Cadernos do LOGIS

Cluster Branching for Vehicle Routing Problems

João Marcos P. Silva, Eduardo Uchoa,
Anand Subramanian

Volume 2024, Number 2

July, 2024

LOGIS
Núcleo de Logística Integrada e Sistemas

Universidade Federal Fluminense

Cluster Branching for Vehicle Routing Problems

João Marcos Silva[a,*], Eduardo Uchoa[a,c], Anand Subramanian[b]

[a] *Universidade Federal Fluminense - Departamento de Engenharia de Produção*
Niterói - Brasil
[b] *Universidade Federal da Paraíba - Departamento de Sistemas de Computação*
João Pessoa - Brasil
[c] *International Chair 2022-2026 - INRIA Bordeaux – Sud-Ouest*
Talence, France

Abstract

This article introduces Cluster Branching, a novel branching strategy for exact algorithms solving Vehicle Routing Problems (VRPs). While branching is crucial for the efficiency of branch-and-bound-based algorithms, existing branching types such as Edge Branching, CutSet Branching, and Ryan&Foster Branching have their limitations. The proposed branching strategy aggregates multiple edge variables into higher-level decision structures corresponding to clusters of customers. Through extensive computational experiments on Distance-Constrained VRP, Capacitated VRP, and VRP with Time Windows instances, we demonstrate that Cluster Branching significantly enhances the performance of a state-of-the-art Branch-Cut-and-Price algorithm, often improving branching quality and reducing the size of the search trees, which allows more instances to be solved. Beyond that, Cluster Branching is also shown to have a robust performance over different instance types, and may work well even for those with randomly positioned customers.

Keywords: Vehicle routing, Branch-Cut-and-Price, Branching strategy, Cluster branching

1. Introduction

Vehicle Routing Problems (VRPs) are among the most widely studied combinatorial optimization problems. Google Scholar found 12,700 articles containing the string "vehicle routing" published only in 2023. Such a level of interest arises from the needs of numerous real systems that distribute goods, essential to modern economies. Since almost all VRP variants are $\mathcal{NP}$-hard, heuristic algorithms are prevailing in today's practice. However, recent progress in exact algorithms has made it possible to solve in reasonable times many instances with up to 100 customers, sometimes even with up to 200 customers. As surveyed in [PU14, CCD19], the current best exact algorithms for VRPs are based either on Branch-and-Cut (BC) or, more frequently, on

*Corresponding author.
Email addresses: joaomarkco@gmail.com (João Marcos Silva), eduardo_uchoa@id.uff.br (Eduardo Uchoa), anand@ci.ufpb.br (Anand Subramanian)

Branch-Cut-and-Price (BCP), which are extensions of the basic Branch-and-Bound (B&B) algorithm [LD60]. This means that even the most sophisticated BCP is still a B&B algorithm that explores the solution space by performing branching: it builds a search tree and uses bounds to prune nodes that cannot yield improving solutions.

The size of a B&B search tree depends significantly on the root node gap, i.e., the difference between the Lower Bound (LB) obtained at the root node and the value of the optimal solution. Indeed, there is a prolific literature on how to reduce root gaps by separating cutting planes (in the case of a BC) or by performing both cut separation and column generation (in the case of a BCP). However, the size of a search tree also depends crucially on the *branching quality*, i.e., how much each branching increases the LBs in its children nodes. The literature on branching is more modest. Many of its works consider the *Branching Selection* problem: given a set of branching candidates (classically the integer variables that have fractional values in the current Linear Programming (LP) solution of a node), select a candidate that leads to a good quality branching. We provide a brief discussion in the following.

- Performing branching selection only by simple rules based on candidate attributes like fractional value, cost, etc., may not work very well. Recently, some complex rules encoded in neural networks obtained by machine learning approaches have found successes, but only on restricted classes of instances that are similar to those in the training set (see [ZLL+23] for a survey).

- Pseudo-cost branching [BGG+71] improves branching selection by considering the actual increases in LBs observed in the current partial search tree. In fact, the past branching quality of a candidate provides good predictors of its quality in the future. Pseudo-cost branching has limitations. In particular, it performs poorly at the top of the search tree because there is little or no past branching information. Note that those are exactly the most critical branching decisions. While an inadequate choice near the bottom of the tree only has a local impact, unpromising choices at the top of the tree may have a global impact on overall performance.

- Strong branching is a computationally expensive procedure that tests many candidates before choosing one of them. Extensive experiments have shown that many times it pays off, especially on the nodes at the top of a large search tree. This strategy was originally proposed by Applegate, Bixby, Chvátal, and Cook around 1995 in their Concorde TSP solver and it was soon included in general Mixed-Integer Programming (MIP) solvers. It turns out the best overall scheme is often to start with strong branching and switch to pseudo-cost branching after reliable branching quality information on most candidates is available [AKM05, AB09].

This work addresses the Branching Type problem for VRPs, consisting of determining which type of branching candidates should be provided for the Branching Selection. The idea is that instead

of only having the natural candidates corresponding to single variables, one may also branch over structures that can be viewed as the aggregation of several variables. Such structures may hopefully capture some higher-level decisions and lead to better branching quality. We focus on the VRP variants that fit within the following framework. There is a depot identified by 0 and a set of customers $V = \{1, \ldots, n\}$. A route corresponds to a cycle in the complete undirected graph $G = (\{0\} \cup V, E)$ that starts and ends at 0. There are additional intra-route constraints defining what are the feasible routes. These constraints may include capacities (e.g., Capacitated VRP (CVRP)), distance limits (e.g., Distance-Constrained VRP (DCVRP)), time windows (e.g., VRP with Time Windows (VRPTW)), etc. A solution is a set of feasible routes that, together, visit each customer exactly once. There may be additional constraints limiting the number of routes in a solution. The currently used branching types for those VRPs are the following:

- **Edge Branching (EB)**. Branching on variables x_e, $e \in E$, which must be binary if $e \notin \delta(\{0\})$ but can assume values in $\{0, 1, 2\}$ otherwise ($x_{0j} = 2$ indicates a route of format $0 - j - 0$). Such edge variables are available in nearly all BC or BCP algorithms for the considered VRP variants. Note that even in problems that are naturally defined over directed graphs, such as the Asymmetric CVRP or the VRPTW, it is still valid to use EB, which can viewed as branching over the aggregation of two directed arc variables. Indeed, in many of those cases EB can be proved to be *complete*, i.e., if all edge variables are integer then there must be a feasible integer solution over the arc variables with the same cost. However, even if EB is not complete, one can still branch on arc variables too. It should be noted that EB is somehow unbalanced: fixing an edge to one usually moves LBs more than fixing it to zero. EB is employed in some of the best BCP algorithms available, e.g., [PSUV20, EQSU23].

- **CutSet Branching (CSB)**. Branching on variables $\omega_S = \sum_{e \in \delta(S)} x_e/2$, $S \subseteq V$. It was first observed in [CN93] that TSP solutions cross a cutset $\delta(S)$ an even number of times, therefore ω_S should be integer. CSB was first used for the CVRP in the BCP presented in [LLE04]. The CVRPSep Library [Lys03] even has a routine that given a fractional solution in terms of the edge variables finds a collection of sets S such that ω_S is close to a given fractional target. It can be observed that CSB is a generalization of EB: if $S = \{i, j\}$, branching over ω_S is equivalent to branching over x_{ij}. This means that if EB is complete, so is CSB (if x_e is integer for all $e \notin \delta(0)$, simple degree constraints $\sum_{e \in \delta(\{i\})} x_e = 2$, $i \in V$, enforce x_e to be also integer for all $e \in \delta(\{0\})$). CSB, using the CVRPSep routine for identifying the candidates, was adopted in several BCP algorithms [FLL+06, PPPU17, PCDU17].

- **R&F Branching (RFB)**. BCP algorithms also have route variables λ_r, $r \in R$, where R is the set of all feasible routes. Because there is an exponential number of such variables, column generation should be performed. Branching over an individual route variable is not recommended. First, it is highly unbalanced: there are so many such variables that fixing a single one to zero barely changes the problem. Second, it is non-robust, making the pricing

subproblem harder [PdAU03]. Indeed, if λ_r is fixed to zero the pricing algorithm should be modified to avoid generating route r again. However, a much more balanced and effective branching type can be obtained by adapting the scheme proposed by Ryan&Foster (R&F) [RF81]. Given a set $\{i, j\} \subseteq V$, let $P(i, j) \subseteq P$ be the set of routes that visit both i and j (not necessarily in sequence). RFB amounts to branching over aggregated variable $\rho_{ij} = \sum_{r \in P(i,j)} \lambda_r$. RFB is non-robust, each such branching makes the pricing subproblem harder. Nevertheless, when the original pricing subproblem is not that hard, which often happens on instances where the routes are shorter, RFB can still be efficiently applied. RFB is complete and it is available as a non-default option in [PSUV20].

The main contribution of this article is proposing the so-called Cluster Branching (CB), a non-complete branching type that can significantly enhance the existing VRP branching strategies. While CB was initially devised for handling some instances where the customers are tightly clustered, to our surprise, on average, it helps even in instances where customers are randomly positioned! This assertion is supported by extensive computational experiments involving many thousands of runs over a highly diversified set of CVRP and VRPTW instances. Additional long runs could solve several open instances in CVRPLIB for the first time.

The remainder of this work is organized as follows. Section 2 presents two metrics for evaluating branching quality, which will be employed to assess both the proposed branching scheme and those found in the literature. Section 3 introduces the proposed CB scheme and its underlying motivation. Section 4 discusses the clustering algorithms that can be employed with CB. Section 5 reports and discusses the findings from extensive computational experiments. Finally, Section 6 provides the concluding remarks of this work.

2. Measuring Branching Quality

In what follows, we define some metrics that will be used for assessing branching quality.

2.1. Estimating Tree Size

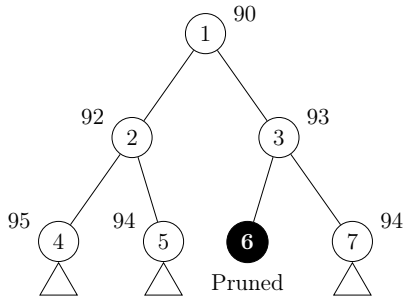
Given a partial run of a B&B algorithm, possibly stopped due to a time limit, we seek to estimate the fraction of the total search tree already explored. We first estimate the Remaining Tree Size (RTS) by summing the expected size of the subtrees in all open nodes (i.e. the nodes that were created but not yet explored). Let $\mathcal{O}$ represent the set of open nodes in the search tree. Assuming a minimization problem, we define:

$$\text{RTS} = \sum_{i \in \mathcal{O}} (2^{\frac{\text{gap}(i)}{\Delta_{avg}}+1} - 2), \tag{1}$$

where $\text{gap}(i)$ is the difference between the LB of node i and the Best Known Solution (BKS), and Δ_{avg} is the average LB improvement achieved by a branching in the explored portion of the tree.

In the calculation of Δ_{avg}, pruned nodes (i.e. with a LB greater than or equal to the BKS) are considered to have an improvement equal to the gap of their parent nodes. The ratio $\text{gap}(i)/\Delta_{avg}$ is the expected depth of the subtree rooted at node i, considering that all branchings in that tree will keep improving bounds in both children nodes by Δ_{avg} units. The Expected Tree Size (ETS) is then obtained by adding the Current Tree Size (CTS) to the RTS. Finally, ETS% = CTS/ETS.

To illustrate the computation of RTS, consider the partial tree with 7 nodes depicted in Figure 1. The BKS is 100 in this example, and each LB value is given next to its corresponding node. The open nodes are $\mathcal{O} = \{4, 5, 7\}$. For each non-root node i, we provide the values of $\text{gap}(i)$, $\Delta(i)$, and the remaining tree size below it. Note that $\Delta(6) = 7$, even if the LB of node 6 is possibly larger than 100. Hence, $\Delta_{avg} = 3$, resulting in RTS = 16.35 and ETS = 7 + RTS = 23.35. If the



i	$\text{gap}(i)$	$\Delta(i)$	$2^{\frac{\text{gap}(i)}{\Delta_{avg}}+1} - 2$
2	8	2	–
3	7	3	–
4	5	3	4.35
5	6	2	6
6	0	7	–
7	6	1	6
		$\Delta_{avg} = 3$	RTS = 16.35

Figure 1: An example of computing the RTS.

run was stopped at that point by time limit, then ETS% = 7/23.25 = 30%.

Some previous work proposes more advanced statistical models for estimating the B&B tree size [CKL06, HALBP21]. Nevertheless, it was thought advisable to adopt the simpler and more intuitive calculation mentioned above, thus justifying our choice of using the ETS% at some fixed time limit for assessing how different branching strategies are close to solving certain hard instances.

2.2. Branching Score

Branching Selection methods that employ strong branching use score functions to transform pairs of LB improvements into a single number, and the larger the score the better the branching. We employ the following adaptation of the product score function [Ach09] not only for choosing candidates in strong branching but also for evaluating branching strategies. For a certain node j of the search tree where branching had been performed, let $l(j)$ and $r(j)$ denote its left and right child, respectively. Moreover, let gap(1) denote the gap of the root node with respect to the BKS. Then,

$$\nabla^l(j) = \frac{\Delta(l(j))}{\text{gap}(1)} \times 100\%, \qquad \nabla^r(j) = \frac{\Delta(r(j))}{\text{gap}(1)} \times 100\%,$$

denote the percentage of the root gap closed in the left and right children. The overall score of the branching at j is given by:

$$\nabla(j) = \sqrt{\max\{\nabla^l(j), \epsilon\} \cdot \max\{\nabla^r(j), \epsilon\}}, \tag{2}$$

where ϵ is a small constant. In addition to scaling the improvements with respect to the root gap, Equation (2) differs from that in [Ach09] because we take the square root of the product of the improvements. This does not change the ranking of candidates in the strong branching but allows $\nabla(j)$ to be interpreted as a geometric mean. For example, a branching that leads to improvements of 20% and 5% is considered as good as a branching producing an improvement of 10% in both children. Given a partial or complete search tree, it is possible to compute ∇_{avg}, the average $\nabla(j)$ over all nodes j where branching was performed. In the tree depicted in Figure 1, gap(1) = 10, $\nabla(1) = 24.49\%$, $\nabla(2) = 24.49\%$, and $\nabla(3) = 26.46\%$, thus resulting in $\nabla_{avg} = 25.15\%$.

3. Cluster Branching and its Motivation: the CMT13 instance

Instance `CMT13` is the last open in the very classic 14-instance benchmark set proposed in [CMT79]. Such an instance is associated with the Distance-constrained CVRP (DCVRP), a CVRP variant that includes a limit for the maximum travel distance in a route. Despite its modest size, even the most advanced BCP algorithms failed to solve `CMT13` in a reasonable time. The topology of that Euclidean instance, where the depot is represented by a yellow square, and its BKS are shown in Figure 2. It can be seen that the customers are split into six well-defined and very separated clusters. We define the depot alone as the cluster C_0. Cluster 1 (C_1) comprises the customers in the cluster surrounding the depot. The remaining clusters are numbered from C_2 to C_6 in clockwise order. As can be seen in Table 1, which presents an analysis of the BKS with respect to that cluster structure, the number of intra-cluster edges in it is significantly higher than the edges between clusters. However, the average cost of those inter-cluster edges is approximately 11.6 times larger.

Table 1: Analysis of the CMT13 BKS with respect to the cluster structure depicted in Figure 2.

Feature	**Value**	**Feature**	**Value**
Number of customers	120	Total cost of inter-cluster edges	1196.16
Total cost of intra-cluster edges	344.98	Number of inter-cluster edges	30
Number of intra-cluster edges	101	Avg. cost of inter-cluster edges	39.87
Avg. cost of intra-cluster edges	3.42	Total cost	1541.14

Why is EB ineffective in a BCP algorithm for CMT13? Let us examine:

- Branching on intra-cluster edges results in small LB increases in both child nodes. This happens simply because such edges have small costs and can be easily replaced in a fractional solution by other intra-cluster edges with almost the same cost.

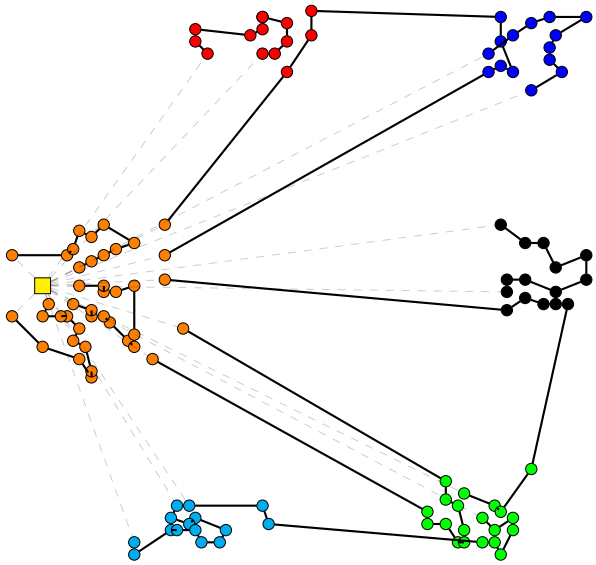


Figure 2: `CMT13` DCVRP instance and its BKS.

- Inter-cluster edges have large costs. Consider an edge $e = \{i, j\}$, $i \in C_1$, $j \in C_2$. Fixing x_e to 1 is likely to be effective, resulting in significantly increased LBs. However, fixing the same x_e to 0 is not likely to be effective. This happens because there are many other edges $e' = \{i', j'\}$, $i' \in C_1$, $j' \in C_2$, that are almost "parallel" to e with nearly the same cost. Therefore, a new fractional solution that uses e' instead of e will yield almost the same LB. As branching is only effective if it works on both child nodes, the overall result is still poor.

CSB over sets $S \subset V$ provided by the procedure in CVRPSep, which looks for sets where the fractional part of ω_S is close to a given target (and completely disregards the cluster structure), while better than EB, is still poor. RFB is in principle more effective. However, it complicates the pricing subproblem, making it prohibitively expensive after some such branchings are performed. Hence, RFB is not a practical alternative.

CB was first devised as an attempt to improve the branching quality on `CMT13`. It can be described as follows. Let $\mathcal{C} = (C_0, \ldots, C_m)$ be a clustering of an instance such that $C_0 = \{0\}$ and $(C_1, \ldots, C_m)$ is a partition of V. Define the following variables for branching:

- ω_{C_k}, $k \in \{0, \ldots, m\}$. They are similar to those used in CSB, and the difference is that in CB only the sets S corresponding to the clusters are used.

- $\psi_{C_k C_l} = \sum_{e \in \delta(C_k, C_l)} x_e$, $k, l \in \{0, \ldots, m\}, k < l$. They correspond to the sum of the edge variables between a pair of clusters.

The experiment on `CMT13` was performed over the CVRP/DCVRP VRPSolver application presented in [PSUV20] (see Appendix A.2). The BCP algorithm uses a two-phase hierarchical strong branching approach. In the first phase, up to 100 candidates are roughly evaluated. The best

three candidates from this phase are then more precisely evaluated in the second phase. Both phases rank candidates using the score specified in Equation (2). The number of candidates in each phase may be reduced at non-root nodes, depending on the expected size of the subtree at that node. In addition, at non-root nodes, half of the candidates in the first phase are selected based on pseudo-costs, increasing the likelihood that candidates with good previous scores will be evaluated again.

The experiment consisted of including the 7 variables ω_{C_k} and the 21 variables $\psi_{C_kC_l}$ corresponding to the clustering exhibited in Figure 2 (note that only the CB variables with fractional value in the current solution are included) among the phase one candidates. The candidate list for the first phase is filled using the existing branching types, so the total number of evaluated candidates does not change. An Upper Bound (UB) of 1541.15 (BKS value + 0.01) was given in all tests, which were performed using a single core of machines with Intel Xeon Gold 6240 processors running at 2.60 GHz.

As can be seen in Table 2, CB always substantially improved the BCP performance and allowed for proving (starting from the root LB of 1471.88) that the BKS is indeed optimal in less than 2.5 hours. Column #**nds** represents the number of explored nodes at the end of the execution, with the corresponding time recorded in column **Time**. For EB and CSB, the execution was stopped due to the time limit of 24 hours. As for RFB, the execution was stopped prematurely because VRPSolver automatically detected that the dynamic programming labeling algorithm was about to crash due to memory overflow. The columns below #**Branchings** show how many times each type of branching was selected by the strong branching mechanism. Column ∇_{avg} reports the average branching scores observed. Finally, columns **ETS**, **ETS%**, and **ETime** are estimations of the total tree size, the fraction of that tree that was already explored, and the total BCP time, respectively. The last estimate assumes that nodes that were not explored will take about the same time as the nodes already explored, which is only reasonable for robust branching. Hence, we do not report the **ETime** for RFB. Additional information about the experiment is given in the branching score charts in Figure 3. The charts show, for each node j where branching had been performed, a point corresponding to the pair $(LB(j), \nabla(j))$. The color and style of the point identify the branch type. The charts also contain lines corresponding to ∇_{avg}. We discuss the results shown in the charts as follows:

- The CVRP/DCVRP VRPSolver application, as presented in [PSUV20], adopts EB with an enhancement: it includes in the candidate list the (robust) branching over the variable $\rho = \sum_{r \in P} \lambda_r$, which corresponds to the number of routes. Notably, $\rho = \omega_{C_0}$, making this approach a form of CB. To avoid worsening existing algorithms, we retained the Number of Routes Branching (NRB) in our tests. As shown in the score charts, NRB was selected at the root node for both EB and RFB, achieving $\nabla(1) = 27.31\%$, a score significantly higher than any obtained by an actual edge or RFB.

- The chart corresponding to CSB shows that it achieved $\nabla(1) = 50.94\%$. This excellent score at the root node was obtained by branching over a variable ω_S where $S = C_3$. This means that the CVRPSep procedure, which identifies sets S based on fractionality, included by chance a very good CB candidate in the list of 100 candidates evaluated at the root node. This did not happen again in subsequent branchings.
- As indicated in Table 2, combining CB with other strategies significantly enhanced the overall branching quality, resulting in a notable increase in the ∇_{avg} metric to over 10%.

Table 2: BCP solution of CMT13 with different branch types.

Type	**#nds**	**Time**	**#Branchings**				∇_{avg}	**ETS**	**ETS%**	**ETime**
			EB	CSB	RFB	CB	(%)			
Edge	655	24h	326	–	–	–	4.37	977K	0.07	4 years
CSB	511	24h	–	254	–	–	3.46	13626	3.75	26.7 days
RFB	37	8h35	–	–	17	–	6.01	2530	1.46	–
Edge + CB	79	2h30	5	–	–	34	10.79	79	100	2h30
CSB + CB	55	1h56	–	1	–	26	13.38	55	100	1h56
RFB + CB	73	2h31	–	–	2	34	11.45	73	100	2h31

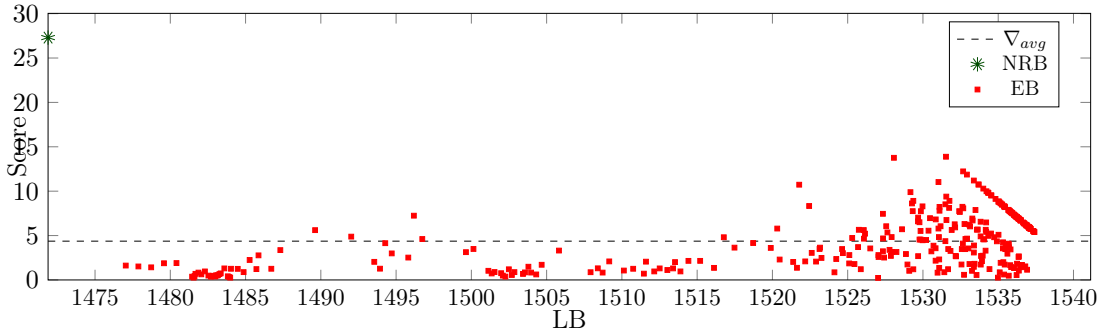
4. Clustering Methods for CB

The promising results achieved on `CMT13` by means of manual clustering motivated us to explore CB on other instances. To this end, we needed to select suitable clustering methods to automatically obtain the partition $\mathcal{C}$.

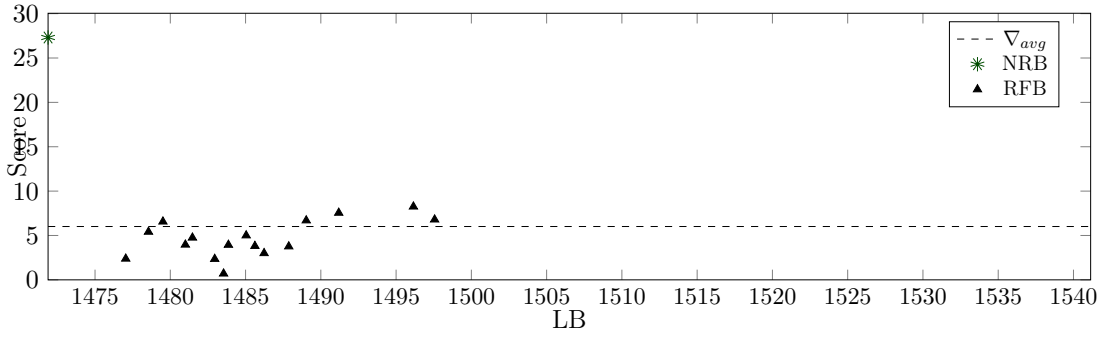
4.1. Minimum Spanning Tree Clustering

The first clustering method tested is based on the Minimum Spanning Tree (MST) approach proposed in [GR69], which is closely related to the classic single-linkage hierarchical clustering. To begin, one computes an MST T of the complete graph $H = (V, E_H)$, where edge costs are equivalent to distances between customers. Next, by removing from T the edge-set $T(m-1)$ formed by the $m-1$ most costly edges, the connected components of $(V, T \setminus T(m-1))$ define m clusters of customers. MST-based clustering is not considered to be a good choice for most applications because it can result in long thin clusters where elements at the opposite ends of those clusters may be very distant. However, as will be demonstrated, it turned out to be an excellent choice for CB. In this context, where the desired number of clusters is unspecified, we decided to remove from T edges $T(\Theta)$ with costs exceeding the threshold value

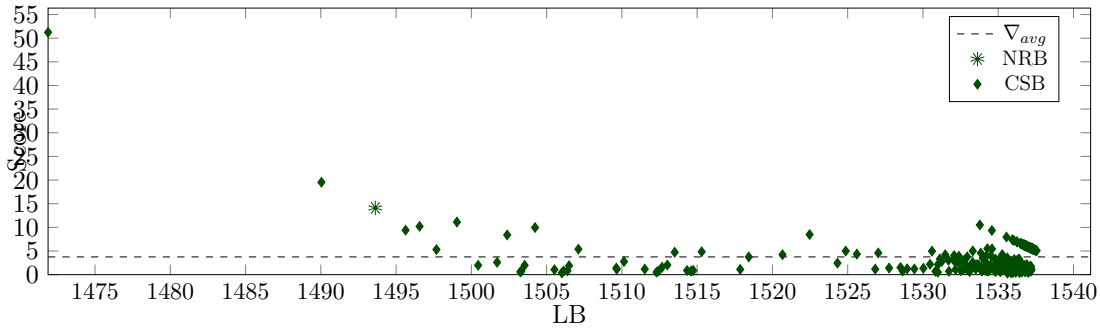
$$\Theta = \text{avg}(T) + \vartheta \, \text{std}(T), \tag{3}$$



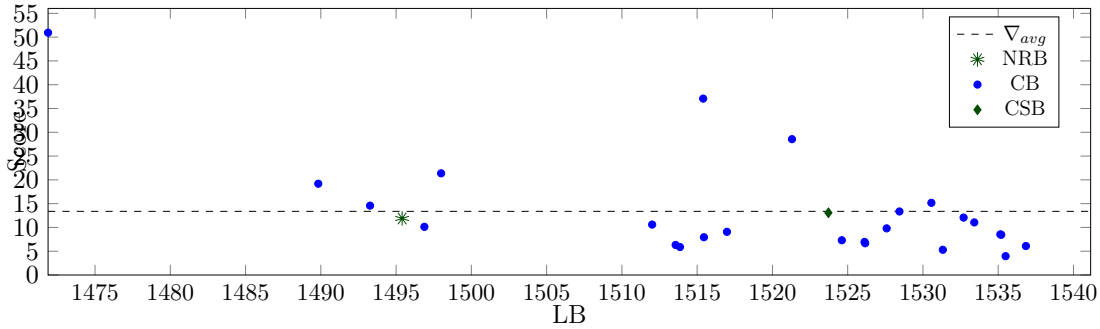
(a) Edge branching



(b) R&F branching



(c) CutSet branching



(d) CutSet + Cluster branching

Figure 3: Branching quality charts of the experiments on the `CMT13`

.

where avg(T) is the average cost of an edge in T, std(T) is the standard deviation of such costs, and ϑ is a parameter. We experimented with $\vartheta \in \{0.5, 1.0, 1.5\}$. The CB with clusters obtained by the MST method with ϑ is referred to as MSTϑ.

4.2. K-means, K-medoids, DBSCAN Clustering

Cluster analysis comprises a wide range of methods and algorithms. Among these, the K-means algorithm [Llo57] is the most widely recognized. Another classic method is the K-medoids algorithm [KR87], which is noted for its robustness to noise and outliers compared to K-means. Another frequently cited method is the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm [EKSX96]. The K-means and K-medoids algorithms fall into the category of centroid-based clustering, where each cluster is represented by a central vector. In contrast, DBSCAN belongs to the family of density-based clustering techniques, defining clusters as regions of higher density and identifying items situated within sparser regions as noise.

In the MST-based clustering discussed in Section 4.1, the number of clusters is automatically determined by removing the edges with weights greater than or equal to the threshold value. Nevertheless, for the aforementioned clustering methods, additional parametrization is required, which directly influences the number of clusters. For both K-means and K-medoids, the number of clusters must be explicitly specified. On the other hand, DBSCAN requires the definition of two hyperparameters: the maximum distance (ε) between two points for them to be considered neighbors, and the minimum number of points ($MinPts$) required to form a dense region. In our experiments, the $MinPts$ parameter was set to 3. Moreover, for the number of clusters and the ε hyperparameter, various values were tested, and the clustering with the highest Silhouette Score (SS) [Rou87] was selected as the preferred clustering for each method on each test instance.

Figure 4 illustrates examples of the clustering obtained by these algorithms over three instances with different customer positions from the XML100 benchmark [QSUV21], namely: `XML100_1135_19` (Random), `XML100_1226_01` (Clustered), and `XML100_3362_03` (Random-Clustered). The SS is displayed at the bottom right of each plot, along with the number of clusters obtained. For K-means, the centroid of each cluster is also depicted; while for K-medoids the center of each cluster (which is a customer) is depicted.

5. Computational experiments

The experiments were conducted on a server equipped with dual 18-core Cascade Lake Intel Xeon Skylake Gold 6240 processors operating at 2.60 GHz. Each such processor has 192 GB of RAM, which in our experiments were shared among nine single-thread parallel copies of the BCP algorithm, each one running a different instance. The cores of such a cluster are not particularly fast by today's standards, having a Passmark single-thread score of 1990; while a typical modern laptop processor has Passmark single-thread scores above 4,000. However, using up to five such

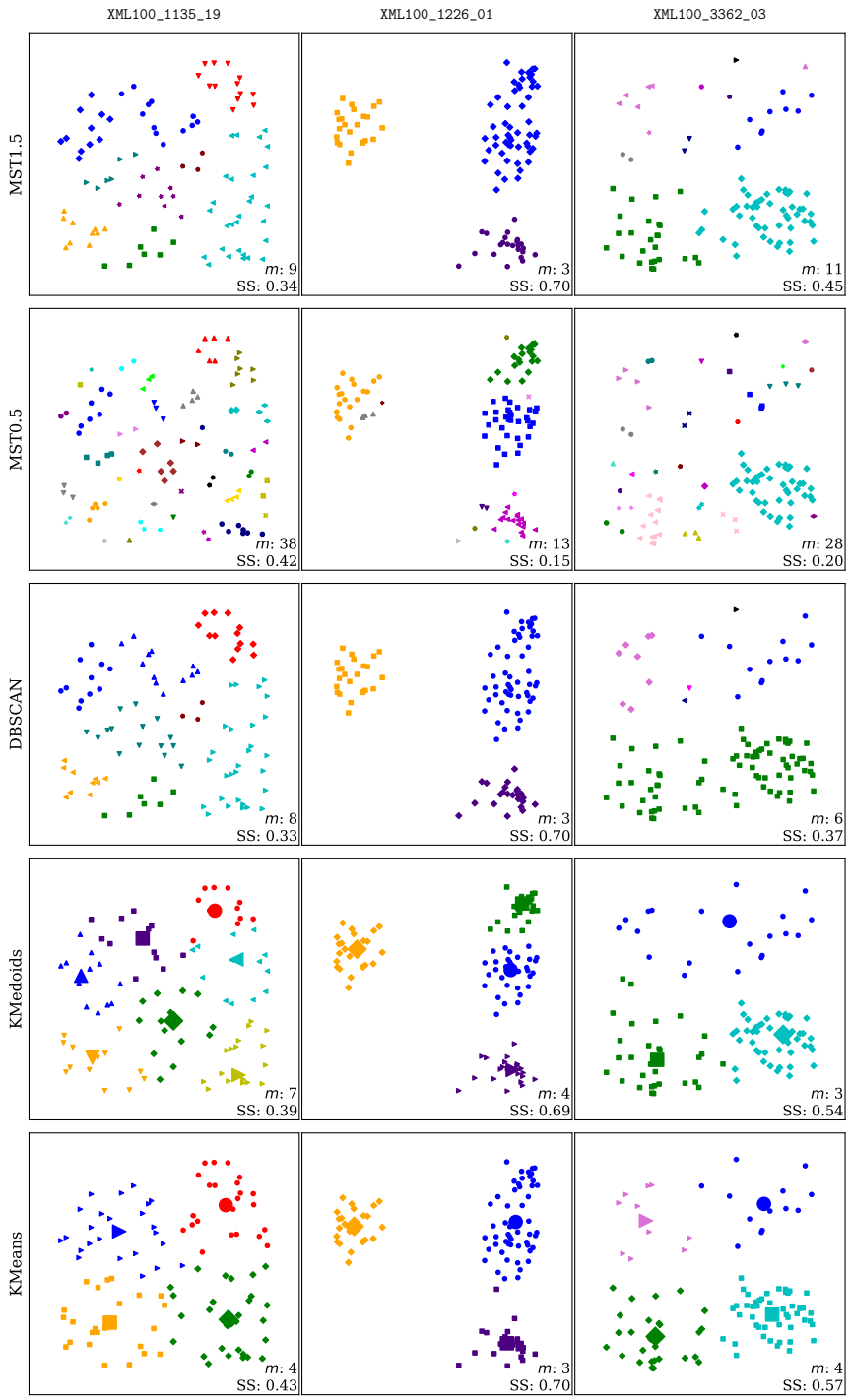


Figure 4: Clustering plots for the instances `XML100_1135_19`, `XML100_1226_01`, and `XML100_3362_03`.

processors in the referred server was very convenient for our extensive experiments, due to its capacity to run dozens of instances in parallel.

The experiments were carried out on three benchmark sets, namely: XML [QSUV21], X [UPP+17], and Homberger and Gehring [GH99]. The former two are for the CVRP, while the latter is for the VRPTW. The experiments on these sets were performed using the CVRP and VRPTW VRPSolver applications presented in [PSUV20] (see Appendix A.2). This section presents a summary of the computational experiments, with the detailed results provided in the supplementary material.

The BCP algorithms were coded in C++ using the BaPCod C++ library [SV21]. This BCP framework is combined with the C++ implementations described in [SUP21], which contain: (i) a labeling algorithm for solving the pricing subproblems based on bucket graphs; (ii) path enumeration; (iii) a bucket arc elimination routine; (iv) a routine for separating limited-memory rank-1 cuts. CPLEX 20.1 is used to solve the LP relaxations and the MIP over the enumerated paths.

BaPCod employs a strong branching technique similar to that of [PSU18]. It also offers the possibility of using one of, or combining, two node search strategies: "breadth-first" and "depth-first". The combination of both consists of a primary and a secondary B&B tree. The search begins by exploring the primary tree using a breadth-first search strategy. When the primary tree reaches a given number of open nodes, newly created nodes are pushed to the secondary tree. The secondary tree is always explored in a depth-first fashion, and when it becomes empty, the search returns to the primary tree.

Since the breadth-first strategy for the primary tree prioritizes exploring the open node with the smallest LB, it results in a tree with more nodes at the top levels. This characteristic is beneficial for analyzing branching decisions, as exploration occurs in different parts of the tree, rather than proceeding deeper along the same branch direction, which is typical in depth-first strategy.

5.1. XML instances

The XML benchmark [QSUV21] is a set of 10,000 CVRP instances with 100 customers. This set was carefully designed, extending the generation scheme in [UPP+17], to cover a wide variety of instance characteristics, more specifically:

- *Customer positioning*: **1.** Random (Rd); **2.** Random-Clustered (RC); **3.** Clustered (Cl);
- *Depot positioning*: **1.** Centered (Ct); **2.** Random (Rd); **3.** Cornered (Cr);
- *Demand distribution*: **1.** Unitary (Un); **2.** Small values with large Coefficient of Variation (CV) (SL); **3.** Small values with small CV (SS); **4.** Large values with large CV (LL); **5.** Large values with small CV (LS); **6.** Depending on quadrant (DQ); **7.** Many small values and few large values (MF);

- *Average route size* r: **1.** very short (VS), r from $U[3,5]$; **2.** short (Sh), r from $U[5,8]$; **3.** medium (Md), r from $U[8,12]$; **4.** long (Lg), r from $U[12,16]$; **5.** very long (VL), r from $U[16,25]$; **6.** ultra long (UL), r from $U[25,50]$.

The tested BCP algorithm could solve 7585 of the XML instances in the root node. Therefore, the experiments with branching were conducted on the remaining 2415 instances. All tests on the same instance used the same external UB value obtained by the heuristics described in [Vid22]. The time limit was set to 1 hour.

Table 3 presents the results comparing EB, CSB, and RFB, along with their combinations using CB, across several clustering methods. The latter include MST0.5, MST1.0, MST1.5, Kmeans, Kmedoids, and DBSCAN. Given that RFB is the only non-robust branching strategy, thus requiring more computational effort, we decided to test it exclusively with MST1.0. Hence, the tests were conducted for 16 different branch types, resulting in a total of 38,640 instance tests.

Table 3 includes the column #**Solved**, displaying the number of instances solved, alongside columns for the average score ∇_{avg} and estimated explored tree size **ETS%**. The results are provided for the set of 2415 instances and the subset formed by the 536 instances not solved by any robust strategy, which can be considered the hardest ones. Note that the filter does not include RFB because the analysis would be dominated by instances not solved by this strategy. If RFB were included, 761 instances would be analyzed, representing at least 30% of instances unsolved solely by RFB, potentially inflating the statistics for the robust branching strategies.

MST1.0 was selected as a baseline clustering method for a detailed comparison between the robust strategies EB and CSB, as well as their respective combinations with MST CB. The value $\vartheta = 1.0$ was chosen arbitrarily to avoid overfitting to a parameterization that might produce better results. The detailed results are exhibited in Table 4, which presents the same data as shown in Table 3 for the robust strategies and their combinations with CB MST1.0, but now including disaggregated statistics based on the instance characteristics. Furthermore, the results of a statistical test are also provided.

Regarding the statistical test, the non-parametric paired left-tailed Wilcoxon signed-rank test [Wil45] was conducted to assess the differences in average scores between the robust branching strategies and their combinations with CB MST1.0. This choice was made to analyze scores before and after implementing the proposed CB scheme, requiring a statistical paired test. In addition, a Shapiro-Wilk test [SW65] with a significance level of 5% ($\alpha = 0.05$) confirmed that the scores do not follow a normal distribution, thus justifying the use of a non-parametric test.

The Wilcoxon signed-rank test compares the medians of two dependent distributions by calculating the difference between paired data values and ranking the absolute value of the differences. Under $\alpha = 0.05$, the hypotheses are as follows:

$$\text{Null Hypothesis } (H_0) : \text{The use of CB does not result in an increase in average branching scores.}$$

Table 3: Overall results on XML100. Runs with 1 hour TL.

	Overall (2415 instances)			Not solved by some robust branch type (536 instances)		
Branch type	**#Solved**	∇_{avg}	**ETS%**	**#Solved**	∇_{avg}	**ETS%**
EB	1961	57.89	87.51	82	22.86	43.71
EB + MST0.5	2032	60.18	89.86	153	26.08	53.90
EB + MST1.0	**2041**	60.71	90.06	**162**	27.08	55.21
EB + MST1.5	2028	60.12	89.72	149	26.35	53.48
EB + Kmeans	2026	**60.95**	90.00	147	**27.88**	54.69
EB + Kmedoids	2026	60.14	89.93	147	27.57	54.28
EB + DBSCAN	2034	60.61	**90.30**	155	27.87	**56.02**
CSB	1998	59.84	90.45	119	27.87	56.34
CSB + MST0.5	2067	61.49	91.81	188	30.17	62.63
CSB + MST1.0	2069	62.03	91.97	190	30.37	63.23
CSB + MST1.5	**2073**	**62.07**	**92.03**	**194**	31.06	**63.70**
CSB + Kmeans	2044	61.85	91.68	165	30.83	61.92
CSB + Kmedoids	2045	61.47	91.30	166	30.19	60.39
CSB + DBSCAN	2064	61.43	91.85	185	**31.66**	62.85
RFB	1692	54.62	76.96	31	17.29	20.11
RFB + MST1.0	**1923**	**59.98**	**86.25**	**116**	**25.59**	**44.65**

Alternative Hypothesis (H_1) : The use of CB leads to an increase in average branching scores.

Table 4 provides the results of the test statistic W and the corresponding p-value. A larger value of W indicates a greater difference between the paired data points in the direction specified by the alternative hypothesis, while a smaller p-value suggests stronger evidence against the null hypothesis.

Following the presentation of Tables 3 and 4, we discuss the analysis and conclusions drawn from these tests:

- To the best of our knowledge, we have conducted the first extensive comparison between standard EB and CSB, showing that CSB is indeed better on average. Nevertheless, the detailed results in Table 4 suggest that CSB is only better than EB for instances with larger routes, i.e., those from classes VL and UL. In classes VS, Sh, Md, and Lg, both branching types are almost equivalent. This fact remains true even after adding CB, with CSB+MST1.0 being significantly better than EB+MST1.0 only for larger routes.

- From Table 3, it can be observed that *CB can significantly improve the performance of all existing branching types, regardless of the clustering method used among the six tested alternatives.* The best overall results were obtained by CSB + MST1.5, but the baseline

Table 4: Detailed results of EB and CSB before and after using CB MST1.0 on XML100. Runs with 1 hour TL.

Instance group			Edge branching								CutSet branching							
			EB			EB + MST1.0			Statistical test		CSB			CSB + MST1.0			Statistical test	
Feature		**#nb**	**#Sol.**	∇_{avg}	**ETS%**	**#Sol.**	∇_{avg}	**ETS%**	W	p-**value**	**#Sol.**	∇_{avg}	**ETS%**	**#Sol.**	∇_{avg}	**ETS%**	W	p-**value**
Customer	Cl	906	746	58.21	88.10	**777**	**61.69**	**91.04**	4.36e+04	**2.64e–16**	754	58.77	90.67	**791**	**62.84**	**92.83**	4.24e+04	**4.36e–18**
	RC	843	686	58.72	88.04	**717**	**61.33**	**90.62**	4.52e+04	**1.21e–10**	692	60.68	90.30	**725**	**62.22**	**92.44**	4.72e+04	**1.37e–07**
	Rd	1498	529	56.42	86.02	**547**	**58.60**	**88.01**	3.27e+04	**1.61e–08**	552	60.24	**90.34**	**553**	**60.70**	90.19	3.39e+04	**4.51e–03**
Depot	Ct	438	384	63.52	92.41	**397**	**67.32**	**94.50**	8.70e+03	**6.84e–08**	391	66.34	94.83	**407**	**67.90**	**95.73**	9.65e+03	**6.29e–04**
	Rd	1498	677	58.38	87.24	**702**	**61.79**	**89.41**	3.85e+04	**2.83e–14**	686	61.22	89.90	**705**	**63.17**	**91.47**	4.17e+04	**2.24e–08**
	Cr	1145	900	55.39	85.83	**942**	**57.41**	**88.83**	9.82e+04	**3.65e–13**	921	56.35	89.16	**957**	**58.97**	**90.89**	9.37e+04	**6.21e–14**
Deamand	Un	87	59	63.77	75.96	**62**	**66.92**	**80.13**	2.72e+02	**2.79e–03**	68	69.87	87.40	**70**	**71.74**	**88.78**	2.69e+02	**4.57e–02**
	SL	261	217	63.80	88.84	**228**	**68.60**	**91.07**	2.44e+03	**1.44e–08**	221	64.79	92.13	**232**	**69.33**	**94.07**	2.80e+03	**7.21e–07**
	SS	330	270	60.70	87.42	**278**	**63.60**	**89.43**	5.92e+03	**1.61e–05**	280	64.27	90.93	**285**	**65.67**	**91.62**	7.15e+03	**2.24e–03**
	LL	419	358	58.36	90.78	**370**	**60.26**	**92.80**	1.23e+04	**5.12e–05**	360	59.24	92.47	**368**	**61.18**	**93.75**	1.17e+04	**1.29e–05**
	LS	388	316	56.37	87.02	**327**	**59.26**	**89.38**	1.07e+04	**4.31e–06**	317	58.49	89.69	**329**	**60.07**	**91.08**	1.11e+04	**4.20e–04**
	DQ	527	427	53.11	88.34	**451**	**56.29**	**91.62**	2.15e+04	**1.90e–08**	435	54.43	90.94	**457**	**56.88**	**92.81**	2.11e+04	**1.91e–07**
	MF	403	314	57.74	85.19	**325**	**59.55**	**87.83**	1.00e+04	**1.31e–05**	317	59.84	87.63	**328**	**61.76**	**89.48**	9.36e+03	**8.08e–04**
Route	VS	270	263	68.24	98.33	**268**	**70.65**	**99.66**	3.28e+03	**2.86e–04**	263	67.76	98.27	**269**	**70.46**	**99.85**	3.32e+03	**1.72e–04**
	Sh	470	447	62.92	97.48	**456**	**64.35**	**98.57**	1.54e+04	**3.76e–03**	444	62.68	96.95	**457**	**64.34**	**98.65**	1.30e+04	**2.96e–06**
	Md	397	360	60.81	94.93	**369**	**61.60**	**96.19**	1.18e+04	**1.74e–02**	361	60.04	95.12	**367**	**62.59**	**95.61**	9.90e+03	**7.78e–06**
	Lg	347	311	62.99	94.28	**320**	**66.27**	**95.18**	6.44e+03	**1.20e–05**	315	64.64	94.76	**321**	**66.65**	**95.36**	6.50e+03	**3.45e–04**
	VL	464	355	51.51	86.07	**376**	**55.02**	**89.54**	1.49e+04	**6.27e–11**	358	54.82	88.72	**381**	**56.98**	**91.59**	1.50e+04	**1.12e–07**
	UL	467	225	46.92	61.29	**252**	**52.09**	**67.46**	1.20e+04	**2.38e–17**	257	53.64	73.57	**274**	**55.95**	**75.06**	1.78e+04	**1.36e–03**
Overall		2415	1961	57.89	87.51	**2041**	**60.71**	**90.06**	3.63e+05	**2.15e–31**	1998	59.84	90.45	**2069**	**62.03**	**91.97**	3.70e+05	**1.06e–22**

CSB + MST1.0 was almost as good.

- Table 4 shows that *the improvement achieved by combining CB MST1.0 with EB and CSB is consistent across all instance groups.* The statistical test obtained strong p-values on all scenarios, indicating that the null hypothesis is rejected. This rejection implies that there are significant differences between the compared methods, confirming the effectiveness of the proposed enhancements. The consistent rejection of the null hypothesis across varied instance groups ratifies the robustness of the improvements introduced by combining CB with EB and CSB.

We now discuss the performance of CB according to the instance characteristics:

- Customer positioning. As expected, CB brings large improvements in instances where the customers are clustered. The null hypothesis that CB does not improve EB or CSB is rejected with highly significant p-values of 2.64e–16 and 4.36e–18, respectively. For random-clustered instances, where half of the customers are clustered and half are random, p-values are still very significant. However, *CB surprisingly yields modest yet still statistically significant improvements even in instances where the customers are random.* As can be seen in Figure 4, when 100 customers are randomly positioned, there may still exist some "natural clusters" (blocks of customers that are well-separated), a structure that may be captured by CB.

- Depot positioning. CB achieves considerable improvements for all three classes of instances, however, p-values are more significant when the depot is placed in the corner, when compared to the cases in which the depot is randomly located or placed in the center.

- Demand distribution. CB attains systematic improvements over all seven classes of instances. The smallest CSB p-value of 4.57e–02 (barely below the 5% bar) are associated with the instances with unitary demand. The explanation here is that unitary demand instances are much easier to current BCP algorithms than other instances (see [UPP+17]), hence providing less room for improvements due to CB, at least for instances with 100 customers.

- Route Size. CB leads to consistent improvements over all six classes of instances. Interestingly, for ultra-long routes, CB achieves the highly significant p-value of 2.38-e17 for EB, but only a p-value of 1.36-e03 for CSB. As mentioned before, CSB is much better than EB for this class of instances, leaving less opportunities for further improvements by CB.

5.2. X instances

The X benchmark instances [UPP+17] comprise 100 CVRP instances, ranging from 100 to 1000 customers and encompassing three distinct customer positioning scenarios: random, clustered, and random-clustered. Instances solved at the root node (considered easy) and those requiring over

two hours to solve the root node (designated as hard) were excluded from the analysis of the proposed strategy. After applying these filters, 70 instances remained for further examination.

The same UBs were applied to each instance across all tests. These UBs are the BKS from the CVRPLib[1] plus one, with a time limit of 24 hours per instance. The primary tree was limited to 100 open nodes. The 24-hour limit per instance was set due to the increased complexity of the X benchmark instance set.

Table 5 reports the results of the tests on the X set in terms of number of instances solved, along with the average score (∇_{avg}) and the explored tree size estimation (**ETS%**). The results are disaggregated by customer positioning and for all instances (column **Overall**). Results for CB with clustering algorithms K-means, K-medoids, and DBSCAN, as well as RFB, are excluded because they yielded inferior performance in previous experiments on the XML benchmark.

Table 5: Results over X benchmark (70 instances). Runs with 24-hour TL.

		∇_{avg}				**ETS%**			
Branch type	**#Solved**	**Overall**	**Cl**	**RC**	**Rd**	**Overall**	**Cl**	**RC**	**Rd**
EB	**25**	18.86	18.03	24.44	13.31	39.07	36.00	40.78	39.95
EB + MST0.5	**25**	19.81	18.48	26.19	13.80	39.82	38.42	43.38	37.08
EB + MST1.0	**25**	19.85	16.42	**27.83**	**13.95**	40.58	**39.11**	**44.06**	37.98
EB + MST1.5	**25**	**19.91**	**18.69**	26.17	13.94	**41.08**	38.37	44.05	**40.20**
CSB	**26**	19.79	19.53	24.32	**14.89**	40.99	38.99	41.56	**42.16**
CSB + MST0.5	25	19.50	19.10	25.33	13.28	41.61	**41.73**	44.39	38.36
CSB + MST1.0	25	**20.16**	19.27	**26.72**	13.56	**42.37**	40.38	**46.07**	40.02
CSB + MST1.5	25	19.61	**19.62**	24.69	13.87	41.46	40.04	44.65	39.14

The comparison between standard EB and CSB reveals some advantages to the latter. CSB could solve one instance (X-n331-k15) not solved by EB and has better ∇_{avg} and $ETS\%$ statistics for all three classes of customer positioning. Moreover, one can observe that MST CB, for any value of ϑ, gives overall improvements for both EB and CSB. However, by looking at the statistics disaggregated by customer positioning it is possible to verify that on pure CSB is better than CSB + MST on the Rd instances. It is worth mentioning that instance X-n331-k15 belongs to that class.

The discussion above brings us to the following question: why CB works on random XML instances but not on random X instances? We believe that this is related to the instance sizes. On the one hand, the XML instances have only 100 customers, meaning that "natural clusters" appear even when customers are randomly positioned, a structure that favors CB. On the other hand, as many times more points are randomly positioned on the X instances, there are no natural clusters and CB becomes unfavorable.

[1] http://vrp.galgos.inf.puc-rio.br/index.php/en/

5.3. Homberger and Gehring instances

In the context of the VRPTW, experiments were conducted on the Homberger and Gehring benchmark set [GH99]. This set includes instances with short and long routes, ranging from 200 to 1000 customers, categorized into clustered, random, and random-clustered scenarios, resulting in a total of 300 instances. However, after applying the same filter as for the X benchmark, 93 instances remained for further examination.

The same UB was applied as a cutoff to each instance in all tests. These UBs were derived by adding 0.1 to the BKS reported on the CVRPLib, with a time limit of 24 hours per instance. The results are presented in Table 6 and include the same statistics as Table 5 for the X benchmark.

Table 6: Overall results on Homberger and Gehring benchmark. Runs with 24-hour TL.

		∇_{avg}				**ETS%**			
Branch type	**#Solved**	**Overall**	**Cl**	**RC**	**Rd**	**Overall**	**Cl**	**RC**	**Rd**
EB	**31**	18.49	26.20	15.53	**17.24**	39.71	61.60	31.07	**36.36**
EB + MST0.5	**31**	19.86	30.88	19.10	14.48	40.58	60.64	34.16	35.98
EB + MST1.0	**31**	**19.91**	31.52	**19.31**	14.03	**41.55**	60.68	**36.15**	36.25
EB + MST1.5	**31**	19.70	**32.42**	17.88	14.52	41.03	**61.98**	34.88	35.65
CSB	29	20.14	34.17	18.42	14.00	40.87	61.54	34.86	**35.32**
CSB + MST0.5	**30**	21.13	36.07	**19.77**	14.15	40.62	59.65	35.69	34.92
CSB + MST1.0	**30**	**21.62**	**37.40**	19.23	**15.11**	**41.51**	62.92	**37.14**	33.80
CSB + MST1.5	29	20.26	35.50	18.12	13.94	40.87	**63.73**	34.78	34.36

By comparing standard EB and CSB, it can be seen that the former is better in the Rd class, while the latter is better in the Cl and RC classes. Once again, MST CB provides overall improvements for both EB and CSB, regardless of the value of ϑ. Nevertheless, when analyzing the statistics disaggregated by customer positioning, we can verify that the sole application of EB is better than EB + MST on the Rd instances.

While there are advantages of applying the current CB to non-random VRPTW instances, perhaps better results could be obtained by clustering methods that somehow consider the time windows. For example, two spatially close customers with very disjoint time windows are potentially better viewed as belonging to different clusters.

5.4. Long runs

In this section, we present the results of additional long runs involving some "promising" CVRP and VRPTW open instances, i.e., those where the 24-hour ETS% forecasts that the instance may be solved by spending extra days of CPU time. Two of these long runs extended for several months (an exceptional effort to close the last Homberger and Gehring instances with 200 customers). Such very long runs were only possible because we could run the subtrees rooted at some open nodes

of the same instance in parallel, using different processors and cores of the server. For each open instance solved, a customized BCP parameterization was employed.

Before presenting the results of the long runs, we list the instances and their results from the 24-hour experiments across various CB methods. The CVRP instances selected for the extended run are displayed in Table 7, while the VRPTW instances are given in Table 8. These tables show the ∇_{avg} and ETS% results from the 24-hour CB experiments for each MST-based clustering method, as well as for the standard EB and CSB.

Among the CVRP instances, three (`X-n256-k16`, `X-n351-k40`, and `X-n367-k17`) have clustered customer positioning, while `X-n344-k43` is randomly clustered, and `X-n670-k126` has randomly positioned customers. Only the last instance was not improved by combining EB or CSB with CB. Also, the performance of the pure version of EB was slightly better than the pure CSB test. We believe this happens because it is an example of a large random instance with no "hidden" cluster structures, and consequently, CB and CSB do not have the desired effect.

Table 7: Results for some open X instances. Runs with 24 hour TL.

	ETS%					∇_{avg}				
Branch type	X-n256-k16	X-n344-k43	X-n351-k40	X-n367-k17	X-n670-k126	X-n256-k16	X-n344-k43	X-n351-k40	X-n367-k17	X-n670-k126
EB	0.89	6.70	9.03	9.04	**39.26**	6.09	9.07	9.04	11.17	**15.04**
EB + MST0.5	7.61	**12.72**	16.19	13.89	14.45	8.16	**9.28**	**9.57**	12.05	12.40
EB + MST1.0	**8.86**	7.96	**20.57**	22.60	22.91	**8.56**	8.55	9.35	**12.68**	14.22
EB + MST1.5	5.60	11.76	14.24	**24.49**	17.62	7.77	9.20	8.98	11.47	14.08
CSB	5.79	7.04	13.82	14.12	**38.76**	8.34	8.58	**9.99**	15.29	**17.28**
CSB + MST0.5	**11.38**	12.90	**21.97**	17.33	21.81	7.71	9.78	8.94	13.36	13.75
CSB + MST1.0	10.86	**21.70**	19.13	18.22	25.36	8.35	**9.79**	9.27	12.80	11.05
CSB + MST1.5	11.00	13.76	16.02	**19.46**	26.32	**8.70**	9.16	8.66	**16.44**	9.11

A similar behavior is observed for the random instances `R1_4_5` and `R1_4_6` in Table 8. In these cases, the CSB version was not improved by CB, while in the EB, only `R1_4_6` was slightly improved by combining it with CB MST1.0. Note that, in both instances, the pure EB outperformed CSB and any combination of it with CB MST-based clustering.

Tables 9 and 10 present the results of the long runs for the CVRP and VRPTW instances, respectively. For these extended runs, only one strategy — either EB+CB, CSB+CB, pure EB, or pure CSB — was considered. Although Tables 7 and 8 identify the best configurations in terms of branching strategy, these optimal configurations were not always applied in the long runs. The choice of branching strategy during the long runs followed the progression of the research, during which some of the previously shown strategies or evaluation metrics were not yet considered.

Table 8: Results for some open VRPTW Homberger instances. Runs with 24-hour TL.

	ETS%										∇_{avg}									
Branch type	RC1_2_9	RC1_2_10	C2_2_4	RC2_2_10	RC1_4_1	RC1_4_2	R1_4_2	R1_4_5	R1_4_6	C1_6_10	RC1_2_9	RC1_2_10	C2_2_4	RC2_2_10	RC1_4_1	RC1_4_2	R1_4_2	R1_4_5	R1_4_6	C1_6_10
EB	20.63	40.11	2.43	**3.14**	10.34	11.46	26.75	**38.84**	58.85	6.26	10.75	13.8	8.08	**9.18**	3.79	10.38	**7.90**	8.31	9.87	9.71
EB+MST0.5	25.42	61.71	3.22	2.24	12.42	9.94	25.30	30.77	59.51	12.80	11.03	**15.00**	7.88	8.89	**5.65**	11.25	7.46	**8.43**	**10.34**	10.91
EB+MST1.0	24.08	61.22	1.31	2.23	**21.39**	**15.28**	**30.36**	31.94	**59.76**	**18.66**	10.68	13.9	7.70	8.52	4.62	**11.97**	7.69	8.22	9.64	11.00
EB+MST1.5	**32.82**	**62.79**	**3.78**	2.03	18.84	13.17	24.98	24.04	51.71	17.88	**12.17**	13.61	**8.89**	8.76	4.00	11.32	7.51	8.18	9.91	**11.28**
CSB	23.58	50.76	23.18	1.10	10.05	9.40	13.96	**27.90**	**52.05**	6.68	9.41	13.08	11.44	7.97	2.73	10.81	6.80	**7.90**	**9.61**	8.98
CSB+MST0.5	26.32	57.25	5.94	3.31	11.68	12.11	**17.02**	17.33	48.16	9.59	10.51	**13.47**	9.09	8.96	**6.44**	**11.91**	6.96	7.67	9.44	8.81
CSB+MST1.0	**30.96**	**62.29**	9.72	**3.65**	**13.51**	**21.86**	16.91	15.86	31.07	18.00	**10.55**	11.59	9.21	8.91	3.38	10.66	**6.99**	7.29	8.33	**10.67**
CSB+MST1.5	28.20	61.56	**28.60**	3.28	6.04	17.17	12.07	18.91	50.71	**18.68**	10.34	11.63	**12.39**	**9.23**	4.93	11.47	6.80	7.55	9.38	9.69

Tables 9 and 10 show the optimal cost (column **Opt. cost**), the size of the B&B tree (column #**bb**), and the time required to solve the instance (column **t (days)**). These columns provide statistics related to the BCP algorithm. In addition, the last set of columns indicates the branching strategy used, and for those utilizing CB, the number of clusters m obtained by the clustering method is also presented.

By employing the EB+MST1.0, the instance `X-n256-k16` was eventually solved after exploring 2897 nodes, requiring a total CPU time of 35.2 days. Despite the extensive duration, it is worth noting that the ETS% of the standard edge branching, as shown in Table 7, exceeds that of the CB utilizing the MST1.0 by almost a factor of 10. Even if the edge branching were to operate within the same time limit, it would still fall significantly short of solving it.

Regarding the other CVRP instances, the time consumed is acceptable given that these are large and challenging instances for exact methods. In addition, it is surprising to note that `X-n670-k126` was solved in a very reasonable time by applying only the standard edge branching as a branching strategy, which was expected based on the results in Table 7.

Table 9: Results obtained on the CVRP X instances with extended running time.

	BCP			**Branching**	
Instance	**Opt. cost**	#**bb**	**t (days)**	**strategy**	m
`X-n256-k16`	**18839**	2897	35.2	EB+MST1.0	43
`X-n344-k43`	**42050**	897	4.4	EB+MST1.5	30
`X-n351-k40`	**25896**	2569	8.4	EB+MST1.0	37
`X-n367-k17`	**22814**	265	2.8	EB+MST1.0	42
`X-n670-k126`	**146332**	1691	8.3	EB	–

From Table 10, the experiments conducted on problem instances `RC1_2_9`, `RC1_2_10`, `RC1_4_1`, `RC1_4_2`, `R1_4_2`, `R1_4_5`, `R1_4_6`, and `C1_6_10` ratified the positive results anticipated from the 24-hour experiments. However, for `RC2_2_10` and `C2_2_4`, the same level of success was not observed, as both instances required a significantly long time to solve. This outcome was expected, as anticipated in Table 8. Given that both instances involve solutions with very long routes, they naturally pose significant challenges for BCP algorithms. It is worth noting that instances `RC1_2_9`, `RC1_2_10`, `C2_2_4`, and `RC2_2_10` were the last four open instances with 200 customers in the Homberger and Gehring benchmark.

6. Conclusion

In this paper, we introduced a novel branching strategy, Cluster Branching (CB), to enhance the performance of solving Vehicle Routing Problems. Our approach clusters customers based on their spatial positioning and defines additional variables associated with these clusters. These variables, representing aggregated edge variables incident to or connecting clusters, are then incor-

Table 10: Experiment results on the VRPTW Homberger instances with extended running time.

	BCP			Branching	
Instance	**Opt. cost**	**#bb**	**t (days)**	**strategy**	m
RC1_2_9	**3073.3**	1434	5.2	EB+MST1.5	23
RC1_2_10	**2990.5**	562	1.5	EB+MST1.5	23
C2_2_4	**1695.0**	20738	188.6	EB+MST0.5	45
RC2_2_10	**1989.2**	8765	93.8	EB+MST1.0	32
RC1_4_1	**8522.9**	3289	6.8	EB+MST1.0	70
RC1_4_2	**7878.2**	839	7.8	EB+MST1.0	70
R1_4_2	**8873.2**	2355	3.2	EB+MST1.0	61
R1_4_5	**9184.6**	7043	5.8	EB+MST1.0	62
R1_4_6	**8340.4**	1073	2.4	EB+MST1.0	62
C1_6_10	**13617.5**	1940	12.3	EB+MST1.5	43

porated into a B&B algorithm, providing new dimensions for branching. One of the key advantages of this strategy is its simplicity, as the clustering is performed before running the BCP algorithm, eliminating the need to consider, for example, fractional solutions. Furthermore, it is robust in the sense of not complicating the pricing subproblem.

Our extensive computational experiments, conducted on BCP algorithms for Distance-Constrained VRP, Capacitated VRP, and VRP with Time Windows, demonstrate the efficacy of Cluster Branching. The primary evaluation metrics, branching score and estimated explored tree size, both proposed in this study, showed that including CB significantly enhances the performance of the traditional branching strategies. Indeed, CB combined with either Edge Branching, Ryan & Foster Branching, or CutSet Branching consistently yielded better results compared to their standalone versions. This synergy highlights the potential of CB as a powerful enhancement to conventional branching techniques. The only situation where CB was found to not improve results was in large instances where customers were totally randomly located. We remark that such instances are not likely to be found in most practical situations (e.g., in delivery of parcels). Cities have neighborhoods with different urban densities and levels of affluence, so clusters of customers exist.

We also explored various clustering methods, including K-means, K-medoids, DBSCAN, and Minimum Spanning Tree-based clustering. The results indicate that across almost all combinations of clustering methods and CB, there were significant improvements in the analyzed metrics. This robustness across different clustering techniques attests the versatility and general applicability of our proposed strategy.

The implementation of CB led to the proof of optimality of several previously open instances for the three VRP variants, including large instances. In particular, this includes the last four open instances with 200 customers of the classic Homberger and Gehring VRPTW set. This achievement not only validates the effectiveness of CB but also contributes valuable solutions to

the VRP research community.

Future research could further refine CB, explore its integration with other advanced clustering algorithms, such as those based on Machine Learning, that could help find attractive clusters even in instances with randomly positioned customers, and extend its application to other combinatorial optimization problems. In addition, research could focus on developing new branching strategies that consider specific characteristics of the given VRP problem, not only spatial distribution but also factors such as demand, vehicle capacity, and others. By tailoring branching strategies to these unique problem attributes, we can further enhance the optimization process for various VRP variants.

7. acknowledgements

This research was partially supported by the following Brazilian research agencies: Comissão de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Grant [Finance Code 001]; CNPq, grants 406245/2021-5, 309580/2021-8, and 313601/2018-6; Paraíba State Research Foundation (FAPESQ), grant 2021/3182; Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro (FAPERJ), Grant E-26/202.887/2017; CAPES PrInt UFF Nº 88881; as well as by Universidade Federal da Paraíba through the Public Call No. 03/2020 "Produtividade em Pesquisa PROPESQ/PRPG/UFPB", proposal code PVL13400-2020.

References

[AB09] Tobias Achterberg and Timo Berthold. Hybrid branching. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 309–311. Springer, 2009.

[Ach09] Tobias Achterberg. Scip: solving constraint integer programs. *Mathematical Programming Computation*, 1:1–41, 2009.

[AKM05] Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[BCM08] Roberto Baldacci, Nicos Christofides, and Aristide Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115:351–385, 2008.

[BGG+71] Michel Bénichou, Jean-Michel Gauthier, Paul Girodet, Gerard Hentges, Gerard Ribière, and Olivier Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1:76–94, 1971.

[CCD19] L. Costa, C. Contardo, and G. Desaulniers. Exact branch-price-and-cut algorithms for vehicle routing. *Transportation Science*, 53:946–985, 2019.

[CKL06] Gérard Cornuéjols, Miroslav Karamanov, and Yanjun Li. Early estimates of the size of branch-and-bound trees. *INFORMS Journal on Computing*, 18(1):86–96, 2006.

[CM14] Claudio Contardo and Rafael Martinelli. A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. *Discrete Optimization*, 12:129–146, 2014.

[CMT79] N Christofides, A Mingozzi, and P Toth. The vehicle routing problem. In N Christofides, A Mingozzi, P Toth, and C Sandi, editors, *Combinatorial optimization*, pages 325–338. Wiley, Chichester, 1979.

[CN93] Jean-Maurice Clochard and Denis Naddef. Using path inequalities in a branch and cut code for the symmetric traveling salesman problem. In *IPCO*, pages 291–311, 1993.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226—231. AAAI Press, 1996.

[EQSU23] Najib Errami, Eduardo Queiroga, Ruslan Sadykov, and Eduardo Uchoa. VRP-SolverEasy: a Python library for the exact solution of a rich vehicle routing problem. *INFORMS Journal on Computing*, 2023.

[FLL+06] Ricardo Fukasawa, Humberto Longo, Jens Lysgaard, Marcus Poggi de Aragão, Marcelo Reis, Eduardo Uchoa, and Renato F Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical programming*, 106:491–511, 2006.

[GDDS95] Sylvie Gélinas, Martin Desrochers, Jacques Desrosiers, and Marius M Solomon. A new branching strategy for time constrained routing problems with application to backhauling. *Annals of Operations Research*, 61:91–109, 1995.

[GH99] Hermann Gehring and Jörg Homberger. A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In *Proceedings of EUROGEN99*, volume 2, pages 57–64. Citeseer, 1999.

[GR69] John C Gower and Gavin JS Ross. Minimum spanning trees and single linkage cluster analysis. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 18(1):54–64, 1969.

[HALBP21] Gregor Hendel, Daniel Anderson, Pierre Le Bodic, and Marc E Pfetsch. Estimating the size of branch-and-bound trees. *INFORMS Journal on Computing*, 34(2):934–952, 2021.

[JPSP08] Mads Jepsen, Bjørn Petersen, Simon Spoorendonk, and David Pisinger. Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, 56(2):497–511, 2008.

[KR87] Leonard Kaufmann and Peter Rousseeuw. Clustering by means of medoids. In *Data Analysis based on the L1-Norm and Related Methods*, pages 405–416, 01 1987.

[LD60] Ailsa H Land and Alison G Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[LLE04] Jens Lysgaard, Adam N Letchford, and Richard W Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical programming*, 100:423–445, 2004.

[Llo57] S P Lloyd. Least squares quantization in pcm. Technical Report RR–5497, Bell Lab, September 1957.

[LN83] Gilbert Laporte and Yves Nobert. A branch and bound algorithm for the capacitated vehicle routing problem. *Operations-Research-Spektrum*, 5:77–85, 1983.

[Lys03] J. Lysgaard. *CVRPSEP: A package of separation routines for the capacitated vehicle routing problem*. Aarhus School of Business, Department of Management Science and Logistics, 2003.

[PCDU17] Diego Pecin, Claudio Contardo, Guy Desaulniers, and Eduardo Uchoa. New enhancements for the exact solution of the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 29(3):489–502, 2017.

[PdAU03] M Poggi de Aragao and Eduardo Uchoa. Integer program reformulation for robust branch-and-cut-and-price algorithms. In L A Wolsey, editor, *Proc. Conf. Math. Programming: Conf. Honour Nelson Maculan*, pages 56–61, 2003.

[PPP+17] Diego Pecin, Artur Pessoa, Marcus Poggi, Eduardo Uchoa, and Haroldo Santos. Limited memory rank-1 cuts for vehicle routing problems. *Operations Research Letters*, 45(3):206 – 209, 2017.

[PPPU17] D. Pecin, A. Pessoa, M. Poggi, and E. Uchoa. Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, 9(1):61–100, 2017.

[PSU18] Artur Pessoa, Ruslan Sadykov, and Eduardo Uchoa. Enhanced branch-cut-and-price algorithm for heterogeneous fleet vehicle routing problems. *European Journal of Operational Research*, 270(2):530–543, 2018.

[PSUV19] Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. A generic exact solver for vehicle routing and related problems. In Andrea Lodi and Viswanath Nagarajan, editors, *Integer Programming and Combinatorial Optimization*, pages 354–369, Cham, 2019. Springer International Publishing.

[PSUV20] Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. A generic exact solver for vehicle routing and related problems. *Mathematical Programming B*, 183:483–523, 2020.

[PU14] Marcus Poggi and Eduardo Uchoa. Chapter 3: New exact algorithms for the capacitated vehicle routing problem. In *Vehicle Routing: Problems, Methods, and Applications, Second Edition*, pages 59–86. SIAM, 2014.

[QSUV21] Eduardo Queiroga, Ruslan Sadykov, Eduardo Uchoa, and Thibaut Vidal. 10,000 optimal cvrp solutions for testing machine learning based heuristics. In *AAAI-22 Workshop on Machine Learning for Operations Research (ML4OR)*, 2021.

[RF81] David M Ryan and Brian A Foster. An integer programming approach to scheduling. In A Wren, editor, *Computer scheduling of public transport: Urban passenger vehicle and crew scheduling*, pages 269–280. North-Holland, Amsterdam, 1981.

[Rou87] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[SUP21] Ruslan Sadykov, Eduardo Uchoa, and Artur Pessoa. A bucket graph–based labeling algorithm with application to vehicle routing. *Transportation Science*, 55(1):4–28, 2021.

[SV21] Ruslan Sadykov and François Vanderbeck. BaPCod – a generic branch-and-price code. Technical Report HAL-03340548, Inria Bordeaux Sud-Ouest, Bordeaux, France, 2021.

[SW65] Samuel S Shapiro and M B Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965.

[UPP+17] Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017.

[Vid22] Thibaut Vidal. Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood. *Computers & Operations Research*, 140:105643, 2022.

[Wil45] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[ZLL+23] Jiayi Zhang, Chang Liu, Xijun Li, Hui-Ling Zhen, Mingxuan Yuan, Yawen Li, and Junchi Yan. A survey for solving mixed integer programming via machine learning. *Neurocomputing*, 519:205–217, 2023.

Appendix A. Appendices

Appendix A.1. Additional results on the `CMT13` instance

The six well-defined customer clusters in Figure 2 are numbered, with the depot designated as C_0. Cluster 1 (C_1) comprises the vertices surrounding the depot, while the remaining clusters are numbered clockwise. The Branch-and-Bound (B&B) tree in Figure A.5 illustrates the outcome of applying CSB and CB strategies to solve the `CMT13` instance. Cyan nodes denote instances where strong branching (SB) favored the Cluster Branching over ω_C variables, while gray nodes represent CB branching on ψ_{C_1,C_2} variables. Notably, SB preferred CB branching in a total of 26 branching, with 15 on ω_C and 11 on ψ_{C_1,C_2}. The yellow node denotes the sole node where SB opted for a branch on cutsets ω_S. Moreover, the green node marks the point where the solution with a cost of 1541.14 was proven optimal.

Appendix A.2. Branch-Cut-and-Price algorithms

This section presents the VRPSolver models for the Capacitated Vehicle Routing Problem (CVRP),VRP with Time Windows (VRPTW), and Distance-Constrained CVRP (DCCVRP).

VRPSolver [PSUV19, PSUV20] is a framework designed to facilitate the creation Branch-Cut-and-Price (BCP) algorithms for different VRPs variants. VRPSolver model is a MIP that contains variables associated with resource-constrained paths over directed graphs defined by the user. Given that the number of paths can be exponentially large, these variables are generated dynamically, solving as pricing problems Resource Constrained Shortest Path (RCSP).

It is the user's task to properly define the RCSP graphs and the *mappings* $\mathcal{M}(x_j)$ of the variables x_j in the user-defined objective function and constraints of the Master Problem (MP). These mappings connect the variables in the MP to a subset of arcs induced by the RCSPs. In addition, the set $\mathcal{R}$ of resources, and the lower (L) and upper (U) bounds on the number of paths from the RCSP in a solution must be defined. To define the feasibility of the paths, each RCSP graph has special vertices v_{source} and v_{sink} representing the start and the end of the paths. Each arc a must have a consumption q_a^r of resource $r \in \mathcal{R}$, and accumulated resource consumption intervals $[l_i^r, u_i^r]$ of r must be defined on each vertex i. The consumption intervals can also be defined over arcs a, denoted as $[l_a^r, u_a^r]$. In all cases, it is mandatory for the accumulated resource consumption to satisfy the UB of the consumption interval. When a resource is defined as *disposable*, resources may be dropped if necessary to meet the lower bound of the consumption interval. However, for non-disposable resources, the consumption interval must be strictly satisfied.

To strengthen the formulation, it is possible to define the so-called *elementarity sets* ($\mathcal{E}$) and *packing sets* ($\mathcal{S}$) on vertices or arcs. Within each $S \in \mathcal{S}$, the elements (arcs or vertices) appear

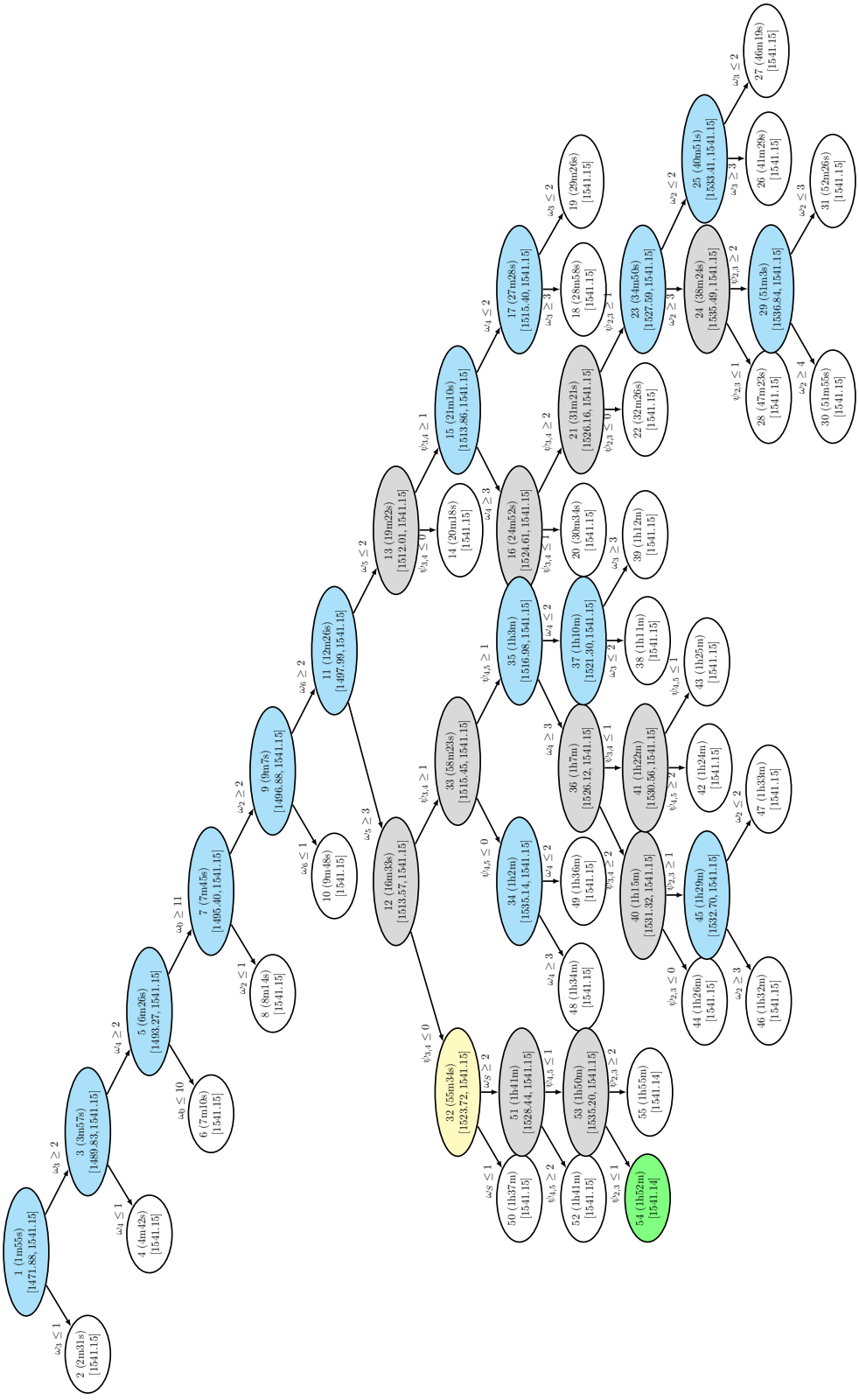


Figure A.5: The B&B tree of applying CSB and CB strategies to solve the `CMT13` instance.

at most once across all paths within any optimal solution. In contrast, within each $S \in \mathcal{E}$, the elements appear at most once in each path that comprises any optimal solution.

It is also possible to define Rounded Capacity Cut (RCC) [LN83] separators, route enumeration [BCM08, CM14], Rank-1 cuts with limited memory [JPSP08, PPPU17, PPP+17], and branching strategies [RF81, GDDS95]. For a comprehensive understanding of VRPSolver models, please refer to [PSUV20].

VRPSolver model for the CVRP proposed by [PSUV20]

Data: Undirected graph $G = (V = \{0\} \cup V_+, E)$, where 0 is the depot and $V_+ = \{1, \dots, n\}$ are the customers; positive costs $c_e, e \in E$ and positive demands $d_i, i \in V_+$; vehicle capacity Q.

Goal: Find a minimum cost set of routes that start and end at the depot, visit all customers, and ensure that the sum of the demands of the customers on each route does not exceed the vehicle capacity.

RCSP graph: A single graph $\mathcal{G} = (\mathcal{V} = V, \mathcal{A}), \mathcal{A} = \{(i,j),(j,i) : \{i,j\} \in E\}$. $v_{\text{source}} = v_{\text{sink}} = 0$. $\mathcal{R} = \{1\}$, that is a single monotone disposable resource defined over the demands of the customers (define $d_0 = 0$); arc consumptions $q^1_a = (d_i + d_j)/2, a = (i,j) \in \mathcal{A}$; and interval consumptions $[l^1_i, u^1_i] = [0, Q], i \in V$.

MP: Define the integer variables $x_e, e \in E$. The formulation is:

$$\min \sum_{e \in E} c_e x_e \tag{A.1a}$$

$$\text{s.t.} \sum_{e \in \delta(i)} x_e = 2 \qquad i \in V_+ \tag{A.1b}$$

$$x_e \leq 1 \qquad e \in E \setminus \delta(0); \tag{A.1c}$$

$\mathcal{M}(x_e) = \{(i,j),(j,i)\}, e = \{i,j\} \in E$; $L = \lceil \sum_{i \in N} d_i/Q \rceil$ and $U = n$.

BCP elements: $\mathcal{S}^{\mathcal{V}} = \mathcal{E}^{\mathcal{V}} = \cup_{i \in V_+} \{\{i\}\}$. RCC separator given by $(\cup_{i \in V_+} \{(\{i\}, d_i)\}, Q)$. Branching is over variables x. The enumeration procedure is on.

Comments: Constraints (A.1b) are the degree constraints over the customers. Constraints (A.1c) are dynamically separated as user cuts. Packing and elementarity sets are defined on vertices. The RCC separator is defined by setting the capacity as the vehicle capacity Q and a demand function on the $d_i, i \in V_+$.

VRPSolver model for VRPTW proposed by [PSUV20]

Data: The data of the CVRP with the inclusion of a time window $[l_i, u_i]$ and positive service time s_i for each customer $i \in V_+$. There are also positive travel times $t_e, e \in E$.

Goal: The same goal as for CVRP, with the additional constraints that the service at customer i must start within their time window, and the service at each customer has a duration of s_i.

RCSP graph: The same graph as CVRP, with the inclusion of a second main monotone disposable resource defining time, $\mathcal{R} = \{1, 2\}$. For each $t_{e=\{i,j\}}$, let $t_{a=(i,j)} = t_e + s_i$ and $t_{a=(j,i)} = t_e + s_j$.

For each arc $a = (i, j) \in A$, the arc consumptions $q_a^1 = d_j$ and $q_a^2 = t_a$; and interval consumptions $[l_i^1, u_i^1] = [0, Q]$ and $[l_i^2, u_i^2] = [l_i, u_i]$ for each $i \in V$.
MP and *BCP elements*: The same as CVRP model
Comments: Note that it is also possible to define only time as a graph resource, while capacity is enforced by RCCs.

VRPSolver model for DCCVRP
Data: The data of the CVRP; maximum travel distance (or time duration) T; a constant service time $s_i = s, i \in V_+$.
Goal: The same goal as for CVRP, with the additional constraint that each vehicle must not exceed a maximum travel distance T in its route, and the service at each customer has a duration of s_i.
RCSP graph: The same graph as CVRP, with the inclusion of a second main monotone disposable resource defining the travel distance, $\mathcal{R} = \{1, 2\}$. For each arc $a = (i, j) \in A$, the arc consumptions $q_a^1 = d_j$ and $q_a^2 = c_a + (s_i + s_j)/2$ (define $s_0 = 0$); and interval consumptions $[l_i^1, u_i^1] = [0, Q]$ and $[l_i^2, u_i^2] = [0, T]$ for each $i \in V$.
MP and *BCP elements*: The same as CVRP model
Comments: Note that it is also possible to define only travel distance as a graph resource, while capacity is enforced by RCCs.