Improving Column-Generation for Vehicle Routing Problems via Random Coloring and Parallelization

Miao Yu, Viswanath Nagarajan, Siqian Shen
Department of Industrial and Operations Engineering,
University of Michigan, Ann Arbor
Email: {miaoyu, viswa, siqian}@umich.edu

May 25, 2021

Abstract

We consider a variant of the Vehicle Routing Problem (VRP) where each customer has a unit demand and the goal is to minimize the total cost of routing a fleet of capacitated vehicles from one or multiple depots to visit all customers. We propose two parallel algorithms to efficiently solve the column-generation based linear-programming relaxation for this VRP. Specifically, we focus on algorithms for the "pricing problem" which corresponds to the resource-constrained elementary shortest path problem. The first algorithm extends the pulse algorithm by Lozano et al. (2015), for which we derive a new bounding scheme on the maximum load of any route. The second algorithm is based on random coloring from parameterized complexity (Alon et al., 1995), which can be also combined with other techniques in the literature for improving VRPs, including cutting planes and column enumeration. We conduct numerical studies using VRP benchmarks (with 50–957 nodes) and instances of a medical home care delivery problem using census data in Wayne County, Michigan. Using parallel computing, both pulse and random coloring can significantly improve column generation for solving the linear programming relaxations and we can obtain heuristic integer solutions with small optimality gaps. Combining random coloring with column enumeration, we can obtain improved integer solutions having less than 2% optimality gaps for most VRP benchmark instances, and less than 1% optimality gaps for the medical home care delivery instances, both under a 30-minute computational time limit. The use of cutting planes (e.g., robust cuts) can further reduce optimality gaps on some hard instances, without much increase in the runtime.

Keywords: vehicle routing problem (VRP), elementary shortest path, random coloring, column generation, parallel computing

1 Introduction

In the Vehicle Routing Problem (VRP) with Unit Demand (VRPUD), a fleet of vehicles is routed to visit a set of customers, and each vehicle has a capacity, i.e., an upper bound on the maximum number of customers to visit. Mathematically, let $G = (V \cup D, E)$ be an undirected graph, where V is a set of nodes representing the locations of all customers each having a unit demand and D is a set of depot nodes. The set $E = \{(i,j) |\ i,j \in V \cup D\}$ contains all the edges that correspond to the best travel routes between each pair of nodes in graph G. Each edge $(i,j) \in E$ is associated with a travel cost $c_{ij} > 0$, which satisfies the triangle inequality, i.e., $c_{ij} + c_{jp} \geq c_{ip}$ for any $(i,j),(j,p),(i,p) \in E$. At each depot node $d \in D$, a fleet of identical vehicles with capacity Q, denoted by set K_d, is deployed to serve customers in V. Our goal is to find a set of vehicle routes with the minimum total travel cost such that: (i) each node in V is visited by exactly one route, (ii) each vehicle in K_d starts and ends its route at depot node $d \in D$, and (iii) each route contains at most Q customer nodes.

When there is no capacity constraint and a single depot ($|D| = 1$), VRPUD extends the classic traveling salesman problem (Kruskal, 1956). VRPUD is also a special case of the more general capacitated VRP (CVRP) (see Toth and Vigo, 2014; Fukasawa et al., 2006; Baldacci et al., 2011; Pecin et al., 2017b), where a fleet of vehicles, each having a limited capacity, is routed to visit a set of customers having different demand volumes. VRPUD finds natural applications in service systems in transportation, logistics, and healthcare. One application is in a patient-centered medical home system, where we need to route caregivers to provide medical care services at patients' homes (American Academy of Family Physicians, 2008). Such a system can especially benefit those patients having limited mobility and can decrease the number of admissions in the hospitals (The National Association for Home Care & Hospice, 2010; Adaji et al., 2018). However, the routing and scheduling tasks are highly complex, often designed manually, and therefore can have unnecessarily high cost in practice (Eveborn et al., 2006). Fikar and Hirsch (2017) provided a review of different approaches for medical home care delivery, e.g., modeling the routing and scheduling problem as VRP with time windows (VRPTW) or VRP with pickup and delivery. To the best of our knowledge, the problem has not been tackled as a VRPUD with a goal of minimizing the workload of individual caregivers. According to The National Association for Home Care & Hospice (2010), nationwide, the number of patients visited by one caregiver ranges from 4 to 6 during a workday, depending on the types of caregivers. Thus, the capacity Q of each vehicle can be set to a relatively small number when we solve the corresponding VRPUD model.

1.1 Solution Methods for VRPs

Column generation is a prominent approach for solving VRPs (see, e.g., Desaulniers et al., 2006; Fukasawa et al., 2006; Baldacci et al., 2008, 2010, 2011; Pecin et al., 2017a,b). Here, a set-partitioning formulation is used where we associate with each feasible route a binary variable indicating whether or not to take the route. A key first step is to optimize the linear programming (LP) relaxation of this integer program. Because the number of decision variables (i.e., columns) in the overall LP formulation can be huge, we repeatedly solve a restricted LP model containing only a subset of columns. In each iteration, optimizing the restricted model provides a dual solution, based on which we can either improve the current solution or prove optimality. This requires solving the so-called "pricing problem" that finds a new column with the minimum reduced cost. If the minimum reduced cost is non-negative, then the current solution is optimal to the overall LP relaxation; otherwise, the column with the least reduced cost enters the basis and we re-optimize the restricted model with an expanded set of columns. Note that in each iteration, any column with negative reduced cost can improve the current solution, and thus we do not have to only add one column but can add multiple to speed up the computation (Desaulniers et al., 2006).

Algorithms for optimizing the pricing problem play an essential role in searching for the best columns to add, aiming to improve the computational efficacy of column generation. Finding a column with negative reduced cost is equivalent to solving an elementary shortest path problem with resource constraints (ESPPRC), which finds the shortest path starting and ending at a depot, traversing customer nodes obeying some resource constraints. When the underlying network contains negative-cost arcs, the problem is NP-hard (Dror, 1994). Desrosiers et al. (1995) proposed a dynamic programming method to solve a relaxed version of ESPPRC by allowing cycles. Based on their work, Feillet et al. (2004) proposed a label correcting algorithm that is the first exact approach for ESPPRC, which was later improved by Feillet et al. (2007). This algorithm solves the pricing problem for VRPTW very efficiently when time windows are tight, but it fails to handle instances with wide time windows. Later, Righini and Salani (2006) proposed a bi-directional label correcting algorithm based on state-space relaxation for ESPPRC and significantly reduced the computational time. Recently, Lozano et al. (2015) proposed a pulse algorithm that efficiently solved ESPPRC when using column generation for solving the vehicle routing problem with time-windows (VRPTW), via implicit enumeration and a bounding procedure. The algorithm worked with a depth-first-search-based enumeration to construct the partial paths starting from the depot to some end nodes. With several pruning strategies to discard the search on partial paths early, the algorithm pruned large regions of the solution space. The algorithm was later extended and generalized by Duque et al. (2015) as a general-purpose framework for solving hard shortest path problems and the orienteering problem with time windows.

The pricing problem in column generation for VRPUD has a parameter Q, which is the maximum number of nodes involved in any route. Although the ESPPRC is NP-hard, it is possible to find a polynomial-time algorithm with respect to the network size for fixed Q (Downey and Fellows, 2012). Indeed, polynomial-time algorithms exist for finding a simple path/cycle with fixed length based on a technique called color-coding (Alon et al., 1995), where each node is randomly assigned a color from a set with a fixed number of colors. Finding a simple path of a fixed length then reduces to finding a path containing nodes with distinct colors. The complexity of the latter is exponential with respect to the number of colors but polynomial with respect to the number of nodes–this is because we only need to track a subset of assigned colors rather than nodes. However, as one color-coding could color two distinct nodes with the same color, it forbids the exploration of a path containing the two nodes. Therefore, we need to investigate multiple independent trials of color-coding when applying the idea to ESPPRC. The color-coding concept has wider applications, e.g., in bioinformatics to explore complex structures in protein-protein interaction networks (Alon et al., 2008). To the best of our knowledge, our paper is the first to apply this technique in column generation algorithms.

The solution obtained at the end of column generation may not be integral. One way is to use all the generated columns to form an integer program by solving which we can obtain a heuristic integer solution for VRPs along with an optimization gap. To solve a VRP instance exactly, additional steps are needed to enforce integrality. The branch-and-price algorithm (Barnhart et al., 1998) is used to close the optimization gap by applying branch-and-bound atop the column generation approach, and branch-cut-and-price (Fukasawa et al., 2006) improves the branch-and-price by introducing cutting planes to strengthen the lower bound on each branching node. Recently, Baldacci et al. (2011); Contardo and Martinelli (2014); Pecin et al. (2017b,a) proposed a column enumeration approach to close the optimality gap early in the branching tree and to solve VRPs exactly. In this paper, we derive parallelized pulse and random-coloring algorithms for ESPPRC used in column generation for solving the LP relaxation of VRPUD. We also demonstrate how to combine random coloring with various types of cutting planes (Lysgaard et al., 2004; Jepsen et al., 2008) to reduce optimality gaps and with column enumeration for improving the quality of integer solutions.

1.2 Contributions of the Paper

The main contributions of the paper are threefold. Firstly, we design an algorithm using random coloring for solving ESPPRC to improve column generation for the LP relaxation of VRPUD. The approach is applicable more broadly to other VRPs with some (possibly implicit) limits on the number of nodes visited in each route. Because different coloring schemes are completely independent, it is natural to implement this algorithm in parallel, and we observe a high speed-up ratio in our parallel implementation. In addition, as the proposed approach specializes in finding multiple elementary paths quickly, it is particularly suitable for improving the recently-proposed column-enumeration procedure for seeking exact VRP solutions. We note that techniques to accelerate classical label-correcting algorithms for ESPPRC, such as bi-directional search and bounding functions, can also be combined with the random-coloring algorithm, yielding further speed-up. The random coloring algorithm can also be extended to handle all "robust" and some "non-robust" cutting planes, which are often useful in improving the LP bound.

Secondly, we evaluate the random-coloring approach against a state-of-the-art algorithm for ESPPRC, called pulse. The pulse algorithm has previously been tested in the column generation approach for VRPTW. In this paper, we extend the pulse algorithm to solve VRPUD by (i) extending the bounding scheme to consider vehicle capacity, and (ii) allowing the algorithm to stop early and return multiple negative-cost paths instead of just one optimal path. The comparison between pulse and random coloring is conducted by testing single-depot VRPUD on modified Solomon's instances (Solomon, 1987) and unitary demand CVRP X-instances (Uchoa et al., 2017) with the size of the instances ranging from 50 nodes to 957 nodes. We also implement one family of robust cuts, the rounded capacity cuts (see Lysgaard et al., 2004), and observe significant improvement in the optimality gaps by largely improving lower bounds on the optimal objective values, with only a small increase in runtime. Our results show that the parallel implementations of both algorithms can solve the LP relaxations of these instances efficiently, and the resulting heuristic integer solutions have optimality gaps less than 2%. Furthermore, after combining random coloring with column enumeration, the optimality gaps can be improved to less than 1% for most classical VRP instances.

Thirdly, we compare different algorithms on multi-depot VRPUD instances of a medical home care delivery problem using census data in Wayne County, Michigan. The parallel implementations of both pulse and random coloring can solve the LP relaxations with up to 500 nodes within reasonable time: about 6 minutes for random coloring and 15 minutes for pulse. Using the generated columns found by random coloring, we obtain high-quality integer solutions that have less than 2% optimality gaps. Further, combined with column enumeration, we improve the integer solutions to have less than 0.5% optimality gaps.

1.3 Organization

The remainder of the paper is organized as follows. In Section 2, we review the most relevant VRP literature including formulations and solution methods. In Section 3, we build a set-partitioning-based formulation for VRPUD that can be solved via column generation. In Section 4, provide details of the pulse and random-coloring algorithms for solving the ESPPRC pricing problem. In Section 5, we present computational results by combining our algorithms with column-generation and column-enumeration approaches and testing them on VRP benchmark instances in the literature and on medical home delivery instances with different sizes and complexities. Section 6 summarizes the paper and states future research directions.

2 Literature Review of VRP Variants

In recent years, the branch-cut-and-price (BCP) approach, which combines column generation with branch-and-cut framework, has been considered the most efficient exact solution approach for VRPs and has been widely applied to various types of VRPs (Pecin et al., 2017a,b). Nevertheless, solving the LP relaxation efficiently via column generation is an important step involved in the overall exact approach for VRPs.

The branch-and-price method particularly performs very well for VRPTW. Desrosiers et al. (1995) considered a column generation method for VRPTW by modeling the pricing problem as a shortest path problem with resource constraints by allowing cycles. The method was later improved by Kohl et al. (1999) and Irnich and Villeneuve (2006) by forbidding cycles with a fixed length. Feillet et al. (2004) solved VRPTW using column generation with ESPPRC as the pricing problem. They proposed the first exact algorithm for ESPPRC to improve the lower bounds obtained at each branching node. Jepsen et al. (2008) extended the BCP framework by introducing so-called "subset-row" cuts which effectively enhanced the bound from root node relaxation. Baldacci et al. (2010) proposed a column-and-cut generation algorithm and used non-elementary route relaxation approach to bound the pricing problem. Pecin et al. (2017a) proposed a BCP approach that combined several recently developed algorithms with limited-memory subset-row cuts and improved elementary inequalities.

CVRP can be viewed as a special case of VRPTW with arbitrary large time windows. However, with resource (i.e., time) being less constrained, the ESPPRC pricing problem becomes more challenging to solve. To solve CVRP through column-generation-based approaches, the main stream of the research considered non-elementary relaxations of the pricing problem and strengthened them through cutting planes. Christofides et al. (1981) first introduced q-route relaxation, which is a walk with at most q units of demand starting from the depot, traversing a sequence of nodes and returning to the depot. In a q-route, a vehicle is allowed to visit the same node multiple times, which may create loops. It is easy to avoid 2-node loops but it is hard to avoid k-node loops with $k \geq 3$. Fukasawa et al. (2006) initiated a BCP method to solve CVRP by combining branch-and-cut and column generation. They considered the pricing problem as a minimum cost q-route problem without 2-node loops, which significantly reduced the runtime of the pricing problem. Fukasawa et al. (2015) extended the previous algorithm to solve a variant of CVRP, where the cost of an arc was defined as the product of arc length and load of a vehicle that travels on this arc.

A variety of column-generation studies for CVRP focused on finding columns associated with elementary routes, whose efficiency relies on bounding functions to reduce the search space of a dynamic program. The bounds are computed through different state-space relaxations. Baldacci et al. (2008) proposed a column-and-cut generation approach using a bounding procedure combining three dual ascent heuristics. Baldacci et al. (2011) introduced the concept of ng-route that is more effective than the q-route. The ng-route is a non-elementary route limiting the visit to a node that was previously visited if such a node belongs to a dynamically computed no-good (ng) set associated with the route. Adding another dual ascent heuristic with ng-route relaxation and non-robust subset-row cuts, they improved the speed and stability of the BCP algorithm. Recently, Pecin et al. (2017b) improved the BCP algorithm by incorporating and enhancing various techniques from the past decade.

Table 1 summarizes the reviewed literature that applied column-generation-based approaches for VRPs. We classify them based on solution approaches used for solving the pricing problem. We refer the interested readers to a survey paper by Braekers et al. (2016) for state-of-the-art classification and theory development for broader VRPs.

Table 1: Summary of the reviewed papers based on solution methods for the pricing problem

Class of VRP	ESPPRC	Non-elementary Route Relaxation
VRPTW	Feillet et al. (2004), Feillet et al. (2007), Jepsen et al. (2008), Righini and Salani (2006), Lozano et al. (2015), Pecin et al. (2017a).	Desrosiers et al. (1995), Kohl et al. (1999), Irnich and Villeneuve (2006), Baldacci et al. (2010).
CVRP	Baldacci et al. (2008), Baldacci et al. (2011), Pecin et al. (2017b).	Fukasawa et al. (2006), Fukasawa et al. (2015), Baldacci et al. (2008), Baldacci et al. (2011), Pecin et al. (2017b).
Other VRPs	Bettinelli et al. (2011), Dabia et al. (2013), Duque et al. (2015).	Dabia et al. (2013).

3 VRPUD and Column Generation

Consider the VRP with unit demand (VRPUD) as defined in Section 1. A set-partitioning-based formulation of our problem is given as follows. Recall that D is the set of the depots. For $d \in D$, let P_d be the set of feasible routes rooted at depot d that can be assigned to vehicles in K_d (each route contains at most Q nodes and starts/ends at d). We assume that the number of vehicles $|K_d|$ at each depot is large enough so that any number of routes can be assigned to each depot d. For each feasible route $p \in P_d$ and node $i \in V$, let c_p be the cost of the route and a_{ip} be a binary coefficient such that $a_{ip} = 1$ if route p contains i and $a_{ip} = 0$ otherwise. We define a binary decision variable x_p for each $p \in P_d$ such that $x_p = 1$ if we pick the route p

in our solution and $x_p = 0$ otherwise. We formulate the overall model for VRPUD as

$$\textbf{(MP)}\ \text{minimize:}\ \sum_{d\in D}\sum_{p\in P_d} c_p x_p \tag{1}$$

$$\text{subject to:}\ \sum_{d\in D}\sum_{p\in P_d} a_{ip} x_p = 1 \qquad \forall\, i \in V, \tag{2}$$

$$x_p \in \{0,1\} \qquad \forall\, p \in P_d,\ d \in D, \tag{3}$$

where the objective function (1) minimizes the cost of all routes; constraints (2) ensure that each node in V is covered by exactly one vehicle; constraints (3) enforce that all the decision variables are binary valued.

Model MP is hard to solve because each $P_d,\ d \in D$ contains a number of feasible routes that grows exponentially with the size of the input instance. In column generation, instead of solving the problem with all variables explicitly, we solve the following restricted master problem (RMP), where a relatively small subset of P_d, $\tilde{P}_d \subset P_d$, is used to replace P_d:

$$\textbf{(RMP)}\ \text{minimize:}\ \sum_{d\in D}\sum_{p\in \tilde{P}_d} c_p x_p \tag{4}$$

$$\text{subject to:}\ \sum_{d\in D}\sum_{p\in \tilde{P}_d} a_{ip} x_p = 1 \qquad \forall\, i \in V, \tag{5}$$

$$x_p \in \{0,1\} \qquad \forall\, p \in \tilde{P}_d,\ d \in D. \tag{6}$$

In each iteration, we solve the LP relaxation of RMP and obtain an optimal dual solution, using which we can then search for new routes with negative cost to improve the current solution. Let $\pi_i,\ i \in V$ be the dual variables associated with constraints (5). Then, for each depot $d \in D$, the reduced cost of a route p (a column in the RMP) rooted at d is computed as $\bar{c}_p = \sum_{(i,j)\in p} \bar{c}_{ij}$, where for each arc (i, j), $\bar{c}_{ij}$ is then calculated as $\bar{c}_{ij} = c_{ij} - \pi_j$. When we find such routes, we add them into $\tilde{P}_d$ and continue to the next iteration of column generation. We obtain the optimal solution to the LP relaxation of MP when no more routes with negative costs can be added.

The main difficulty is the step of finding routes with negative cost, i.e., solving the pricing problem to generate columns. In this paper, we formulate the pricing problem as an ESPPRC described as follows. Consider a directed graph $G' = (V \cup \{s, t\}, A)$, where $V \cup \{s, t\}$ is the set of nodes with a source node s and a terminal node t, which correspond to a depot node in VRPUD, and $A = \{(i, j)|i \in V \cup \{s\}, j \in V \cup \{t\}\}$ is the set of arcs. Each arc $(i, j) \in A$ has a cost $\bar{c}_{ij}$ that can be negative. We are also given a capacity Q and a unit consumption associated with each node $i \in V$. Our goal is to find an elementary path from source node s to terminal node t with the minimum cost while the total consumption is no more than Q.

Consider a binary decision vector $y = (y_{ij}, (i,j) \in A)^\top$ such that $y_{ij} = 1$ if we visit node j right after node i and $y_{ij} = 0$ otherwise. Let $\pi_i,\ \forall i \in V$ represent the optimal dual solutions obtained at the end of the current iteration of solving RMP and let $\pi_s = \pi_t = 0$. The pricing problem (ESPPRC) is given by a flow-based integer program as follows.

$$
\begin{aligned}
\textbf{(PP)}\ z_{\mathrm{PP}}(\pi) = \underset{y}{\text{minimize}} \quad & \sum_{(i,j)\in A} (c_{ij} - \pi_j) y_{ij} && (7)\\
\text{subject to} \quad & \sum_{j:(s,j)\in A} y_{sj} = 1 && (8)\\
& \sum_{j:(j,t)\in A} y_{jt} = 1 && (9)\\
& \sum_{j:(i,j)\in A} y_{ij} - \sum_{j:(j,i)\in A} y_{ji} = 0 \quad \forall i \in V && (10)\\
& \sum_{(i,j)\in A} y_{ij} \le Q + 1 && (11)\\
& \sum_{(i,j)\in A,\ i,j\in S} y_{ij} \le |S| - 1 \quad \forall S \subset V && (12)\\
& y_{ij} \in \{0,1\} \quad \forall (i,j) \in A. && (13)
\end{aligned}
$$

Here (8)–(10) are flow balance constraints to ensure a path solution from s to t, and the objective function (7) minimizes the cost of the path by assuming $(c_{ij} - \pi_i)$ as the cost of arc (i,j), $\forall (i,j) \in A$. Constraint (11) ensures that the path visits no more than Q nodes. Constraints (12) are subtour elimination constraints. After solving PP, if $z_{\mathrm{PP}}(\pi) < 0$, then we find a path with negative cost. Otherwise, our solution is optimal to the LP relaxation of RMP. It is well known that PP is NP-hard and solving it as a binary integer program is challenging given that there are exponentially many constraints (12).

Also, as reviewed in Section 2, the use of non-elementary route relaxation can significantly improve the performance of column generation at each branching node. However, it yields a worse lower bound and thus results in a larger branch-and-bound tree (Desaulniers et al., 2006). In this paper, we focus on exact approaches for ESPPRC that is equivalent to finding the minimum-cost route in a network where arcs may have negative costs, while obeying resource/capacity-related side constraints. Different from tightly constrained VRP instances (e.g., the pricing problem associated with VRPTW) where the ESPPRC can be efficiently solved directly, it becomes very challenging to solve ESPPRC for less-constrained VRP instances, e.g., CVRP with no time-window information (Lysgaard et al., 2004). Next we introduce new, efficient algorithms for optimizing ESPPRC, which can be implemented in parallel.

4 Algorithms for Solving ESPPRC

Efficiently solving the pricing problem (PP) is crucial to improving the performance of the column generation approach. In this section, we propose two exact algorithms for solving ESPPRC. The first algorithm (called pulse) utilizes bounding and pruning strategies to accelerate the computational time of a dynamic program. The second algorithm (called random coloring) is a randomized algorithm that solves a dynamic program with significantly reduced state space.

4.1 Pulse Algorithm

This algorithm is based on a method proposed by Lozano and Medaglia (2013) for solving a constrained shortest path problem and also an extension by Lozano et al. (2015) for solving ESPPRC related to VRPTW. In VRPUD, the resource consumption is related to vehicle capacity rather than time (as in VRPTW).

The overall approach is to compute values $b(v, q)$ representing the minimum cost of a path from v to t that starts with resource consumption q. These values are computed in a backward manner, starting with $q = Q$ (which is trivial) and iteratively decreasing q by a step-size Δ. In order to compute $b(v, q)$ for some q, the algorithm performs a depth-first exploration from v and uses the $b(\cdot, q + \Delta)$ values as lower bounds to prune the search (after Δ nodes have been explored).

In more detail, the algorithm computing $b(v, q)$ constructs paths from some starting node v to the terminal node t by propagating from each current node to its successors. The propagation recursively explores the graph to construct partial paths while recording needed information. At each node, the algorithm tries to explore all outgoing arcs unless certain pruning strategies are triggered to stop the propagation. Each time, when the propagation reaches the terminal node t, we find a feasible solution (which updates the current best solution) and the algorithm will then backtrack to explore other options. At the end, the algorithm enumerates all possible paths from s to t following a depth-first search scheme. Crucially, by implementing pruning strategies to stop exploration early, the algorithm cleverly avoids full enumeration.

The implementation uses two procedures: pulse (see Algorithm 1) and bound (see Algorithm 2). The pulse procedure takes as input a current path P, its cost $r(P)$, its load $q(P)$ and a node w to which the path is being extended. It also maintains a pair of global variables: the best path P^* found so far and its cost $r(P^*)$. The global variables are updated whenever the propagation reaches the end node t and the resulting path is better than P^*. To find more columns with negative reduced costs per iteration, we introduce a global list $\mathcal{L}$ containing paths with negative costs. We add a path to $\mathcal{L}$ whenever the propagation reaches the end node t and the resulting path has a negative cost. We terminate the algorithm early when the size of $\mathcal{L}$ reaches a preset limit, *nSol*. Note that finding the optimal path P^* is critical for the bounding procedure (to be discussed later) and list $\mathcal{L}$ is only used when calling pulse procedure to solve the entire problem. To efficiently explore the graph, the pulse procedure utilizes a set of pruning strategies: infeasibility, rollback, and bounds, which will be detailed in Section 4.1.1. The most important strategy is bounds pruning, which relies on the already-computed $b(\cdot, q+\Delta)$ values. The bound procedure implements a backward dynamic program to compute the values $b(v,q)$ for $q = Q-\Delta, Q-2\Delta, \cdots$, each time invoking the pulse procedure. In particular, we start with obtaining the elementary shortest path (using pulse) from every node $v \in V$ to t given a resource consumption $Q-\Delta$. Then, we continue searching for the elementary shortest path from every node $v \in V$ to t given a resource consumption $Q-2\Delta$. We repeat the same procedure backwards until we reach a desired lower bound on the bounding resource consumption $\underline{Q}$. Therefore, this procedure collects all $b(v,q)$ values for all $v \in V$ and $q \in \mathcal{Q}$, where $\mathcal{Q} = \{\underline{Q}, \underline{Q}+\Delta, \ldots, Q-2\Delta, Q-\Delta\}$.

Algorithm 1: Pulse procedure

input : Current node w; cost $r(P)$; path load $q(P)$; current path P
output: Void

Let u and v be the second last and the last node visited in P, respectively
if $w == t$ **then**
  if $r(P) + c'_{vw} < r(P^*)$ **then**
    $P^* \leftarrow P \cup \{t\}$
    $r(P^*) \leftarrow r(P) + c'_{vw}$
  end
  ▷ update optimal path **if** $r(P) + c'_{vw} < 0$ ▷ *skip when executed inside bounding procedure* **then**
    $\mathcal{L} \leftarrow \mathcal{L} \cup \{P \cup \{t\}\}$
  end
  stop
end
if $|\mathcal{L}| \geq nSol$ **then** stop ▷ skip when executed inside bounding procedure

if $q(P) == Q$ *or* $w \in P$ **then** stop ▷ pruned by infeasibility

if $|P| \geq 2$ *and* $c'_{uv} + c'_{vw} > c'_{uw}$ **then** stop ▷ pruned by rollback

let $\underline{q(P)}$ be the greatest q such that $q \leq q(P)$ and $q \in \mathcal{Q}$ **if** $r(P) + b(w, \underline{q(P)}) \geq r(P^*)$ **then** stop ▷ pruned by bounds

$P' \leftarrow P \cup \{w\}$
$q(P') \leftarrow q(P) + 1$
$r(P') \leftarrow r(P) + c'_{vw}$ ▷ $r(P') \leftarrow 0$ if $P = \emptyset$
for $(w, w') \in A$ **do**
  $pulse(w', r(P'), q(P'), P')$
end

The overall algorithm works as follows. We start by executing the bounding procedure to compute the lower bound matrix B. Note that we do not maintain the list of negative-cost paths $\mathcal{L}$ when executing pulse within the bounding procedure. Next, we run the pulse procedure with $P = \{s\}$, $r(P) = 0$, and $q(P) = 0$.

Algorithm 2: Bounding procedure

input : Graph $G' = (V \cup \{s,t\}, A)$; step size Δ; bounding cap $[\underline{Q}, Q]$
output: Lower bound matrix $B = [b(v,q) : \; v \in V, \; q \in \mathcal{Q}]$

$q \leftarrow Q$ **while** $q > \underline{Q} + \Delta$ **do**
  $q \leftarrow q - \Delta$ **for** $v \in V$ **do**
    $P^* \leftarrow \{\}$ ▷ initialize global variables
    $r(P^*) \leftarrow \infty$
    $P \leftarrow \{\}$
    $r(P) \leftarrow 0$
    $q(P) \leftarrow q$
    $pulse(v, r(P), q(P), P)$ ▷ find the optimal partial path from v to t given q consumed
    $b(v,q) \leftarrow r(P^*)$
  end
end
return B

When the program terminates, the global list $\mathcal{L}$ contains at most $nSol$ many s–t paths with negative costs.

4.1.1 Pruning Strategy

The efficiency of the pulse algorithm depends on the pruning strategies to stop the exploration of partial paths as soon as possible. Lozano et al. (2015) proposed three pruning strategies: infeasibility, bound and rollback. Based on the problem setting of VRPUD, we detail how to modify each pruning strategy as follows.

Infeasibility pruning. Infeasibility pruning terminates an exploration when a partial path violates any feasibility constraints: the partial path visits more than Q nodes, or the partial path forms a cycle when it reaches a new node. For each partial path, we maintain an indicator vector of length $|V|$ to indicate if such a path has visited each node $v \in V$. We can then identify if any cycle is created in constant time, i.e., if the path is extended to a node that has been previously visited.

Bounds pruning. Bounds pruning is a key component that significantly improves the performance of the pulse algorithm. The idea is to fathom suboptimal partial paths using the continuously updated primal bound $r(P^*)$ (the cost from the current best feasible solution) and pre-calculated conditional lower bounds $b(v, \underline{q(P)})$, which store the minimum reduced cost that can be achieved for every node $v \in V$ and for a given resource consumption $\underline{q(P)}$. We terminate the exploration for a partial path P when it reaches a node $v \in V$ where its cost, $r(P)$, plus the conditional lower bound at v with $\underline{q(P)}$ resource consumption is at least the current primal bound, i.e., $r(P) + b(v, \underline{q(P)}) \geq r(P^*)$. Note that we may not have a valid s–t path of cost $r(P) + b(v, \underline{q(P)})$, but it is still a lower bound.

Rollback pruning As the pulse algorithm implicitly enumerates the search space in a depth-first search fashion, a poor decision made at early stages may lead to an unpromising region of the search space. To avoid this, we impose the rollback pruning strategy that examines the last choice made. Let P_{ij} be a partial path with end node j and it visits node i right before j. When we extend P_{ij} to next node v, we check if $\bar{c}_{ij} + \bar{c}_{jv} > \bar{c}_{iv}$. If yes, we terminate the current exploration as a better propagation is to roll back to the partial path with end node i and extending it to v ("Rollback" is automatically done when we propagate the path from node i); otherwise, we continue the exploration. This helps to avoid bad early explorations.

4.1.2 Parallelization

In the pulse framework, Algorithm 1 explores partial paths in a depth-first search fashion. Along the search, it runs the pulse procedure on one node at a time until the search reaches the end node. Starting from node s, the extensions starting on different outgoing arcs are independent, and therefore we can implement Algorithm 1 in parallel on different computer threads to accelerate the search while maintaining the global information properly. Lozano and Medaglia (2013) proposed to trigger a fixed number of threads at node s and explore the extensions on different outgoing arcs from s independently. We only need to maintain the record of the visited nodes for each thread and the bound information globally. Multiple threads can run Algorithm 1 on the same node at the same time except for the end node t, where the global lower bound can only be updated by one thread at a time.

4.2 Random Coloring Algorithm

A traditional way to solve ESPPRC is through the label correcting algorithm (e.g., Lysgaard et al., 2004; Feillet et al., 2004). However, to make sure the path is elementary, the algorithm needs to record the full path for each state variable. Therefore, it requires exponentially many state variables. To be specific, the size of the state space for label correcting algorithm is in the order of $O(2^{|V|}|V|)$. In this section, we discuss how to utilize the idea of color-coding from Alon et al. (1995) to extend the label correcting algorithm and efficiently cut the size of the state space to $O(2^Q|V|)$.

In VRPUD, each route can visit at most Q nodes. Suppose that we are given a color-coding, which is a function $\phi : V \to \{1, 2, \ldots, Q\}$ that maps each node in V to a color attribute labeled from $1, 2, \ldots, Q$. We say that a path in G' is *colorful* if the nodes in the path are colored by distinct colors. Clearly, every colorful path is elementary, and each colorful path contains no more than Q nodes in V. Then if we can find a colorful s-t path with negative cost, we find an elementary path connecting nodes s and t with a negative cost. To find a colorful path in G', we can modify the label correcting algorithm from Feillet et al. (2004).

Let P_{si} be a partial path from source node s to node $i \in V$. Different from the original algorithm, we record the information of color history instead of node history of the path. A state $R_i = (n_i, V_i^1, \ldots, V_i^Q)$ corresponds to the number of visited nodes and a binary indication vector that is used to record color usage, where $V_i^k = 1$ if P_{si} visits a node colored $k \in \{1, 2, \ldots, Q\}$ and $V_i^k = 0$ otherwise. Although n_i is implied by $\sum_{i=1}^{Q} V_i$, we keep it to save the computational time when implementing path domination. Let $C_i = c(P_{si})$ be the cost of such path. A dominance rule is enforced to eliminate additional paths P_{si} in the label correcting algorithm. Let P'_{si} and P^*_{si} be two distinct paths from s to i with associated labels (R'_i, C'_i) and (R^*_i, C^*_i). We say that P'_{si} dominates P^*_{si} if and only if $C'_i \leq C^*_i$, $n'_i \leq n^*_i$, $V_i'^k \leq V_i^{*k}$ for all $k \in \{1, 2, \ldots, Q\}$, and $(R'_i, C'_i) \neq (R^*_i, C^*_i)$. Note that the number of possible states R_i is at most $|V| \cdot 2^Q$.

The label correcting algorithm works as follow. For each node $i \in V$, we maintain a list Λ_i of paths from source node s to node i. We start with a set of active nodes containing s only. In each iteration, we poll an active node i from the active node set and extend the paths in Λ_i. Let P_{sj} be the extended path that is feasible. Suppose P_{sj} is not dominated by other paths in Λ_j; then we put j into the active node set and iterate the previous procedure. We stop the algorithm when no active nodes exist. The details of the label correcting algorithm are displayed in Algorithm 3. For any partial path P_{si}, we record the history of colors instead of nodes: during the extension process, we can extend a path to a new node only if we have not visited a node with the same color before.

Theorem 1 (Theorem 3.4 from Alon et al. (1995))**.** Let $G' = (V \cup \{s, t\}, A)$ be a directed graph. Any pairs of vertices connected by a path with Q vertices in G can be found in $O(2^Q|V||A|)$ worst-case time.

Recall that our pricing problem is defined on a network $G' = (V \cup \{s, t\}, A)$ and it suffices to output any route with negative cost. Hence, we can terminate the algorithm early to output such a solution.

Note that any negative-cost colorful path found by Algorithm 3 is indeed an elementary path with negative cost. On the other hand, Algorithm 3 may fail to find a negative-cost colorful path even if there is some elementary path with negative cost. We now bound this "failure" probability. For a randomly chosen coloring ϕ, any elementary path in G' with at most Q nodes (in particular, any feasible path with negative cost) has a probability $\frac{Q!}{Q^Q} > e^{-Q}$ to be colorful. So the probability that the algorithm fails to identify a negative-cost elementary path with at most Q nodes is less than $1 - e^{-Q}$. Then, if we repeat k independent runs of the color-coding algorithm, the probability of failing to identify a negative-cost path in all repetitions is at most $(1 - e^{-Q})^k$, which is decreasing exponentially in k. Therefore, we repeat this algorithm multiple times to increase the probability of finding a colorful path with negative cost. For example, with $Q = 4$ and $k = 40$, the probability of failure is at most 0.02.

Our overall algorithm works as follows. We pre-define a stopping criterion in terms of the maximum number of iterations and a threshold count for the number of output routes. In each iteration, we randomly generate a color-coding ϕ that assigns color labels to each node in G'. Then, based on the color-coding ϕ, we solve the ESPPRC through Algorithm 3 and store all solution routes found with negative cost. If we

Algorithm 3: Algorithm for ESPPRC with Colors

input : Graph $G' = (V \cup \{s, t\}, A)$, color-coding ϕ.
output: A set T of routes with negative cost.

Initialization $\Lambda_s \leftarrow \{(0, \dots, 0)\}$
for $i \in V \cup \{t\}$ **do**
  $\Lambda_i \leftarrow \emptyset$
end
$S = \{s\}$
while $S \neq \emptyset$ **do**
  Pick $i \in S$
  if $i == t$ **then**
    add corresponding routes from Λ_t with negative cost to T
  end
  else
    forall $j : (i, j) \in E$ **do**
      forall $\lambda_i = (R_i, C_i) \in \Lambda_i$ *with* $R_i = (n_i, V_i^1, \dots, V_i^Q)$ **do**
        if $V_i^{\phi(j)} = 0$ **then**
          extend λ_i to get λ_j
          if λ_j *is not dominated by any path in* Λ_j **then**
            add λ_j to Λ_j and $S = S \cup \{j\}$
            remove any path in Λ_j that is dominated by λ_j
          end
        end
      end
    end
  end
  remove i from S
end
return T

reach the maximum number of iterations or the set of solutions contains more than the threshold number of output routes, we stop the algorithm; otherwise, we move to the next iteration. The detail of our random coloring algorithm for ESPPRC is presented in Algorithm 4.

Algorithm 4: Random Coloring Algorithm for ESPPRC

input : Graph $G = (V \cup \{s, t\}, A)$, maximum iteration *maxIter* to execute random coloring algorithm, number of the solutions triggered early stop *nSol*.
output: A set T of routes with negative cost.

Initialization $T = \emptyset$ as solution set and $k = 0$
while $k < maxIter$ *or* $|T| < nSol$ **do**
  Generate a random coloring scheme $\phi_k : V \rightarrow \{1, \dots, Q\}$
  Use Algorithm 3 to solve ESPPRC based on current color-coding ϕ_i
  Add routes with negative cost to T
  $k = k + 1$
end
return T

Irrespective of the number of repetitions *maxIter*, the random coloring algorithm has a non-zero probability of failure (i.e., it does not find any negative-cost route even if one exists). To address this issue, we can either implement the de-randomized algorithm (which has the same asymptotic time complexity) as described in Section 4 of Alon et al. (1995) or any other exact algorithm (e.g., the pulse algorithm), as a "safe vault", to ensure that no more negative-cost routes can be found in such cases. In our computational experiments, we used the pulse algorithm as the safe vault as it was already implemented.

It is worth highlighting that the random coloring idea could be extended to other label-correcting algorithms for ESPPRC, as the label requires maintaining a binary vector recording the nodes of corresponding partial path visited. By randomly assigning nodes with a fixed set of colors, we can reduce the length of such vector and decrease the total number of labels to explore in the algorithm. One can also apply bidirectional search techniques to further improve the random coloring algorithm.

4.2.1 Cutting Planes

Valid inequalities (or cuts) can strengthen LP relaxations of integer programs and help to obtain integer solutions at the extreme points of LP relaxations. In the BCP approach, the effective use of cuts yields better root-node bounds and shortens the overall solution time. Poggi de Aragao and Uchoa (2003) proposed to classify valid inequalities into "robust cuts" and "non-robust cuts". In the context of VRP, robust cuts apply to the "flow-based" formulation (which can be transformed into RMP) and these cuts do not affect the complexity of the pricing problem. On the other hand, non-robust cuts are applied directly on the RMP relaxation and thus increase the complexity of the pricing problem as their associated dual variables cannot be incorporated into arc costs (of the pricing problem). In this section, we will discuss how to incorporate robust and non-robust cuts to our proposed algorithm, which can improve the lower bounds and thus decrease optimality gaps.

Robust cuts. Lysgaard et al. (2004) summarized various robust cuts for CVRP, including rounded capacity cuts, bound cuts, framed capacity cut, strengthened comb, multistar, partial multistar, and hypotour cuts (also, see Fukasawa et al., 2006). In our computations, we implement the rounded capacity cuts described as follows. For any set $S \subset V$, let $\delta(S)$ be a set of edges having exactly one end-node in set S. The rounded capacity cuts for VRPUD are:

$$\sum_{p \in \tilde{P}} \sum_{e \in \delta(S)} x_p \ \geq \ 2 \cdot \left\lceil \frac{|S|}{Q} \right\rceil, \qquad \forall S \subseteq V \tag{14}$$

Above, $\tilde{P}$ denotes all routes in the RMP. To see why these constraints are valid, note that each route can visit at most Q nodes of S and each route must enter/leave set S at least twice. Although the separation problem for these cuts is NP-hard, Lysgaard et al. (2004) gave a number of efficient heuristics. We also use some of these heuristics in our computation later.

Non-robust cuts. We now discuss a class of non-robust cuts, known as subset-row cuts introduced by Jepsen et al. (2008) for CVRP and explain how the random coloring algorithm can be extended to solve the resulting pricing problem. The cuts are defined over route variables and are applied directly to the RMP. Recall that a_{ip} is a binary coefficient indicating whether a route $p \in \tilde{P}$ visits a node $i \in V$. For any set $S \subset V$ and a multiplier $0 < k < 1$, a subset-row cut is given by

$$\sum_{p \in \tilde{P}} \left\lfloor k \sum_{i \in S} a_{ip} \right\rfloor x_p \leq \lfloor k|S| \rfloor . \tag{15}$$

Inequalities (15) are valid as they can be obtained by a Chvátal-Gomory rounding of constraints (5). Various combinations of $|S|$ and p yield effective subset-row cuts to improve the lower bounds given by the LP relaxation of RMP. For example, when $|S| = 3$ and $k = \frac{1}{2}$, cuts (15) are 3-subset-row cuts and when $|S| = 4$ and $k = \frac{2}{3}$, cuts (15) are 4-subset-row cuts.

However, introducing these cuts changes the pricing subproblems. Let $\mathcal{S}$ denote all the added subset-row cuts. For each $S \in \mathcal{S}$, let σ_S be the dual variable associated with inequality (15) when solving the LP relaxation of RMP. Then, the reduced cost of a column/route p is given by

$$\bar{c}_p = \sum_{(i,j) \in p} (c_{ij} - \pi_j) - \sum_{S \in \mathcal{S}} \sigma_S \left\lfloor k \sum_{i \in S} a_{ip} \right\rfloor .$$

Recall that π_i is the dual variable associated with constraint (5) in RMP. To incorporate the subset-row cuts into the random coloring algorithm, we follow an idea from Jepsen et al. (2008). In each iteration of column generation, we maintain a vector corresponding to the subset-row cuts $\mathcal{S}$ with non-zero dual variables. This vector $\kappa = \langle \kappa_S : S \in \mathcal{S} \rangle$ maintains counters for each subset-row cut: when a label extends to a node in a subset-row cut $S \in \mathcal{S}$, we increase κ_S by k. When the value of any κ_S (for $S \in \mathcal{S}$) exceeds one, we (i) update the cost by subtracting σ_S and (ii) reduce κ_S by 1. Therefore, for any path P_{si} from s to i, we maintain a label (R, C, κ) where C denotes the cost of the path, the state R corresponds to the set of visited colors (as in Section 4.2) and vector κ corresponds to the subset-row cuts (as defined above).

We also need to modify the dominance rule based on the above changes to the label of a path. Let P'_{si} and P^*_{si} be two distinct paths from s to i with labels (R', C', κ') and (R^*, C^*, κ^*) respectively. For the pricing algorithm without subset-row cuts, recall that P'_{si} dominates P^*_{si} if and only if $C' \leq C^*$, $n' \leq n^*$, $V'^j \leq V^{*j}$ for all $j \in \{1, 2, \ldots, Q\}$. With our modification for subset-row cuts, we say that P'_{si} dominates P^*_{si} if and only if $C' \leq C^* + \sum_{S \in \mathcal{S}: \kappa'_S > \kappa^*_S} \sigma_S$, $n' \leq n^*$, $V'^j \leq V^{*j}$ for all $j \in \{1, 2, \ldots, Q\}$. (See Proposition 6 in Jepsen et al. (2008) for more details.)

To keep the pricing problem tractable, only a small number of subset-row cuts are included in the pricing problem. Pecin et al. (2017b) introduced a weak version of subsets row cuts called limited-memory subset-row cuts where each subset-row cut has a memory set, and the state counter of subset-row cut resets when a label extends to a node outside such a memory set. Our proposed algorithm can also be easily modified to incorporate limited-memory subset-row cuts following a similar idea.

4.2.2 Column Enumeration

Solving the LP relaxation for RMP through column generation does not guarantee an integer solution. Furthermore, some routes in an optimal integer solution may not even be generated, and therefore solving the RMP as an integer program at the end of the column generation cannot guarantee optimality. Most of the exact approaches for VRPs use BCP approach to tackle this problem, but often lead to a large branch-and-bound tree. Baldacci et al. (2008) and Baldacci et al. (2011) proposed a column-and-cut generation approach that avoided branching to find optimal solutions for CVRP and VRPTW, respectively. The algorithm is based on the idea of column enumeration: given an upper bound, UB, on the objective value of an integer VRP solution and a lower bound, LB, on the objective value of an optimal solution to the LP relaxation of RMP along with its optimal dual solution, one can enumerate a set of routes P' such that each route $p \in P'$ has a reduced cost $\bar{c}_p < UB - LB$. Solving RMP with all routes (i.e., columns) in P' as an integer program warrants that we find an optimal solution to MP. Such an approach has been proposed in recent literature for solving a broad class of VRPs (see, e.g., Contardo and Martinelli, 2014; Pecin et al., 2014, 2017b,a).

The column enumeration step can be solved using the label-correcting algorithm. However, certain modifications need to be made from the original one. First, we need to change the stopping criteria as we now aim to enumerate routes with reduced cost less than $UB - LB$, instead of 0. Additionally, the early stop should be disabled. Second, to enumerate several possible routes, we adopt a more restricted domination criterion. Recall that for a partial path P_{si} connecting nodes s and i, we define a state $R_i = (n_i, V_i^1, \ldots, V_i^Q)$ that corresponds to the number of visited nodes, a binary indication vector for color usage and $C_i = c(P_{si})$ being the cost of the path. Then in column enumeration, for any two partial paths P'_{si} and P^*_{si} from s to i with associated labels (R'_i, C'_i) and (R^*_i, C^*_i), we say that P'_{si} dominates P^*_{si} if and only if

$$C'_i \leq C^*_i,\ n'_i = n^*_i,\ V_i'^k = V_i^{*k},\ \forall k \in \{1, 2, \ldots, Q\}. \tag{16}$$

With more restricted domination criteria, the pricing problems becomes harder to solve by a general label-correcting algorithm, which considers a domination criteria that is equivalent to letting $Q = |V|$ in (16). The resulting search space has size $O(|V| \times 2^{|V|})$, which counts the number of distinct labels. However, as the label structure in random coloring algorithm only records color visited information, it keeps a smaller search space with size $O(|V| \times 2^Q)$, and therefore can achieve high computational efficiency. The details of the modified label-correcting algorithm are presented in Algorithm 5.

Algorithm 5: Algorithm for ESPPRC with Colors for Enumeration

---

input : Graph $G' = (V \cup \{s,t\}, A)$, color-coding ϕ, upper bound and lower bound, UB and LB, to RMP

output: A set T of routes with reduced costs less than $UB - LB$.

Initialization $\Lambda_s \leftarrow \{(0,\dots,0)\}$
for $i \in V \cup \{t\}$ **do**
    $\Lambda_i \leftarrow \emptyset$
end
$S = \{s\}$
while $S \neq \emptyset$ **do**
    Pick $i \in S$
    if $i == t$ **then**
        add corresponding routes from Λ_t with cost less than $UB - LB$ to T
    end
    else
        forall $j : (i,j) \in E$ **do**
            forall $\lambda_i = (R_i, C_i) \in \Lambda_i$ *with* $R_i = (n_i, V_i^1, \dots, V_i^Q)$ **do**
                if $V_i^{\phi(j)} = 0$ **then**
                    extend λ_i to get λ_j
                    if λ_j *is not dominated* **then**
                        add λ_j to Λ_j and $S = S \cup \{j\}$
                        replace the dominated label as needed
                    end
                end
            end
        end
    end
    remove i from S
end
return T

Despite that column enumeration avoids branching in an extensive search tree, it relies on commercial solvers to solve the RMP with enumerated columns. On the one hand, such an approach benefits from cutting-edge tools offered by the state-of-the-art solvers and therefore achieves high efficiency for some VRP instances (Costa et al., 2019). On the other hand, when the number of enumerated columns is large, e.g., more than 100,000 columns, it is not realistic to only rely on off-the-shelf solvers for optimizing the corresponding RMP (Pecin et al., 2017b). Usually, the number of enumerated columns depends on the optimality gap between the lower bound and the upper bound of RMP. To overcome the drawback, one may apply a hybrid algorithm that combines column enumeration and branching (see, e.g., Pecin et al., 2017b,a). In practice, we can still use commercial solvers to compute the RMP with many enumerated columns to quickly obtain a feasible solution that has an optimality gap within a certain time limit.

Because random coloring is a randomized algorithm, we may fail to enumerate all routes with reduced cost bounded by the optimality gap. However, as shown in Section 4.2, when repeating the independent runs of the color-coding algorithm, the "failure" probability to find a particular route decreases exponentially. Also, when we aim to find a high-quality integer solution instead of an optimal solution, missing some enumerated columns is tolerable.

4.2.3 Parallelization

The random coloring algorithm requires to explore different color-codings to increase the success probability of recovering all potential routes. In each iteration, a label correcting algorithm is executed based on the current color-coding, which is completely independent from all other iterations. For this reason, it is natural to perform a parallel implementation of the random coloring algorithm. We can invoke each iteration using parallel computer threads to accelerate the algorithm while maintaining the solution set as global information. The number of threads, therefore, determines the number of color-coding iterations that can be implemented simultaneously.

5 Computational Experiments

We conduct numerical studies and demonstrate the performance of the proposed algorithms on different types of VRP instances. We embed our proposed algorithms inside the column generation approach for VRPUD. In experiments, we solve the root node LP relaxation of MP mentioned in Section 3. We also solve a strengthened LP relaxation that incorporates rounded-capacity-cuts (for some experiments). Then, we use the generated columns to obtain an integer solution to RMP. This integer solution provides an upper bound UB_1 on the optimal value. We further use UB_1 in a column enumeration approach to generate an improved integer solution. We conduct three sets of experiments: (i) a set of tailored instances from the Solomon's and Gehring & Homberger benchmark[1], (ii) selected unitary demand CVRP instances from CVRPLIB[2], and (iii) a multi-depot VRPUD which has potential application in patient-centered medical home systems.

We implement column generation based on the conventional set-partitioning formulation RMP. We start with a series of heuristics that initialize the columns pool following a common practice (see, e.g., Feillet et al.,

[1] https://www.sintef.no/projectweb/top/vrptw/
[2] http://vrp.galgos.inf.puc-rio.br/index.php/en/

2004; Lozano et al., 2015). The heuristics are based on tabu search: we start with a set of feasible solutions (e.g., routes visiting only one node per vehicle) and then execute insertion and deletion operations until no further improvements can be made to these routes. After the initialization, we only solve subproblems as ESPPRC to generate columns.

After tuning parameters in a few preliminary tests, we choose our parameters for pulse and random-coloring algorithms as follows. For the pulse algorithm (Algorithm 2), we set $\Delta = 1$, $\underline{Q} = 2$ and $nSol = 30$. For random coloring (Algorithm 4), we set $maxIter = 39$ and $nSol = 30$. Also, when the random-coloring algorithm fails to find any route with negative cost, we trigger a run of the pulse algorithm as a safe vault to ensure that no more route with negative cost exists. For column enumeration, we utilize the upper bound solution obtained at the end of column generation and set $maxIter = 78$. In our tests, we implement the multi-thread versions of the proposed algorithms unless otherwise noted.

We remark that although column enumeration aims to close the optimality gap using a massive number of columns, our solutions from column enumeration are not guaranteed optimal due to the following reasons. First, we may fail to enumerate all columns due to the randomness in the coloring, even though the probability of missing any particular negative-cost route is small. Second, due to the number of columns enumerated, Gurobi cannot solve some problems to optimality within the preset time limit. Still, we find that the proposed random coloring algorithm performs very well to find high-quality solutions with small optimality gaps. (The lower bounds are significantly improved by the addition of rounded capacity cuts.)

We code our algorithms in Java on a computer with two Intel Xeon E5-2630v4 processors with 20 cores each (40 total), and 128GB DDR4-2400 registered RAM. We use Gurobi 7.5.2 as the LP and mixed-integer linear programming solver for all computation. All the data instances and our code can be downloaded from the GitHub repository at `https://github.com/myu23/VRPUD_RandCol` for non-commercial use, where we include a Readme document to define the contents and formats of the files.

5.1 Numerical Results on Single Depot VRPUD

5.1.1 Solomon and Gehring & Homberger Instances

The first set of test instances are modified from the Solomon benchmark with 100 customers[3] and Gehring & Homberger benchmark with up to 600 customers[4]. Both benchmark instances contain three types of node distributions: Type R instances where customers are randomly distributed, Type C instances where customers form several clusters, and Type RC where some customers are randomly distributed while others are clustered. For each customer node in the test instances, we ignore its time windows and assign a unit

[3] `https://www.sintef.no/projectweb/top/vrptw/solomon-benchmark/`
[4] `https://www.sintef.no/projectweb/top/vrptw/homberger-benchmark/`

demand. The travel distances between any two nodes are calculated as the Euclidean distance based on the coordinates given by the original data.

We solve the LP relaxation of RMP that is strengthened with rounded capacity cuts (described in Section 4.2.1). As discussed before, the separation problem for rounded capacity cuts is NP-hard and we rely on efficient heuristics for the cut separation. We note that because of the heuristic cut separation procedures, the LP bounds obtained by the pulse and random-coloring algorithms may differ. We use the LP bound obtained from random coloring as our lower-bound (LB), while noting that the LP bounds from both algorithms are very similar.

Serial implementation. We first compare both our proposed algorithms with the label correcting algorithm for ESPPRC from Feillet et al. (2004). As the original label correcting algorithm is implemented in serial, we run the serial implementation of our algorithms as well. We test the performance of the algorithms on instances with number of customers ranging from 50 to 150. For instances with number of nodes $|V| \leq 100$, we use the first $|V|+1$ nodes from Solomon's instance and for $|V| > 100$, we use the first $|V|+1$ nodes from Gehring & Homberger's instances with the first node being the depot node in all benchmark instances. We consider $Q = 4$ and set the time limit for column generation as 15 minutes for each instance. Figure 1 summarizes the computational results.

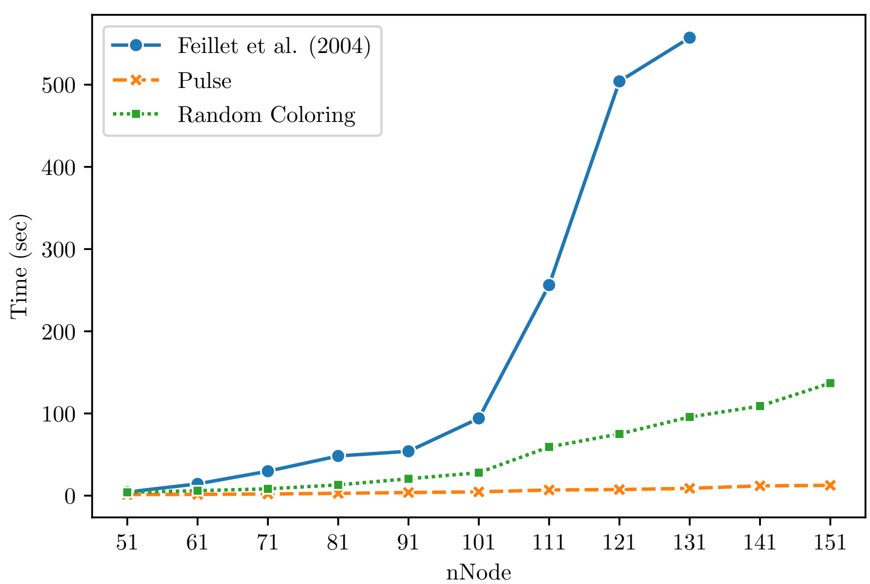

Figure 1: Numerical results for proposed algorithms in serial implementation ($Q = 4$)

In Figure 1, we observe the efficiency of the pulse algorithm as its runtime for solving the LP relaxation of RMP is significantly shorter than the other two algorithms. Compared to the original label-correcting algorithm, random coloring significantly improves the solution time. When the number of nodes increases, the label correcting algorithm encounters the curse of dimensionality as it fails to solve instances with more than 140 nodes given the limit of time. We can also see the advantage of using random coloring for larger instances as the problem can be consistently solved. The speed-up factor of the random coloring algorithm compared to the original label-correcting algorithm is between 2 and 10, and this factor increases for larger instances.

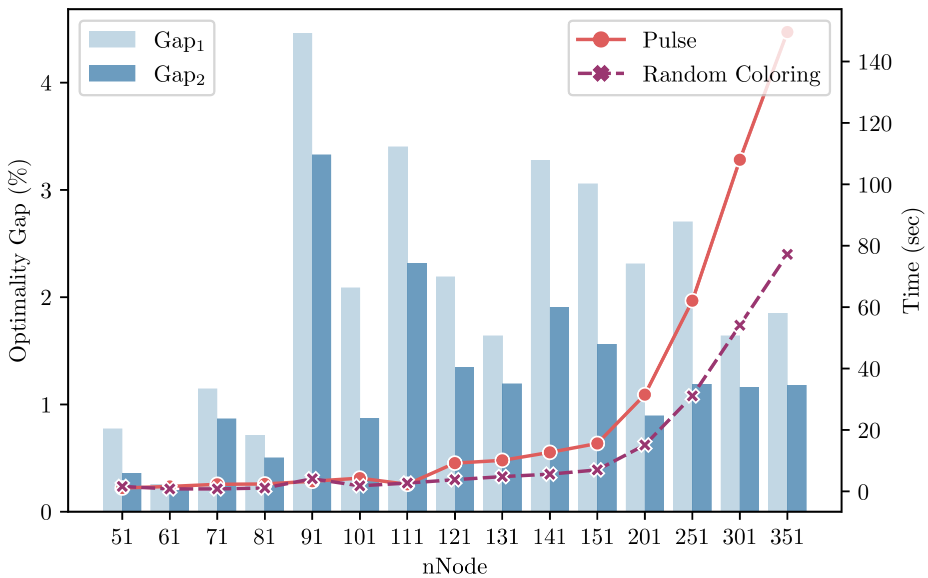

Figure 2: Numerical results for proposed algorithms for Type C instance in parallel implementation ($Q = 4$)

All remaining experiments involve the **parallel implementation** of both pulse and random-coloring. For instances with $|V| \leq 100$, we use the first $|V|+1$ nodes from Solomon's instance and for instances having $|V| > 100$, we use the first $|V|+1$ nodes from Gehring & Homberger's instances. In the following figures, we plot (i) the computational time (in seconds) for solving the column generation LP (Time (sec)), (ii) the optimality gap for random-coloring at the end of column generation (Gap$_1$), and (iii) the optimality gap for random-coloring after column enumeration (Gap$_2$). We note that the optimality gaps under the pulse pricing algorithm are similar. The detailed numerical tables are presented Tables 7–9 and 16–18 in the online supplement, Section A.1. Figures 2–4 summarize the performance of the proposed algorithms on Type C, Type R, and Type RC instances with $Q = 4$. We observe significant improvements for both algorithms when they are implemented in parallel. When implemented in parallel, the random coloring algorithm outperforms the pulse algorithm (in terms of lower bound runtime). In the detailed tables, we highlight (in bold) the

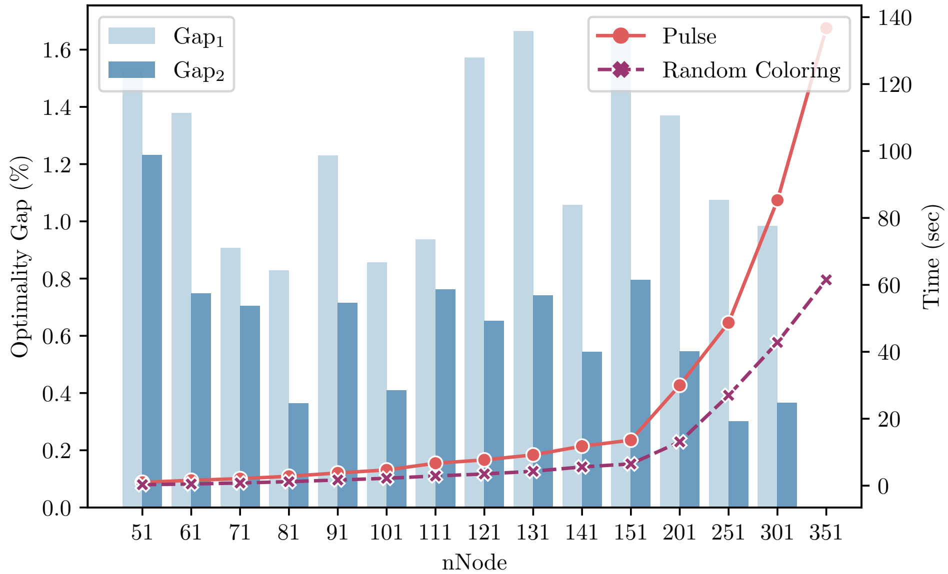

Figure 3: Numerical results for proposed algorithms for Type R instance in parallel implementation ($Q = 4$)

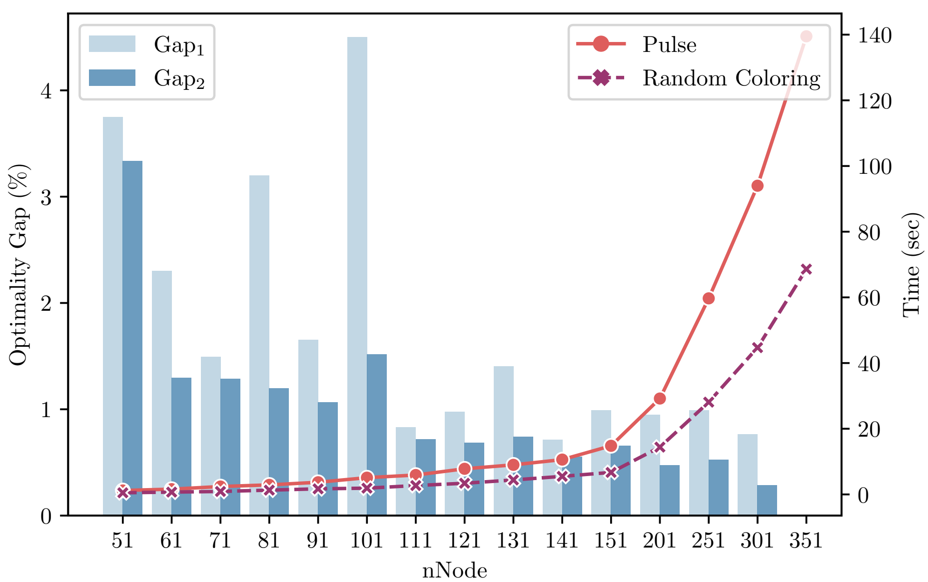

Figure 4: Numerical results for proposed algorithms for Type RC instance in parallel implementation ($Q = 4$)

instances where the random coloring algorithm is faster than pulse. Notice that, although we allow both algorithms to stop early when the number of generated columns reaches a preset limit, we still have one algorithm generating more columns than the other for some instances. This is because when the global number of generated columns reaches the preset bound, there are still some threads keeping a small set of paths pending to update to global column set. We decide to not waste those generated columns. In any case, the number of generated columns is much smaller than the total number of possible columns: For example, the maximum number of generated columns in instances with 301 nodes and $Q = 4$ is less than 7000 (whereas the total possible number is more than 7.9 billion). With the help of cutting planes, the optimality gap, Gap_1, is small for column generation method (even without column enumeration): 2% on average for Type C instances, and 1% on average for the other two types.

Results of column enumeration. We notice that the random coloring algorithm can enumerate columns very efficiently (generating over 100,000 columns within 1.6 seconds), which barely affects the overall runtime. During column enumeration, the number of enumerated columns, whose reduced costs are smaller than $UB_1 - LB$, is approximately 10–20 times larger than the number of generated columns in the initial column generation procedure. Despite the fact that the Gurobi solver generally cannot solve problems of this size to optimality, it still obtains high-quality solutions under the 30-minute time limit. For example, for RC type instances, the average of optimality gaps, Gap_2, is 0.98%, which is improved from 1.71% of Gap_1 obtained earlier using column generation.

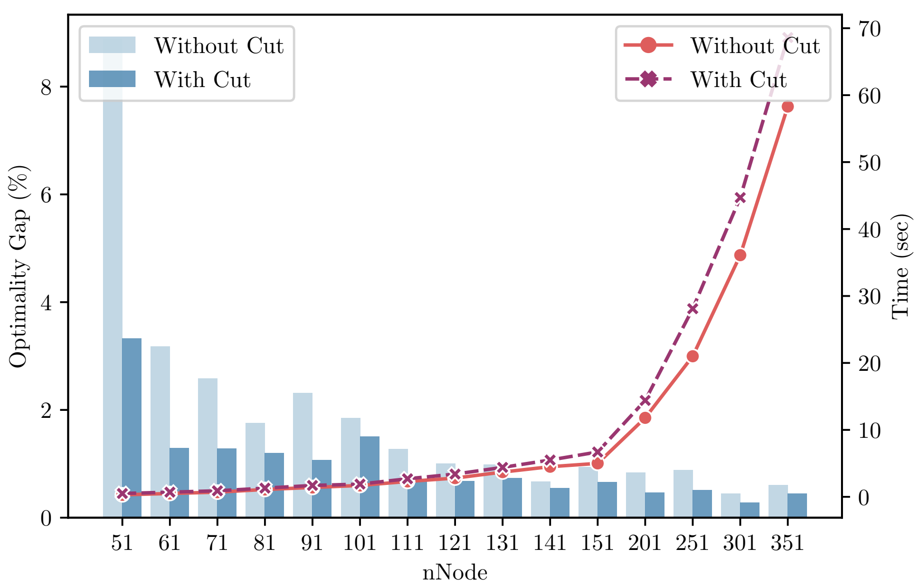

Figure 5: Effect of cutting planes in column generation for Type RC instances ($Q = 4$)

Results of cutting planes. We also study the effect of cutting planes to our solution approaches. Figure 5 demonstrates the differences, in terms of solution time and optimality gap for column generation using random coloring algorithm, for models with and without rounded capacity cuts. We consider the RC instances with $Q = 4$. As shown in the figure, there is a trade-off of using cutting planes. On one hand, introducing cutting planes will increase the solution time for the algorithm. On the other hand, it allows us to obtain better lower bound solution and therefore leads to better optimality gap. On average, for the model with rounded capacity cuts, the solution time increases by 20% while the optimality gap decreases by 50%.

Parallel runtime speedup. We also study the effect of using parallel computing. When running in parallel, the speedup for the pulse algorithm is limited while the random coloring algorithm gets significantly boosted because the runs for different color-codings are completely independent. In this computation, to avoid different lower bounds resulting from different cuts added, we do not enforce any cuts. Figure 6 shows the effect of parallel-implementation for the two algorithms by plotting the average speedup factor across three different types of instances with $Q = 4$ on 50–150 nodes. As shown in the figure, we can observe a significant improvement for the random coloring algorithm as the speedup factor ranges from 5 to 14 using two 20-core processors. On the other hand, the speedup for pulse algorithm is limited.

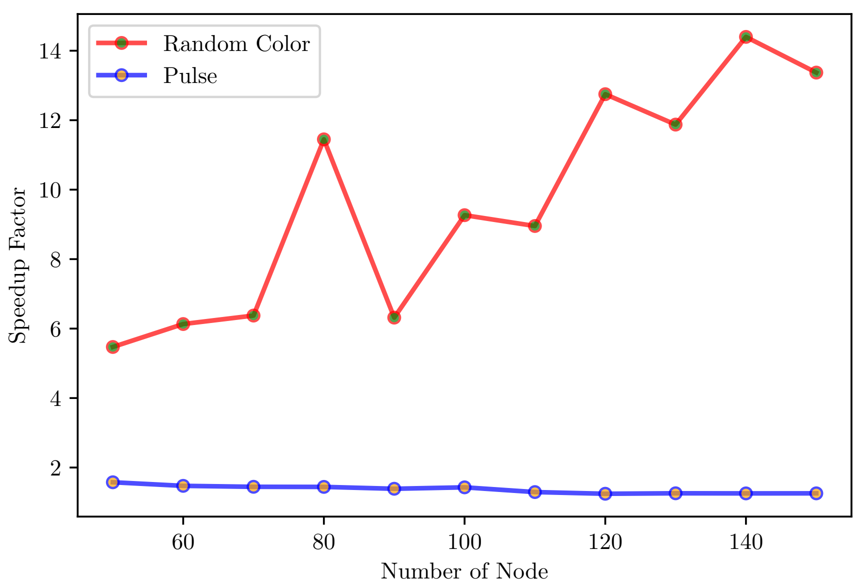

Figure 6: Average speedup factor in parallel implementation for instances with $Q = 4$

Lastly, we conduct numerical experiments with different vehicle capacities. The results for the proposed algorithms on the instances with $Q = 3$ and $Q = 5$ are also presented in the online supplement, Section A.1. It shows that both pulse and random coloring algorithms perform similarly as for the cases of $Q = 4$, but we observe that the pulse algorithm becomes more efficient than the random coloring algorithm as Q increases. In terms of computational time (t_{LB}) for computing LB, random coloring is on average 5 times faster and 1.1 times faster than the pulse algorithm for instances with $Q = 3$ and $Q = 5$, respectively. The optimality gaps, (Gap_2), found by proposed algorithms are on average 0.6% and 0.7% for instances with $Q = 3$ and $Q = 5$, respectively.

5.1.2 Unitary Demand CVRP X-instances

As a special case of CVRP, some unitary demand CVRP instances have been tested in the literature. In particular, Uchoa et al. (2017) proposed a set of new benchmark instances for CVRP. They are generated on a $[0, 1000] \times [0, 1000]$ two-dimensional space with different settings on the number of customers, vehicle capacity, depot locations, and demand distribution. Out of 100 instances in Uchoa et al. (2017), 16 are unitary demand CVRP instances. The original capacity of the vehicle ranges from 3 to 23 in those instances. We test both algorithms (pulse and random coloring) on those instances with modified vehicle capacity $Q = 3$ to $Q = 5$. We use the same node locations of the original instances and compute the distance between any pair of nodes as their Euclidean distance rounded to the nearest integer.

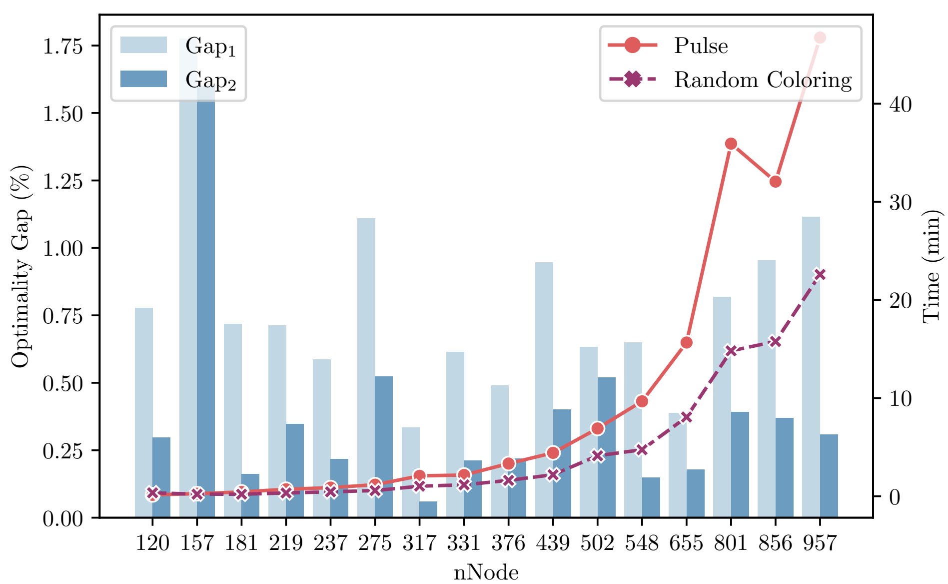

Figure 7: Numerical results for proposed algorithms for X instance in parallel implementation ($Q = 4$)

Figure 7 summarizes the computational time and optimality gap of our proposed approaches for instances derived from unitary demand CVRP X-instances in Uchoa et al. (2017) with $Q = 4$. We report the computational time (in minutes) of column generation (Time (min)) for both pulse and random coloring approaches, the optimality gap at the end of column generation (Gap_1), and the optimality gap obtained using enumerated columns after the column generation (Gap_2). The detailed results are presented in Tables 23–26 in the online supplement, Section A.2. When the capacity of the vehicle is small, both algorithms are capable of solving the root node LP relaxations with up to 957 customer nodes within a reasonable amount of time. The random coloring algorithm outperforms the pulse algorithm in terms of the speed of solving the LP relaxations: The former is on average 2.08 times faster than the latter. We note the efficiency of random coloring when the number of nodes is large. The optimality gaps, Gap_1, obtained by the two algorithms are similar and ranges between 0.29%–1.78% with the average being 0.79%.

Next, we examine the effects of applying column enumeration atop column generation while using random coloring. First, we notice that, as observed previously, the speed for random coloring to enumerate columns is fast, even for the instance with 957 customers. The time taken by column enumeration is only a small fraction of the time taken by column generation (noting that the unit of the runtime is in seconds for enumeration as compared to in minutes when we present the lower-bound results in Table 26). During column enumeration, the number of enumerated columns having reduced costs being smaller than $UB_1 - LB$, is approximately 10–15 times larger than the number of columns generated in the initial column generation procedure. Despite the large number of columns enumerated, we still manage to obtain solutions with excellent quality under the 30-minute time limit. The average optimality gap is only 0.37%, which is a significant improvement from the 0.79% average gap obtained before without using column enumeration.

We provide detailed results of modified X-instances with $Q = 3$ to $Q = 5$ in the online supplement, Section A.2. We observe that the advantages of using the random coloring algorithm, compared with the pulse algorithm, diminish as the capacity of the vehicle increases. The random coloring algorithm is on average 4.31 and 1.11 times faster for instances with $Q = 3$ and $Q = 5$, respectively. The average optimality gaps are 0.13% and 0.89% for instances with $Q = 3$ and $Q = 5$, respectively.

Comparison with results in existing literature. In Uchoa et al. (2017), authors provided the numerical results of BCP from Pecin et al. (2014) and two heuristic approaches, UHGS from Vidal et al. (2012) and ILS-SP from Subramanian et al. (2013) to solve all X-instances. Among them, we solve the instances X-n219-k73, X-n376-k94, and X-n655-k131, which correspond to $Q = 3, 4$, and 5, respectively. We provide a direct comparison between our proposed approach and exiting ones in Table 2. For each approach, we provide the best upper bound solution found (UB) and the computational time (in minutes) to obtain such a solution (Time (min)). We also show the best known solution for each instance (BKS). We note that the computing environments used in these algorithms are different.

Table 2: Comparison of the proposed approach with existing results

Instance	ILS-SP		UHGS		BCP		Random Coloring		BKS
	UB	Time (min)	UB	Time (min)	UB	Time (min)	UB	Time (min)	
X-n219-k73	117595	0.9	117605	7.3	117595	0.5	117595	0.8	117595
X-n376-k94	147713	7.1	147750	28.3	147713	3.3	147721	31.6*	147713
X-n655-k131	106782	47.2	106899	150.5	106780	41.5	107543	60.2*	106780

*: we terminate the solver for MIP to compute UB at 30-minute time limit.

We acknowledge that the BCP approach performs the best among all solution approaches in those instances. However, we note that the BCP approach used in Pecin et al. (2014) is very complex, combines several ideas in the literature and further improves them to solve the CVRP. We were not able to obtain the full BCP code to fairly compare our approach in combination with the BCP with other speed-up tricks. For example, the BCP approach utilizes the value of the best solution found by two metaheuristics as the upper bound, which improves the computational performance in various ways. These two metaheuristics still take a long time to solve VRPUD with small capacity, and their runtime is not included in the BCP time reported above. When using the same upper bound obtained from the metaheuristic method, our approach is also able to close the optimality gap for X-n219-k73 and the total computational time is only 0.2 minute, including the time for solving the MIP.

5.2 Numerical Results on Multi-depot VRPUD

In this section, we discuss an application of VRPUD in a patient-centered medical home system where caregivers route one or multiple fleets of vehicles to serve/treat patients in their homes. Patient-centered medical home has been considered as an effective and economical way to serve patients and is experiencing a fast-growing development (Musich et al., 2015). In 2012, over 4.7 million patients received services from about 12,000 registered home health agencies (Harris-Kojetin et al., 2013) and nowadays patient-centered medical home makes up more than 35% of post-acute care in the market.

According to Fikar and Hirsch (2017), different objectives and constraints of the patient-centered medical home problem have been studied. They summarize that possible objectives are total traveling time, operational cost, total wait time, total overtime, workload balance, number of tasks, etc.; and possible constraints include time windows, skill requirements, working time regulations, breaks, uncertainties, and so on (see, e.g., Allaoua et al., 2013; Bachouch et al., 2011; Dohn et al., 2009; Fernandez et al., 1974; Lanzarone and Matta, 2014). In this section, we model the patient-centered medical home problem as a VRPUD motivated by the observation that the average number of patients that can be visited by a crew is small during one working period. We generalize the problem as VRPUD allowing caregivers to operate the system with multiple bases to start and end their service routes.

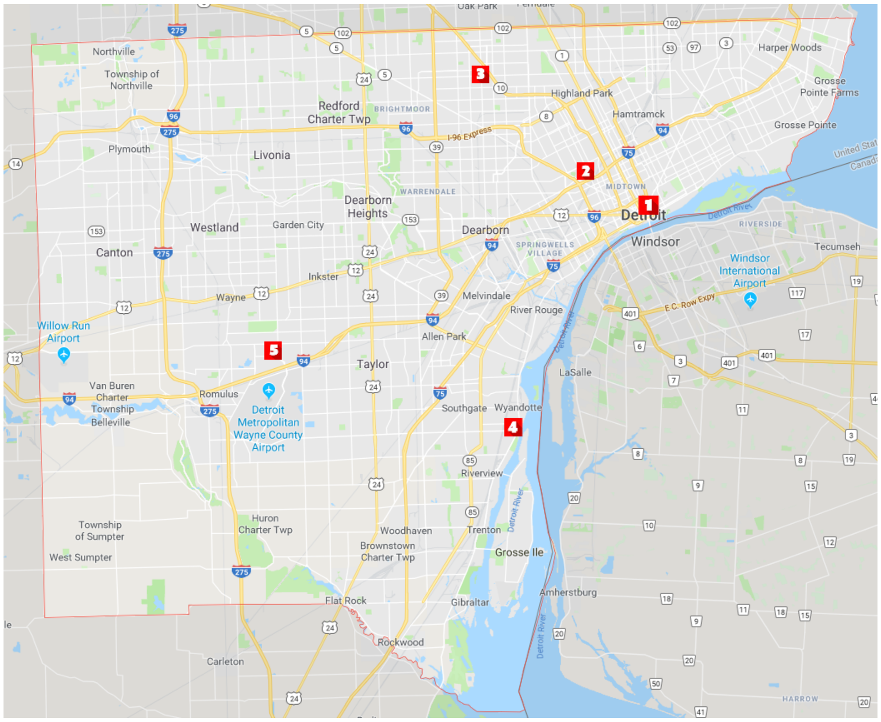

Figure 8: Distribution of hospitals in Wayne County

The test instances are based on the most updated United States Census data for Wayne County in Michigan (see United States Census Bureau 2010[5]). The census data divides Wayne County into 610 different census tracts, and each contains the geographical information (longitude and latitude of the geographical center). The detailed reference map can be found at *Michigan 2010 Census - Census Tract Reference Maps*[6]. We assume that its geographical center represents each census tract and construct a corresponding network with 610 nodes. In addition, we use the geographical information of the top five hospitals in Wayne County as the depot nodes. The five hospitals are (1) Harper University Hospital, (2) Henry Ford Hospital, (3) DMC Sinai-Grace Hospital, (4) Henry Ford Wyandotte Hospital, and (5) Beaumont Hospital-Wayne. The distribution of the hospitals is shown in Figure 8. The travel time between any of two nodes is calculated through Haversine Equation[7]: for any two points with longitude φ_1, φ_2 and latitude λ_1, λ_2, the distance is given by:

$$d((\varphi_1, \lambda_1), (\varphi_2, \lambda_2)) = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

[5] https://www2.census.gov/
[6] https://www2.census.gov/geo/maps/dc10map/tract/st26_mi/c26163_wayne/
[7] https://en.wikipedia.org/wiki/Haversine_formula

In this experiment, we assume that vehicles start and end the route at the same depot (hospital) while covering all the patients. We test both proposed algorithms on the instances with the number of patient nodes ranging from 100 to 500. We test against the instances with 1, 3, or 5 depots and use the parallel implementation of the algorithms. We consider $Q = 4$ in our test instances. For any instance with $|V|$ patient nodes, we randomly pick $|V|$ data points from 610 census tracts as patient nodes. (We do not include the rounded capacity cuts in these computations because the optimality gap is very good even without the additional cuts.)

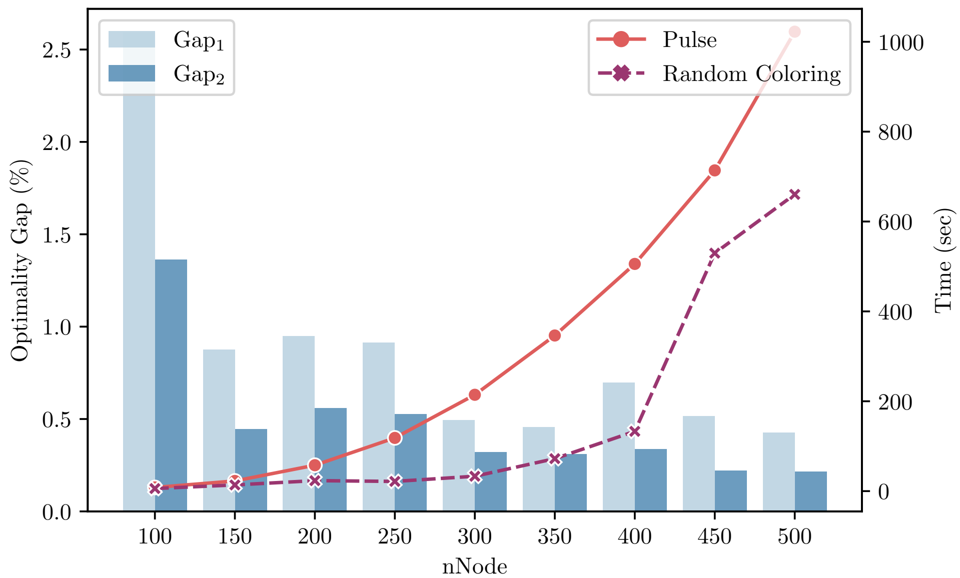

Figure 9: Numerical results on patient-centered medical home instances with 3 depots

Figure 9 and Figure 10 summarize the numerical results of the proposed algorithms on the multi-depot VRPUD embedded in the patient-centered medical home problem. (We demonstrate detailed solutions and other details of the results in the online supplement, Section A.3.) In the two figures, we report the computational time (in seconds) for column generation (Time (sec)), the optimality gap at the end of column generation (Gap$_1$), and the optimality gap after column enumeration (Gap$_2$). As the multi-depot VRPUD requires us to solve the pricing problem based on each depot, the solution time increases when we have three depots instead of one. However, the increase factor is less than 3. Surprisingly, when the number of depots increases to five, the solution time for the instance is shorter than the cases where three depots allowed. We believe that the involvement of more depots, especially new depots (Hospital 4 and 5) separated away from the existing ones in our test instance, would reduce the empirical complexity of the problem. Between using pulse algorithm and random coloring algorithm to solve the column generation, the random coloring algorithm is

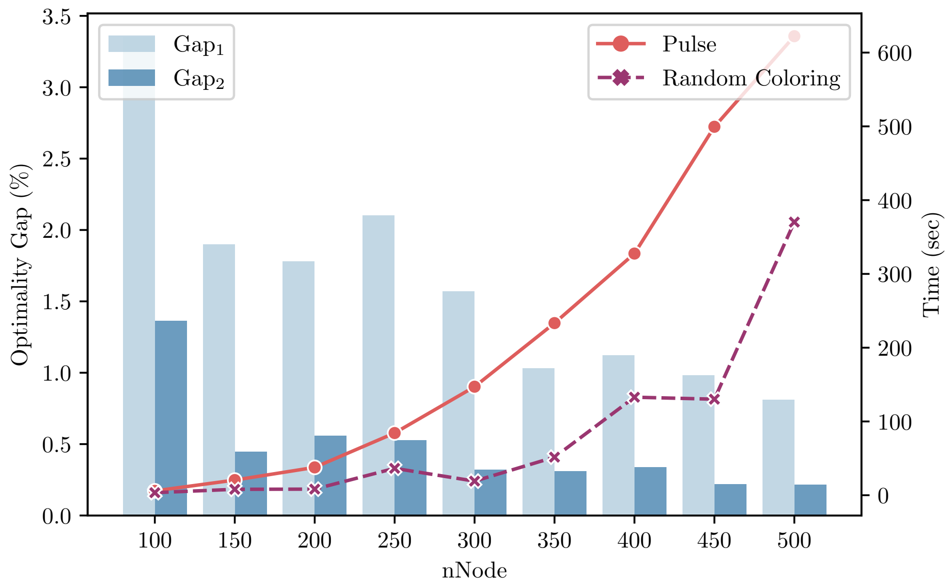

Figure 10: Numerical results on patient-centered medical home instances with 5 depots

more efficient in solving multi-depot instances especially when the number of patient is large.The optimality gap yielded by two algorithms are small (less than 2% in general). After implementing column enumeration with the random coloring algorithm, we notice that the proposed algorithm is capable of enumerating a large number of columns within a small amount of time. For example, it takes 4.9 seconds to enumerate about 310,000 routes for instances with up 500 customer nodes and 5 depots. We find better integer VRPUD solutions with the help of additional enumerated columns, and achieve much better optimality gaps that are less than 0.5% for most of instances. Throughout the experiments, our results show that both algorithms, within a reasonable amount of time, are capable of solving large multi-depot VRPUD instances containing up to 500 patient nodes, which is a practical amount under the context of a patient-centered medical home system in Wayne County. Furthermore, as shown in the numerical results, the optimality gap using the column generation approach is negligibly small considering the size of the instance.

6 Conclusion

In this paper, we studied VRPUD, a special case of multi-depot CVRP where each customer has a unit demand, and applied the column generation method to solve the problem. To efficiently solve the exact pricing problem (ESPPRC) in the column generation approach, we proposed two parallel pricing algorithms: an extension of the pulse algorithm from Lozano and Medaglia (2013) and a randomized algorithm based on the color-coding approach from Alon et al. (1995). Both algorithms could be implemented in parallel to achieve better computational efficiency. In our numerical tests, the random coloring algorithm was typically faster for smaller vehicle capacities. We further combined our methods with other techniques developed in the existing literature for speeding up the computation of diverse VRP instances, including (robust) cutting planes and column enumeration approaches, and observed that they can significantly improve the optimality gaps and lead to high-quality integer solutions.

Acknowledgement

We thank Dr. Leonardo Lozano for providing to us the code used in their paper Lozano et al. (2015). We thank the Associate Editor and reviewers for their constructive feedback and suggestions. The authors are grateful for the support of the NSF grants CMMI-1636876, CMMI-1727618, CCF-1750127 and CMMI-1940766.

References

Adaji, A., Melin, G. J., Campbell, R. L., Lohse, C. M., Westphal, J. J., and Katzelnick, D. J. (2018). Patient-centered medical home membership is associated with decreased hospital admissions for emergency department behavioral health patients. *Population Health Management*, 21(3):172–179.

Allaoua, H., Borne, S., Létocart, L., and Calvo, R. W. (2013). A matheuristic approach for solving a home health care problem. *Electronic Notes in Discrete Mathematics*, 41:471–478.

Alon, N., Dao, P., Hajirasouliha, I., Hormozdiari, F., and Sahinalp, S. C. (2008). Biomolecular network motif counting and discovery by color coding. *Bioinformatics*, 24(13):i241–i249.

Alon, N., Yuster, R., and Zwick, U. (1995). Color-coding. *J. ACM*, 42(4):844–856.

American Academy of Family Physicians (2008). Joint principles of the patient-centered medical home. *Delaware Medical Journal*, 80(1):21.

Bachouch, R. B., Guinet, A., and Hajri-Gabouj, S. (2011). A decision-making tool for home health care nurses' planning. In *Supply Chain Forum: An International Journal*, volume 12, pages 14–20. Taylor & Francis.

Baldacci, R., Bartolini, E., Mingozzi, A., and Roberti, R. (2010). An exact solution framework for a broad class of vehicle routing problems. *Computational Management Science*, 7(3):229–268.

Baldacci, R., Christofides, N., and Mingozzi, A. (2008). An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115(2):351–385.

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

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W., and Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329.

Bettinelli, A., Ceselli, A., and Righini, G. (2011). A branch-and-cut-and-price algorithm for the multi-depot heterogeneous vehicle routing problem with time windows. *Transportation Research Part C: Emerging Technologies*, 19(5):723–740.

Braekers, K., Ramaekers, K., and Van Nieuwenhuyse, I. (2016). The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*, 99:300–313.

Christofides, N., Mingozzi, A., and Toth, P. (1981). Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical Programming*, 20(1):255–282.

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

Dabia, S., Ropke, S., Van Woensel, T., and De Kok, T. (2013). Branch and price for the time-dependent vehicle routing problem with time windows. *Transportation Science*, 47(3):380–396.

Desaulniers, G., Desrosiers, J., and Solomon, M. M. (2006). *Column Generation*. Springer Science & Business Media.

Desrosiers, J., Dumas, Y., Solomon, M. M., and Soumis, F. (1995). Time constrained routing and scheduling. *Handbooks in Operations Research and Management Science*, 8:35–139.

Dohn, A., Kolind, E., and Clausen, J. (2009). The manpower allocation problem with time windows and job-teaming constraints: A branch-and-price approach. *Computers & Operations Research*, 36(4):1145–1157.

Downey, R. G. and Fellows, M. R. (2012). *Parameterized Complexity*. Springer Science & Business Media.

Dror, M. (1994). Note on the complexity of the shortest path models for column generation in vrptw. *Operations Research*, 42(5):977–978.

Duque, D., Lozano, L., and Medaglia, A. L. (2015). Solving the orienteering problem with time windows via the pulse framework. *Computers & Operations Research*, 54:168–176.

Eveborn, P., Flisberg, P., and Rönnqvist, M. (2006). Laps care—an operational system for staff planning of home care. *European Journal of Operational Research*, 171(3):962–976.

Feillet, D., Dejax, P., Gendreau, M., and Gueguen, C. (2004). An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229.

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

Fernandez, J., Blacking, J., Dundes, A., Edmonson, M. S., Etzkorn, K. P., Haydu, G. G., Kearney, M., Kehoe, A. B., Loveland, F., McCormack, W. C., et al. (1974). The mission of metaphor in expressive culture. *Current Anthropology*, pages 119–145.

Fikar, C. and Hirsch, P. (2017). Home health care routing and scheduling: A review. *Computers & Operations Research*, 77:86–95.

Fukasawa, R., He, Q., and Song, Y. (2015). A branch-cut-and-price algorithm for the energy minimization vehicle routing problem. *Transportation Science*, 50(1):23–34.

Fukasawa, R., Longo, H., Lysgaard, J., de Aragão, M. P., Reis, M., Uchoa, E., and Werneck, R. F. (2006). Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming*, 106(3):491–511.

Harris-Kojetin, L., Sengupta, M., Park-Lee, E., and Valverde, R. (2013). Long-term care services in the united states: 2013 overview. *Vital & health statistics. Series 3, Analytical and epidemiological studies/[US Dept. of Health and Human Services, Public Health Service, National Center for Health Statistics]*, 3(37):1–107.

Irnich, S. and Villeneuve, D. (2006). The shortest-path problem with resource constraints and k-cycle elimination for $k \geq 3$. *INFORMS Journal on Computing*, 18(3):391–406.

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

Kohl, N., Desrosiers, J., Madsen, O. B., Solomon, M. M., and Soumis, F. (1999). 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, 33(1):101–116.

Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50.

Lanzarone, E. and Matta, A. (2014). Robust nurse-to-patient assignment in home care services to minimize overtimes under continuity of care. *Operations Research for Health Care*, 3(2):48–58.

Lozano, L., Duque, D., and Medaglia, A. L. (2015). An exact algorithm for the elementary shortest path problem with resource constraints. *Transportation Science*, 50(1):348–357.

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

Lysgaard, J., Letchford, A. N., and Eglese, R. W. (2004). A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100(2):423–445.

Musich, S., Wang, S. S., Hawkins, K., and Yeh, C. S. (2015). Homebound older adults: Prevalence, characteristics, health care utilization and quality of care. *Geriatric Nursing*, 36(6):445–450.

Pecin, D., Contardo, C., Desaulniers, G., and 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., and Uchoa, E. (2014). Improved branch-cut-and-price for capacitated vehicle routing. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 393–403. Springer.

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

Poggi de Aragao, M. and Uchoa, E. (2003). Integer program reformulation for robust branch-and-cut-and-price algorithms. In *In Proceedings of the Conference Mathematical Program in Rio: A Conference in Honour of Nelson Maculan*. Citeseer.

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

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265.

Subramanian, A., Uchoa, E., and Ochi, L. S. (2013). A hybrid algorithm for a class of vehicle routing problems. *Computers & Operations Research*, 40(10):2519–2531.

The National Association for Home Care & Hospice (2010). Basic statistics about home care. Technical report, The National Association for Home Care & Hospice.

Toth, P. and Vigo, D. (2014). *Vehicle Routing: Problems, Methods, and Applications*, volume 18. SIAM.

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

Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., and Rei, W. (2012). A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, 60(3):611–624.

Online Supplement of the Paper

"Improving Column-Generation for Vehicle Routing Problems via Random Coloring and Parallelization"

Miao Yu, Viswanath Nagarajan, Siqian Shen

Department of Industrial and Operations Engineering,

University of Michigan, Ann Arbor, MI, USA

A Detailed Numerical Results

We summarize all the numerical results for the column generation approach on VRPUD and multi-depot VRPUD with different vehicle capacities Q. We consider both pulse algorithm and the random coloring algorithm as the pricing algorithm for the column generation approach. Both algorithms have been implemented in parallel using 40 computer threads unless otherwise noted. Table 3 summarizes the results for original label correcting algorithm (Feillet et al., 2004), pulse algorithm, and proposed random coloring algorithm in their serial implementation to solve modified Solomon instances. Tables 4–21 summarize the results for modified Solomon and Gehring & Homerberger's benchmark. For instances with a number of customers $|V| \leq 100$, we use the first $|V|+1$ nodes from Solomon's instance and for $|V| > 100$, we use the first $|V|+1$ nodes from Gehring & Homberger's instances. Tables 22–27 summarize the results for modified unitary CVRP X-instances with $Q=3$ to $Q=5$. Lastly, Tables 28–29 summarize the results for multi-depot VRPUD in patient-centered medical home problem with number of depots ranging from 1 to 5.

In all our result tables, for solving the LP relaxation of MP through column generation, *nIter* represents the number of iterations; *nCol* represents the number of columns generated; *LB* represents the lower bound of the optimal objective value of MP; t_{LB} represents the runtime of column generation. When solving the RMP as an integer program using the generated columns through column generation, UB_1 represents the upper bound of MP from the resulting integer solution; Gap_1 (in %) represents the optimality gap computed as $\frac{UB_1 - LB}{LB} \times 100\%$; t_{UB_1} represents the runtime of computing the upper bound. The time limit is set as 30 minutes. We also apply column enumeration to obtain improved integer solutions. In all related tables, $nCol_e$ represents the number of columns enumerated in column enumeration; t_e (in seconds) represents the runtime of enumerating columns; UB_2 represents the upper bound of MP found by Gurobi solver using enumerated columns; Gap_2 (in %) represents the optimality gap computed as $\frac{UB_2-LB}{LB} \times 100\%$; t_{UB_2} represents the runtime of computing UB_2. The time limit of using solver to find upper bound is set as 30 minutes. In all the tables, we remark columns reporting time with "(s)" if the time unit is in seconds; otherwise, the reporting time unit is in minutes and remarked by "(m)".

A.1 Solomon and Gehring & Homberger Instances

Table 3: Numerical results for proposed algorithms in serial implementation ($Q = 4$)

Type	nNodes	Label Correcting			Pulse			Random Coloring		
		nCol	LB	t_{LB} (s)	nCol	LB	t_{LB} (s)	nCol	LB	t_{LB} (s)
C	51	685	694.31	4.96	863	694.31	1.44	764	694.31	5.67
	61	764	902.46	20.13	1077	902.46	1.64	842	902.46	3.93
	71	920	1088.31	34.98	1171	1088.31	1.94	957	1088.31	6.13
	81	1027	1266.57	47.04	1394	1266.57	2.71	1047	1266.57	8.58
	91	1128	1437.82	37.77	1632	1437.82	3.73	1307	1437.82	13.42
	101	1193	1643.44	49.37	1710	1643.44	4.42	1376	1643.44	18.93
	111	1449	3211.32	367.57	2233	3211.32	8.06	1517	3211.32	61.02
	121	1868	3501.8	420.60	2673	3501.80	8.76	1777	3501.80	70.50
	131	2049	3798.77	557.21	2825	3798.77	10.30	1964	3798.77	106.54
	141	2053	4136.82	–	3089	4096.78	15.95	2063	4096.78	119.24
	151	2263	4442.25	–	3191	4398.63	13.61	2188	4398.63	137.89
R	51	569	916.81	4.08	707	916.81	1.42	754	916.81	3.67
	61	749	1029.79	14.00	843	1029.79	1.51	898	1029.79	7.10
	71	940	1235.64	28.42	987	1235.64	1.89	984	1235.64	10.50
	81	1038	1375.34	45.58	1234	1375.34	2.98	1132	1375.34	15.99
	91	1103	1510.59	52.83	1356	1510.59	3.51	1291	1510.59	26.61
	101	1267	1612.58	124.91	1456	1612.58	4.66	1481	1612.58	34.60
	111	1533	3508	283.43	1771	3508.00	5.26	1661	3508.00	61.99
	121	1829	3775.6	725.47	1847	3775.60	5.15	1834	3775.60	75.67
	131	1943	4123.27	–	2171	4107.18	6.58	1981	4107.18	93.00
	141	1946	4393.12	–	2538	4364.25	8.87	2082	4364.25	112.32
	151	2104	4669.05	–	2563	4624.60	10.40	2322	4624.60	150.79
RC	51	596	1124.15	4.01	596	1124.15	1.13	688	1124.15	2.85
	61	758	1349.72	8.78	836	1349.72	1.46	872	1349.72	7.07
	71	819	1488.27	25.76	1245	1488.27	2.28	920	1488.27	8.10
	81	991	1717.44	52.22	1247	1717.44	2.69	1130	1717.44	14.74
	91	1033	1871.52	71.30	1524	1871.53	4.09	1176	1871.53	21.53
	101	1244	1994.13	107.79	1684	1994.13	4.70	1394	1994.13	30.36
	111	1582	3459.84	117.58	2010	3459.84	7.45	1726	3459.84	54.83
	121	1688	3832.77	366.14	2224	3832.77	8.25	1892	3832.77	79.02
	131	1893	4190.07	–	2428	4174.07	9.74	2043	4174.07	87.58
	141	2039	4428.03	–	2646	4396.25	10.85	2145	4396.25	95.44
	151	2104	4731.66	–	2763	4685.09	13.53	2162	4685.09	121.90

– : runtime exceeds time limit of 15 minutes

Table 4: Numerical results for proposed algorithms for Type C instance in parallel implementation for $Q = 3$

nNode	Pulse							Random Coloring						
	nIter	nCol	LB	UB_1	Gap_1	t_{LB} (s)	t_{UB_1} (s)	nIter	nCol	LB	UB_1	Gap_1	t_{LB} (s)	t_{UB_1} (s)
51	52	444	902.6	924.4	2.41%	1.1	0.2	21	421	903.2	933.0	3.30%	1.4	0.5
61	73	485	1174.8	1196.3	1.83%	1.5	0.3	29	535	1180.3	1184.7	0.37%	**0.8**	0.1
71	57	557	1377.6	1419.7	3.05%	1.4	0.6	26	572	1383.5	1411.9	2.05%	**0.8**	1.1
81	75	746	1654.6	1687.5	1.99%	2.1	0.9	37	666	1660.1	1695.5	2.13%	**1.1**	1.3
91	80	692	1843.6	1905.4	3.35%	2.5	0.9	34	720	1859.5	1882.3	1.23%	**0.8**	0.6
101	89	862	2133.5	2158.5	1.17%	3.5	0.7	42	825	2139.8	2168.0	1.32%	**1.3**	1.2
111	96	978	4227.0	4313.8	2.05%	1.9	0.1	45	1013	4226.6	4286.1	1.41%	**1.0**	3.0
121	106	1231	4597.8	4685.4	1.90%	1.9	2.2	52	1092	4604.0	4682.8	1.71%	**1.3**	6.7
131	123	1177	4969.3	5069.3	2.01%	2.1	0.2	56	1258	4998.2	5070.5	1.45%	**1.7**	3.3
141	116	1284	5359.7	5575.3	4.02%	2.4	4.5	56	1218	5391.5	5553.2	3.00%	**1.9**	20.7
151	127	1399	5753.5	5879.1	2.18%	2.9	0.8	60	1259	5789.8	5867.1	1.33%	**2.2**	4.5
201	180	1993	14575.4	14893.9	2.18%	22.5	14.7	83	1736	14632.5	14813.8	1.24%	**5.2**	19.7
251	226	2598	17956.0	18243.2	1.60%	44.2	13.5	101	2226	18004.7	18276.9	1.51%	**8.6**	44.3
301	242	3294	21698.3	22078.5	1.75%	68.9	49.4	122	2792	21766.2	22053.1	1.32%	**15.4**	48.5
351	290	3925	24945.7	25381.4	1.75%	113.7	78.1	147	3174	25039.4	25319.5	1.12%	**23.3**	63.2

Table 5: Numerical results for proposed algorithms for Type R instance in parallel implementation ($Q = 3$)

nNode	Pulse							Random Coloring						
	nIter	nCol	LB	UB_1	Gap_1	t_{LB} (s)	t_{UB_1} (s)	nIter	nCol	LB	UB_1	Gap_1	t_{LB} (s)	t_{UB_1} (s)
51	51	386	1123.9	1131.4	0.67%	0.8	0.1	18	402	1125.5	1131.9	0.57%	**0.2**	0.2
61	59	446	1269.6	1275.7	0.48%	1.2	0.1	24	501	1269.1	1270.6	0.12%	**0.3**	0.1
71	68	588	1528.3	1541.0	0.83%	1.6	0.5	25	558	1528.7	1539.8	0.73%	**0.4**	0.8
81	80	662	1703.9	1713.8	0.58%	2.2	0.5	29	652	1703.1	1714.3	0.66%	**0.5**	0.8
91	93	815	1872.9	1881.8	0.47%	3.1	0.3	34	749	1872.9	1879.0	0.33%	**0.6**	0.7
101	99	898	2004.1	2017.9	0.69%	3.8	0.8	34	775	2005.5	2015.4	0.49%	**0.7**	1.3
111	100	994	4399.4	4427.6	0.64%	4.5	1.0	39	873	4398.8	4422.2	0.53%	**1.0**	1.1
121	108	1107	4749.1	4771.4	0.47%	5.5	0.7	45	995	4748.8	4774.3	0.54%	**1.2**	1.0
131	129	1152	5174.2	5210.2	0.69%	7.3	1.1	47	1077	5176.1	5216.4	0.78%	**1.4**	2.9
141	118	1299	5526.8	5589.4	1.13%	7.7	2.4	50	1124	5525.3	5567.5	0.76%	**1.8**	2.3
151	137	1400	5879.6	5932.2	0.89%	10.2	2.5	54	1213	5874.2	5922.0	0.81%	**2.1**	5.1
201	174	2028	15948.8	16074.3	0.79%	21.7	3.6	73	1680	15975.1	16066.4	0.57%	**4.8**	2.9
251	211	2625	19584.6	19653.5	0.35%	41.8	3.3	95	2097	19590.5	19679.3	0.45%	**8.2**	8.7
301	238	3154	23309.0	23433.6	0.53%	67.8	8.1	106	2501	23321.4	23469.6	0.64%	**13.2**	20.3
351	279	3803	27241.4	27396.3	0.57%	108.8	13.1	131	2987	27253.4	27422.9	0.62%	**20.9**	41.9

Table 6: Numerical results for proposed algorithms for Type RC instance in parallel implementation ($Q = 3$)

nNode	Pulse							Random Coloring						
	nIter	nCol	LB	UB_1	Gap_1	t_{LB} (s)	t_{UB_1} (s)	nIter	nCol	LB	UB_1	Gap_1	t_{LB} (s)	t_{UB_1} (s)
51	54	436	1521.9	1591.6	4.58%	0.8	0.2	20	412	1517.4	1564.4	3.10%	**0.2**	0.7
61	59	483	1737.5	1775.4	2.18%	1.1	0.9	24	517	1736.0	1770.0	1.96%	**0.3**	1.0
71	78	618	1896.1	1917.8	1.14%	1.8	0.4	34	601	1910.2	1915.8	0.29%	**0.5**	0.4
81	84	712	2182.6	2209.0	1.21%	2.4	1.4	34	708	2187.7	2199.3	0.53%	**0.6**	0.9
91	82	806	2380.7	2403.8	0.97%	2.8	0.8	32	683	2380.8	2397.5	0.70%	**0.6**	0.5
101	93	893	2541.8	2548.1	0.25%	3.6	0.3	38	796	2532.1	2568.1	1.42%	**0.8**	0.5
111	106	1014	4396.7	4444.6	1.09%	5.0	0.9	45	1015	4405.8	4428.2	0.51%	**1.0**	1.1
121	115	1095	4864.2	4893.3	0.60%	5.8	0.8	45	1039	4864.9	4906.2	0.85%	**1.1**	3.2
131	111	1272	5309.5	5343.2	0.63%	6.4	1.2	48	1096	5313.2	5335.8	0.43%	**1.4**	1.4
141	123	1338	5607.9	5656.1	0.86%	7.9	1.9	62	1278	5613.7	5648.2	0.61%	**2.2**	4.0
151	137	1554	5992.3	6047.3	0.92%	10.1	3.8	61	1348	5998.7	6049.0	0.84%	**2.3**	5.8
201	173	2081	15804.8	15915.6	0.70%	21.6	2.8	80	1704	15813.5	15911.2	0.62%	**5.1**	6.2
251	219	2719	19208.7	19327.8	0.62%	43.3	17.6	96	2140	19201.6	19335.9	0.70%	**8.0**	11.1
301	245	3276	22918.4	23013.8	0.42%	70.6	6.2	114	2560	22911.6	23109.1	0.86%	**14.4**	45.6
351	275	3706	26738.8	26866.2	0.48%	107.6	17.1	134	2989	26757.0	26865.2	0.40%	**21.1**	22.9

Table 7: Numerical results for proposed algorithms for Type C instance in parallel implementation ($Q = 4$)

nNode	Pulse							Random Coloring						
	nIter	nCol	LB	UB_1	Gap_1	t_{LB} (s)	t_{UB_1} (s)	nIter	nCol	LB	UB_1	Gap_1	t_{LB} (s)	t_{UB_1} (s)
51	57	949	719.9	725.1	0.72%	1.1	0.4	22	1370	721.3	726.9	0.78%	1.6	1.3
61	70	1082	915.7	948.5	3.58%	1.6	0.9	30	1604	929.3	931.7	0.26%	**0.8**	1.9
71	76	1352	1114.0	1125.9	1.07%	2.3	1.0	30	1737	1113.9	1126.7	1.15%	**0.8**	1.8
81	77	1322	1289.5	1290.2	0.06%	2.4	1.2	31	1808	1283.4	1292.6	0.72%	**1.1**	1.8
91	86	1612	1480.8	1524.1	2.93%	3.3	3.2	33	2082	1456.8	1521.8	4.46%	4.1	4.5
101	96	1830	1694.9	1715.6	1.22%	4.3	1.7	34	2105	1691.7	1727.1	2.09%	**1.8**	4.2
111	139	2371	3307.8	3359.0	1.55%	2.3	1.3	42	2598	3272.0	3383.4	3.41%	2.7	5.2
121	135	2602	3612.6	3698.0	2.36%	9.2	11.8	50	2988	3617.8	3697.2	2.20%	**3.8**	15.0
131	136	2623	3882.8	4027.3	3.72%	10.1	11.0	53	3156	3884.0	3947.9	1.65%	**4.8**	7.9
141	147	2916	4176.6	4321.2	3.46%	12.7	19.2	57	3290	4170.7	4307.5	3.28%	**5.6**	18.8
151	161	3271	4513.4	4624.3	2.46%	15.6	34.3	62	3849	4498.8	4636.4	3.06%	**7.0**	59.1
201	211	4275	11378.7	11631.6	2.22%	31.5	50.5	79	4718	11372.9	11636.1	2.31%	**15.1**	61.3
251	271	5690	13908.9	14181.5	1.96%	62.1	116.3	101	5871	13889.1	14264.8	2.71%	**31.1**	138.2
301	327	6714	16651.2	16885.0	1.40%	108.0	57.1	128	7194	16641.7	16915.4	1.64%	**54.1**	90.1
351	342	7791	19109.4	19491.6	2.00%	149.6	571.6	139	7996	19103.8	19457.5	1.85%	**77.2**	487.3

Table 8: Numerical results for proposed algorithms for Type R instance in parallel implementation ($Q = 4$)

nNode	Pulse							Random Coloring						
	nIter	nCol	LB	UB_1	Gap_1	t_{LB} (s)	t_{UB_1} (s)	nIter	nCol	LB	UB_1	Gap_1	t_{LB} (s)	t_{UB_1} (s)
51	57	816	918.1	944.8	2.90%	1.0	0.8	17	1199	916.8	930.8	1.53%	**0.3**	0.7
61	70	1071	1029.9	1044.2	1.39%	1.6	0.3	21	1494	1029.8	1044.0	1.38%	**0.5**	0.3
71	76	1134	1236.7	1256.7	1.62%	2.1	1.0	24	1640	1235.6	1246.8	0.90%	**0.8**	0.8
81	85	1414	1375.3	1382.7	0.53%	2.8	0.4	30	1852	1375.5	1386.9	0.83%	**1.2**	1.3
91	99	1553	1510.6	1537.0	1.75%	3.8	1.5	32	2115	1511.7	1530.3	1.23%	**1.7**	2.0
101	103	1727	1612.7	1628.2	0.96%	4.7	0.5	36	2276	1612.7	1626.5	0.85%	**2.2**	1.5
111	125	2105	3515.3	3561.8	1.32%	6.7	4.0	41	2488	3514.9	3547.8	0.94%	**2.9**	3.9
121	128	2193	3777.3	3831.3	1.43%	7.7	0.9	42	2686	3783.9	3843.4	1.57%	**3.5**	6.5
131	134	2382	4116.6	4173.6	1.39%	9.2	3.4	46	2890	4115.5	4184.0	1.66%	**4.3**	6.0
141	155	2694	4382.2	4422.3	0.92%	11.8	2.1	50	3140	4376.5	4422.8	1.06%	**5.6**	5.3
151	155	2936	4627.6	4674.2	1.01%	13.6	4.8	52	3436	4625.3	4702.6	1.67%	**6.5**	14.6
201	201	3867	12503.8	12657.5	1.23%	30.0	13.6	66	4208	12499.4	12670.7	1.37%	**13.1**	10.0
251	211	4732	15263.0	15473.4	1.38%	48.7	80.2	85	5453	15280.6	15444.8	1.07%	**27.0**	12.3
301	259	5660	18122.3	18270.3	0.82%	85.3	16.7	100	6511	18125.2	18303.5	0.98%	**42.8**	94.8
351	311	6923	21119.0	21317.5	0.94%	136.7	77.7	109	6987	21123.8	21333.3	0.99%	**61.5**	172.1

Table 9: Numerical results for proposed algorithms for Type RC instance in parallel implementation ($Q = 4$)

nNode	Pulse							Random Coloring						
	nIter	nCol	LB	UB_1	Gap_1	t_{LB} (s)	t_{UB_1} (s)	nIter	nCol	LB	UB_1	Gap_1	t_{LB} (s)	t_{UB_1} (s)
51	60	1079	1217.3	1221.3	0.33%	1.2	1.5	23	1236	1184.6	1229.0	3.75%	**0.5**	5.5
61	74	1103	1375.0	1418.0	3.12%	1.6	3.3	27	1565	1372.0	1403.6	2.30%	**0.7**	6.8
71	87	1269	1510.6	1543.2	2.16%	2.4	1.9	28	1665	1507.4	1529.9	1.49%	**0.9**	3.7
81	89	1392	1740.0	1763.2	1.33%	2.9	2.3	31	1853	1728.4	1783.7	3.20%	**1.3**	10.5
91	92	1551	1899.3	1913.5	0.75%	3.7	1.3	34	2068	1893.3	1924.6	1.65%	**1.7**	3.7
101	110	1850	2018.0	2057.8	1.97%	5.1	2.6	34	2222	2000.7	2090.7	4.50%	**1.9**	17.6
111	117	2073	3469.4	3506.2	1.06%	5.9	1.6	41	2621	3478.8	3507.6	0.83%	**2.7**	2.5
121	132	2403	3837.1	3919.6	2.15%	7.8	3.3	45	2793	3844.9	3882.5	0.98%	**3.4**	3.5
131	133	2471	4178.5	4240.7	1.49%	9.0	5.2	50	2961	4184.4	4243.2	1.40%	**4.4**	8.8
141	138	2741	4400.8	4458.1	1.30%	10.6	2.3	53	3208	4403.9	4435.2	0.71%	**5.5**	2.8
151	167	3194	4698.0	4750.8	1.12%	14.8	18.0	58	3344	4698.2	4744.6	0.99%	**6.7**	5.1
201	199	3982	12400.2	12551.4	1.22%	29.2	14.5	70	4222	12413.8	12531.2	0.95%	**14.4**	10.7
251	257	5147	14928.3	15144.8	1.45%	59.7	97.0	96	5698	14945.3	15093.5	0.99%	**28.1**	137.0
301	284	6222	17717.7	17877.0	0.90%	94.0	44.8	106	6441	17732.7	17867.9	0.76%	**44.7**	22.2
351	320	7130	20685.4	20907.9	1.08%	139.5	98.2	125	7748	20689.6	20920.3	1.12%	**68.6**	361.5

Table 10: Numerical results for proposed algorithms for Type C instance in parallel implementation ($Q = 5$)

nNode	Pulse							Random Coloring						
	nIter	nCol	LB	UB_1	Gap$_1$	t_{LB} (s)	t_{UB_1} (s)	nIter	nCol	LB	UB_1	Gap$_1$	t_{LB} (s)	t_{UB_1} (s)
51	68	1312	589.45417	612.6	3.93%	1.5	1.0	28	2975	594.9	594.9	0.00%	2.8	0.8
61	78	1639	770.03333	770.1	0.01%	2.1	0.2	28	3029	768.8	770.1	0.17%	**1.6**	0.9
71	86	1921	934.52969	956.9	2.39%	2.8	2.6	29	3893	937.6	947.8	1.09%	**2.7**	1.8
81	106	2182	1088.4875	1101	1.15%	4.2	0.6	34	3915	1084.9	1117.0	2.96%	**3.2**	6.1
91	126	2459	1226.0092	1257.3	2.55%	6.0	3.1	37	4802	1226.6	1252.4	2.10%	11.3	9.1
101	141	2850	1393.4448	1401.4	0.57%	8.1	0.7	37	4987	1388.3	1412.7	1.75%	**6.0**	8.6
111	175	3814	2691.3854	2770.4	2.94%	13.8	7.2	50	6057	2702.6	2754.0	1.90%	14.1	13.0
121	179	4076	3000.4116	3092.3	3.06%	14.5	23.5	51	7030	2997.8	3100.8	3.43%	**13.9**	104.0
131	167	4021	3184.4195	3302.5	3.71%	15.3	10.8	47	6571	3173.4	3301.5	4.04%	**14.0**	45.9
141	193	4621	3429.9881	3531.4	2.96%	21.6	30.6	54	7032	3432.0	3554.2	3.56%	**19.8**	91.7
151	208	5013	3703.9952	3821.5	3.17%	25.5	156.2	59	7810	3728.5	3770.9	1.14%	**23.3**	69.9

Table 11: Numerical results for proposed algorithms for Type R instance in parallel implementation ($Q = 5$)

nNode	Pulse							Random Coloring						
	nIter	nCol	LB	UB_1	Gap$_1$	t_{LB} (s)	t_{UB_1} (s)	nIter	nCol	LB	UB_1	Gap$_1$	t_{LB} (s)	t_{UB_1} (s)
51	67	1128	793.725	816.7	2.89%	1.4	0.7	20	2392	793.9	816.1	2.80%	**1.0**	0.8
61	76	1428	892.58981	921.4	3.23%	2.0	1.2	20	3175	892.7	915.1	2.51%	**1.4**	0.8
71	88	1640	1067.3075	1086.9	1.84%	2.9	1.2	26	3505	1067.3	1084.8	1.64%	**2.4**	3.3
81	102	1944	1178.4755	1212.1	2.85%	4.0	2.5	28	3842	1178.5	1200.8	1.89%	**3.2**	5.6
91	127	2300	1291.238	1335.5	3.43%	6.1	4.1	30	4341	1290.8	1308.3	1.36%	**4.4**	7.4
101	122	2432	1373.7051	1418.8	3.28%	6.9	5.8	33	4647	1373.7	1414.2	2.95%	**5.9**	13.9
111	141	2978	2980.9473	3040	1.98%	9.3	4.6	41	5398	2973.4	3042.9	2.34%	10.1	11.5
121	146	3186	3197.1936	3234.2	1.16%	11.3	3.6	42	5991	3191.8	3226.8	1.10%	12.0	4.6
131	170	3600	3459.5512	3532.4	2.11%	14.6	8.4	45	6198	3451.9	3525.9	2.14%	**14.0**	23.0
141	174	3996	3664.3558	3743.8	2.17%	18.7	20.9	49	6671	3664.2	3748.5	2.30%	**17.7**	21.0
151	183	4036	3883.3897	3945.8	1.61%	21.1	3.6	53	7235	3882.6	3934.2	1.33%	21.9	8.1

Table 12: Numerical results for proposed algorithms for Type RC instance in parallel implementation ($Q = 5$)

nNode	Pulse							Random Coloring						
	nIter	nCol	LB	UB_1	Gap$_1$	t_{LB} (s)	t_{UB_1} (s)	nIter	nCol	LB	UB_1	Gap$_1$	t_{LB} (s)	t_{UB_1} (s)
51	76	1371	922.5	922.5	0.00%	1.5	0.0	18	2194	922.5	922.5	0.00%	**0.8**	0.1
61	80	1573	1145.2688	1211.7	5.80%	2.2	12.8	24	2936	1142.9	1177.2	3.00%	**1.7**	6.3
71	92	1825	1258.3929	1272.3	1.11%	3.1	0.8	30	3485	1251.9	1288.6	2.93%	**2.9**	4.3
81	115	2262	1440.6	1440.6	0.00%	4.6	0.1	32	4164	1440.6	1440.6	0.00%	**3.7**	0.1
91	118	2316	1569.275	1587.8	1.18%	5.5	1.3	33	4463	1568.4	1587.5	1.22%	**4.8**	2.0
101	132	2729	1676.1122	1704.6	1.70%	7.4	3.1	38	4738	1670.2	1706.1	2.15%	**6.8**	6.8
111	146	3218	2911.4224	2956.2	1.54%	9.7	4.2	39	5660	2911.0	2918.7	0.26%	**8.3**	1.1
121	142	3122	3222.6403	3269.5	1.45%	10.7	4.5	45	5859	3222.9	3265.8	1.33%	11.0	5.7
131	163	3563	3502.6785	3654.4	4.33%	14.2	46.9	48	6426	3511.6	3596.5	2.42%	14.9	31.3
141	177	3882	3689.6869	3794.5	2.84%	18.3	68.1	55	6898	3691.8	3770.3	2.13%	18.7	73.7
151	179	4166	3918.7123	4071.2	3.89%	20.6	121.7	57	7233	3923.0	4041.6	3.02%	21.6	115.5

Table 13: Random coloring with column enumeration for Type C instance ($Q = 3$)

nNode	Random Coloring w/o Column Enumeration							Random Coloring w/ Column Enumeration				
	nIter	nCol	LB	t_{LB} (s)	UB_1	Gap_1	t_{UB_1}	$nCol_e$	t_e (s)	UB_2	Gap_2	t_{UB_2} (s)
51	21	421	903.2	1.4	933	3.30%	0.5	3044	0.3	920.8	1.95%	2.0
61	29	535	1180.3	0.8	1184.7	0.37%	0.1	1370	0.4	1184.7	0.37%	1.8
71	26	572	1383.5	0.8	1411.9	2.05%	1.1	4222	0.0	1407.6	1.74%	15.5
81	37	666	1660.1	1.1	1695.5	2.13%	1.3	4917	0.1	1683.9	1.44%	10.4
91	34	720	1859.5	0.8	1882.3	1.23%	0.6	4798	0.0	1878.3	1.01%	5.6
101	42	825	2139.8	1.3	2168	1.32%	1.2	3784	0.0	2152.4	0.59%	2.7
111	45	1013	4226.6	1.0	4286.1	1.41%	3.0	8014	0.0	4274	1.12%	14.9
121	52	1092	4604.0	1.3	4682.8	1.71%	6.7	9271	0.0	4652.1	1.04%	25.4
131	56	1258	4998.2	1.7	5070.5	1.45%	3.3	9436	0.0	5054.7	1.13%	139.3
141	56	1218	5391.5	1.9	5553.2	3.00%	20.7	11866	0.1	5485.1	1.74%	315.5
151	60	1259	5789.8	2.2	5867.1	1.33%	4.5	12208	0.0	5839.1	0.85%	231.9
201	83	1736	14632.5	5.2	14813.8	1.24%	19.7	16112	0.1	14745.2	0.77%	270.1
251	101	2226	18004.7	8.6	18276.9	1.51%	44.3	20599	0.2	18171.4	0.93%	1113.5
301	122	2792	21766.2	15.4	22053.1	1.32%	48.5	23319	0.2	21932.5	0.76%	1105.4
351	147	3174	25039.4	23.3	25319.5	1.12%	63.2	29735	0.3	25184.5	0.58%	1450.7

–: solution time reaches 30-minute time limit.

Table 14: Random coloring with column enumeration for Type R instance ($Q = 3$)

nNode	Random Coloring w/o Column Enumeration							Random Coloring w/ Column Enumeration				
	nIter	nCol	LB	t_{LB} (s)	UB_1	Gap_1	t_{UB_1}	$nCol_e$	t_e (s)	UB_2	Gap_2	t_{UB_2} (s)
51	18	402	1125.5	0.2	1131.9	0.57%	0.2	703	0.0	1131.1	0.50%	0.5
61	24	501	1269.1	0.3	1270.6	0.12%	0.1	225	0.0	1270.6	0.12%	0.1
71	25	558	1528.7	0.4	1539.8	0.73%	0.8	2125	0.0	1536.6	0.52%	1.6
81	29	652	1703.1	0.5	1714.3	0.66%	0.8	2614	0.0	1707.9	0.28%	0.7
91	34	749	1872.9	0.6	1879	0.33%	0.7	1788	0.0	1876.8	0.21%	0.7
101	34	775	2005.5	0.7	2015.4	0.49%	1.3	3348	0.0	2013	0.37%	1.7
111	39	873	4398.8	1.0	4422.2	0.53%	1.1	4078	0.0	4416.5	0.40%	2.9
121	45	995	4748.8	1.2	4774.3	0.54%	1.0	4962	0.0	4763.3	0.31%	2.2
131	47	1077	5176.1	1.4	5216.4	0.78%	2.9	6917	0.0	5196.8	0.40%	5.3
141	50	1124	5525.3	1.8	5567.5	0.76%	2.3	8190	0.0	5559.9	0.63%	18.0
151	54	1213	5874.2	2.1	5922	0.81%	5.1	9039	0.0	5902.3	0.48%	19.4
201	73	1680	15975.1	4.8	16066.4	0.57%	2.9	12408	0.1	16021.7	0.29%	8.3
251	95	2097	19590.5	8.2	19679.3	0.45%	8.7	16289	0.1	19633	0.22%	51.2
301	106	2501	23321.4	13.2	23469.6	0.64%	20.3	23562	0.2	23372.6	0.22%	64.6
351	131	2987	27253.4	20.9	27422.9	0.62%	41.9	27834	0.3	27338.8	0.31%	96.5

Table 15: Random coloring with column enumeration for Type RC instance ($Q = 3$)

nNode	Random Coloring w/o Column Enumeration							Random Coloring w/ Column Enumeration				
	nIter	nCol	LB	t_{LB} (s)	UB_1	Gap_1	t_{UB_1}	$nCol_e$	t_e (s)	UB_2	Gap_2	t_{UB_2} (s)
51	20	412	1517.4	0.2	1564.4	3.10%	0.7	3525	0.0	1558.2	2.69%	5.7
61	24	517	1736.0	0.3	1770	1.96%	1.0	3095	0.0	1762.6	1.54%	6.1
71	34	601	1910.2	0.5	1915.8	0.29%	0.4	1362	0.0	1915.8	0.29%	2.6
81	34	708	2187.7	0.6	2199.3	0.53%	0.9	2380	0.0	2199.3	0.53%	7.4
91	32	683	2380.8	0.6	2397.5	0.70%	0.5	3108	0.0	2390.8	0.42%	4.0
101	38	796	2532.1	0.8	2568.1	1.42%	0.5	5352	0.0	2546.1	0.55%	6.6
111	45	1015	4405.8	1.0	4428.2	0.51%	1.1	4454	0.0	4417.2	0.26%	7.6
121	45	1039	4864.9	1.1	4906.2	0.85%	3.2	6731	0.0	4885.4	0.42%	12.2
131	48	1096	5313.2	1.4	5335.8	0.43%	1.4	5275	0.0	5332.6	0.37%	15.5
141	62	1278	5613.7	2.2	5648.2	0.61%	4.0	7890	0.0	5628.6	0.26%	17.9
151	61	1348	5998.7	2.3	6049	0.84%	5.8	9949	0.1	6020.1	0.36%	11.1
201	80	1704	15813.5	5.1	15911.2	0.62%	6.2	13315	0.1	15878.7	0.41%	41.0
251	96	2140	19201.6	8.0	19335.9	0.70%	11.1	18999	0.1	19272.7	0.37%	126.8
301	114	2560	22911.6	14.4	23109.1	0.86%	45.6	21348	0.2	22967.7	0.24%	93.0
351	134	2989	26757.0	21.1	26865.2	0.40%	22.9	25402	0.3	26822.4	0.24%	145.2

Table 16: Random coloring with column enumeration for Type C instance ($Q = 4$)

nNode	Random Coloring w/o Column Enumeration							Random Coloring w/ Column Enumeration				
	nIter	nCol	LB	t_{LB} (s)	UB_1	Gap_1	t_{UB_1}	$nCol_e$	t_e (s)	UB_2	Gap_2	t_{UB_2} (s)
51	22	1370	721.3	1.6	726.9	0.78%	5.0	3451	0.0	723.9	0.36%	2.5
61	30	1604	929.3	0.8	931.7	0.26%	3.4	1522	0.0	931.2	0.21%	4.2
71	30	1737	1113.9	0.8	1126.7	1.15%	6.6	9713	0.0	1123.6	0.87%	12.0
81	31	1808	1283.4	1.1	1292.6	0.72%	3.8	5284	0.0	1289.9	0.51%	7.9
91	33	2082	1456.8	4.1	1521.8	4.46%	9.4	22713	0.1	1505.3	3.33%	43.2
101	34	2105	1691.7	1.8	1727.1	2.09%	8.2	22726	0.1	1706.5	0.87%	23.7
111	42	2598	3272.0	2.7	3383.4	3.41%	6.8	30749	0.1	3347.9	2.32%	77.5
121	50	2988	3617.8	3.8	3697.2	2.20%	19.1	38411	0.1	3666.7	1.35%	533.0
131	53	3156	3884.0	4.8	3947.9	1.65%	13.7	37484	0.2	3930.5	1.20%	48.0
141	57	3290	4170.7	5.6	4307.5	3.28%	21.2	48653	0.2	4250.4	1.91%	-
151	62	3849	4498.8	7.0	4636.4	3.06%	90.7	51330	0.2	4569.2	1.56%	1320.8
201	79	4718	11372.9	15.1	11636.1	2.31%	56.3	67710	0.4	11475	0.90%	232.5
251	101	5871	13889.1	31.1	14264.8	2.71%	62.6	86116	0.6	14054.2	1.19%	-
301	128	7194	16641.7	54.1	16915.4	1.64%	17.3	102548	0.8	16835.6	1.17%	-
351	139	7996	19103.8	77.2	19457.5	1.85%	680.1	127956	1.0	19329.8	1.18%	-

–: solution time reaches 30-minute time limit.

Table 17: Random coloring with column enumeration for Type R instance ($Q = 4$)

nNode	Random Coloring w/o Column Enumeration							Random Coloring w/ Column Enumeration				
	nIter	nCol	LB	t_{LB} (s)	UB_1	Gap_1	t_{UB_1}	$nCol_e$	t_e (s)	UB_2	Gap_2	t_{UB_2} (s)
51	17	1199	916.8	0.3	930.8	1.53%	0.654	3420	0.0	928.1	1.23%	1.2
61	21	1494	1029.8	0.5	1044.0	1.38%	0.263	4513	0.0	1037.5	0.75%	1.2
71	24	1640	1235.6	0.8	1246.8	0.90%	0.781	3596	0.0	1244.3	0.70%	1.5
81	30	1852	1375.5	1.2	1386.9	0.83%	1.257	6080	0.0	1380.5	0.36%	1.1
91	32	2115	1511.7	1.7	1530.3	1.23%	2.025	13007	0.1	1522.5	0.71%	7.5
101	36	2276	1612.7	2.2	1626.5	0.85%	1.532	11283	0.1	1619.3	0.41%	1.3
111	41	2488	3514.9	2.9	3547.8	0.94%	3.906	14425	0.1	3541.7	0.76%	51.6
121	42	2686	3783.9	3.5	3843.4	1.57%	6.479	28268	0.2	3808.6	0.65%	9.6
131	46	2890	4115.5	4.3	4184.0	1.66%	6.031	34794	0.2	4146	0.74%	124.7
141	50	3140	4376.5	5.6	4422.8	1.06%	5.281	28854	0.2	4400.3	0.54%	12.9
151	52	3436	4625.3	6.5	4702.6	1.67%	14.577	41635	0.3	4662.1	0.80%	160.0
201	66	4208	12499.4	13.1	12670.7	1.37%	10	60369	0.4	12567.7	0.55%	17.4
251	85	5453	15280.6	27.0	15444.8	1.07%	12.281	77788	0.6	15326.7	0.30%	78.0
301	100	6511	18125.2	42.8	18303.5	0.98%	94.766	92284	0.8	18191.5	0.37%	168.4
351	109	6987	21123.8	61.5	21333.3	0.99%	172.126	114496	1.0	21208.7	0.40%	237.0

Table 18: Random coloring with column enumeration for Type RC instance ($Q = 4$)

nNode	Random Coloring w/o Column Enumeration							Random Coloring w/ Column Enumeration				
	nIter	nCol	LB	t_{LB} (s)	UB_1	Gap_1	t_{UB_1}	$nCol_e$	t_e (s)	UB_2	Gap_2	t_{UB_2} (s)
51	23	1236	1184.6	0.5	1229.0	3.75%	5.5	10472	0.0	1224.1	3.33%	11.4
61	27	1565	1372.0	0.7	1403.6	2.30%	6.8	10603	0.0	1389.8	1.30%	19.6
71	28	1665	1507.4	0.9	1529.9	1.49%	3.7	7940	0.1	1526.8	1.29%	23.3
81	31	1853	1728.4	1.3	1783.7	3.20%	10.5	21362	0.1	1749.1	1.20%	14.4
91	34	2068	1893.3	1.7	1924.6	1.65%	3.7	17072	0.1	1913.5	1.07%	57.6
101	34	2222	2000.7	1.9	2090.7	4.50%	17.6	30147	0.1	2031.0	1.51%	34.1
111	41	2621	3478.8	2.7	3507.6	0.83%	2.5	14469	0.1	3503.7	0.72%	31.3
121	45	2793	3844.9	3.4	3882.5	0.98%	3.5	19413	0.1	3871.2	0.68%	50.0
131	50	2961	4184.4	4.4	4243.2	1.40%	8.8	32915	0.2	4215.4	0.74%	33.3
141	53	3208	4403.9	5.5	4435.2	0.71%	2.8	21255	0.1	4428.3	0.55%	46.6
151	58	3344	4698.2	6.7	4744.6	0.99%	5.1	33437	0.2	4729.0	0.66%	119.8
201	70	4222	12413.8	14.4	12531.2	0.95%	10.7	52831	0.4	12472.3	0.47%	165.2
251	96	5698	14945.3	28.1	15093.5	0.99%	137.0	75244	0.6	15023.7	0.52%	1076.9
301	106	6441	17732.7	44.7	17867.9	0.76%	22.2	93267	0.8	17783.0	0.28%	522.5
351	125	7748	20689.6	68.6	20920.3	1.12%	361.5	120756	1.1	20782.5	0.45%	1801.6

Table 19: Random coloring with column enumeration for Type C instance ($Q = 5$)

nNode	Random Coloring w/o Column Enumeration							Random Coloring w/ Column Enumeration				
	nIter	nCol	LB	t_{LB} (s)	UB_1	Gap_1	t_{UB_1}	$nCol_e$	t_e (s)	UB_2	Gap_2	t_{UB_2} (s)
51	28	2975	594.9	2.8	594.9	0.00%	0.8	38	0.0	594.9	0.00%	0.8
61	28	3029	768.8	1.6	770.1	0.17%	0.9	763	0.1	770.1	0.17%	0.9
71	29	3893	937.6	2.7	947.8	1.09%	1.8	11849	0.1	939.1	0.16%	2.7
81	34	3915	1084.9	3.2	1117	2.96%	6.1	54388	0.2	1101.8	1.56%	103.2
91	37	4802	1226.6	11.3	1252.4	2.10%	9.1	54548	0.3	1240.5	1.13%	127.0
101	37	4987	1388.3	6.0	1412.7	1.75%	8.6	50095	0.3	1396.7	0.60%	21.8
111	50	6057	2702.6	14.1	2754	1.90%	13.0	89092	0.5	2741.7	1.45%	1226.2
121	51	7030	2997.8	13.9	3100.8	3.43%	104.0	125698	0.7	3047.7	1.66%	-
131	47	6571	3173.4	14.0	3301.5	4.04%	45.9	153927	0.7	3258	2.66%	-
141	54	7032	3432.0	19.8	3554.2	3.56%	91.7	149970	0.8	3485.5	1.56%	-
151	59	7810	3728.5	23.3	3770.9	1.14%	69.9	109314	0.8	3772	1.17%	-

–: solution time reaches 30-minute time limit.

Table 20: Random coloring with column enumeration for Type R instance ($Q = 5$)

nNode	Random Coloring w/o Column Enumeration							Random Coloring w/ Column Enumeration				
	nIter	nCol	LB	t_{LB} (s)	UB_1	Gap_1	t_{UB_1}	$nCol_e$	t_e (s)	UB_2	Gap_2	t_{UB_2} (s)
51	20	2392	793.9	1.0	816.1	2.80%	0.8	13323	0.0	804.4	1.32%	1.4
61	20	3175	892.7	1.4	915.1	2.51%	0.8	18944	0.0	897.8	0.57%	1.0
71	26	3505	1067.3	2.4	1084.8	1.64%	3.3	15417	0.0	1072.3	0.47%	1.1
81	28	3842	1178.5	3.2	1200.8	1.89%	5.6	30492	0.0	1185.5	0.60%	3.5
91	30	4341	1290.8	4.4	1308.3	1.36%	7.4	25936	0.0	1299.6	0.68%	6.0
101	33	4647	1373.7	5.9	1414.2	2.95%	13.9	87191	0.0	1384.6	0.79%	14.6
111	41	5398	2973.4	10.1	3042.9	2.34%	11.5	84934	0.0	2998.8	0.85%	36.7
121	42	5991	3191.8	12.0	3226.8	1.10%	4.6	30820	0.0	3206.5	0.46%	6.3
131	45	6198	3451.9	14.0	3525.9	2.14%	23.0	104694	0.0	3489.6	1.09%	127.5
141	49	6671	3664.2	17.7	3748.5	2.30%	21.0	126291	0.0	3691.5	0.74%	38.9
151	53	7235	3882.6	21.9	3934.2	1.33%	8.1	84715	0.1	3905.7	0.60%	51.8

Table 21: Random coloring with column enumeration for Type RC instance ($Q = 5$)

nNode	Random Coloring w/o Column Enumeration							Random Coloring w/ Column Enumeration				
	nIter	nCol	LB	t_{LB} (s)	UB_1	Gap_1	t_{UB_1}	$nCol_e$	t_e (s)	UB_2	Gap_2	t_{UB_2} (s)
51	18	2194	922.5	0.8	922.5	0.00%	0.1	28	0.1	922.5	0.00%	0.0
61	24	2936	1142.9	1.7	1177.2	3.00%	6.3	26159	0.1	1157.8	1.31%	85.7
71	30	3485	1251.9	2.9	1288.6	2.93%	4.3	36754	0.2	1268.6	1.34%	16.5
81	32	4164	1440.6	3.7	1440.6	0.00%	0.1	62	0.2	1440.6	0.00%	0.1
91	33	4463	1568.4	4.8	1587.5	1.22%	2.0	16570	0.2	1572.3	0.25%	1.5
101	38	4738	1670.2	6.8	1706.1	2.15%	6.8	56078	0.3	1680.4	0.61%	8.4
111	39	5660	2911.0	8.3	2918.7	0.26%	1.1	5097	0.3	2916.3	0.18%	1.1
121	45	5859	3222.9	11.0	3265.8	1.33%	5.7	57420	0.4	3242.1	0.60%	42.5
131	48	6426	3511.6	14.9	3596.5	2.42%	31.3	116524	0.6	3537	0.72%	69.1
141	55	6898	3691.8	18.7	3770.3	2.13%	73.7	117008	0.6	3729.5	1.02%	937.6
151	57	7233	3923.0	21.6	4041.6	3.02%	115.5	156365	0.8	3976.9	1.38%	1122.7

A.2 Unitary Demand CVRP X-instances

Table 22: Numerical results for unitary X instances with $Q = 3$

Instance	nNode	Pulse					Random Coloring				
		LB	UB_1	Gap_1	t_{LB} (m)	t_{UB_1} (m)	LB	UB_1	Gap_1	t_{LB} (m)	t_{UB_1} (m)
X-n120-k6	120	61050.8	61296.0	0.40%	0.1	0.0	61082.0	61324.0	0.40%	0.1	0.0
X-n157-k13	157	56791.2	56920.0	0.23%	0.2	0.1	56784.5	56935.0	0.27%	**0.1**	0.0
X-n181-k23	181	59806.8	60005.0	0.33%	0.3	0.1	59799.2	60216.0	0.70%	**0.1**	0.3
X-n219-k73	219	117306.1	117893.0	0.50%	0.5	0.4	117325.9	117761.0	0.37%	**0.1**	0.1
X-n237-k14	237	124261.1	124765.0	0.41%	0.6	0.4	124324.8	124653.0	0.26%	**0.1**	0.5
X-n275-k28	275	56713.8	56962.0	0.44%	0.9	0.4	56759.3	56984.0	0.40%	**0.2**	0.6
X-n317-k53	317	150795.9	151179.0	0.25%	1.4	-	150816.3	151199.0	0.25%	**0.3**	-
X-n331-k15	331	180868.0	181366.0	0.28%	1.7	0.3	180955.3	181495.0	0.30%	**0.3**	0.7
X-n376-k94	376	193400.3	193913.0	0.27%	2.1	0.3	193420.7	193986.0	0.29%	**0.5**	0.7
X-n439-k37	439	116146.1	116671.0	0.45%	3.4	0.2	116194.4	116633.0	0.38%	**0.7**	0.4
X-n502-k39	502	278048.1	278207.0	0.06%	5.5	0.4	278048.4	278251.0	0.07%	**1.3**	0.6
X-n548-k50	548	287916.8	288554.0	0.22%	6.4	2.0	287950.5	288610.0	0.23%	**1.6**	0.8
X-n655-k131	655	173036.6	173321.0	0.16%	12.3	1.4	173068.1	173343.0	0.16%	**2.7**	1.1
X-n801-k40	801	415868.9	416613.0	0.18%	19.5	2.7	415931.6	416600.0	0.16%	**4.7**	1.8
X-n856-k95	856	240621.2	241196.0	0.24%	25.9	12.7	240663.9	241300.0	0.26%	**5.4**	-
X-n957-k87	957	276006.7	276807.0	0.29%	29.8	6.9	276091.3	276890.0	0.29%	**7.6**	-

–: solution time reaches 30-minute time limit.

Table 23: Numerical results for unitary X instances with $Q = 4$

Instance	nNode	Pulse					Random Coloring				
		LB	UB_1	Gap_1	t_{LB} (m)	t_{UB_1} (m)	LB	UB_1	Gap_1	t_{LB} (m)	t_{UB_1} (m)
X-n120-k6	120	47221.2	47496.0	0.58%	0.1	0.1	47225.7	47593.0	0.78%	0.4	0.0
X-n157-k13	157	43512.1	44171.0	1.51%	0.3	-	43541.9	44315.0	1.78%	**0.2**	-
X-n181-k23	181	45955.8	46175.0	0.48%	0.4	0.3	45946.5	46277.0	0.72%	**0.2**	0.9
X-n219-k73	219	89899.3	90756.0	0.95%	0.7	4.2	89902.8	90544.0	0.71%	**0.3**	0.5
X-n237-k14	237	95128.0	95579.0	0.47%	0.9	0.9	95130.9	95689.0	0.59%	**0.5**	0.8
X-n275-k28	275	43946.9	44476.0	1.20%	1.2	13.3	43956.9	44445.0	1.11%	**0.6**	25.9
X-n317-k53	317	114514.2	114745.0	0.20%	2.1	0.2	114523.6	114907.0	0.33%	**1.0**	1.4
X-n331-k15	331	137819.7	138772.0	0.69%	2.2	-	137819.7	138667.0	0.61%	**1.2**	5.4
X-n376-k94	376	147428.4	148116.0	0.47%	3.3	14.8	147397.3	148120.0	0.49%	**1.6**	7.0
X-n439-k37	439	89546.1	90493.0	1.06%	4.4	13.3	89540.5	90388.0	0.95%	**2.2**	4.0
X-n502-k39	502	209892.9	211201.0	0.62%	6.9	-	209900.2	211228.0	0.63%	**4.1**	-
X-n548-k50	548	218820.0	219725.0	0.41%	9.7	-	218817.3	220240.0	0.65%	**4.7**	-
X-n655-k131	655	131569.8	132050.0	0.36%	15.7	-	131572.5	132084.0	0.39%	**8.1**	18.1
X-n801-k40	801	315127.1	316191.0	0.34%	35.9	-	315162.3	317741.0	0.82%	**14.8**	-
X-n856-k95	856	183768.8	184630.0	0.47%	32.1	-	183733.5	185486.0	0.95%	**15.8**	-
X-n957-k87	957	210524.0	211425.0	0.43%	46.7	-	210542.6	212890.0	1.11%	**22.6**	-

–: solution time reaches 30-minute time limit.

A.3 Multi-depot VRPUD

We also study the solution routes computed from the column generation using two different pricing algorithms. For each instance with the different number of patient nodes (nNode) and hospitals (nDepot), we report the number of solution routes (nRoute) and their average cost (AvgCost) based at each depot. Table 30 and 31 summarize the solution results. Comparing the instances with the same number of customer nodes (patients) but a different number of depots (hospitals), we notice the total number of routes used to cover

Table 24: Numerical results for unitary X instances with $Q = 5$

Instance	nNode	Pulse					Random Coloring				
		LB	UB_1	Gap_1	t_{LB} (m)	t_{UB_1} (m)	LB	UB_1	Gap_1	t_{LB} (m)	t_{UB_1} (m)
X-n120-k6	120	38726.3	39159.0	1.12%	0.2	0.1	38695.4	39025.0	0.85%	0.2	0.2
X-n157-k13	157	35581.2	36134.0	1.55%	0.5	-	35625.0	36216.0	1.66%	0.6	-
X-n181-k23	181	37686.6	38445.0	2.01%	0.7	23.3	37684.9	38190.0	1.34%	0.7	17.4
X-n219-k73	219	73324.6	74120.0	1.08%	1.4	2.9	73328.2	74131.0	1.09%	**1.1**	6.5
X-n237-k14	237	77640.4	78431.0	1.02%	1.7	3.6	77639.8	78598.0	1.23%	**1.4**	12.4
X-n275-k28	275	36358.3	36826.0	1.29%	2.0	6.9	36342.5	37131.0	2.17%	**1.7**	-
X-n317-k53	317	92730.0	93561.0	0.90%	4.6	-	92773.9	93719.0	1.02%	**3.5**	-
X-n331-k15	331	111945.6	113641.0	1.51%	4.2	-	111926.1	113611.0	1.51%	**3.8**	-
X-n376-k94	376	119634.3	120369.0	0.61%	6.9	14.5	119607.3	121447.0	1.54%	**5.4**	-
X-n439-k37	439	73375.0	74450.0	1.47%	6.7	-	73405.3	74907.0	2.05%	7.6	-
X-n502-k39	502	169081.8	170765.0	1.00%	14.0	-	169083.8	171768.0	1.59%	14.1	-
X-n548-k50	548	177331.1	181087.0	2.12%	21.3	-	177332.4	180340.0	1.70%	**16.2**	-
X-n655-k131	655	106568.6	108434.0	1.75%	27.1	-	106559.2	109359.0	2.63%	30.2	-
X-n801-k40	801	254747.6	258542.0	1.49%	76.1	-	254727.9	263320.0	3.37%	**57.7**	-
X-n856-k95	856	149461.0	152721.0	2.18%	54.9	-	149468.3	152965.0	2.34%	65.0	-
X-n957-k87	957	171173.4	174174.0	1.75%	115.4	-	171170.3	176515.0	3.12%	**93.9**	-

–: Solution time reaches 30-minute time limit.

Table 25: Random coloring with column enumeration for unitary X instances with $Q = 3$

nNode	Random Coloring w/o Column Enumeration							Random Coloring w/ Column Enumeration				
	nIter	nCol	LB	t_{LB} (m)	UB_1	Gap_1	t_{UB_1} (m)	nCol_e	t_e (s)	UB_2	Gap_2	t_{UB_2} (m)
120	50	1125	61082.0	0.1	61324.0	0.40%	0.0	5731	1.7	61236.0	0.25%	0.2
157	63	1435	56784.5	0.1	56935.0	0.27%	0.0	8523	0.5	56851.0	0.12%	0.0
181	83	1848	59799.2	0.1	60216.0	0.70%	0.3	13311	0.1	59926.0	0.21%	0.6
219	89	2021	117325.9	0.1	117761.0	0.37%	0.1	13376	0.1	117595.0	0.23%	0.7
237	101	2182	124324.8	0.1	124653.0	0.26%	0.5	12937	0.1	124505.0	0.14%	6.3
275	104	2360	56759.3	0.2	56984.0	0.40%	0.6	17014	0.1	56845.0	0.15%	3.6
317	136	3084	150816.3	0.3	151199.0	0.25%	-	20867	0.2	151058.0	0.16%	-
331	134	3051	180955.3	0.3	181495.0	0.30%	0.7	21870	0.2	181138.0	0.10%	1.0
376	154	3379	193420.7	0.5	193986.0	0.29%	0.7	25647	0.3	193678.0	0.13%	2.9
439	172	3923	116194.4	0.7	116633.0	0.38%	0.4	33172	0.4	116367.0	0.15%	1.4
502	238	5335	278048.4	1.3	278251.0	0.07%	0.6	28125	0.5	278148.0	0.04%	0.7
548	236	5161	287950.5	1.6	288610.0	0.23%	0.8	39342	0.5	288233.0	0.10%	-
655	291	6495	173068.1	2.7	173343.0	0.16%	1.1	44404	0.8	173172.0	0.06%	2.9
801	338	7591	415931.6	4.7	416600.0	0.16%	1.8	60586	1.2	416246.0	0.08%	8.0
856	352	7959	240663.9	5.4	241300.0	0.26%	-	66588	1.4	240911.0	0.10%	-
957	387	8833	276091.3	7.6	276890.0	0.29%	-	79422	1.8	276386.0	0.11%	-

–: solution time reaches 30-minute time limit.

the patients are similar as most of the routes contain four patients. However, the average cost of each route reduces 30%-40% (from approximately 40 to approximately 25) as the number of depots increases from 1 to 3. Further reduction, though diminishing, is observed as we increase the number of depots to 5, reducing the average cost per route from 25 to 20. An example solution for instances with 500 patients and five hospitals has been displayed in Figure 11.

Table 26: Random coloring with column enumeration for unitary X instances with $Q = 4$

nNode	Random Coloring w/o Column Enumeration							Random Coloring w/ Column Enumeration				
	nIter	nCol	LB	t_{LB} (m)	UB_1	Gap_1	t_{UB_1} (m)	$nCol_e$	t_e (s)	UB_2	Gap_2	t_{UB_2} (m)
120	47	2942	47225.7	0.4	47593.0	0.78%	0.0	20382	4.1	47366.0	0.30%	0.6
157	68	4047	43541.9	0.2	44315.0	1.78%	-	51868	0.3	44239.0	1.60%	-
181	74	4592	45946.5	0.2	46277.0	0.72%	0.9	40854	0.3	46021.0	0.16%	0.2
219	82	5215	89902.8	0.3	90544.0	0.71%	0.5	57460	0.5	90215.0	0.35%	11.8
237	93	5683	95130.9	0.5	95689.0	0.59%	0.8	58772	0.5	95339.0	0.22%	4.9
275	100	6291	43956.9	0.6	44445.0	1.11%	25.9	85111	0.7	44187.0	0.52%	-
317	128	8113	114523.6	1.0	114907.0	0.33%	1.4	72284	0.8	114593.0	0.06%	0.3
331	130	7721	137819.7	1.2	138667.0	0.61%	5.4	103195	1.1	138113.0	0.21%	-
376	145	8972	147397.3	1.6	148120.0	0.49%	7.0	113954	2.0	147721.0	0.22%	-
439	157	9632	89540.5	2.2	90388.0	0.95%	4.0	150900	2.6	89900.0	0.40%	-
502	226	12801	209900.2	4.1	211228.0	0.63%	-	164436	2.1	210991.0	0.52%	-
548	217	12908	218817.3	4.7	220240.0	0.65%	-	185357	3.2	219142.0	0.15%	20.2
655	257	15651	131572.5	8.1	132084.0	0.39%	18.1	213913	4.7	131808.0	0.18%	-
801	316	19145	315162.3	14.8	317741.0	0.82%	-	273588	5.5	316398.0	0.39%	-
856	303	17830	183733.5	15.8	185486.0	0.95%	-	281052	6.3	184412.0	0.37%	-
957	341	20789	210542.6	22.6	212890.0	1.11%	-	327607	7.8	211192.0	0.31%	-

–: solution time reaches 30-minute time limit.

Table 27: Random coloring with column enumeration for unitary X instances with $Q = 5$

nNode	Random Coloring w/o Column Enumeration							Random Coloring w/ Column Enumeration				
	nIter	nCol	LB	t_{LB} (m)	UB_1	Gap_1	t_{UB_1} (m)	$nCol_e$	t_e (s)	UB_2	Gap_2	t_{UB_2} (m)
120	45	6174	38695.4	0.2	39025.0	0.85%	0.2	40883	0.4	38875.0	0.46%	2.5
157	74	9835	35625.0	0.6	36216.0	1.66%	-	159998	0.8	36171.0	1.53%	-
181	74	9622	37684.9	0.7	38190.0	1.34%	17.4	161859	1.0	37800.0	0.31%	6.6
219	80	11158	73328.2	1.1	74131.0	1.09%	6.5	200115	1.5	73662.0	0.46%	-
237	89	12547	77639.8	1.4	78598.0	1.23%	12.4	246750	1.9	77932.0	0.38%	-
275	90	13026	36342.5	1.7	37131.0	2.17%	-	302297	4.2	36503.0	0.44%	-
317	128	17026	92773.9	3.5	93719.0	1.02%	-	356056	3.4	93882.0	1.19%	-
331	123	16451	111926.1	3.8	113611.0	1.51%	-	385527	3.8	112667.0	0.66%	-
376	138	18724	119607.3	5.4	121447.0	1.54%	-	436722	6.9	119933.0	0.27%	-
439	149	20077	73405.3	7.6	74907.0	2.05%	-	515734	6.7	74029.0	0.85%	-
502	219	26837	169083.8	14.1	171768.0	1.59%	-	566745	12.3	170408.0	0.78%	-
548	196	26977	177332.4	16.2	180340.0	1.70%	-	613757	14.8	178120.0	0.44%	-
655	244	32206	106559.2	30.2	109359.0	2.63%	-	744221	18.1	107543.0	0.92%	-
801	280	37742	254727.9	57.7	263320.0	3.37%	-	949217	29.6	259321.0	1.80%	-
856	280	36127	149468.3	65.0	152965.0	2.34%	-	990298	41.1	151460.0	1.33%	-
957	294	41178	171170.3	93.9	176515.0	3.12%	-	1121190	45.4	175186.0	2.35%	-

–: solution time reaches 30-minute time limit.

Table 28: Numerical result for the proposed algorithm on patient-centered medical home instances

nNode	nDepot	Pulse							Random Coloring						
		nIter	nCol	LB	UB_1	Gap_1	t_{LB} (s)	t_{UB_1} (s)	nIter	nCol	LB	UB_1	Gap_1	t_{LB} (s)	t_{UB_1} (s)
100	1	99	2409	971.62	991.92	2.09%	6.50	5.45	35	1875	971.62	992.33	2.13%	**6.05**	5.14
	3	57	3085	759.23	778.90	2.59%	8.17	3.83	18	2779	759.23	778.69	2.56%	**2.87**	5.73
	5	25	2036	486.91	503.24	3.35%	5.88	1.94	12	2245	486.91	496.77	2.02%	**3.02**	3.46
150	1	148	3826	1398.61	1423.14	1.75%	16.15	17.76	43	2589	1398.61	1417.99	1.39%	**9.15**	19.45
	3	78	4893	1091.60	1101.16	0.88%	22.66	4.95	26	3961	1091.60	1111.10	1.79%	**8.29**	13.75
	5	44	3370	669.27	681.97	1.90%	20.52	4.81	16	3125	669.27	679.10	1.47%	**7.93**	8.02
200	1	211	5671	1852.60	1875.21	1.22%	40.26	178.24	66	3704	1852.60	1875.51	1.24%	**12.80**	160.01
	3	112	7365	1428.08	1441.64	0.95%	57.68	32.00	34	4844	1428.08	1444.44	1.15%	**17.85**	23.32
	5	46	4279	852.30	867.48	1.78%	37.73	6.22	19	4001	852.30	867.42	1.77%	**15.20**	8.25
250	1	303	8408	2274.12	2293.87	0.87%	89.82	63.99	79	4277	2274.12	2293.49	0.85%	**22.71**	90.64
	3	147	9746	1748.24	1764.20	0.91%	118.44	49.34	40	6191	1748.24	1765.39	0.98%	**33.47**	21.61
	5	65	5494	1033.53	1055.22	2.10%	84.31	68.46	21	4851	1033.53	1054.21	2.00%	**26.66**	36.43
300	1	388	11226	2736.67	2751.82	0.55%	164.92	51.40	93	5372	2736.67	2761.15	0.89%	**38.29**	164.73
	3	187	12002	2094.00	2104.35	0.49%	214.41	32.31	50	6844	2094.00	2106.56	0.60%	**56.55**	33.13
	5	79	6902	1235.72	1255.16	1.57%	147.11	42.35	26	5794	1235.72	1248.57	1.04%	**46.79**	19.12
350	1	477	13727	3173.38	3188.32	0.47%	274.14	530.63	111	6242	3173.38	3199.79	0.83%	**59.53**	1012.17
	3	226	15152	2418.36	2429.37	0.46%	346.35	131.20	60	8428	2418.36	2432.85	0.60%	**91.38**	72.25
	5	93	8255	1404.23	1418.66	1.03%	233.33	38.64	31	6812	1404.23	1417.29	0.93%	**74.40**	51.59
400	1	596	17313	3580.96	3599.36	0.51%	431.60	411.34	121	6888	3580.96	3620.50	1.10%	**85.54**	1129.43
	3	260	17964	2719.65	2738.64	0.70%	505.62	213.47	63	9181	2719.65	2741.33	0.80%	**123.64**	133.02
	5	103	9253	1586.06	1603.77	1.12%	327.58	91.37	37	7505	1586.06	1603.30	1.09%	**111.06**	132.86
450	1	682	19942	3981.56	3997.29	0.40%	611.65	2502.78	137	7721	3981.56	4011.20	0.74%	**116.24**	1481.64
	3	298	20372	3011.01	3026.54	0.52%	713.83	727.64	77	9782	3011.01	3028.74	0.59%	**186.63**	529.59
	5	126	11209	1799.96	1817.65	0.98%	499.46	120.01	39	8204	1799.96	1817.77	0.99%	**148.03**	130.18
500	1	836	24471	4439.35	4455.34	0.36%	913.61	2437.67	153	8743	4439.35	4466.88	0.62%	**166.67**	2791.23
	3	349	23711	3370.00	3384.41	0.43%	1022.49	169.83	80	11263	3370.00	3392.68	0.67%	**245.36**	660.38
	5	129	12494	2006.88	2023.09	0.81%	622.25	304.67	44	9483	2006.88	2027.25	1.02%	**208.37**	370.29

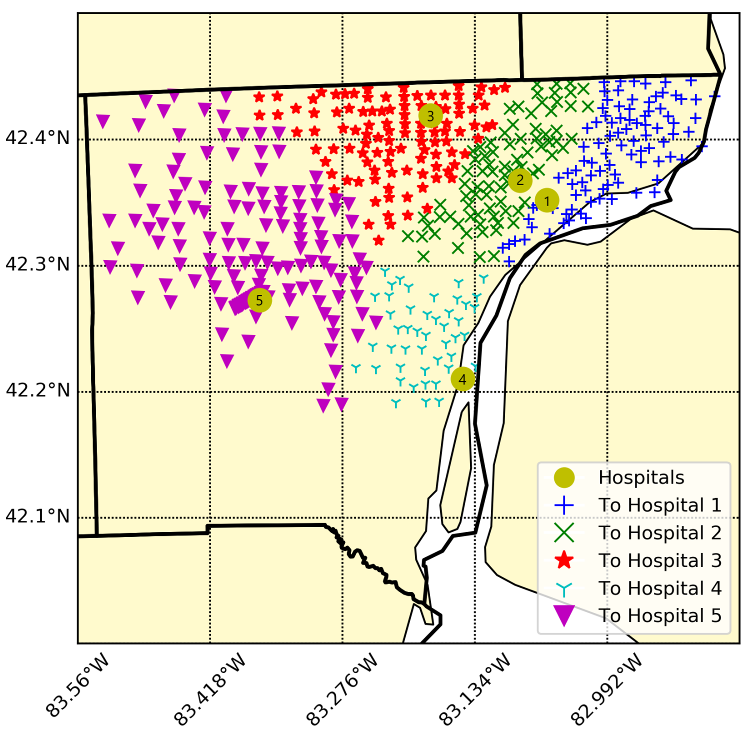

Figure 11: Example solution of assignments to the instance with 500 patients and 5 hospitals

Table 29: Random coloring with column enumeration for patient-centered medical home instances

nNode	nDepot	Random Coloring w/o Column Enumeration							Random Coloring w/ Column Enumeration				
		nIter	nCol	LB	t_{LB} (s)	UB_1	Gap_1	t_{UB_1} (s)	$nCol_e$	t_e (s)	UB_2	Gap_2	t_{UB_2} (s)
100	1	35	1875	971.62	6.0	992.33	2.13%	5.1	19872	2.5	981.11	0.98%	6.3
	3	18	2779	759.23	2.9	778.69	2.56%	5.7	48014	0.2	769.58	1.36%	22.1
	5	12	2245	486.91	3.0	496.77	2.02%	3.5	23203	0.3	496.20	1.91%	4.2
150	1	43	2589	1398.61	9.1	1417.99	1.39%	19.4	31227	0.3	1405.09	0.46%	20.0
	3	26	3961	1091.60	8.3	1111.10	1.79%	13.7	68022	0.3	1096.46	0.45%	5.3
	5	16	3125	669.27	7.9	679.10	1.47%	8.0	39691	0.6	674.44	0.77%	6.3
200	1	66	3704	1852.60	12.8	1875.51	1.24%	160.0	43480	0.3	1860.91	0.45%	311.8
	3	34	4844	1428.08	17.8	1444.44	1.15%	23.3	89259	0.6	1436.07	0.56%	543.0
	5	19	4001	852.30	15.2	867.42	1.77%	8.2	82328	0.8	861.72	1.11%	206.0
250	1	79	4277	2274.12	22.7	2293.49	0.85%	90.6	46853	0.3	2283.13	0.40%	1600.7
	3	40	6191	1748.24	33.5	1765.39	0.98%	21.6	118636	0.9	1757.44	0.53%	210.2
	5	21	4851	1033.53	26.7	1054.21	2.00%	36.4	140038	1.3	1043.01	0.92%	177.7
300	1	93	5372	2736.67	38.3	2761.15	0.89%	164.7	67823	0.5	2743.00	0.23%	160.7
	3	50	6844	2094.00	56.6	2106.56	0.60%	33.1	112536	1.2	2100.69	0.32%	150.1
	5	26	5794	1235.72	46.8	1248.57	1.04%	19.1	113158	2.0	1244.15	0.68%	131.6
350	1	111	6242	3173.38	59.5	3199.79	0.83%	1012.2	76853	0.6	3200.29	0.85%	-
	3	60	8428	2418.36	91.4	2432.85	0.60%	72.2	153475	1.7	2425.83	0.31%	1112.1
	5	31	6812	1404.23	74.4	1417.29	0.93%	51.6	140965	2.5	1411.46	0.52%	136.2
400	1	121	6888	3580.96	85.5	3620.50	1.10%	1129.4	95003	0.8	3589.30	0.23%	722.4
	3	63	9181	2719.65	123.6	2741.33	0.80%	133.0	242063	2.2	2728.82	0.34%	1673.9
	5	37	7505	1586.06	111.1	1603.30	1.09%	132.9	190925	3.2	1594.89	0.56%	1135.1
450	1	137	7721	3981.56	116.2	4011.20	0.74%	1481.6	107861	0.8	3997.29	0.40%	155.1
	3	77	9782	3011.01	186.6	3028.74	0.59%	529.6	238318	3.0	3017.63	0.22%	351.0
	5	39	8204	1799.96	148.0	1817.77	0.99%	130.2	246107	4.0	1807.56	0.42%	421.6
500	1	153	8743	4439.35	166.7	4466.88	0.62%	2791.2	114403	1.1	4447.10	0.17%	421.0
	3	80	11263	3370.00	245.4	3392.68	0.67%	660.4	284804	3.5	3377.22	0.21%	265.8
	5	44	9483	2006.88	208.4	2027.25	1.02%	370.3	309440	4.9	2027.25	1.02%	-

–: solution time reaches 30-minute time limit.

Table 30: Solution summary of multi-depot VRPUD with pulse pricing algorithm

nNode	nDepot	Depot 1		Depot 2		Depot 3		Depot 4		Depot 5	
		nRoute	avgCost	nRoute	avgCost	nRoute	avgCost	nRoute	avgCost	nRoute	avgCost
100	1	25	39.88	–	–	–	–	–	–	–	–
	3	6	24.69	7	26.80	13	35.17	–	–	–	–
	5	5	23.39	5	14.63	6	18.26	2	15.51	9	20.06
150	1	38	38.01	–	–	–	–	–	–	–	–
	3	8	23.70	10	22.28	20	35.20	–	–	–	–
	5	8	21.07	7	13.79	8	16.76	3	17.93	14	16.92
200	1	51	37.08	–	–	–	–	–	–	–	–
	3	12	26.20	14	21.56	25	34.15	–	–	–	–
	5	9	20.41	10	14.94	11	16.25	5	15.60	16	18.22
250	1	63	36.67	–	–	–	–	–	–	–	–
	3	15	22.29	17	21.22	32	33.94	–	–	–	–
	5	14	20.09	10	12.76	13	14.92	7	17.28	20	17.31
300	1	76	36.58	–	–	–	–	–	–	–	–
	3	19	24.30	18	19.60	39	33.35	–	–	–	–
	5	14	20.20	13	12.72	16	15.85	10	14.03	25	16.96
350	1	88	36.56	–	–	–	–	–	–	–	–
	3	23	24.13	19	19.66	46	33.10	–	–	–	–
	5	16	18.86	15	13.72	20	15.42	9	14.64	29	16.84
400	1	100	36.21	–	–	–	–	–	–	–	–
	3	23	22.10	25	21.56	53	32.29	–	–	–	–
	5	19	18.39	17	12.11	24	15.18	11	14.52	33	16.20
450	1	113	35.56	–	–	–	–	–	–	–	–
	3	27	22.31	26	20.54	60	31.67	–	–	–	–
	5	22	18.88	19	13.07	25	14.89	12	15.80	36	16.86
500	1	126	35.55	–	–	–	–	–	–	–	–
	3	32	22.05	27	18.99	67	32.74	–	–	–	–
	5	25	18.39	21	13.22	28	15.47	13	15.16	40	16.71

Table 31: Solution summary of multi-depot VRPUD with random coloring pricing algorithm

nNode	nDepot	Depot 1		Depot 2		Depot 3		Depot 4		Depot 5	
		nRoute	avgCost	nRoute	avgCost	nRoute	avgCost	nRoute	avgCost	nRoute	avgCost
100	1	25	39.3	–	–	–	–	–	–	–	–
	3	6	26.5	6	25.44	13	35.66	–	–	–	–
	5	5	23.18	4	15.93	6	18.07	2	14.45	9	20.06
150	1	38	37.07	–	–	–	–	–	–	–	–
	3	9	22.89	9	22.18	20	34.88	–	–	–	–
	5	7	20.7	8	16.82	8	17.59	3	15.22	12	17.64
200	1	50	37.31	–	–	–	–	–	–	–	–
	3	11	25.16	13	20.44	26	34.5	–	–	–	–
	5	10	20.62	9	13.54	11	16.79	5	15.51	16	17.11
250	1	63	36.3	–	–	–	–	–	–	–	–
	3	16	22.91	15	21.18	32	33.6	–	–	–	–
	5	13	19.62	11	13.34	13	15.46	6	15.32	20	17.81
300	1	75	36.62	–	–	–	–	–	–	–	–
	3	19	23.59	18	19.73	39	33.33	–	–	–	–
	5	14	19.5	13	12.81	17	16.81	9	15.26	23	16.64
350	1	88	36.16	–	–	–	–	–	–	–	–
	3	20	23.55	22	20.32	46	32.86	–	–	–	–
	5	16	18.64	15	13.45	19	15.21	9	14.64	29	16.95
400	1	100	35.94	–	–	–	–	–	–	–	–
	3	24	23.44	23	20.04	53	32.2	–	–	–	–
	5	18	19.18	17	13.02	22	14.65	10	15.22	33	16.81
450	1	113	35.33	–	–	–	–	–	–	–	–
	3	26	21.99	28	20.58	59	31.72	–	–	–	–
	5	21	18.72	20	12.93	25	15.16	11	15.48	36	16.95
500	1	125	35.6	–	–	–	–	–	–	–	–
	3	31	22.37	29	20.1	65	32.38	–	–	–	–
	5	24	18.95	21	12.93	27	14.47	12	15.24	42	17.07