	213.5	25.4	1444.9	317
X-n256-k16	662.3	18640	18692	18839	422.2	228.7	5726.7	39	509.0	129.0	5115.8	45	703.1	357.1	5360.5	41	658.8	120.8	4903.1	41
X-n261-k13	873.2	26461	26502	26558	398.8	353.0	7531.2	43	365.1	203.0	6208.3	31	727.9	417.7	7172.1	39	648.2	164.5	8236.0	41
X-n270-k35	170.8	35124	35220	35291	561.7	706.0	4002.9	209	794.2	283.7	3602.9	243	651.4	629.3	3114.8	183	607.5	220.3	2638.2	173
X-n280-k17	968.4	33302	33348	33503	962.8	623.2	12479.4	55	520.4	179.9	7442.1	31	842.6	342.9	8325.2	31	666.6	151.6	7794.1	29
X-n284-k15	1069.6	20163	20183	20215	208.1	169.6	4462.1	21	285.4	87.0	5303.2	29	476.7	291.6	4570.0	25	227.3	91.2	3191.5	11
X-n289-k60	197.8	94950	95063	95151	736.7	481.4	5556.8	405	1239.5	228.0	9009.1	407	756.8	668.6	6509.0	549	474.0	136.0	4261.5	349
X-n294-k50	93.7	46936	47058	47161	2004.7	1224.4	8057.6	1139	1670.2	347.3	5404.0	889	1590.1	1218.8	7711.4	1251	904.4	221.0	4508.6	837
X-n303-k21	752.2	21549	21601	21736	839.9	1020.3	9316.2	73	690.4	415.0	8102.8	65	546.9	800.8	4581.8	25	469.0	298.4	3968.9	25
X-n313-k71	114.3	93840	93956	94043	1166.6	767.5	8799.5	1587	843.9	178.8	5708.9	1015	783.1	740.4	7009.1	1575	432.9	95.7	3738.8	877
X-n322-k28	653.4	29727	29785	29834	161.6	159.2	4962.8	33	193.4	68.6	4917.1	35	342.5	174.0	3984.9	27	317.2	96.7	3947.9	29
X-n327-k20	724.9	27401	27431	27532	716.3	801.9	9488.2	55	223.1	45.5	3328.7	21	385.9	218.4	3716.2	23	327.1	126.1	3383.9	19
X-n331-k15	1405.5	31063	31085	31102	352.7	409.6	7835.5	31	326.9	116.2	6270.8	27	466.9	369.3	5924.3	21	401.3	91.6	5434.4	23
X-n336-k84	159.1	138773	138916	139111	352.0	337.1	3397.8	153	170.2	58.8	1629.5	71	350.7	351.0	2439.1	137	208.2	54.3	1679.8	81
X-n344-k43	220.9	41929	41993	42050	547.1	782.5	3172.6	151	404.7	119.9	1777.3	99	554.9	653.9	2885.2	131	684.6	332.8	3262.2	191
X-n351-k40	673.6	25819	25858	25896	679.0	693.3	10626.5	149	227.5	88.2	3924.8	43	474.2	291.5	4749.9	61	393.4	79.9	3922.7	51
X-n359-k29	564.2	51387	51435	51505	327.7	268.2	4160.0	51	527.6	188.9	5090.7	85	420.8	192.8	2859.4	35	377.6	117.5	2734.8	33
X-n384-k52	306.5	65776	65824	65928	298.5	535.9	1979.2	77	264.3	84.6	1259.6	57	285.6	355.4	1591.9	51	261.2	137.0	1282.2	53
X-n420-k130	72.9	107688	**107798**	107798	282.2	86.5	1117.2	417	192.8	10.5	654.0	275	145.7	43.5	630.2	227	125.1	9.6	595.5	267
X-n429-k61	321.3	65201	65267	65449	1279.2	2123.6	7147.7	405	856.7	398.2	3571.3	225	1089.9	1557.1	5530.6	303	895.9	396.0	3832.2	265
X-n439-k37	611.9	36312	36345	36391	430.3	593.3	5820.5	43	534.3	223.7	5868.3	47	752.9	783.8	5940.8	49	880.9	505.1	6071.8	63
X-n469-k138	30.7	221483	221664	221824	1472.0	510.4	4418.2	1989	836.1	70.7	2177.3	1021	581.8	295.3	2390.7	1271	572.9	36.4	2134.4	1285
X-n480-k70	304.2	89286	89334	89449	298.4	376.7	2154.7	73	336.2	154.9	1778.4	71	283.2	285.1	1645.7	45	268.7	76.7	1376.8	39
X-n502-k39	1378.9	69151	69172	69226	767.3	946.9	6470.4	67	453.4	206.6	3859.3	41	848.0	964.2	5269.7	43	582.7	601.6	4006.2	33
X-n524-k153	167.6	154533	**154593**	154593	536.4	984.0	4054.6	1021	161.9	143.0	1212.9	335	353.0	344.4	1765.2	499	179.3	138.9	1060.3	235
X-n536-k96	699.3	94586	94633	94846	619.7	1303.4	9190.2	205	276.1	277.1	3899.2	81	347.5	492.5	3097.9	59	312.7	389.0	3058.9	61
X-n548-k50	632.1	86650	86680	86700	655.0	1228.7	5597.0	53	416.0	132.7	3209.4	31	651.8	410.6	3438.2	33	661.2	89.1	3474.0	43
X-n586-k159	67.6	190011	190139	190316	2373.6	796.7	5311.5	1365	3097.8	188.7	5622.9	1661	1530.7	750.6	4485.9	1443	1186.5	78.5	3254.8	1451
X-n670-k130	1549.1	146228	146280	146332	379.8	1101.1	9328.7	169	297.9	230.9	5499.8	63	154.2	350.8	3020.7	27	70.4	86.9	2369.3	15
X-n837-k142	819.0	193348	193391	193737	258.7	373.7	2499.5	49	352.7	128.2	2468.5	61	372.1	272.6	2210.2	43	294.9	101.5	2045.7	43
X-n856-k95	984.3	88891	88918	88965	264.3	424.6	4376.9	23	127.4	42.7	2227.0	11	275.0	381.7	2592.9	11	223.5	215.0	2478.3	11
X-n916-k207	488.3	328718	328788	329179	814.8	1105.5	5463.8	357	1152.3	288.2	4974.9	403	718.8	824.6	4336.1	273	676.5	94.4	3138.5	231
G.M.	366.8	-	-	-	389.2	329.2	4324.3	100.3	321.9	91.2	3184.3	74.9	410.6	256.4	3120.7	69.3	329.0	86.1	2691.5	62.7

EC.10.4. Detailed Results of Comparison with VRPSolver

Table EC.13: Comparison of VRPSolver and 2LBB-dy for CVRP benchmark: CVRP-V-42-X

Instance	UB	BKS	VRPSolver						2LBB-dy					
			LB	Root	LP	Heur	CPU	Node	LB/1	Root	LP	Heur	CPU	Node
X-n120-k6	**13332**	13332	13297	1698.0	246.7	16.3	2544.9	5	13281	327.3	161.1	11.4	1238.9	25
X-n134-k13	**10916**	10916	10875	842.3	86.9	32.5	5927.0	15	10882	364.5	164.3	13.0	1575.5	19
X-n143-k7	**15700**	15700	15652	1467.3	251.7	165.0	5425.4	15	15639	382.3	568.2	135.8	5896.9	45
X-n186-k15	24130	24145	24073	1160.6	207.7	39.8	2940.2	13	24051	304.2	278.7	71.8	3396.2	53
X-n190-k8	16962	16980	16957	2520.1	266.5	55.4	3073.5	3	16957	1495.5	115.1	24.9	3948.7	7
X-n200-k36	**58578**	58578	58532	83.4	59.8	26.3	885.1	47	58530	50.1	43.3	9.1	459.2	73
X-n209-k16	30644	30656	30599	1787.4	244.0	29.2	3613.8	13	30585	447.8	163.4	38.6	2297.2	17
X-n214-k11	10830	10856	10823	3373.7	485.1	130.9	4314.0	3	10812	858.8	190.0	159.1	3450.7	15
X-n228-k23	**25742**	25742	25697	1197.0	385.9	123.1	8444.8	33	25688	490.2	264.4	132.0	3142.6	87
X-n233-k16	19170	19230	19130	2150.2	217.8	57.9	7350.1	17	19118	440.0	438.3	69.6	4593.2	27
X-n242-k48	**82751**	82751	82639	124.0	450.2	221.3	7126.6	181	82634	116.6	236.5	49.1	2090.0	363
X-n256-k16	18692	18839	18652	2602.5	588.9	222.8	8230.8	17	18642	691.3	554.5	297.8	4498.1	31
X-n261-k13	26502	26558	26466	3404.1	819.6	411.7	11287.7	21	26458	1104.2	487.2	319.4	8323.6	31
X-n270-k35	35220	35291	35120	276.1	1105.7	398.9	16169.8	269	35124	197.0	532.2	238.7	2615.9	181
X-n280-k17	33348	33503	33327	3680.2	531.3	131.2	7317.8	9	33303	1500.4	542.8	477.1	6856.1	25
X-n284-k15	20183	20226	20172	4954.2	760.4	114.0	6971.4	3	20164	1395.0	263.9	408.2	4171.7	17
X-n289-k60	95063	95151	94963	180.7	188.6	115.2	4020.1	135	94953	268.1	522.0	170.8	3578.5	449
X-n294-k50	47058	47161	46939	214.0	740.7	407.3	15552.3	405	46934	102.9	812.8	313.3	5378.9	991
X-n303-k21	21601	21736	21569	4535.7	948.7	308.4	13633.7	9	21551	1079.2	359.2	513.8	3941.3	15
X-n313-k71	93956	94043	93873	148.1	321.7	198.9	5700.5	249	93841	127.7	382.2	200.7	3961.3	917
X-n322-k28	29785	29834	29752	2114.8	471.7	94.7	3774.8	7	29726	707.0	399.3	134.9	4845.3	35
X-n327-k20	27431	27532	27417	4476.0	991.1	128.8	6052.8	3	27406	874.8	247.0	307.7	2694.7	9
X-n331-k15	31085	31102	31085	5317.7	0.0	0.0	5317.7	1	31063	1645.4	363.6	441.7	6493.6	17
X-n336-k84	138916	139111	138812	1062.6	83.9	64.8	6770.7	41	138774	209.2	315.2	129.2	2479.6	141
X-n344-k43	41993	42050	41930	503.6	698.0	228.9	9829.7	107	41925	213.2	640.7	327.4	2691.8	147
X-n351-k40	25858	25896	25821	930.9	588.4	262.1	8299.5	77	25819	719.0	223.7	223.5	4669.1	45
X-n359-k29	51435	51505	51414	3803.1	1202.8	165.7	5787.8	3	51391	710.1	507.4	153.6	4174.9	45
X-n384-k52	65824	65940	65778	852.6	212.0	49.0	3882.0	33	65775	295.0	309.5	141.6	1308.3	63
X-n420-k130	**107798**	107798	107708	154.9	134.5	104.2	3317.3	117	107688	71.9	58.0	6.5	489.3	225
X-n429-k61	65267	65449	65199	884.0	616.1	271.3	14753.1	131	65198	261.3	662.1	347.0	3109.7	227
X-n439-k37	36345	36391	36318	1656.8	1047.8	313.0	8499.4	35	36312	932.4	1152.8	1061.4	9091.1	47
X-n469-k138	221664	221824	221553	221.4	21.9	18.8	1303.2	37	221483	34.7	241.3	32.2	1733.8	1277
X-n480-k70	89334	89449	89290	954.4	361.7	120.4	7895.8	43	89288	327.2	315.0	172.3	1740.3	49
X-n502-k39	69172	69226	69164	3272.1	850.5	199.6	5616.4	5	69153	2171.5	362.5	549.3	4439.4	11
X-n524-k153	**154593**	154593	154534	258.6	141.7	181.7	3105.6	149	154533	180.1	80.8	124.2	743.7	193
X-n536-k96	94633	94846	94618	2319.6	124.7	51.6	2924.5	3	94585	885.0	324.3	469.1	4328.4	61
X-n548-k50	86680	86700	86652	1328.5	2020.0	578.2	10342.7	45	86652	744.5	1340.2	1056.9	8264.1	65
X-n586-k159	190139	190316	190052	428.0	161.6	152.6	9238.5	161	190009	67.7	717.3	129.7	2607.4	1519
X-n670-k130	146280	146332	146262	2103.2	114.5	47.4	2972.1	7	146218	2102.7	267.6	560.1	5863.8	63
X-n837-k142	193391	193737	193364	4838.9	461.8	162.5	10145.5	13	193358	832.2	180.0	91.2	1483.3	31
X-n856-k95	88918	88965	88903	4591.1	626.0	305.8	7739.9	7	88891	1314.4	180.2	196.1	2952.8	9
X-n916-k207	328788	329179	328761	4313.7	114.6	34.9	7736.8	11	328717	564.4	729.1	172.9	3454.9	269
G.M.	-	-	48592.0	1178.8	336.1	116.4	5773.3	21.8	48576.9	429.1	316.6	146.2	3055.1	64.0

EC.10.5. Detailed Results of VRPTW-180-60-1 (UB=u^*)

Table EC.14: Comparison of 3PB, 2LBB, 3PB-dy and 2LBB-dy for VRPTW batch: VRPTW-180-60-1 (UB=u^*)

Instance	Root	LB	UB	3PB				2LBB				3PB-dy				2LBB-dy			
				LP	Heur	CPU	Node	LP	Heur	CPU	Node	LP	Heur	CPU	Node	LP	Heur	CPU	Node
XML180_2142_01_TW	32.9	25419.5	25456.1	145.6	81.6	309.0	121	135.3	23.6	229.6	109	66.7	38.2	166.8	77	57.5	10.3	132.9	83
XML180_2142_02_TW	21.4	25911.8	25973.0	384.8	302.1	876.5	375	247.2	40.3	389.8	223	227.4	239.9	662.1	389	119.8	26.3	313.8	353
XML180_2142_03_TW	37.6	24630.2	24669.2	169.4	109.2	388.2	123	175.2	23.2	306.6	129	141.7	100.6	363.0	139	85.3	16.1	233.4	153
XML180_2142_04_TW	24.6	24493.6	24550.8	295.3	248.8	719.7	283	216.7	38.1	370.8	205	158.0	180.6	500.3	245	89.6	19.6	240.1	201
XML180_2142_05_TW	27.5	23971.1	24004.9	71.0	41.9	168.5	71	48.3	6.2	100.2	47	39.0	19.4	100.5	35	32.5	6.6	81.7	37
XML180_2142_06_TW	20.4	24958.3	24991.2	67.9	48.0	164.7	81	55.4	9.3	105.7	65	55.8	39.4	143.0	75	31.9	4.9	77.5	53
XML180_2142_07_TW	19.5	24690.5	24745.3	350.6	306.4	848.3	425	268.4	44.3	428.9	319	200.5	216.1	591.5	411	106.0	31.8	292.1	365
XML180_2142_08_TW	17.4	24575.4	24626.1	132.4	95.4	299.1	165	116.6	20.4	191.8	145	120.9	90.0	293.4	193	84.4	18.6	197.7	251
XML180_2142_09_TW	13.3	23929.9	23976.9	206.4	174.8	487.4	337	106.7	19.3	175.9	169	81.3	78.5	221.7	213	58.7	14.0	150.1	259
XML180_2142_10_TW	19.3	24558.5	24597.2	53.6	36.5	133.6	57	40.3	8.7	82.1	43	39.1	20.4	95.1	43	34.4	5.8	79.2	51
XML180_2142_11_TW	41.6	24457.8	24495.4	182.4	111.6	412.2	119	153.8	28.0	283.7	99	122.7	96.4	342.7	119	88.9	18.9	223.0	111
XML180_2142_12_TW	17.0	25722.1	25767.5	34.3	22.9	89.2	55	33.5	6.4	68.6	49	26.8	14.5	68.3	43	21.2	3.6	51.8	43
XML180_2142_13_TW	19.4	25693.0	25742.7	187.8	151.7	449.9	255	131.7	22.5	223.3	171	105.4	105.6	301.7	211	77.9	22.3	205.1	267
XML180_2142_14_TW	27.3	24380.1	24426.3	416.9	411.3	1097.8	409	249.9	49.0	424.7	223	235.2	291.3	763.8	333	136.2	43.1	377.0	313
XML180_2142_15_TW	37.6	25066.3	25092.7	67.7	36.9	164.4	41	55.2	10.0	123.6	33	62.1	32.3	153.3	33	54.8	9.3	125.3	37
XML180_2142_16_TW	21.4	25498.5	25546.9	138.9	83.3	296.0	149	142.8	26.7	240.4	145	132.3	117.7	358.6	223	86.7	17.9	208.2	215
XML180_2142_17_TW	37.8	23517.0	23567.4	160.6	103.1	380.7	81	130.9	17.1	241.8	71	122.7	73.7	289.2	65	122.5	35.2	329.5	137
XML180_2142_18_TW	19.4	24835.1	24872.0	91.6	65.2	216.7	109	94.1	14.6	163.4	107	72.6	54.5	185.0	103	50.1	10.9	129.8	123
XML180_2142_19_TW	20.3	24941.1	24988.9	243.9	181.8	555.2	259	233.1	31.7	365.1	259	337.1	286.0	842.6	457	126.1	28.0	286.9	305
XML180_2142_20_TW	42.8	25567.3	25605.4	138.5	86.8	324.7	91	124.0	18.4	232.7	83	104.6	69.5	271.8	89	82.1	15.0	211.4	105

Continued on next page

Table EC.14 – *Continued from previous page*

Instance	Root	LB	UB	3PB				2LBB				3PB-dy				2LBB-dy			
				LP	Heur	CPU	Node	LP	Heur	CPU	Node	LP	Heur	CPU	Node	LP	Heur	CPU	Node
XML180_2242_01_TW	47.5	18941.9	18962.6	125.5	80.1	313.0	73	123.1	19.3	238.4	71	92.8	63.7	254.6	59	81.4	18.2	195.1	65
XML180_2242_02_TW	37.5	16009.3	16026.6	74.5	44.3	188.0	59	63.2	12.0	138.6	49	81.2	46.2	204.2	63	55.3	10.6	135.8	61
XML180_2242_03_TW	17.2	21883.4	21946.9	635.6	613.1	1634.2	673	469.1	90.8	790.8	477	470.0	503.7	1374.7	685	200.5	65.1	628.5	679
XML180_2242_04_TW	31.8	22418.3	22455.9	60.6	43.4	166.1	51	54.5	9.7	122.3	49	53.8	31.9	144.3	53	42.3	6.2	106.4	49
XML180_2242_05_TW	31.1	22444.8	22488.0	322.1	239.3	745.0	259	204.7	33.0	349.0	163	167.7	148.4	459.8	221	120.1	38.9	315.6	235
XML180_2242_06_TW	16.5	12256.0	12273.6	43.9	25.2	102.3	67	32.0	4.7	66.4	49	36.2	21.0	88.6	63	26.5	4.9	65.6	67
XML180_2242_07_TW	19.3	16713.7	16740.2	446.8	403.7	1105.4	517	279.4	59.5	484.4	307	170.7	223.1	570.5	329	125.4	39.2	368.8	363
XML180_2242_08_TW	27.0	23049.0	23076.6	194.9	150.0	466.1	159	161.6	26.3	277.3	131	139.4	139.0	401.0	159	99.6	23.3	246.7	175
XML180_2242_09_TW	17.2	15545.9	15558.1	15.4	9.4	48.8	25	25.1	5.3	59.1	39	25.3	13.2	65.4	37	26.0	3.1	61.3	47
XML180_2242_10_TW	28.3	19312.1	19356.4	273.2	187.7	611.8	209	264.7	42.7	440.6	199	139.5	111.6	373.8	157	120.8	30.0	318.6	231
XML180_2242_11_TW	19.7	20693.3	20722.8	119.4	88.8	283.6	157	110.2	27.9	200.1	125	63.3	51.9	171.9	107	46.9	11.5	114.9	113
XML180_2242_12_TW	15.8	15093.2	15125.1	503.5	392.0	1153.1	913	619.1	115.1	1001.2	1013	353.3	341.0	965.7	907	173.9	46.8	530.8	995
XML180_2242_13_TW	17.4	19579.7	19633.7	626.5	498.3	1429.8	671	422.0	79.5	687.3	443	396.6	387.8	1090.9	627	188.0	65.0	563.8	663
XML180_2242_14_TW	19.8	20177.8	20212.9	423.8	337.9	990.3	513	290.5	53.7	484.5	337	204.9	232.8	633.3	449	101.0	25.2	293.7	395
XML180_2242_15_TW	29.8	14281.4	14301.9	746.4	648.5	1827.2	639	626.9	122.2	1063.1	507	439.2	498.4	1361.0	625	202.1	52.1	583.9	509
XML180_2242_16_TW	20.6	22055.1	22111.4	549.5	432.5	1258.4	687	662.3	104.3	1025.3	815	435.6	389.0	1141.5	813	212.7	54.5	575.1	877
XML180_2242_17_TW	31.6	15375.9	15387.6	102.4	76.7	260.8	93	77.1	13.8	154.4	59	60.1	39.7	159.1	61	54.0	5.8	119.8	63
XML180_2242_18_TW	39.5	21496.9	21524.1	216.5	133.8	467.3	127	211.1	36.2	359.5	117	178.8	130.8	424.9	131	128.4	28.0	305.8	181
XML180_2242_19_TW	24.6	18484.5	18532.8	279.5	228.6	677.9	229	147.5	24.0	255.5	121	81.8	53.7	205.9	95	71.3	19.1	199.4	161
XML180_2242_20_TW	16.4	16627.2	16655.5	361.6	339.1	907.6	531	238.0	46.7	399.6	337	165.9	178.8	484.5	377	73.6	21.9	239.4	383
XML180_2342_01_TW	44.6	22942.9	22991.1	516.5	362.0	1161.3	347	315.2	44.2	531.9	211	249.0	224.3	712.9	299	151.3	29.6	421.0	281
XML180_2342_02_TW	42.4	23852.3	23896.3	179.3	96.5	386.2	101	177.5	26.8	316.3	101	130.6	84.0	337.9	109	88.5	12.6	218.4	99
XML180_2342_03_TW	42.9	22772.9	22826.8	266.5	159.0	570.8	145	328.0	40.3	523.5	173	248.9	191.6	642.3	209	130.5	22.0	297.3	145
XML180_2342_04_TW	12.7	23461.6	23624.5	738.5	741.0	1908.2	705	601.2	154.7	1029.1	559	494.0	473.2	1304.7	563	400.2	140.0	1022.7	869
XML180_2342_05_TW	40.8	22234.8	22266.5	153.4	97.7	362.9	119	123.8	20.7	229.1	95	90.9	57.7	232.6	83	76.7	18.5	188.6	103
XML180_2342_06_TW	5.3	21132.2	21151.2	25.0	18.4	57.9	143	27.6	4.9	47.7	149	43.4	28.0	92.4	219	10.3	1.3	23.4	95
XML180_2342_07_TW	20.8	22495.8	22551.1	351.5	240.3	738.1	277	182.4	24.4	288.0	145	155.5	137.4	410.1	229	88.2	18.1	208.7	185
XML180_2342_08_TW	11.4	23746.4	23816.0	323.9	318.0	845.5	549	364.0	71.9	627.3	603	277.5	296.1	784.7	577	186.2	58.9	538.6	827
XML180_2342_09_TW	18.8	22832.9	22888.8	511.9	399.0	1173.2	603	348.0	61.2	564.4	393	467.9	400.1	1171.8	743	183.9	47.4	503.0	651
XML180_2342_10_TW	18.7	24020.0	24044.3	183.2	143.4	422.1	259	131.5	19.8	218.7	179	82.9	75.3	224.9	161	56.9	12.6	135.1	161
XML180_2342_11_TW	15.9	24078.1	24114.3	121.1	76.0	256.8	153	102.2	16.9	166.9	129	96.3	75.5	240.5	179	53.9	10.5	126.3	185
XML180_2342_12_TW	26.1	22752.1	22779.7	141.3	81.1	304.3	133	102.0	14.6	176.8	91	71.8	42.2	178.7	91	55.6	6.9	130.9	101
XML180_2342_13_TW	16.5	22609.5	22635.5	35.6	18.9	81.7	53	35.4	5.2	69.4	53	35.1	19.7	84.0	59	36.7	7.8	76.1	61
XML180_2342_14_TW	76.7	25371.4	25391.7	104.4	67.2	292.5	47	60.9	16.5	180.9	27	85.4	39.7	226.0	27	76.2	17.7	199.5	35
XML180_2342_15_TW	41.6	21436.5	21467.3	154.6	85.2	336.5	111	130.2	17.1	229.5	87	134.7	92.2	328.2	133	89.9	20.6	211.4	113
XML180_2342_16_TW	23.8	25068.6	25126.8	560.6	436.4	1280.7	485	348.3	58.8	570.7	293	353.7	352.6	1001.4	491	206.5	60.0	586.6	543
XML180_2342_17_TW	39.5	25414.3	25459.9	698.5	527.6	1574.0	353	482.1	68.8	776.0	251	329.1	298.7	892.9	283	211.9	53.3	547.5	305
XML180_2342_18_TW	13.6	21293.3	21307.9	39.2	26.2	93.2	79	27.5	6.0	58.1	55	22.9	14.1	60.0	49	19.0	3.4	45.5	45
XML180_2342_19_TW	19.3	24657.8	24701.5	60.6	35.0	134.2	83	34.9	4.5	69.5	49	34.8	18.1	84.5	51	30.0	5.9	71.3	67
XML180_2342_20_TW	23.9	22601.7	22645.2	144.3	83.0	304.1	161	143.5	21.0	234.9	157	93.8	69.3	241.7	169	60.9	9.8	144.1	153
G.M.	24.0	-	-	175.3	124.0	414.0	176.0	141.9	24.1	254.1	138.8	119.2	92.2	317.0	157.6	79.1	17.1	201.5	164.5

EC.10.6. Detailed Results of VRPTW-180-60-1 (UB=u^0)

Table EC.15: Comparison of 3PB, 2LBB, 3PB-dy and 2LBB-dy for VRPTW batch: VRPTW-180-60-1 (UB=u^0)

Instance	Root	LB	UB	3PB				2LBB				3PB-dy				2LBB-dy			
				LP	Heur	CPU	Node	LP	Heur	CPU	Node	LP	Heur	CPU	Node	LP	Heur	CPU	Node
XML180_2142_01_TW	27.2	25417.5	25481.6	212.9	128.4	419.5	173	236.7	41.4	341.8	191	151.0	74.1	282.6	103	104.9	25.3	181.4	107
XML180_2142_02_TW	20.1	25908.9	25999.0	690.2	546.6	1430.2	685	413.0	79.9	596.8	397	601.6	384.5	1182.8	499	147.2	33.5	367.8	379
XML180_2142_03_TW	33.0	24629.2	24693.9	414.4	329.1	914.6	301	346.0	66.8	497.8	249	348.9	254.2	742.2	243	222.2	58.8	396.9	231
XML180_2142_04_TW	17.2	24483.1	24575.4	468.5	413.3	1080.3	501	423.9	112.6	685.5	423	423.8	326.5	935.7	443	175.4	51.1	366.3	339
XML180_2142_05_TW	34.0	23977.7	24028.9	93.4	56.2	221.6	63	118.5	24.9	208.6	75	130.9	60.2	262.4	55	117.8	28.4	211.5	77
XML180_2142_06_TW	16.3	24953.0	25016.2	99.0	65.3	212.9	131	82.9	16.4	137.6	107	100.4	64.6	222.8	129	65.2	13.9	117.1	121
XML180_2142_07_TW	17.1	24689.8	24770.0	515.3	433.7	1108.6	617	467.8	99.2	668.7	537	528.2	323.0	999.6	503	196.3	57.0	347.1	439
XML180_2142_08_TW	28.9	24585.4	24650.7	326.4	236.4	675.0	253	179.7	29.9	266.9	143	310.0	152.1	567.7	181	118.3	25.3	209.2	121
XML180_2142_09_TW	12.3	23928.0	24000.9	313.5	282.9	717.6	523	214.1	46.6	339.0	331	393.4	311.5	806.3	539	145.3	51.5	274.7	357
XML180_2142_10_TW	30.0	24562.5	24621.8	58.3	41.3	154.1	39	46.0	9.9	105.5	31	65.5	32.7	155.8	35	64.7	12.4	132.0	49
XML180_2142_11_TW	30.9	24450.5	24519.9	411.6	323.5	961.9	269	293.6	52.2	470.1	187	529.0	354.6	1086.1	291	188.3	43.7	379.1	219
XML180_2142_12_TW	15.0	25718.4	25793.3	33.8	17.8	81.1	51	46.4	8.6	83.3	67	39.7	16.9	88.0	49	28.1	6.9	60.9	39
XML180_2142_13_TW	22.8	25693.4	25768.4	293.8	232.8	639.6	267	270.3	73.1	412.1	239	459.7	362.7	1020.0	365	296.3	89.7	503.1	421
XML180_2142_14_TW	27.2	24378.5	24450.7	764.3	774.5	1886.3	645	668.9	166.0	1058.6	503	937.1	584.2	1829.2	525	489.7	133.5	812.4	545
XML180_2142_15_TW	29.3	25064.1	25117.8	84.6	50.8	191.5	67	94.0	19.9	166.9	69	100.7	47.7	214.3	67	96.5	21.7	176.0	77
XML180_2142_16_TW	18.4	25497.3	25572.4	220.8	170.4	507.0	231	224.1	47.0	349.4	221	201.1	171.0	511.8	281	241.8	73.0	414.0	369
XML180_2142_17_TW	30.9	23513.0	23591.0	226.4	148.2	475.9	161	171.4	34.1	277.4	133	295.9	220.8	644.5	195	117.8	22.9	219.1	135
XML180_2142_18_TW	21.9	24837.9	24896.9	84.9	50.5	192.5	79	148.4	37.5	233.0	129	113.9	53.8	237.4	85	90.5	21.6	170.0	105
XML180_2142_19_TW	19.7	24938.4	25013.9	688.5	530.4	1406.0	745	456.8	99.1	658.0	483	578.0	381.6	1155.8	543	286.2	76.2	496.6	547
XML180_2142_20_TW	35.1	25563.2	25631.0	201.0	127.0	437.4	123	218.7	49.7	379.8	125	255.7	166.4	518.4	141	175.3	41.1	314.5	151
XML180_2242_01_TW	26.6	18936.4	18981.6	255.3	176.6	516.9	209	233.0	46.8	358.6	185	353.7	203.4	648.3	223	167.8	39.5	268.9	171
XML180_2242_02_TW	29.3	16007.4	16042.6	157.5	96.5	338.2	115	201.4	42.9	309.1	139	173.0	81.4	320.2	107	100.7	16.5	186.2	87
XML180_2242_03_TW	16.0	21881.9	21968.8	1025.8	935.7	2332.9	1141	790.0	202.8	1211.9	819	798.4	634.9	1697.2	775	563.8	228.0	1045.7	1123
XML180_2242_04_TW	31.9	22418.0	22478.4	59.8	39.5	164.4	49	65.5	10.3	144.2	55	64.6	43.9	177.3	55	73.6	14.5	148.3	61

Continued on next page

Table EC.15 – *Continued from previous page*

Instance	Root	LB	UB	3PB				2LBB				3PB-dy				2LBB-dy			
				LP	Heur	CPU	Node	LP	Heur	CPU	Node	LP	Heur	CPU	Node	LP	Heur	CPU	Node
XML180_2242_05_TW	22.8	22441.6	22510.5	539.2	431.1	1212.4	453	530.7	130.2	787.4	441	481.3	339.0	995.3	383	283.8	81.3	492.0	405
XML180_2242_06_TW	16.9	12255.8	12285.9	76.2	44.5	155.0	87	61.8	12.2	108.1	71	74.4	33.9	145.7	75	58.9	14.6	111.9	77
XML180_2242_07_TW	17.8	16712.6	16756.9	518.1	489.6	1258.8	577	459.3	131.9	733.4	515	376.3	292.8	857.2	371	275.6	76.8	510.2	395
XML180_2242_08_TW	31.9	23048.8	23099.7	212.8	168.6	507.6	145	341.1	81.7	554.8	225	320.3	264.9	771.8	227	140.2	35.9	364.6	197
XML180_2242_09_TW	15.2	15543.1	15573.7	61.6	40.7	134.5	87	58.7	13.9	105.4	85	57.4	26.9	112.6	59	50.1	8.7	88.0	75
XML180_2242_10_TW	29.4	19310.6	19375.8	472.5	343.0	974.7	303	358.8	78.4	551.4	225	430.7	306.7	912.9	265	273.4	67.5	495.7	305
XML180_2242_11_TW	18.5	20693.4	20743.5	157.6	124.2	358.7	179	125.1	29.1	209.6	135	176.0	121.2	384.6	179	78.8	16.7	157.3	145
XML180_2242_12_TW	14.3	15092.6	15140.2	543.1	487.5	1336.1	825	1074.1	257.9	1667.2	1555	619.2	408.4	1252.5	797	279.8	70.7	603.8	819
XML180_2242_13_TW	19.9	19581.4	19653.3	725.0	587.7	1636.5	653	562.7	124.1	886.1	467	725.6	401.6	1420.6	509	288.0	68.8	619.0	501
XML180_2242_14_TW	19.1	20179.3	20233.1	367.2	300.2	855.7	425	298.3	80.7	492.0	325	439.1	314.0	875.8	421	171.8	47.9	351.9	315
XML180_2242_15_TW	27.1	14279.9	14316.2	865.0	797.2	2170.4	769	1306.3	368.7	1975.4	1083	1174.3	687.2	2271.4	759	702.4	249.3	1262.8	789
XML180_2242_16_TW	17.0	22051.2	22133.5	1242.1	886.2	2520.9	1265	1087.2	245.8	1577.4	1101	1064.0	674.3	1959.4	1053	511.2	159.0	957.0	1003
XML180_2242_17_TW	29.6	15376.0	15403.0	135.2	79.7	311.3	103	176.0	38.8	313.1	125	270.7	142.2	504.4	127	144.3	29.0	248.0	127
XML180_2242_18_TW	32.5	21495.0	21545.6	389.4	227.3	729.8	239	183.8	34.8	315.7	107	206.6	121.4	428.1	125	154.2	31.2	284.4	179
XML180_2242_19_TW	18.4	18481.0	18551.3	282.3	249.9	650.6	265	221.3	42.8	340.5	201	210.9	179.7	484.2	219	127.7	48.5	262.7	203
XML180_2242_20_TW	16.8	16622.5	16672.2	448.0	398.6	1056.8	597	513.1	150.3	801.9	665	469.8	337.1	1002.7	547	241.8	93.2	514.9	531
XML180_2342_01_TW	26.5	22935.9	23014.1	805.5	630.0	1727.7	661	557.3	113.4	885.7	451	719.9	514.6	1480.8	563	279.2	72.7	601.0	469
XML180_2342_02_TW	42.9	23850.9	23920.2	342.7	222.9	683.8	187	407.1	77.3	609.1	219	265.6	168.8	555.8	149	208.4	51.9	379.2	183
XML180_2342_03_TW	36.6	22770.4	22849.6	338.0	197.8	701.2	179	312.4	46.3	477.4	155	297.4	171.2	594.3	171	233.2	57.2	456.0	263
XML180_2342_04_TW	13.6	23466.6	23648.1	661.0	713.0	1750.4	633	544.4	168.9	933.4	475	470.5	434.3	1168.3	445	302.5	123.9	716.3	589
XML180_2342_05_TW	37.6	22233.4	22288.8	181.5	99.6	382.6	117	268.5	64.8	413.5	163	149.8	82.1	340.1	99	102.1	19.8	213.2	85
XML180_2342_06_TW	5.6	21131.9	21172.4	45.9	36.1	101.8	185	30.8	7.2	53.1	125	71.0	33.1	125.4	163	41.5	11.1	62.7	169
XML180_2342_07_TW	15.1	22491.5	22573.7	480.0	388.0	1018.1	523	274.6	69.8	419.1	293	475.8	253.0	856.1	361	242.1	66.8	396.2	447
XML180_2342_08_TW	10.2	23746.4	23839.8	752.4	715.6	1708.3	1111	661.3	180.5	965.6	937	630.5	397.0	1241.8	723	306.9	101.4	644.8	813
XML180_2342_09_TW	18.1	22831.4	22911.7	737.6	573.1	1593.3	809	346.4	72.9	562.2	351	499.5	340.0	1045.7	517	274.2	84.9	527.8	517
XML180_2342_10_TW	16.6	24018.3	24068.3	253.0	224.0	595.4	303	169.5	40.6	258.3	191	439.4	206.3	741.9	287	188.3	41.0	287.9	227
XML180_2342_11_TW	14.5	24076.4	24138.4	203.8	155.9	428.6	247	107.5	19.6	173.9	131	174.3	115.7	351.6	189	97.6	25.3	172.0	215
XML180_2342_12_TW	28.7	22751.2	22802.5	148.3	79.7	313.3	115	135.2	23.5	234.7	105	142.8	64.2	282.1	105	110.1	16.9	198.7	105
XML180_2342_13_TW	12.9	22606.0	22658.1	60.1	36.1	129.7	93	83.0	24.9	145.2	121	59.8	29.2	119.4	77	55.5	13.8	106.9	101
XML180_2342_14_TW	52.9	25365.7	25417.1	71.5	45.3	200.3	33	116.6	31.8	231.6	53	142.4	74.8	314.5	49	139.4	32.0	257.8	65
XML180_2342_15_TW	40.9	21434.7	21488.8	161.5	91.9	345.2	115	296.6	55.4	426.2	185	379.6	235.6	716.0	215	205.4	43.3	333.1	167
XML180_2342_16_TW	22.1	25062.2	25151.9	1022.0	670.1	1936.9	1007	596.9	110.8	833.5	593	467.8	320.1	924.6	477	283.2	79.2	507.6	543
XML180_2342_17_TW	28.0	25410.1	25485.4	1147.9	918.9	2462.0	621	888.8	201.6	1363.7	453	1070.7	713.9	2107.3	493	749.8	229.7	1197.3	539
XML180_2342_18_TW	11.9	21291.3	21329.2	47.1	31.4	109.1	101	64.1	15.5	102.8	129	106.2	44.6	184.5	123	33.5	5.6	68.0	85
XML180_2342_19_TW	16.7	24653.0	24726.2	63.4	36.2	131.6	87	76.0	15.7	119.0	105	71.8	41.3	142.9	101	44.7	10.8	87.0	73
XML180_2342_20_TW	19.5	22598.4	22667.8	294.5	219.2	627.8	323	188.6	34.2	288.4	197	211.1	147.6	457.3	243	140.3	37.4	257.1	255
G.M.	21.8	-	-	258.0	187.8	574.1	245.9	237.3	52.0	381.2	218.1	269.4	164.9	553.3	220.1	158.0	39.8	297.1	219.9

EC.10.7. Detailed Results of VRPTW-200-22-H2

Table EC.16: Comparison of 3PB, 2LBB, 3PB-dy and 2LBB-dy for VRPTW batch: VRPTW-200-22-H2

Instance	Root	LB	UB	BKS	3PB				2LBB				3PB-dy				2LBB-dy			
					LP	Heur	CPU	Node	LP	Heur	CPU	Node	LP	Heur	CPU	Node	LP	Heur	CPU	Node
C2_2_2	60.3	1846.1	**1851.4**	1851.4	16.6	84.7	202.7	11	42.0	56.8	276.8	27	34.0	94.5	264.0	17	24.1	120.9	256.4	15
C2_2_3	140.7	1745.1	1753.4	1763.4	230.1	838.2	1657.3	107	136.4	217.2	741.3	71	790.8	1497.8	4031.0	347	139.5	373.9	948.8	67
C2_2_4	223.0	1662.1	1670.0	1695.0	1578.1	6312.9	14614.2	721	640.3	604.7	3282.1	213	359.3	417.3	2215.2	125	458.8	325.7	3053.2	231
C2_2_5	53.1	1863.1	**1869.6**	1869.6	126.6	527.7	928.9	71	131.6	149.0	522.2	83	107.6	222.1	535.3	45	74.3	42.7	304.3	45
C2_2_6	66.2	1839.1	**1844.8**	1844.8	120.5	439.6	828.4	59	42.4	46.2	260.2	29	78.2	128.6	420.9	35	70.0	265.2	523.8	35
C2_2_7	62.3	1839.1	**1842.2**	1842.2	19.3	46.3	155.8	13	23.4	17.6	121.7	13	30.5	44.4	169.3	13	20.3	29.7	146.2	9
C2_2_8	58.2	1809.1	**1813.7**	1813.7	88.0	253.4	521.7	47	82.3	79.1	308.3	47	68.9	112.6	314.4	31	65.5	65.0	322.2	45
C2_2_9	99.4	1806.1	**1815.0**	1815.0	1078.0	5425.6	9611.1	569	672.0	755.7	2604.2	319	966.8	2469.9	5455.6	377	368.7	302.4	2148.8	283
C2_2_10	117.1	1783.1	**1791.2**	1791.2	604.3	2101.0	4389.3	275	911.6	573.5	3151.0	399	1405.6	3049.0	7808.4	493	528.1	269.3	2097.6	271
R2_2_5	108.1	3047.1	**3061.1**	3061.1	552.7	2383.9	4589.2	235	676.9	1116.0	3521.4	257	576.8	1904.9	4160.4	197	245.8	185.9	1535.1	165
R2_2_6	244.0	2664.1	**2675.4**	2675.4	1819.4	15987.2	28469.9	609	1458.2	3281.2	11027.7	547	1839.1	11786.5	20209.4	393	673.9	1641.5	7448.1	339
R2_2_7	603.2	2289.1	2299.7	2304.7	2553.8	19836.5	38023.4	889	5182.5	11430.1	44542.9	1537	2364.4	10741.7	22794.5	499	1264.7	2930.2	12380.6	477
R2_2_9	132.4	2833.1	**2843.3**	2843.3	435.2	1513.0	2949.5	251	318.4	231.8	1106.7	169	149.5	190.4	638.2	57	190.5	95.4	857.1	129
R2_2_10	150.1	2537.1	**2549.4**	2549.4	1052.6	3909.1	8193.0	571	801.0	609.3	3040.2	371	361.7	589.8	1690.8	139	281.7	132.3	1395.3	197
RC2_2_1	69.8	2790.1	**2797.4**	2797.4	45.9	195.8	424.1	29	27.5	49.0	215.5	17	37.0	87.4	266.4	13	21.3	206.8	369.6	11
RC2_2_2	368.2	2463.1	2475.0	2481.6	1814.8	17055.6	29749.8	749	2442.1	5666.6	26919.2	1467	1120.7	6984.3	12444.2	307	506.9	4321.4	7754.8	181
RC2_2_3	757.9	2209.1	2217.7	2227.7	196.1	1043.0	4975.5	77	116.4	121.1	3597.6	59	274.3	1099.3	4258.2	53	129.3	936.8	2809.3	33
RC2_2_4	1185.1	1787.1	1809.8	1854.8	324.8	3766.4	10592.6	99	499.6	858.1	22582.8	293	581.1	6016.9	14653.2	89	117.8	3811.0	6514.5	29
RC2_2_5	172.4	2481.1	**2491.4**	2491.4	248.7	1791.7	3265.0	103	256.0	599.1	2124.7	113	243.3	1420.7	2821.0	73	184.5	430.7	1457.5	79
RC2_2_6	164.0	2490.1	**2495.1**	2495.1	152.8	911.2	1846.7	57	59.7	127.1	600.9	33	141.4	387.0	992.8	35	92.9	332.0	884.9	33
RC2_2_7	282.8	2275.1	2282.7	2287.7	83.0	722.6	1854.0	31	200.2	354.7	3569.0	107	54.6	299.5	1088.5	13	97.1	786.6	2059.9	35
RC2_2_10	670.3	1929.1	1943.0	1989.2	38.3	257.6	2008.3	11	100.6	124.6	6432.0	51	199.6	1305.0	8905.1	27	84.4	982.8	5348.2	19
G.M.	169.5	-	-	-	253.4	1284.8	2959.2	114.7	245.4	324.7	1944.8	122.7	253.7	743.9	2112.5	78.2	147.8	351.8	1442.6	68.6

References

Baldacci R, Christofides N, Mingozzi A (2008) An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming* 115:351–385.

Baldacci R, Hadjiconstantinou E, Mingozzi A (2004) An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. *Operations Research* 52(5):723–738.

Baldacci R, Mingozzi A, Roberti R (2011) New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research* 59(5):1269–1283.

Bulhoes T, Pessoa A, Protti F, Uchoa E (2018a) On the complete set packing and set partitioning polytopes: Properties and rank 1 facets. *Operations Research Letters* 46(4):389–392.

Bulhoes T, Sadykov R, Uchoa E (2018b) A branch-and-price algorithm for the minimum latency problem. *Computers & Operations Research* 93:66–78.

Chen T, Guestrin C (2016) Xgboost: A scalable tree boosting system. *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 785–794.

Gasse M, Chételat D, Ferroni N, Charlin L, Lodi A (2019) Exact combinatorial optimization with graph convolutional neural networks. *Advances in neural information processing systems* 32.

Homberger J, Gehring H (2005) A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. *European Journal of Operational Research* 162(1):220–238.

IBM Corporation (2023) IBM ILOG CPLEX Optimization Studio CPLEX User's Manual. URL https://www.ibm.com/docs/en/icos/22.1.0?topic=manuals-cplex-users-manual.

Irnich S, Desaulniers G, Desrosiers J, Hadjar A (2010) Path-reduced costs for eliminating arcs in routing and scheduling. *INFORMS Journal on Computing* 22(2):297–313.

Lima I, Uchoa E, Pecin D, Pessoa A, Poggi M, Vidal T, Subramanian A, Oliveira D, Queiroga E (2014) CVRPLIB: Capacitated vehicle routing problem library. URL http://vrp.galgos.inf.puc-rio.br/index.php/en/updates.

Lysgaard J (2003) CVRPSEP: A package of separation routines for the capacitated vehicle routing problem. Technical Report 03-04, Department of Management Science and Logistics, Aarhus School of Business, Aarhus University, URL http://www.hha.dk/~lys/CVRPSEP.htm.

Lysgaard J, Letchford AN, Eglese RW (2004) A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming* 100:423–445.

Martinelli R, Pecin D, Poggi M (2014) Efficient elementary and restricted non-elementary route pricing. *European Journal of Operational Research* 239(1):102–111.

Morabit M, Desaulniers G, Lodi A (2021) Machine-learning–based column selection for column generation. *Transportation Science* 55:815–831.

Naddef D, Rinaldi G (2002) Branch-and-cut algorithms for the capacitated vrp. *The vehicle routing problem*, 53–84 (SIAM).

Pecin D, Pessoa A, Poggi M, Uchoa E, Santos H (2017) Limited memory rank-1 cuts for vehicle routing problems. *Operations Research Letters* 45(3):206–209.

Pessoa A, Sadykov R, Uchoa E, Vanderbeck F (2020) A generic exact solver for vehicle routing and related problems. *Mathematical Programming* 183:483–523.

Righini G, 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.

Roberti R, Mingozzi A (2014) Dynamic ng-path relaxation for the delivery man problem. *Transportation Science* 48(3):413–424.

Sadykov R, Uchoa E, Pessoa A (2021) A bucket graph–based labeling algorithm with application to vehicle routing. *Transportation Science* 55(1):4–28.

Uchoa E, Pecin D, Pessoa A, Poggi M, Vidal T, Subramanian A (2017) New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research* 257(3):845–858.

Yang Y (2023) An exact price-cut-and-enumerate method for the capacitated multitrip vehicle routing problem with time windows. *Transportation Science* 57(1):230–251.

Yang Y (2024) Deluxing: Deep lagrangian underestimate fixing for column-generation-based exact methods. *Operations Research* .

Yang Y, Boland N, Dilkina B, Savelsbergh M (2022) Learning generalized strong branching for set covering, set packing, and 0–1 knapsack problems. *European Journal of Operational Research* 301(3):828–840.