

Branch-and-Price for the Capacitated Autonomous Vehicle Assisted Delivery Problem

Rui Zhang

Leeds School of Business, University of Colorado Boulder, Colorado 80309, USA, rui.zhang@colorado.edu

In recent years, the exponential growth of package volumes has posed significant challenges for logistics networks, particularly in the realm of last-mile delivery. To mitigate costs while upholding service and delivery commitments, companies are increasingly investigating autonomous assisted delivery as a viable solution. In this paper, we study the Capacitated Autonomous Vehicle Assisted Delivery Problem, where an autonomous vehicle works in conjunction with a delivery person. The autonomous vehicle drops off the delivery person at designated locations, and the delivery person completes the deliveries (with a capacity constraint) on foot to the final addresses. Once the deliveries are completed, the vehicle picks up the delivery person and travels to the next reloading point. The goal is to decide on a route to serve all customers while minimizing the route completion time. We introduce an integer programming formulation with exponentially many variables and develop a branch-and-price approach. For generating promising columns, we present a tailored pulse algorithm to solve the pricing problem. Furthermore, by leveraging the structural properties of optimal solutions, we carefully design algorithmic enhancements, valid inequalities, and preprocessing steps to improve computational tractability. By conducting computational experiments based on instances derived from real-world data, we demonstrate the positive impact of these components. More importantly, we provide optimal certificates for 426 out of the 450 instances documented in the literature. Among the 100 instances in which driving could be slower than walking, we report solutions for the 40 largest instances for the first time.

Key words: Autonomous Vehicle, Routing, Last-Mile Delivery, Branch-and-Price

1. Introduction

In recent years, the rise of e-commerce and online shopping has led to a significant increase in the volume of packages being delivered. Based on data from the U.S. Census Bureau (U.S. Census Bureau 2022), e-commerce sales surged by almost 32% in 2020 compared to the previous year. This statistic has only been increasing ever since. This shift was accelerated by the COVID-19 health crisis (United Nations 2021). For example, it is reported that over 1.5 million packages were delivered daily in New York City in 2019 (Haag and Hu 2019). This number rose to 2.3 million in 2021 (Deffenbaugh and Eisenpress 2021) and further increased to 3.7 million in 2022 (Simon 2022). As the city’s population continues to grow and consumers increasingly demand near-instant deliveries, there is a further pressing need for a better ground distribution system.

Last-mile delivery, the final leg of the delivery process from the distribution center to the customer’s doorstep, has become a crucial part of the logistics network, accounting for a significant portion (30-35%) of costs (Rosenberger 2022). To address the growing demand for efficient and

cost-effective last-mile delivery, companies are turning to new technologies and approaches, including autonomous vehicles. For instance, compartmentalized sidewalk autonomous robots are being tested for delivering small items such as groceries (Mims 2019). These robots offer the potential to improve efficiency and reduce delivery costs, as they can navigate sidewalks and avoid traffic congestion. Despite their potential, these robots have limitations, including the need to have miniders of one sort or another, restricted capacity, and the inability to perform tasks such as ringing a doorbell or climbing stairs. As a result, human delivery personnel will still be required for the foreseeable future to ensure a seamless delivery experience.

An alternative solution is the use of autonomous vehicles in conjunction with delivery personnel, which is being considered by the U.S. Postal Service Office (U.S. Postal Service 2017, Romo and Domonoske 2019). With this approach, the autonomous vehicle drops off the delivery person at designated locations within the city, and the delivery person completes the deliveries on foot to the final addresses. Once the deliveries are completed, the vehicle picks up the delivery person and travels to the next reloading point. There are two key advantages to using autonomous vehicles in this manner. Firstly, while the vehicle travels between customers with the delivery person on board, the delivery person can engage in other tasks, such as sorting mail, which can improve productivity and efficiency. Secondly, significant cost savings can be achieved by eliminating the time-consuming process of finding parking (INRIX Research 2017, Chiara et al. 2020) and reducing the time spent by the delivery person walking back to parking locations after completing multiple deliveries.

Recently, Reed et al. (2022a,b) have proposed a model called the Capacitated Autonomous Vehicle Assisted Delivery Problem (CAVADP) to systematically investigate the benefit of using autonomous vehicles in the aforementioned manner. In Reed et al. (2022b), the authors assume the customer geography to be a complete grid of customers and assume that driving is always faster than walking. They provide a polynomial time algorithm to find an optimal solution with the two restrictions. Computational experiments demonstrate that the CAVADP reduces the route completion time significantly, with higher values achieved for longer parking times, smaller capacities, and lower fixed times for loading packages. In Reed et al. (2022a), the authors lift the two assumptions. They consider the customer geography across the urban-to-rural continuum and allow driving to be slower than walking. An integer program (IP) model with various improvements is proposed as the solving approach. They conduct a case study based on real-world data in Illinois and benchmark the CAVADP with multiple other business models to analyze its effects on productivity and operational costs. They show that autonomous vehicle assisted delivery is the most cost-effective business model across all customer geographies. Autonomous vehicle assisted delivery can save up

to 56% in route completion time and reduce operational costs for businesses by 50% on average, compared to the current practice.

Reed et al. (2022a,b) present two IP formulations for the CAVADP. We provide a detailed discussion on each of them in Section 2 after we formally define the CAVADP. We present a new IP formulation with an exponential number of variables, which serves as the foundation for deriving a branch-and-price procedure. To further improve computational tractability, we carefully design several enhancements, including valid inequalities and preprocessing steps, by exploring the structural properties of optimal solutions.

The CAVADP shares certain similarities with drone delivery problems, which have recently attracted attention in the literature. Comprehensive reviews by Otto et al. (2018) and Chung et al. (2020) provide thorough reviews that effectively summarize the most significant research on this topic. The closest ones are described in Poikonen and Golden (2020a,b). However, the CAVADP and drone delivery problems have several important differences. Firstly, drones can have launch points that are not customer locations. In addition, the capacity constraint is on the fixed energy capacity of the drone, which decreases based on the total weight of the packages, the travel direction, and/or the travel distance. Solution approaches based on column generation, such as branch-and-price, branch-cut-and-price, and price-and-branch, are established techniques for solving large and challenging integer programming problems (Barnhart et al. 1998, Desaulniers et al. 2006). This is especially the case for Vehicle Routing problems, where the current state-of-the-art exact methods are all found in this category (Costa et al. 2019, Florio et al. 2020, Pessoa et al. 2020, Rostami et al. 2021). However, it is not clear if the typical set covering formulation in the literature can be applied to the CAVADP because it is necessary to model the synchronization between the delivery person and the autonomous vehicle. Our formulation is able to capture this synchronization and lays the foundation for developing a branch-and-price approach.

In our branch-and-price approach, the pricing problem can be modeled as a variant of the elementary shortest path problem with resource constraints (ESPPRC). The ESPPRC is known to be NP-hard in the strong sense (Dror 1994). For this reason, one stream of research has focused on the relaxations of the ESPPRC, where the requirement of elementary paths is relaxed. The resulting shortest path problem with resource constraints (SPPRC) is often solved by so-called labeling algorithms, which are based on dynamic programming techniques. There is a large literature on labeling algorithms; we refer to Irnich and Desaulniers (2005), Pugliese and Guerriero (2013), and Costa et al. (2019) for surveys. Another line of research has concentrated on resolving the ESPPRC without any relaxation. The pulse algorithm introduced by Lozano et al. (2016) is one of such algorithms. In addition to its excellent computational performance, an essential feature of the pulse algorithm is its flexibility, as it has been successfully adapted to many applications

(Lozano and Smith 2017, Schrottenboer et al. 2019, Corredor-Montenegro et al. 2021, Yamín et al. 2022). Unlike common routing problems, the CAVADP could use elementary paths between all pairs of customers. Furthermore, the cost function in our pricing problem is not linear. We present a tailored pulse algorithm for the pricing problem.

Overall, we propose a new formulation and develop solution approaches built on it. Our computational experiments, based on a set of 450 instances derived from real-world data (Reed et al. 2022a), demonstrate the efficacy of the proposed approaches. More importantly, we provide optimal certificates for 426 out of the 450 instances documented in the literature. Among the 100 instances in which driving could be slower than walking, we report solutions for the 40 largest instances for the first time. Our results provide further evidence in favor of autonomous vehicle assisted delivery.

The rest of the paper is organized as follows. Section 2 gives a formal definition of the CAVADP and presents detailed discussions of the two IP formulations known in the literature. Section 3 presents the proposed branch-and-price approach for the CAVADP. It provides a new IP formulation with exponentially sized variables and a tailored pulse algorithm. Further, we describe several algorithmic enhancements. Section 5 describes our computational results on a set of instances based on real-world data. Section 6 provides concluding remarks. All proofs and some details of the computational results are included in the Electronic Companion (EC).

2. Problem Definition and Known Formulations

In this section, we present the formal definition of the CAVADP. Then, we discuss the two known formulations from Reed et al. (2022b) and Reed et al. (2022a) in detail.

The CAVADP aims to serve a set $\mathcal{C}$ of n customers using an autonomous vehicle assisted by a delivery person. The vehicle departs from and returns to the depot with the delivery person on board. We denote the depot by 0 and define $\mathcal{V} = \mathcal{C} \cup \{0\}$. We also have a set $\mathcal{A}$ of arcs. An arc (i, j) in $\mathcal{A}$ represents the possible movement between locations i and j such that i, j in $\mathcal{V}$ and $i \neq j$. Thus, we have a directed graph $G = (\mathcal{V}, \mathcal{A})$. We use $n(i)^+$ and $n(i)^-$ to denote the sets of the outgoing and incoming neighbors of node i for all i in $\mathcal{V}$, respectively. The vehicle shuttles the delivery person between sets of customers. For each set, the delivery person is dropped off at one customer location, loads packages, serves customers on foot, and is picked up at a customer location that may be different from the drop-off location. The delivery person can serve up to q customer locations at one time. The time for loading packages f is independent of the number of customers being served in a set and is fixed. A driving time matrix T has entry $t_{i,j}$, showing the driving time from nodes i to j for i, j in $\mathcal{V}$. Similarly, a walking time matrix W has entry $w_{i,j}$, showing the walking time from nodes i to j for i, j in $\mathcal{C}$. Both driving and walking times satisfy the triangular inequality. The objective of the CAVADP is to minimize the route completion time of the delivery tour.

Reed et al. (2022b) propose an IP formulation for the CAVADP under the assumption that driving from customers i to j is always faster than starting at customer i , walking through some service set σ_k of customers, and returning to customer j . We present it here with slight adaptations to our notation. Let Ω be a family of possible subsets of customers served by a delivery person and let $\mathcal{M} = \{1, \dots, |\Omega|\}$, where $|\cdot|$ denotes the cardinality of a set. For every set $\sigma_k \in \Omega$ and $k \in \mathcal{M}$, we have $|\sigma_k| \leq q$. When the delivery person is dropped off at customer i and picked up at customer j while serving the customers in set σ_k , we only need to consider the shortest walking path starting at customer i and ending at customer j through the customers of σ_k . Assume that the shortest walking path is given as the visiting order $(i, k_1, k_2, \dots, k_{|\sigma_k|}, j)$, the time to walk and serve the customers in set σ_k when loading at customer i and returning to customer j is $\hat{w}_{i,j,k} = w_{i,k_1} + w_{k_1,k_2} + \dots + w_{k_{|\sigma_k|},j}$. We refer to this whole operation as a path and denote it as path (i, j, k) . The total time of such a path is $c_{i,j,k} = \max\{t_{i,j}, \hat{w}_{i,j,k}\} + f$. Note that when driving from customers i to j is always faster than starting at customer i , walking through some service set σ_k of customers, and returning to customer j , we have $c_{i,j,k} = \hat{w}_{i,j,k} + f$. Let $\mathcal{R}_i$ be the set of paths that serve node i , i.e., $\mathcal{R}_i = \{(l, j, k) : i \in \sigma_k\}$.

Define the binary decision variables $x_{i,j}$, $z_{i,j}$, and $y_{i,j,k}$ as well as the integer-valued variable u_i as follows: when $x_{i,j}$ is 1, the vehicle drives from customers i to j with the delivery person on board for $i, j \in \mathcal{V}$ such that $i \neq j$. When $z_{i,j}$ is 1, the vehicle drops the delivery person off at customer i and meets the delivery person at customer j for $i, j \in \mathcal{C}$ such that $i \neq j$. When $y_{i,j,k}$ is 1, the delivery person loads at customer i , serves customer set σ_k , and meets the vehicle at customer j for $i, j \in \mathcal{C}$, and $k \in \mathcal{M}$. The u_i variable records the visiting order of node i by the autonomous vehicle in $\mathcal{C}$. The IP formulation is:

$$\text{(RCT1) Min } \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{V}: i \neq j} t_{i,j} x_{i,j} + \sum_{i \in \mathcal{C}} \sum_{j \in \mathcal{C}} \sum_{k \in \mathcal{M}} y_{i,j,k} (\hat{w}_{i,j,k} + f) \quad (1a)$$

$$\text{Subject to } \sum_{i \in \mathcal{C}} x_{0,i} = 1, \quad (1b)$$

$$\sum_{i \in \mathcal{C}} x_{i,0} = 1, \quad (1c)$$

$$\sum_{(l,j,k) \in \mathcal{R}_i} y_{l,j,k} = 1, \quad \forall i \in \mathcal{C}, \quad (1d)$$

$$\sum_{j \in \mathcal{V}: j \neq i} x_{j,i} + \sum_{j \in \mathcal{C}: j \neq i} z_{j,i} = \sum_{j \in \mathcal{V}: j \neq i} x_{i,j} + \sum_{j \in \mathcal{C}: j \neq i} z_{i,j}, \quad \forall i \in \mathcal{C}, \quad (1e)$$

$$y_{i,j,k} \leq \sum_{l \in \mathcal{V}: l \neq i} x_{l,i} + \sum_{l \in \mathcal{C}: l \neq i} z_{l,i}, \quad \forall i, j \in \mathcal{C}, k \in \mathcal{M}, \quad (1f)$$

$$z_{i,j} = \sum_{k \in \mathcal{M}} y_{i,j,k}, \quad \forall i, j \in \mathcal{C}, i \neq j, \quad (1g)$$

$$u_0 = 0, \quad (1h)$$

$$u_j \leq \sum_{i \in \mathcal{C}} \sum_{l \in \mathcal{C}: l \neq i} (x_{i,l} + z_{i,l}) + 1, \quad \forall j \in \mathcal{C}, \quad (1i)$$

$$u_i - u_j + 1 \leq n(1 - x_{i,j}), \quad \forall i \in \mathcal{V}, j \in \mathcal{C}, i \neq j, \quad (1j)$$

$$u_i - u_j + 1 \leq n(1 - z_{i,j}), \quad \forall i, j \in \mathcal{C}, i \neq j, \quad (1k)$$

$$x_{i,j} \in \{0, 1\}, \quad \forall i, j \in \mathcal{V}, i \neq j, \quad (1l)$$

$$z_{i,j} \in \{0, 1\}, \quad \forall i, j \in \mathcal{C}, i \neq j, \quad (1m)$$

$$y_{i,j,k} \in \{0, 1\}, \quad \forall i, j \in \mathcal{C}, k \in \mathcal{M}, \quad (1n)$$

$$u_i \in \mathbb{Z}_{\geq 0}, \quad \forall i \in \mathcal{V}. \quad (1o)$$

The objective function (1a) aims to minimize the delivery tour's route completion time for the delivery person. Constraints (1b) and (1c) require the delivery person to be with the vehicle when leaving from and returning to the depot, respectively. Constraints (1d) make sure all customers are served. For each customer visited by the vehicle, constraints (1e) check that the vehicle will also depart from that customer. Specifically, if the delivery person loads at customer i , serves set σ_k , and returns to customer j (i.e., $y_{i,j,k} = 1$), the vehicle must visit customer i to load there. Constraints (1f) ensure the vehicle travels from any $l \in \mathcal{V} \setminus \{i\}$ to customer i with or without the delivery person (i.e., $x_{l,i} = 1$ or $z_{l,i} = 1$ if $l \neq 0$). If the delivery person is not onboard the autonomous vehicle, constraints (1g) ensure that, between visits to customers i and j (i.e., $z_{i,j} = 1$), the delivery person loads at customer i and rejoins the vehicle at customer j . The subtour elimination constraints are adapted from the Miller-Tucker-Zemlin (MTZ) constraints (Miller et al. 1960) and are stated in Constraints (1h), (1i), (1j), and (1k). Lastly, the domains of the decision variables are defined in constraints (1l), (1m), (1n), and (1o), with nonnegative integers denoted by $\mathbb{Z}_{\geq 0}$. Before proceeding, we summarize all parameters and decision variables in Table 1. For each decision variable, we point out which formulations are using it.

RCT1 is not expected to scale to larger problems. The reasons include the following. First, the subtour elimination constraints are adapted from the MTZ constraints. It is well-known that the MTZ constraints do not provide satisfactory performance in practice. Second, it has exponentially many variables and constraints due to the y variables and constraints (1f). In other words, the resulting formulation could be extremely large. Reed et al. (2022b) actually do not apply this formulation in their experiments at all.

RCT1 presumes that driving from customer i to j is quicker than loading at customer i , walking through some set σ_k , and coming back to customer j . However, in reality, the delivery person might have to wait for the autonomous vehicle, a crucial factor when considering actual data. Figure 1 illustrates one example where driving is slower than walking between two customer locations in New York City. To address this, Reed et al. (2022a) propose another IP formulation, which is

Parameters	Descriptions	
n	Number of customers	
$\mathcal{C}$	Set of customers	
$\mathcal{V}$	Set of customers and depot 0	
$t_{i,j}$	The driving time from nodes i to j for i, j in $\mathcal{V}$ (minutes)	
$w_{i,j}$	The walking time from nodes i to j for i, j in $\mathcal{C}$ (minutes)	
Ω	A family of customer service sets	
$\mathcal{M}$	The index set of Ω	
σ_k	Service set k and $k \in \mathcal{M}$	
f	The fixed time for loading packages (minutes)	
$\hat{w}_{i,j,k}$	The time to walk the shortest walking path between the customers in set σ_k when loading at i and returning to j (minutes)	
q	Capacity of delivery person (number of packages)	
Decision Variables	Formulations	Descriptions
$x_{i,j}$	RCT1, RCT2, IP1, IP2	$x_{i,j} = 1$ if the vehicle drives from customers i to j with the delivery person on board for $i, j \in \mathcal{V}$ such that $i \neq j$. Note that RCT2 allows $i = j$.
$y_{i,j,k}$	RCT1, RCT2, IP1, IP2	$y_{i,j,k} = 1$ if the delivery person loads at customer i , serves customer set σ_k , and meets the vehicle at customer j for $i, j \in \mathcal{C}$, and $k \in \mathcal{M}$.
$z_{i,j}$	RCT1, IP1, IP2	$z_{i,j} = 1$ if the vehicle drops the delivery person off at customer i and meets the delivery person at customer j for $i, j \in \mathcal{C}$ such that $i \neq j$.
u_i	RCT1	It records the visiting order of node i by the autonomous vehicle in $\mathcal{C}$.
$v_{i,j}$	RCT2, IP1, IP2	Its value represents the flow of packages on board the vehicle from customers i to j .
ζ_i	SepEx	$\zeta_i = 1$ if node i is included in $\mathcal{S}$.
$\eta_{i,j}$	SepEx	$\eta_{i,j} = 1$ if the arc (i, j) is included in the term $\sum_{i,j \in \mathcal{S}: i \neq j} (x_{i,j} + z_{i,j})$.
θ_k	SepEx	$\theta_k = 1$ if node k is the node from $\mathcal{S}$ selected for exclusion from the summation of the second term . the left-hand-side of constraints (7c).

Table 1 Descriptions of Parameters and Decision Variables



Figure 1 Illustration of the Driving (Left) is Slower Than Walking (Right) between Two Customer Locations Generated by Google Map (Google, Inc 2024).

referred to as RCT2. In this formulation, we still have x and y variables as defined earlier, but not z variables. In addition, an integer-valued variable $v_{i,j}$ is defined for each i in $\mathcal{V}$ and j in $\mathcal{C}$ such

that $i \neq j$. Its value represents the flow of packages on board the vehicle from customer or depot i to customer j . It is worth noting that, in contrast to RCT1, a crucial difference is that RCT2 includes binary variables $x_{i,i}$ for all i in $\mathcal{V}$. When $x_{i,i}$ is 1, the delivery person meets the vehicle at location i after the delivery person serves a service set and gets ready for the next service set. The formulation is given below:

$$(RCT2) \quad \text{Min} \quad \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{V}: i \neq j} t_{i,j} x_{i,j} + \sum_{i \in \mathcal{C}} \sum_{j \in \mathcal{C}} \sum_{k \in \mathcal{M}} y_{i,j,k} (\max\{t_{i,k}, \hat{w}_{i,j,k}\} + f) \quad (2a)$$

$$\text{Subject to} \quad \sum_{i \in \mathcal{C}} x_{0,i} = 1, \quad (2b)$$

$$\sum_{i \in \mathcal{C}} x_{i,0} = 1, \quad (2c)$$

$$\sum_{(l,j,k) \in \mathcal{R}_i} y_{l,j,k} = 1, \quad \forall i \in \mathcal{C}, \quad (2d)$$

$$\sum_{j \in \mathcal{C}} \sum_{k \in \mathcal{M}} y_{i,j,k} = \sum_{l \in \mathcal{V}} x_{l,i}, \quad \forall i \in \mathcal{C}, \quad (2e)$$

$$\sum_{i \in \mathcal{C}} \sum_{k \in \mathcal{M}} y_{i,j,k} = \sum_{l \in \mathcal{V}} x_{j,l}, \quad \forall j \in \mathcal{C}, \quad (2f)$$

$$\sum_{j \in \mathcal{C}} v_{0,j} = n, \quad (2g)$$

$$v_{0,i} \leq n x_{0,i}, \quad \forall i \in \mathcal{C}, \quad (2h)$$

$$v_{i,j} \leq n(x_{i,j} + \sum_{k \in \mathcal{M}} y_{i,j,k}), \quad \forall i, j \in \mathcal{C}, i \neq j, \quad (2i)$$

$$\sum_{j \in \mathcal{V}: j \neq i} v_{j,i} - \sum_{j \in \mathcal{C}: j \neq i} v_{i,j} = \sum_{j \in \mathcal{C}} x_{i,j} + \sum_{j \in \mathcal{C}} \sum_{k \in \mathcal{M}} (|\sigma_k| - 1) y_{i,j,k}, \quad \forall i \in \mathcal{C}, \quad (2j)$$

$$x_{i,j} \in \{0, 1\}, \quad \forall i, j \in \mathcal{V}, \quad (2k)$$

$$y_{i,j,k} \in \{0, 1\}, \quad \forall i, j \in \mathcal{C}, k \in \mathcal{M}, \quad (2l)$$

$$v_{i,j} \in \mathbb{Z}_{\geq 0}, \quad \forall i \in \mathcal{V}, j \in \mathcal{C}, i \neq j. \quad (2m)$$

The goal of objective function (2a) is to minimize the route completion time of the delivery tour for the delivery person. Note that its second term accounts for the additional waiting time if the delivery person needs to wait for the vehicle by using the “ $\max\{\cdot\}$ ” operator. Constraints (2b) and (2c) ensure the delivery person is with the vehicle when departing and returning to the depot, respectively. Constraints (2d) guarantee all customers are served. Constraints (2e) make sure the vehicle visits customer i when the delivery person starts at node i . Constraints (2f) confirm that when the delivery person returns to customer j after serving a set of customers σ_k , the vehicle visits customer j to take the delivery person to the next customer. Constraints (2g), (2h), (2i), and (2j) are adapted from the single commodity flow subtour constraints (Gavish and Graves 1978). Constraint (2g) shows that at the start of the tour, there are n packages to be delivered.

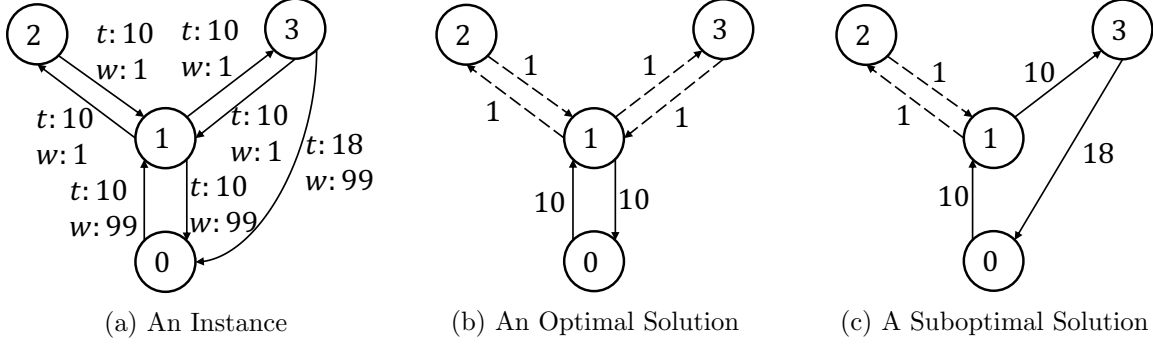


Figure 2 An Illustrative Instance of RCT2.

Constraints (2h) specify that the vehicle must visit each customer from the depot with the delivery person on board (i.e., $x_{0,i} = 1$ for some customer i). Constraints (2i) ensure that, between customers i and j such that $i, j \in \mathcal{C}$ and $i \neq j$, the vehicle can travel either with the delivery person on board (i.e., $x_{i,j} = 1$) or autonomously (i.e., there is a service set σ_k where the delivery person loads at customer i and returns to customer j). Constraints (2j) account for the change in flow at each customer: when the vehicle travels with the delivery person on board, the flow from the last customer to the next one changes by one; when the vehicle travels autonomously (i.e., $y_{i,j,k} = 1$), the flow changes by the number of customers in the service set σ_k minus one, to prevent double counting of customers in the flow constraints. Finally, the domains of the decision variables are imposed in constraints (2k), (2l) and (2m).

While RCT2 still has exponentially many variables, it has polynomially-sized constraints, compared to RCT1. This makes it more suitable for column-generation-based approaches, but existing methods require enumerating y variables nonetheless, which is not a viable option for larger instances. Most importantly, RCT2 implicitly makes the assumption that at each location i in $\mathcal{C}$, the delivery person can deliver to at most two walking paths that start and end at node i , i.e., $\sum_{k \in \mathcal{M}} y_{i,i,k} \leq 2$. The downside of this is that RCT2 can lead to suboptimal solutions.

EXAMPLE: An illustrative instance is shown in Figure 2(a). In this instance, we have four customers and one depot, $\mathcal{C} = \{1, 2, 3\}$ and $\mathcal{V} = \mathcal{C} \cup \{0\}$. For simplicity, we have $f = 0$, and $q = 1$. Also, we have the vehicle driving time $t_{0,1} = t_{1,0} = t_{1,2} = t_{2,1} = t_{1,3} = t_{3,1} = 10$, $t_{3,0} = 18$, and the delivery person walking time $w_{1,2} = w_{2,1} = w_{1,3} = w_{3,1} = 1$, $w_{0,1} = w_{1,0} = w_{3,0} = 99$. All other driving and walking times between two nodes are omitted here, but they can be obtained by using the aforementioned arcs. An optimal solution is illustrated in Figure 2(b). The solid arcs represent the vehicle and the delivery person traveling together. The dashed arcs show the delivery person walking to the service sets. Thus, in this optimal solution, the vehicle and the delivery person travel from the depot to customer 1. Then, the delivery person serves three service sets. The first service set has customer 1, the second one has customer 2, and the third one has customer 3. Then, the

vehicle and the delivery person travel to the depot from customer 1. The overall route completion time is 24.

However, RCT2 cannot obtain this optimal solution. We need to have $\sum_{k \in \mathcal{M}} y_{1,1,k} = 3$ but the right-hand side of constraint (2e) ($x_{1,1} + \sum_{l \in \mathcal{V} \setminus \{1\}} x_{l,1}$) in RCT2 is at most 2 by setting $x_{0,1} = 1$ and $x_{1,1} = 1$ following the solution in Figure 2(b). Consequently, RCT2 returns a suboptimal solution, which is shown in Figure 2(c). After arriving at customer 1, the delivery person can only serve two service sets: one for customer 1 and one for customer 2. Then, the vehicle and the delivery person must travel together to customer 3 and serve the customer there. After that, they travel together to the depot. The overall route time is 40, which is much worse than the optimal one. Moreover, given node i in $\mathcal{C}$, if there is more than one incoming arc (the vehicle and the delivery person travel together) into node i to have $\sum_{l \in \mathcal{V}} x_{l,i} = 3$, without loss of the generality, let the second incoming arc come from node l' ($x_{l',i} = 1$), constraints (2e) and (2f) ensure that the vehicle must leave node i and get to node l' before it can return to node i . That means $\sum_{j \in \mathcal{C}: j \neq i} \sum_{k \in \mathcal{M}} y_{i,j,k} = 1$. The same argument can be applied if there are three more incoming arcs. As a result, we have $\sum_{k \in \mathcal{M}} y_{i,i,k} \leq 2$ in RCT2.

3. A Branch-and-Price Approach

First, we describe an IP formulation, which leads to the branch-and-price approach we develop for the CAVADP. We also discuss in detail the differences between our formulation and the two described above.

We use $x_{i,j}$, $y_{i,j,k}$, $z_{i,j}$ and $v_{i,j}$ variables, as defined earlier in RCT1 and RCT2. Recall that when $x_{i,j}$ is 1, the vehicle drives from customers i to j with the delivery person on board for $i, j \in \mathcal{V}$ such that $i \neq j$. When $z_{i,j}$ is 1, the vehicle drops the delivery person off at customer i and meets the delivery person at customer j for $i, j \in \mathcal{C}$ such that $i \neq j$. When $y_{i,j,k}$ is 1, the delivery person loads at customer i , serves customer set σ_k , and meets the vehicle at customer j for $i, j \in \mathcal{C}$, and $k \in \mathcal{M}$. Variable $v_{i,j}$ takes nonnegative integer values. Its value represents the flow of packages on board the vehicle from customers i to j . Finally, we use $c_{i,j,k}$ to denote a path's duration of time, which is calculated as $c_{i,j,k} = \max\{t_{i,j}, \hat{w}_{i,j,k}\} + f$. We can write the following IP formulation for the CAVADP:

$$\text{(IP1) Min } \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{V}: i \neq j} t_{i,j} x_{i,j} + \sum_{i \in \mathcal{C}} \sum_{j \in \mathcal{C}} \sum_{k \in \mathcal{M}} c_{i,j,k} y_{i,j,k} \quad (3a)$$

$$\text{Subject to } \sum_{i \in \mathcal{C}} x_{i,0} = 1, \quad (3b)$$

$$\sum_{i \in \mathcal{C}} x_{0,i} = 1, \quad (3c)$$

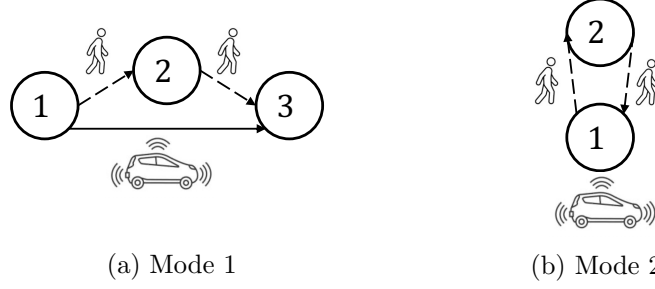


Figure 3 Two Basic Operation Modes in the CAVADP.

$$\sum_{(l,j,k) \in \mathcal{R}_i} y_{l,j,k} = 1, \quad \forall i \in \mathcal{C}, \quad (3d)$$

$$\sum_{j \in \mathcal{V}: j \neq i} x_{j,i} + \sum_{j \in \mathcal{C}: j \neq i} z_{j,i} = \sum_{j \in \mathcal{V}: j \neq i} x_{i,j} + \sum_{j \in \mathcal{C}: j \neq i} z_{i,j}, \quad \forall i \in \mathcal{C}, \quad (3e)$$

$$\sum_{k \in \mathcal{M}} y_{i,i,k} \leq n \left(\sum_{j \in \mathcal{V}: j \neq i} x_{j,i} + \sum_{j \in \mathcal{C}: j \neq i} z_{j,i} \right), \quad \forall i \in \mathcal{C}, \quad (3f)$$

$$z_{i,j} = \sum_{k \in \mathcal{M}} y_{i,j,k}, \quad \forall i, j \in \mathcal{C}, i \neq j, \quad (3g)$$

$$v_{0,i} \leq nx_{0,i}, \quad \forall i \in \mathcal{C}, \quad (3h)$$

$$v_{i,j} \leq n(x_{i,j} + \sum_{k \in \mathcal{M}} y_{i,j,k}), \quad \forall i, j \in \mathcal{V}, i \neq j, \quad (3i)$$

$$\sum_{j \in \mathcal{V}: j \neq i} v_{j,i} = \sum_{j \in \mathcal{V}: j \neq i} v_{i,j} + \sum_{k \in \mathcal{M}} |\sigma_k| y_{i,j,k}, \quad \forall i \in \mathcal{C}, \quad (3j)$$

$$x_{i,j} \in \{0, 1\}, \quad \forall i, j \in \mathcal{V}, i \neq j, \quad (3k)$$

$$y_{i,j,k} \in \{0, 1\}, \quad \forall i, j \in \mathcal{C}, k \in \mathcal{M}, \quad (3l)$$

$$z_{i,j} \in \{0, 1\}, \quad \forall i, j \in \mathcal{C}, i \neq j, \quad (3m)$$

$$v_{i,j} \in \mathbb{Z}_{\geq 0}, \quad \forall i \in \mathcal{V}, j \in \mathcal{C}, i \neq j. \quad (3n)$$

The objective function (3a) minimizes the route's total completion time. Constraints (3b) and (3c) require the truck to enter and leave the depot with the delivery person exactly once. Constraints (3d) require that each node in $\mathcal{C}$ should be serviced exactly once. Constraints (3e) are the flow balance constraints. Constraints (3f) ensure that only when the vehicle visits node i in $\mathcal{C}$ can the delivery person make the delivery on a path starting and ending at node i . Constraints (3g) require a path (i, j, k) to be selected if the vehicle drops off the delivery person at node i and picks up the delivery person at node j . Constraints (3h), (3i), and (3j) ensure the arcs visited by the vehicle form a tour, which are referred to as the Arc Order (AO) constraints. The rest of the constraints enforce the domains on the variables.

Compared to RCT1, IP1 replaces the exponentially-sized constraints (1f) in RCT1 by constraints (3f), which are a set of n constraints. This change allows us to apply column-generation

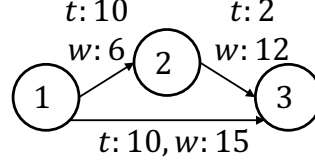


Figure 4 An Example that Does Not Satisfy Triangular Inequalities.

techniques while IP1 still has exponentially many variables. Furthermore, IP1 replaces the MTZ constraints (given in constraints (1h), (1i), (1j), and (1k)) in RCT1 by the AO constraints (given in constraints (3h), (3i) and (3j)). This is necessary to model the CAVADP correctly. A typical routing problem's optimal solution would be a cycle that includes the depot and all customers when the vehicle's travel time satisfies the triangular inequalities. The MTZ constraints impose a unique value for each node's u variable to obtain such a cycle. As a result, it forbids the vehicle from revisiting a node and restricts the operation modes to the two illustrated in Figure 3. We use solid lines for the arcs traveled by the vehicle and dashed lines for the arcs walked by the person. In mode 1, the vehicle drops the person at a node (node 1 here) and meets the person at a different one (node 3 here), while the person walks from node 1 to serve some nodes (node 2 here), and then walks to node 3. In mode 2, the vehicle drops the person at a node (node 1 here) and stays there until the person returns after the person walks to serve some nodes (node 2 here).

However, even though the walking and driving times satisfy triangular inequalities, the route time in the CAVADP does not necessarily satisfy triangular inequalities.

EXAMPLE: A small example in Figure 4 is used to illustrate this. Assume that we have one service set $\sigma_1 = \{2\}$. Then, the delivery person is dropped at node 1 and is picked up at node 3 while the person walks from node 1 to node 2, and then to node 3. Thus, we have $c_{1,3,1} = f + \max\{10, 18\} = f + 18$ for $y_{1,3,1}$, which can be linked to $z_{1,3}$ as $z_{1,3} = y_{1,3,1} = 1$. We can also serve node 2 and get to node 3 in the following way: the delivery person is dropped at node 1 and is picked up at node 2 while the person is walking from node 1 to node 2. Then, the person rides on the vehicle to node 3. We know that $c_{1,2,1} = f + \max\{10, 6\} = f + 10$ for $y_{1,2,1}$, which can be linked to $z_{1,2}$ as $z_{1,2} = y_{1,2,1} = 1$. Then, we have $x_{2,3} = 1$ with $t_{2,3} = 2$. Consequently, comparing $z_{1,2} + x_{2,3}$ and $z_{1,3}$, we have $c_{1,2,1} + t_{2,3} = f + 12 < f + 18 = c_{1,3,1}$. Thus, it violates triangular inequalities, and we should not use the MTZ constraints.

In contrast, the AO constraints track the order of the arcs visited by the vehicle and allow the vehicle to revisit a node, leading to a much richer set of possible tours, compared to the tours allowed by the MTZ constraints. Figure 5 shows four operation modes. A node (node 1 here) can be considered as the base node for an operation mode. The vehicle can transport the person from and/or to the base node, given that an operation mode starts and ends at the base node. In Mode

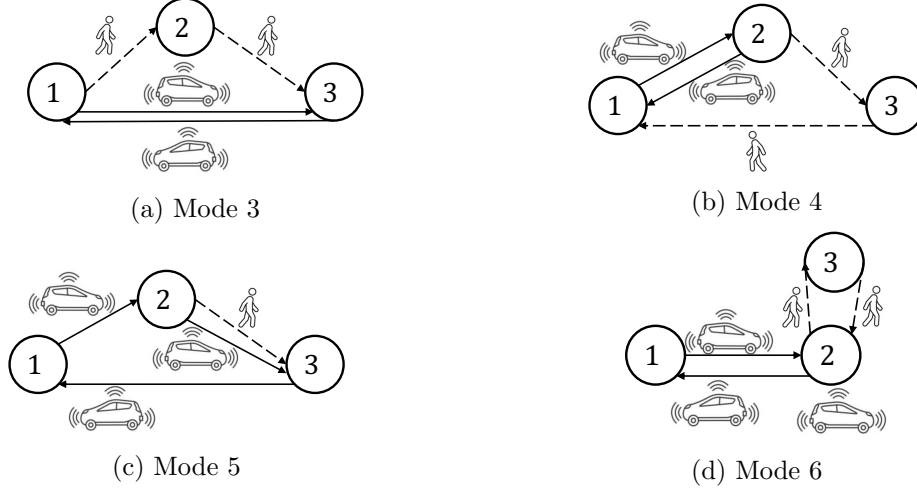


Figure 5 Four More Operation Modes in the CAVADP Allowed by the AO Constraints.

3, after dropping off the delivery person at node 1, the vehicle meets the delivery person at node 3 before they return to node 1 together. In Mode 4, the vehicle drops off the delivery person at node 2, returns to node 1 alone, and meets the delivery person there. In Mode 5, the vehicle drops off the delivery person at node 2 and travels to node 3 to meet the delivery person before they return to node 1 together. In mode 6, the vehicle drops off the delivery person at node 2 and waits there to pick up the delivery person. Then, they return to node 1 together. These additional operation modes can better utilize an autonomous vehicle for efficiency while greatly increasing the CAVADP's complexity.

Compared to RCT2, IP2 resolves the issue mentioned in Section 2 by using constraint (3f). Thus, an optimal solution of IP2 can have more than two paths starting and ending at the same node. Furthermore, the AO constraints are more compact than the single commodity constraints (given in constraints (2g), (2h), (2i), and (2j)) in RCT2. In addition, variables $v_{i,j}$ not only represent the flow but also give the number of customers that are served by all operation modes initiated at the base node $i \in \mathcal{C}$ by the calculation: $\sum_{j \in \mathcal{V}: j \neq i} v_{j,i} - \sum_{j \in \mathcal{V}: j \neq i} v_{i,j}$.

3.1. Basic Components

In this section, we introduce the basic components of our branch-and-price approach. Before detailing the column-generation procedure, let IP1R represent a restricted version of IP1 such that IP1R is identical to IP1, except that it includes a subset of the y variables. Specifically, let $\mathcal{M}'$ be a subset of $\mathcal{M}$, and $\mathcal{R}'_i$ be a subset of $\mathcal{R}_i$ considered so far. We have:

$$(\text{IP1R}) \quad \text{Min} \quad \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{V}: i \neq j} t_{i,j} x_{i,j} + \sum_{i \in \mathcal{C}} \sum_{j \in \mathcal{C}} \sum_{k \in \mathcal{M}'} c_{i,j,k} y_{i,j,k} \quad (4a)$$

$$\text{Subject to} \quad (3b), (3c), (3e), (3h), (3i), (3k), (3l), (3m), (3n),$$

$$\sum_{(l,j,k) \in \mathcal{R}'_i} y_{l,j,k} = 1, \quad \forall i \in \mathcal{C}, \quad (4b)$$

$$\sum_{k \in \mathcal{M}'} y_{i,i,k} \leq n \left(\sum_{j \in \mathcal{V}: j \neq i} x_{j,i} + \sum_{j \in \mathcal{C}: j \neq i} z_{j,i} \right), \quad \forall i \in \mathcal{C}, \quad (4c)$$

$$z_{i,j} = \sum_{k \in \mathcal{M}'} y_{i,j,k}, \quad \forall i, j \in \mathcal{C}, i \neq j, \quad (4d)$$

$$\sum_{j \in \mathcal{V}: j \neq i} v_{j,i} = \sum_{j \in \mathcal{V}: j \neq i} v_{i,j} + \sum_{k \in \mathcal{M}'} |\sigma_k| y_{i,j,k}, \quad \forall i \in \mathcal{C}. \quad (4e)$$

Then, let LP1R be the linear programming (LP) relaxation of IP1R.

3.1.1. Column Generation: IP1 has exponentially many y variables because all paths are included in IP1. A pricing procedure is needed to add them to IP1R dynamically. The column-generation procedure for variable $\mathbf{y}$ is as follows. Let α , β , γ , and δ be the dual variables associated with constraints (4b), (4c), (4d), and (4e). We denote their values as α^* , β^* , γ^* , and δ^* . Recall that a path is denoted as (i, j, k) if the vehicle drops the person at node i , meets the person at node j , and the person walks to serve the customers in service set σ_k . We look for a path (i, j, k) whose reduced cost, denoted as $\bar{c}_{i,j,k}$, is negative. If no such paths exist, we have solved LP1R. The pricing problem can be formulated as follows:

$$\begin{aligned} \min_{i \in \mathcal{C}, j \in \mathcal{C}, \sigma_k \in \Omega} \bar{c}_{i,j,k} &:= c_{i,j,k} - \sum_{l \in \sigma_k} \alpha_l^* - \mathbb{1}(i=j) \beta_i^* - \mathbb{1}(i \neq j) \gamma_{i,j}^* - |\sigma_k| \delta_i^* \\ &= f + \max\{t_{i,j}, \hat{w}_{i,j,k}\} - \sum_{l \in \sigma_k} (\alpha_l^* + \delta_i^*) - \mathbb{1}(i=j) \beta_i^* - \mathbb{1}(i \neq j) \gamma_{i,j}^*. \end{aligned} \quad (5)$$

To solve this problem, we first fix two nodes $\bar{i}$ and $\bar{j}$ in $\mathcal{C}$ as the start node and end node, respectively. Then, we can find the service set σ_k by forming an elementary path from node $\bar{i}$ to node $\bar{j}$ such that the size of σ_k is not larger than q , i.e., $|\sigma_k| \leq q$. Depending on whether $\bar{i}$ and/or $\bar{j}$ is in σ_k or not, we can limit the length of the elementary path to be q , $q+1$, or $q+2$. Overall, the pricing problem can be solved as a variant of the ESPPRC.

All-Pair Pulse Algorithm: We present a tailored pulse algorithm for our pricing problem. From a high level, the pulse algorithm (Lozano et al. 2016) employs a recursive depth-first search method that integrates bounds and rollback pruning strategies to discard partial paths. In this way, the algorithm does not need to search the paths that are not promising.

Two features exist in our algorithm. First, the path has a nonlinear cost function because we need to take the larger value of the driving time and the walking time for serving the customers. This is different from most applications of the ESPPRC and requires a careful design of a major component: the bounds pruning strategy. Second, the CAVADP needs elementary paths between all pairs of customers. Rather than returning exactly the elementary shortest path, this algorithm returns the elementary shortest paths between all pairs of customers. We refer to it as the all-pair pulse algorithm, which is detailed in EC.1.

3.1.2. Initializing y Variables: We initialize IP1R with some basic y variables by constructing paths as follows: for each i in $\mathcal{C}$, we create a path that visits nodes $i, i+1$, and i , where $i+1 = 1$ if $i = n$. In other words, we construct a path that starts at node i and ends at node i while serving node $i+1$. However, when $i = n$, the path serves node 1. For each of these paths, the delivery person serves one node, which is node $i+1$. In addition, for each i in $\mathcal{C}$, there is also a path for the delivery person to stay at node i and serve only node i . Hence, we have $2n$ initial y variables in total.

3.1.3. Branching Rule: In the branching procedure, we branch on fractional z variables first. After that, we consider x variables before y variables. Furthermore, we pick the most fractional one to branch on. Specifically, let $(\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}})$ be the solution value at the current branch node. Assuming we are branching on z variables, we choose $(\hat{i}, \hat{j})$, where $(\hat{i}, \hat{j}) = \arg \min\{i, j \in \mathcal{C} \text{ and } i \neq j : |\hat{z}_{i,j} - 0.5|\}$. We note that this branching rule does not affect the pricing problem.

3.1.4. Path Enumeration: A path is considered potentially beneficial to enhancing the best feasible solution only if its reduced cost is less than the existing gap between the primal and dual bounds. We might enumerate all paths within this gap to close the duality gap following the root node. A similar strategy has been applied to the vehicle routing problem in Baldacci et al. (2008). This process of path enumeration can be accomplished using the all-pair pulse algorithm. However, in practice, the number of paths is often too large. In our implementation, we run the all-pair pulse algorithm one more time once we finish the root node. Then, among the paths found by the all-pair pulse algorithm, we include the ones whose reduced costs are within the gap to the master problem.

4. Enhancements

In this section, we introduce several enhancements. We focus on situations where driving is not slower than walking, which can often be satisfied in real-world practice. The enhancements include preprocessing steps and several valid inequalities based on the properties of optimal solutions.

4.1. Preprocessing

We start with preprocessing. It is possible to conduct a few preprocessing steps to improve the computational tractability. The first preprocessing involves constructing a restricted walking graph. Let $G_w = (\mathcal{C}, \mathcal{A}_w)$ be the restricted walking graph, the graph over which the delivery person might travel while walking. We only include an arc from $\mathcal{A}$ into $\mathcal{A}_w$ when the sum of its driving time and the loading time is strictly larger than its walking time, i.e., $\mathcal{A}_w = \{(i, j) \in \mathcal{A} : t_{i,j} + f > w_{i,j}\}$. These arcs represent the potential to walk between two customers. The restricted walking graph is constructed based on the following proposition. Even though Reed et al. (2022a) propose the

restricted walking graph, we provide a formal proof and show that, to construct the restricted walking graph, a necessary condition is that driving is not slower than walking by presenting a counter-example. Further, we show that is also a sufficient condition for the restricted walking graph.

PROPOSITION 1. *When driving is not slower than walking, i.e., $t_{i,j} \leq w_{i,j}$ for all (i,j) in $\mathcal{A}$, the delivery person should never walk on an arc (i,j) if $t_{i,j} + f \leq w_{i,j}$.*

As a result of Proposition 1, the delivery person must visit a node with the vehicle if this node is isolated in the restricted walking graph. This result is formally stated in Lemma 1.

LEMMA 1. *Let i in $\mathcal{C}$ and assume driving is not slower than walking, i.e., $t_{i,j} \leq w_{i,j}$ for all (i,j) in $\mathcal{A}$. If $t_{i,j} + f \leq w_{i,j}$ and $t_{j,i} + f \leq w_{j,i}$ for all j in $\mathcal{C} : i \neq j$, then, the following valid inequalities hold: $\sum_{j \in \mathcal{C} : i \neq j} x_{j,i} = \sum_{j \in \mathcal{C} : i \neq j} x_{i,j} = 1$ and $\sum_{k \in \mathcal{M}} y_{i,i,k} = 1$.*

We adopt several results from Reed et al. (2022a) and show how to apply them in our branch-and-price approach.

PROPOSITION 2. *Reed et al. (2022a, Claim 2): When driving is not slower than walking, i.e., $t_{i,j} \leq w_{i,j}$ for all (i,j) in $\mathcal{A}$, there exists an optimal solution such that if $y_{i,j,k} = 1$ for some path (i,j,k) that starts at node i and ends at node j , then $i,j \in \sigma_k$.*

Proposition 2 means that the start and end nodes of a path must be served when driving is not slower than walking. This has several implications. First, at most, one path can be included in an optimal solution if the path starts and ends at the same node, i.e., $\sum_{k \in \mathcal{M}} y_{i,i,k} \leq 1$ for all $i \in \mathcal{C}$. Thus, we can strengthen the formulation by utilizing this property. Specifically, using n on the right-hand side of constraint (3f) can impose computational challenges. However, with Proposition 2, we can tighten constraint (3f) as

$$\sum_{k \in \mathcal{M}} y_{i,i,k} \leq \sum_{j \in \mathcal{V} : i \neq j} x_{j,i} + \sum_{j \in \mathcal{C} : i \neq j} z_{j,i}, \quad \forall i \in \mathcal{C}.$$

A more important and interesting consequence is that formulation RCT2 is valid because, at most, one path that starts and ends at node i will be used. This result is formally stated in the following lemma:

LEMMA 2. *When driving is not slower than walking, i.e., $t_{i,j} \leq w_{i,j}$ for all (i,j) in $\mathcal{A}$, formulation RCT2 is valid.*

The next result from Reed et al. (2022a) can further strengthen the master problem.

PROPOSITION 3. *Reed et al. (2022a, Lemma 2): When driving is not slower than walking, i.e., $t_{i,j} \leq w_{i,j}$ for all (i,j) in $\mathcal{A}$, if $y_{i,i,k} = 1$ for some service set $\sigma_k \in \Omega$, then $\sigma_k = \{i\}$.*

With Proposition 3, we can include all $y_{i,i,k}$ variables for all i in $\mathcal{C}$ because there is exactly one path with service set $\{i\}$ that starts and ends at node i for node i in $\mathcal{C}$.

4.2. Valid Inequalities

We can also derive several valid inequalities when driving is not slower than walking. Recall that, as shown in Proposition 2, when driving is not slower than walking, there exists an optimal solution such that a service set always contains the start and end nodes of the path. This leads to the observation that the vehicle will only visit a customer at most once. As a result, we should be able to focus on the basic operation modes shown in Figure 3, and there is no need to worry about the four operation modes shown in Figure 5. This enables us to use the following subtour elimination constraints, which are in the spirit of the classic Danzig-Fulkerson-Johnson (DFJ) subtour elimination constraints (Dantzig et al. 1954), on a tour formed by the vehicle. Let $\mathcal{S}$ be a subset of nodes in $\mathcal{C}$ that form a subtour.

$$\sum_{i,j \in \mathcal{S}: i \neq j} (x_{i,j} + z_{i,j}) \leq |\mathcal{S}| - 1, \quad \forall \mathcal{S} \subseteq \mathcal{C}, |\mathcal{S}| \geq 2. \quad (6)$$

Furthermore, we introduce a strengthened version of the subtour elimination constraints. Although similar results can be found in the literature for vehicle routing problems, these existing results are not applicable to the CAVADP. Our findings are specifically derived for the CAVADP. The set of strengthened valid inequalities is formally presented as follows:

PROPOSITION 4. $\sum_{i,j \in \mathcal{S}: i \neq j} (x_{i,j} + z_{i,j}) - \sum_{i \in \mathcal{S} \setminus \{k\}} (\sum_{j \in \mathcal{V}: i \neq j} x_{j,i} + \sum_{j \in \mathcal{C}: i \neq j} z_{j,i}) \leq 0, \forall \mathcal{S} \subseteq \mathcal{C}, |\mathcal{S}| \geq 2, k \in \mathcal{S}$ is valid for the CAVADP and dominates constraint set (6), $\sum_{i,j \in \mathcal{S}: i \neq j} (x_{i,j} + z_{i,j}) \leq |\mathcal{S}| - 1, \forall \mathcal{S} \subseteq \mathcal{C}, |\mathcal{S}| \geq 2$.

This set of valid inequalities is called the CAVADP-strengthened subtour elimination constraints.

EXAMPLE: To demonstrate the strength of the CAVADP-strengthened subtour elimination constraints, consider $\mathcal{S} = \{1, 2, 3\}$ and $x_{1,2}^* + z_{1,2}^* = x_{2,3}^* + z_{2,3}^* = x_{3,1}^* + z_{3,1}^* = 2/3$. Thus, the corresponding constraint (6) is satisfied because $x_{1,2}^* + z_{1,2}^* + x_{2,3}^* + z_{2,3}^* + x_{3,1}^* + z_{3,1}^* = 2$. Also, we have $\sum_{j \in \mathcal{C}: 1 \neq j} x_{j,1}^* + \sum_{j \in \mathcal{V}: 1 \neq j} z_{j,1}^* = \sum_{j \in \mathcal{C}: 2 \neq j} x_{j,2}^* + \sum_{j \in \mathcal{V}: 2 \neq j} z_{j,2}^* = \sum_{j \in \mathcal{C}: 3 \neq j} x_{j,3}^* + \sum_{j \in \mathcal{V}: 3 \neq j} z_{j,3}^* = 2/3$. Without loss of generality, let $k = 1$. Then, the corresponding CAVADP-strengthened subtour elimination constraint is violated because $x_{1,2}^* + z_{1,2}^* + x_{2,3}^* + z_{2,3}^* + x_{3,1}^* + z_{3,1}^* - (\sum_{j \in \mathcal{C}: 2 \neq j} x_{j,2}^* + \sum_{j \in \mathcal{V}: 2 \neq j} z_{j,2}^*) - (\sum_{j \in \mathcal{C}: 3 \neq j} x_{j,3}^* + \sum_{j \in \mathcal{V}: 3 \neq j} z_{j,3}^*) = 2/3 > 0$. Consequently, the corresponding CAVADP-strengthened subtour elimination constraint cuts off this fractional solution.

The following set of valid inequalities is a strengthened version of the one in Claim 5 of Reed et al. (2022a) because only one of the two variables $x_{i,j}$ and $x_{j,i}$ is included on the left-hand side there. We also provide a different proof. The following inequalities indicate whether an arc (i, j) in $\mathcal{A}$ is either traveled by the autonomous vehicle with the delivery person or is visited by the delivery person alone.

PROPOSITION 5. $x_{i,j} + x_{j,i} + \sum_{(l,m,k) \in \mathcal{R}_i \cap \mathcal{R}_j} y_{l,m,k} \leq 1, \forall i, j \in \mathcal{C}, i \neq j$, is valid for the CAVADP.

Moreover, let $\mathcal{O}_i$ be the set of paths that serve node i , while node i is not the start or end node of the path. We have the following valid inequalities, which ensure that when node i in $\mathcal{C}$ is not the start or end node of a path, node i is either visited by the autonomous vehicle with the delivery person or is visited by the delivery person alone.

PROPOSITION 6. $\sum_{j \in \mathcal{C}: i \neq j} x_{j,i} + \sum_{(l,m,k) \in \mathcal{O}_i} y_{l,m,k} \leq 1, \forall i \in \mathcal{C}$, is valid for the CAVADP.

Putting it together, when driving is not slower than walking, we have a strengthened formulation of IP1 as follows:

$$(IP2) \text{ Min } \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{V}: i \neq j} t_{i,j} x_{i,j} + \sum_{i \in \mathcal{C}} \sum_{j \in \mathcal{C}} \sum_{k \in \mathcal{M}} c_{i,j,k} y_{i,j,k} \quad (7a)$$

Subject to (3b), (3c), (3d), (3e), (3g), (3h), (3i), (3j), (3k), (3l), (3m), (3n),

$$\sum_{k \in \mathcal{M}} y_{i,i,k} \leq \sum_{j \in \mathcal{V}} x_{j,i} + \sum_{j \in \mathcal{C}} z_{j,i}, \quad \forall i \in \mathcal{C}, \quad (7b)$$

$$\sum_{i,j \in \mathcal{S}: i \neq j} (x_{i,j} + z_{i,j}) - \sum_{i \in \mathcal{S} \setminus \{k\}} \left(\sum_{j \in \mathcal{V}: i \neq j} x_{j,i} + \sum_{j \in \mathcal{C}: i \neq j} z_{j,i} \right) \leq 0, \quad \forall \mathcal{S} \subseteq \mathcal{C}, |\mathcal{S}| \geq 2, k \in \mathcal{S}, \quad (7c)$$

$$x_{i,j} + x_{j,i} + \sum_{(l,m,k) \in \mathcal{R}_i \cap \mathcal{R}_j} y_{l,m,k} \leq 1, \quad \forall i, j \in \mathcal{C}, i \neq j, \quad (7d)$$

$$\sum_{j \in \mathcal{C}: i \neq j} x_{j,i} + \sum_{(l,m,k) \in \mathcal{O}_i} y_{l,m,k} \leq 1, \quad \forall i \in \mathcal{C}. \quad (7e)$$

Similar to IP1R and LP1R, we define IP2R and LP2R for IP2.

IP2 contains exponentially many constraints due to constraint (7c), necessitating a row-generation procedure. Our goal is to eliminate subtours, except those that involve the depot. The following row-generation procedure is triggered once we obtain an optimal solution to LP2R, denoted as $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$. Since eliminating subtours implies removing directed cycles, we begin by running a heuristic that applies a shortest path algorithm to detect cycles induced by $\mathbf{x}^*$ and $\mathbf{z}^*$. Define $\mathbf{h} = 1 - \mathbf{x}^* - \mathbf{z}^*$, leading to the inequalities $\sum_{i,j \in \mathcal{S}: i \neq j} h_{i,j} \geq 1$, since $\sum_{i,j \in \mathcal{S}: i \neq j} (x_{i,j} + z_{i,j}) \leq |\mathcal{S}| - 1$ for all $\mathcal{S} \subseteq \mathcal{C}$ with $|\mathcal{S}| \geq 2$. Then, for each node i in $\mathcal{C}$, create a dummy node d_i and add the incoming arcs of node i to it such that $n(d_i)^- = n(i)^-$ and $h_{j,d_i} = h_{j,i}$ for all $j \in n(i)^-$. If a shortest path from nodes i to d_i has a distance less than 1, a violated subtour elimination constraint (a cycle that starts and ends at node i) is detected. Subsequently, we add a new constraint $\sum_{i,j \in \mathcal{S}: i \neq j} (x_{i,j} + z_{i,j}) - \sum_{i \in \mathcal{S} \setminus \{k\}} (\sum_{j \in \mathcal{V}: i \neq j} x_{j,i} + \sum_{j \in \mathcal{C}: i \neq j} z_{j,i}) \leq 0$ to IP2R for all $k \in \mathcal{S}$, where $\mathcal{S}$ contains the nodes in the shortest path, excluding dummy node d_i .

If the heuristic fails to identify a violated constraint, we proceed with an exact separation as follows. For each node i in $\mathcal{C}$, define a binary variable ζ_i , which takes the value of 1 if node i is included in the set $\mathcal{S}$, and 0 otherwise (this helps identify the nodes in set $\mathcal{S}$). For all $i, j \in \mathcal{C}$ and

$i \neq j$, introduce a binary variable $\eta_{i,j}$, which is set to 1 if the arc (i,j) is included in the expression $\sum_{i,j \in \mathcal{S}: i \neq j} (x_{i,j} + z_{i,j})$; otherwise, it is 0. Additionally, for each node k in $\mathcal{C}$, define a binary variable θ_k , which takes the value of 1 if node k is the one from $\mathcal{S}$ that is selected for exclusion from the summation in the second term on the left-hand side of constraint (7c); otherwise, it is 0. With these variables, we formulate the following separation problem.

$$\begin{aligned} (\text{SepEx}) \quad \text{Max} \quad & \sum_{i \in \mathcal{C}} \sum_{j \in \mathcal{C}: i \neq j} (x_{i,j}^* + z_{i,j}^*) \eta_{i,j} - \sum_{i \in \mathcal{C}} \left(\sum_{j \in \mathcal{V}: i \neq j} x_{j,i}^* + \sum_{j \in \mathcal{C}: i \neq j} z_{j,i}^* \right) \zeta_i \\ & + \sum_{k \in \mathcal{C}} \left(\sum_{j \in \mathcal{V}: k \neq j} x_{j,k}^* + \sum_{j \in \mathcal{C}: k \neq j} z_{j,k}^* \right) \theta_k \end{aligned} \quad (8a)$$

$$\text{Subject to} \quad \eta_{i,j} \leq \zeta_i, \eta_{i,j} \leq \zeta_j, \quad \forall i, j \in \mathcal{C}, i \neq j, \quad (8b)$$

$$\sum_{i \in \mathcal{C}} \zeta_i \geq 2, \quad (8c)$$

$$\sum_{k \in \mathcal{C}} \theta_k = 1, \quad (8d)$$

$$\theta_k \leq \zeta_k, \quad \forall k \in \mathcal{C}, \quad (8e)$$

$$\eta_{i,j} \in \{0, 1\}, \quad \forall i, j \in \mathcal{C}, i \neq j, \quad (8f)$$

$$\zeta_i, \theta_i \in \{0, 1\}, \quad \forall i \in \mathcal{C}. \quad (8g)$$

The objective function (8a) is designed to maximize the left-hand side of constraint (7c). If the objective value exceeds zero, it indicates that a violated constraint has been identified. Constraint (8b) ensures that an arc (i,j) can be included only if both nodes i and j are selected in $\mathcal{S}$. Constraint (8c) requires at least two nodes to be selected in $\mathcal{S}$. Constraint (8d) ensures that exactly one node is excluded. Constraint (8e) states that only one of the selected nodes can be excluded. Constraint (8f) and Constraint (8g) enforce the binary nature of the variables.

Before concluding this section, we note that an outline of the branch-and-price procedure is described in Algorithms 7 in EC.2.

5. Computational Experiments

In this section, we present our computational experience with a set of benchmark instances from the literature. Our experiments aim to: (i) assess the overall effectiveness of the proposed approaches and (ii) report the best-found solutions for the instances. The experiments were run on a machine with an M1 Ultra processor, 128 GB RAM, and macOS. We used Gurobi 11.0.0 with the Python API, keeping its default settings and utilizing ten cores for a parallel implementation of the all-pair pulse algorithm. In our implementation, we stop the branch-and-price approach after solving the root node. As will become evident later, this stopping criterion typically yields high-quality solutions already. Unless stated otherwise, we set a time limit of six hours to facilitate comparisons with alternative formulations and approaches.

We conduct the experiments with the instances from Reed et al. (2022a). This set of instances is based on real-world data in the state of Illinois and is generated to reflect real-world changes in customer geographies based on their population densities. There are nine counties in Illinois. For each county, ten graphs are generated. There are 50 customers in each graph. Thus, there are 51 nodes, including the depot, for each graph. The interested reader is referred to Reed et al. (2022a, Section 5.1) for more details. Following the setting in Reed et al. (2022a), we set the loading time $f = 2.8$ (minutes) and the capacity $q = \{1, 2, 3, 4, 5\}$. We consider a capacity of three packages ($q = 3$) to be the standard practice, as shown in a study in London (Allen et al. 2018). In total, we have 450 instances. Among them, 350 instances satisfy the assumption that driving is not slower than walking, which is referred to as instance-set 1, while 100 instances do not, which is referred to as instance-set 2. All instances and source code can be downloaded from this IJOC GitHub repository (Zhang 2024) through this link (<https://github.com/INFORMSJoC/2023.0177>).

5.1. Results When Driving Is Not Slower Than Walking

In this section, we examine the results regarding instance-set 1, which contains 350 instances, and driving is not slower than walking. We conduct experiments with four settings. The first one is referred to as “RCT2”, where we implement all of the model improvements in Reed et al. (2022a, Section 4), including service set reduction, variable reduction, and valid inequalities. We include “RCT2” because RCT2 is a valid formulation when driving is not slower than walking, as shown in Lemma 2. The second and third settings are called “IP1” and “IP2”, where we test the performance of IP1 and IP2 after applying service set reduction and variable reduction like “RCT2”. We note that in our implementation of IP2, the CAVADP-strengthened subtour elimination constraints are implemented as user cuts in Gurobi. The last setting is referred to as “BP”, where we apply the proposed branch-and-price approach. Specifically, we combine the solution approach in Section 3 and Section 4 because the 350 instances satisfy the assumption that driving is not slower than walking. The detailed results can be found in this IJOC GitHub repository (Zhang 2024) through this link (<https://github.com/INFORMSJoC/2023.0177>).

5.1.1. Run Times: We first look at the run times. Figure 6 shows the average run times of RCT2, IP1, IP2, and BP over all 350 instances. The average run time (168.7 seconds) of RCT2 is similar to that (168.3 seconds) of IP1. However, IP2 is significantly faster (over six times faster). Its average run time is 25.3 seconds. While BP has an average run time of 72.0 seconds, it does not solve all instances optimally like the other three settings. Furthermore, we present the average run times of the four settings on instances with various capacities (i.e., $q = 1, 2, 3, 4, 5$). Generally, a larger capacity leads to a longer run time. It is also worth noting that the largest run times are 16,601.1, 20,187.2, 2,922.5, and 1,349.1 seconds for RCT2, IP1, IP2, and BP, respectively.

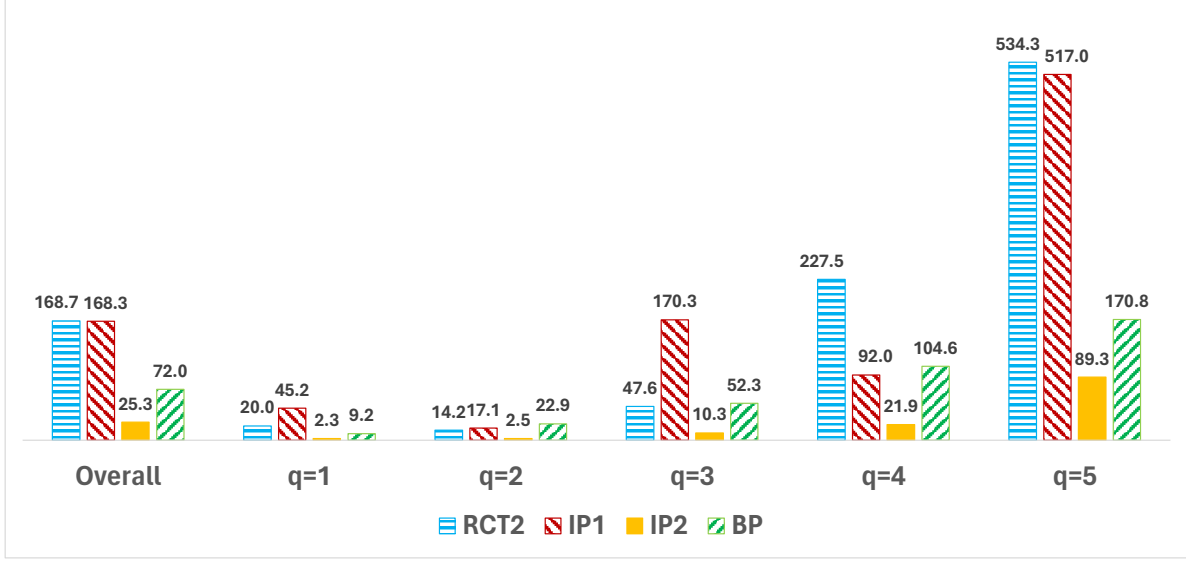


Figure 6 Average Run Times (in Seconds) of RCT2, IP1, IP2 and BP and Those on Instances with $q = 1, 2, 3, 4, 5$ on Instance-Set 1.

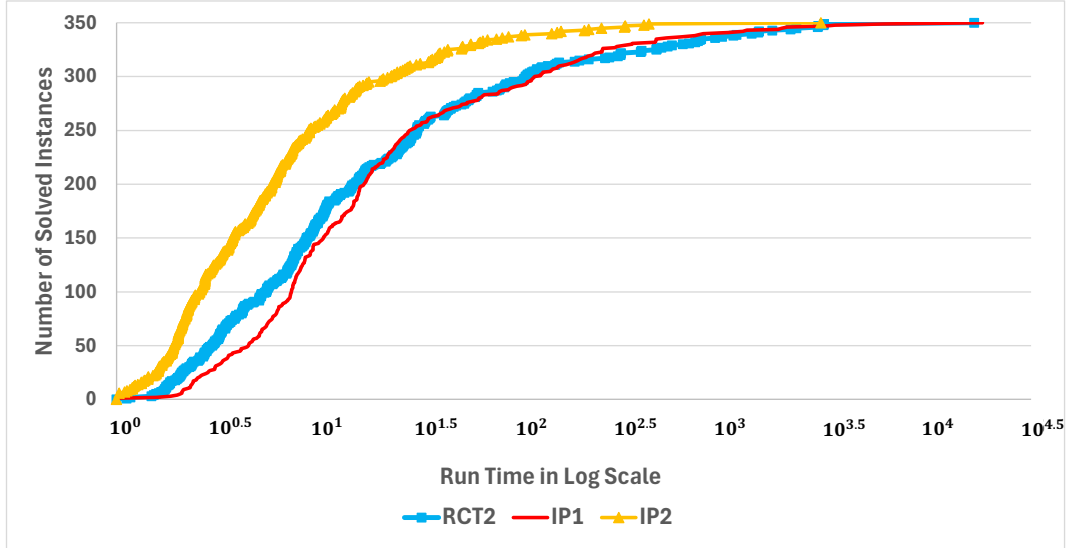


Figure 7 Performance Profile of RCT2, IP1, and IP2 on Instance-Set 1.

We next focus on the three settings, RCT2, IP1, and IP2, because they solve all instances optimally. We create the performance profile plot in Figure 7. The x-axis is the run time in log scale with base 10. The y-axis is the number of solved instances. The blue line with a rectangular marker is for RCT2. The red line is for IP2. The yellow line with a triangular marker is for IP2. A line closer to the top left corner indicates a better performance. On the one hand, it shows that the performances of RCT2 and IP1 are close to each other. On the other hand, IP2 has a better performance, as there is a clear gap between its performance profile and the other two.

5.1.2. LP Relaxations: We solve the LP relaxations of RCT2, IP1, and IP2 to understand our observations further. Using RCT2 as a baseline, the relative comparison of IP1 is calculated as $Z_{LP1}/Z_{RCT2} - 1$, where Z_{RCT2} and Z_{LP1} are the objective values of the LP relaxation of RCT2 and

	LP1						LP2					
	Overall	$q=1$	$q=2$	$q=3$	$q=4$	$q=5$	Overall	$q=1$	$q=2$	$q=3$	$q=4$	$q=5$
Max	0.00	0.00	0.00	0.00	0.00	0.00	49.56	41.16	49.56	47.76	48.54	48.67
Avg	-0.03	0.00	-0.01	-0.03	-0.04	-0.05	17.15	18.67	18.25	16.47	16.25	16.12
Min	-0.25	0.00	-0.06	-0.14	-0.19	-0.25	1.73	3.70	4.18	2.70	1.73	2.03

Table 2 LP Relaxation Comparison (%) of IP1 and IP2 to RCT2 in Instance-Set 1.

LP1, respectively. Similarly, the relative improvement of IP2 is calculated as $Z_{LP2}/Z_{RCT2} - 1$, where Z_{LP2} is the objective value of the LP2. The summary results (the maximum, average, and minimum percentages) are presented in Table 2. Overall, the LP relaxation of IP1 is 0.03% looser than that of RCT2 on average, while the best and worst cases are 0.00% and 0.25%, respectively. Thus, the LP relaxations of RCT2 and IP1 are essentially the same. In contrast, LP2 is, on average, 17.15% tighter than the LP relaxation of RCT2 on average, while the best and worst cases are 49.56% and 1.73%, respectively. These numbers demonstrate the impact obtained by the enhancements in Section 4. It shows that IP2 is a much stronger formulation than RCT2 and IP1, and provides an explanation for the superior performance of IP2. We also present the same information for instances with various capacities (i.e., $q = 1, 2, 3, 4, 5$). Generally, we observe that as capacity increases, the average improvement in LP2 decreases slightly, compared to the LP relaxation of RCT2.

	Overall	$q=1$	$q=2$	$q=3$	$q=4$	$q=5$
Max	3.75	0.00	0.01	0.00	2.19	3.75
Avg	0.25	0.00	0.00	0.00	0.54	0.72
Min	0.00	0.00	0.00	0.00	0.00	0.00

Table 3 Optimality Gaps (%) of BP in Instance-Set 1.

5.1.3. Solution Quality of BP: Next, we evaluate the quality of the solutions obtained by BP. To achieve this, we calculate the optimality gap as $Z_{BP}/Z_{IP2} - 1$, where Z_{BP} and Z_{IP2} are the objective values of BP and IP2, respectively. We present the summary results (the maximum, average, and minimum percentages) in Table 3. Across all of the 350 instances, the average, maximum, and minimum gaps are 0.25%, 3.75%, and 0.00%, respectively. Furthermore, the BP setting finds optimal solutions for 228 instances. In addition, 243 and 317 instances are within 0.1% and 1% gaps, respectively. It should be safe to conclude that the proposed branch-and-price approach is able to find high-quality solutions.

Putting it together, we can see that IP2 is the best one among the four settings in instance-set 1. Not only does it solve all instances optimally, but IP2 also has a faster run-time performance (the average run time is 25.3 seconds) than all of the other three settings. Thus, we recommend using IP2 when driving is not slower than walking.

5.2. Results When Driving Could Be Slower Than Walking

This section presents the results on instance-set 2, which contains 100 instances. Instead of four settings, we conduct experiments with two settings, IP1 and BP, because RCT2 and IP2 are not valid for instances in this set due to the fact that driving could be slower than walking.

	$q = 1$		$q = 2$		$q = 3$		$q = 4$		$q = 5$	
	IP1	BP	IP1	BP	IP1	BP	IP1	BP	IP1	BP
Cook_1	3.8	19.9	13.3	36.1	505.8	225.8	9856.7	278.5	NA	329.6
Cook_4	1.0	23.0	10.5	42.9	164.0	94.4	6970.8	163.9	NA	311.8
Cook_5	1.5	20.0	12.0	66.6	76.2	201.0	1646.6	321.9	NA	642.9
Cook_7	4.3	22.5	13.5	71.3	77.6	121.8	2224.3	152.9	NA	257.6
Cook_8	2.8	21.2	16.0	50.4	425.7	189.1	10053.8	195.4	NA	412.6
Winnebago_1	3.4	20.1	10.3	49.3	196.2	84.7	8785.8	124.7	NA	315.1
Winnebago_5	1.6	18.7	5.8	52.4	44.6	54.7	1544.0	78.0	NA	170.4
Champaign_2	128.9	157.6	107.1	125.0	1779.8	473.0	NA	3661.9	NA	12687.5
Champaign_6	31.0	62.9	196.0	65.7	3198.7	170.8	20098.4	638.3	NA	1054.5
Champaign_8	3.8	21.2	5.4	55.5	70.7	79.9	3863.6	170.2	NA	196.1
LaSalle_6	38.5	54.6	232.4	221.7	1192.5	515.0	NA	274.8	NA	1703.9
LaSalle_7	3.1	22.8	14.9	39.1	497.7	110.1	10284.8	208.5	NA	223.6
Adams_2	3.9	33.1	17.9	93.5	214.2	187.0	9465.1	239.8	NA	698.2
Adams_6	1.7	30.3	3.8	46.4	73.7	103.6	3327.1	142.6	NA	689.2
Jefferson_9	15.9	33.3	258.6	102.5	1107.9	169.0	NA	364.3	NA	590.9
Jefferson_10	4.7	20.8	23.4	48.9	370.4	55.2	NA	259.4	NA	1151.5
Johnson_2	43.5	164.4	104.9	239.4	794.3	495.3	9309.8	489.1	NA	1044.5
Johnson_7	10.9	74.6	21.4	180.1	168.8	234.3	4245.2	376.3	NA	454.0
Johnson_8	7.0	168.8	33.1	341.9	272.5	535.7	1652.8	1261.0	NA	1965.2
Cumberland_1	3.9	28.1	21.1	93.9	222.8	144.4	8460.8	155.5	NA	321.0

Table 4 Run Times (in Seconds) of IP1 and BP on Instance-Set 2.

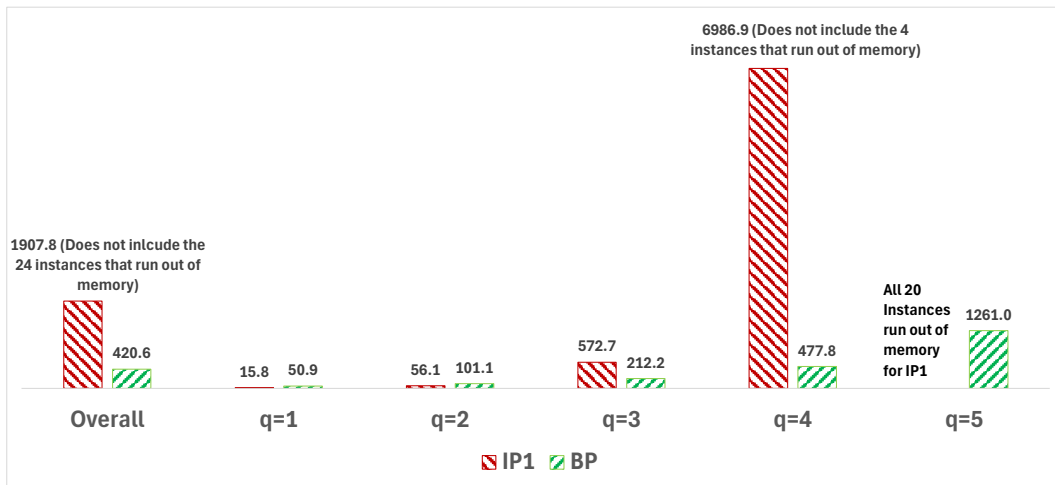


Figure 8 Average Run Times (in Seconds) of IP1 and BP and Those on Instances with $q = 1, 2, 3, 4, 5$ on Instance-Set 2.

5.2.1. Run Times: Table 4 contains the run times of IP1 and BP. IP1 cannot find solutions for 24 instances, which are indicated as “NA” (Not Applicable). They are too big to build models for Gurobi with the 128 GB memory in the machine. The computer reports this in our experiments after a long time (at least four hours). On the other hand, BP can find solutions for all instances in this set. For the instances with $q = 4$ and $q = 5$, their solutions are reported for the first time in the literature (Reed et al. (2022a) only report solutions for instances with $q = 4$ and $q = 5$ in instance-set 1). Figure 8 shows the average run times of IP1 and BP over all instances and those on instances with various capacities (i.e., $q = 1, 2, 3, 4, 5$). Overall, BP has a shorter average run time (420.6 seconds), compared to that of IP1 (1907.8 seconds). However, IP1 is faster when the instances are smaller ($q = 1, 2$). BP demonstrates bigger advantages when the instances become larger ($q = 3, 4, 5$).

	$q = 1$		$q = 2$		$q = 3$		$q = 4$		$q = 5$	
Total y Variables	125000		3187500		52187500		627812500		5921650000	
	IP1	BP	IP1	BP	IP1	BP	IP1	BP	IP1	BP
Cook_1	429	4925	8926	18409	357882	37568	15877998	54044	NA	74192
Cook_4	217	4325	4829	15173	211076	31244	10294287	47593	NA	69987
Cook_5	102	4422	1048	17324	32722	34456	1789805	51691	NA	71853
Cook_7	105	3992	1452	15864	58485	35249	3325320	54665	NA	78707
Cook_8	513	4806	11147	15961	474160	33797	21045987	50421	NA	67326
Winnebago_1	858	3225	13512	17114	533782	39719	26754326	68793	NA	106361
Winnebago_5	106	4242	1009	14511	36278	28502	1981594	42392	NA	57540
Champaign_2	3338	2364	71138	21614	3455114	62262	210806994	116600	NA	226445
Champaign_6	6324	4431	89716	17570	2403520	41040	83269490	74677	NA	108023
Champaign_8	132	4790	1654	15777	71128	30461	3687654	43707	NA	57377
LaSalle_6	7293	3902	157070	19297	6024884	58094	276374630	96011	NA	218359
LaSalle_7	1647	3499	22464	16702	621662	43954	23791694	76553	NA	106301
Adams_2	510	3967	9391	17181	459859	39256	24487465	63865	NA	96056
Adams_6	121	3916	1741	15620	83199	36589	4546613	58241	NA	86040
Jefferson_9	4830	3061	89804	16745	3241730	39216	141224540	63083	NA	100304
Jefferson_10	4759	2803	71151	14675	3426889	37656	195548887	71511	NA	103702
Johnson_2	1574	4053	24787	20267	730861	47321	28936499	79214	NA	122613
Johnson_7	178	3183	2990	15805	182390	38836	11500140	64832	NA	117706
Johnson_8	105	3632	955	17025	36765	43304	2154000	80494	NA	124109
Cumberland_1	454	3437	8141	15438	399369	33839	20931121	55066	NA	83060

Table 5 Numbers of y Variables of IP1 and BP in Instance-Set 2.

5.2.2. Number of y Variables: The difference in the performance between IP1 and BP might be explained by the number of y variables. It is worth noting that when $n = 50$, the total numbers of y variables are 627,812,500 and 5,921,650,000 for $q = 4$ and $q = 5$, respectively. In other words, the resulting formulations are extremely large. Table 5 shows the total number of y variables with various capacities, and that in IP1 and BP after conducting service set reduction and variable reduction, as described in Reed et al. (2022a, section 4). When the capacity is four (i.e., $q = 4$),

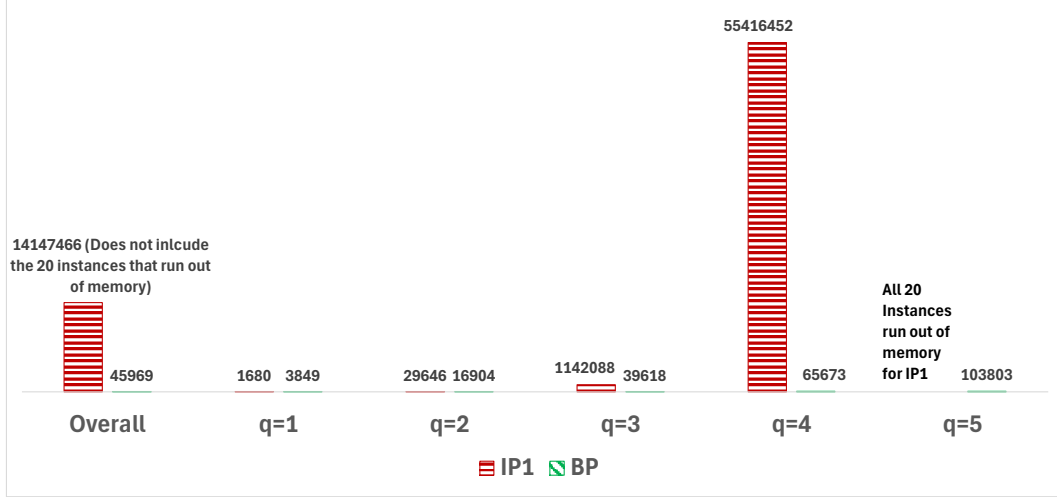


Figure 9 Average Numbers of y variables in IP1 and BP on Instance-Set 2.

the numbers of y variables for the four instances that cannot be solved are in bold. For these four instances, we were able to finish the variable enumeration process. When the capacity is five (i.e., $q = 5$), we could not even finish this process within the six-hour time limit. Thus, we use “NA” to indicate that. Figure 9 shows the average numbers of y variables in IP1 and BP over all instances, and those on instances with various capacities (i.e., $q = 1, 2, 3, 4, 5$). Overall, on average, BP only needs to use a small fraction (less than 0.5%) of y variables, compared to IP1. It should be clear that the number of y variables grows when the capacity increases. Furthermore, it grows extremely fast in IP1 but is more manageable in BP.

	$q = 1$		$q = 2$		$q = 3$		$q = 4$		$q = 5$	
	IP1	BP	IP1	BP	IP1	BP	IP1	BP	IP1	BP
Cook_1	173.71	173.71	130.88	130.88	116.30	116.30	109.30	109.30	NA	104.70
Cook_4	159.16	159.16	112.00	112.00	97.71	97.71	89.66	89.66	NA	85.54
Cook_5	168.57	168.57	122.57	122.57	108.74	108.74	104.19	104.67	NA	100.05
Cook_7	165.18	165.18	115.38	115.38	102.17	102.17	93.03	93.03	NA	90.59
Cook_8	166.86	166.86	117.72	117.78	100.22	100.22	91.48	91.48	NA	87.69
Winnebago_1	165.17	165.17	114.20	114.20	97.44	97.80	90.55	90.55	NA	87.56
Winnebago_5	171.93	171.93	136.07	136.07	123.90	123.90	118.45	118.52	NA	117.54
Champaign_2	151.00	151.00	88.78	88.78	73.28	73.28	NA	64.28	NA	59.29
Champaign_6	215.42	215.97	170.92	171.49	151.49	154.14	142.03	146.03	NA	140.84
Champaign_8	200.03	200.03	166.07	166.07	158.84	158.84	156.20	156.20	NA	154.24
LaSalle_6	177.11*	177.57	130.59*	130.83	116.08	116.96	NA	108.20	NA	104.59
LaSalle_7	180.06	180.06	130.59	130.59	117.80	117.80	109.98	110.02	NA	107.09
Adams_2	204.36	204.94	158.70	158.70	141.71	141.71	136.95	137.11	NA	131.10
Adams_6	179.48	179.48	138.67	138.67	129.76	129.76	123.89	123.89	NA	121.42
Jefferson_9	184.13	184.24	135.61*	135.67	116.50	116.56	NA	111.35	NA	102.17
Jefferson_10	186.30	186.82	137.61	140.78	120.73	122.40	NA	116.58	NA	113.30
Johnson_2	196.77*	197.82	147.51*	147.51	130.28	130.28	123.59	123.59	NA	121.05
Johnson_7	191.22	191.22	143.01	143.01	127.59	127.59	120.91	120.91	NA	115.71
Johnson_8	285.90	285.90	240.60	240.60	229.76	229.76	224.89	225.23	NA	224.21
Cumberland_1	194.84	194.84	155.76	155.76	145.92	146.55	137.07	137.07	NA	136.07

Table 6 Objective Values of the Solutions of IP1 and BP on Instance-Set 2.

	Overall	$q = 1$	$q = 2$	$q = 3$	$q = 4$	$q = 5$
Max	2.82	0.54	2.30	1.75	2.82	NA
Avg	0.17	0.08	0.15	0.24	0.23	NA
Min	0.00	0.00	0.00	0.00	0.00	NA

Table 7 Optimality Gaps (%) of BP in Instance-Set 2.

5.2.3. Solution Quality of BP: We discuss the quality of the solutions obtained by BP here. Table 6 shows the objective values of the solutions of IP1 and BP on instance-set 2. For the instances solved by IP1, we calculate the optimality gap as $Z_{BP}/Z_{IP1} - 1$, where Z_{BP} and Z_{IP1} are the objective values of BP and IP1, respectively. The summary results (the maximum, average, and minimum percentages) are in Table 7. There are 76 instances solved optimally by IP1. The average, maximum, and minimum gaps are 0.17%, 2.80%, and 0.00%, respectively. Furthermore, BP finds optimal solutions for 53 out of the 76 instances. The optimality gaps of BP on instance-set 2 are very similar to those on instance-set 1. Even though we do not have the optimal values for 24 instances, we believe that the solutions obtained by BP are of high quality.

While RCT2 is valid for instance-set1, we show in Section 2 that RCT2 is not valid generally because it implicitly makes the assumption that at each location i in $\mathcal{C}$, the delivery person can deliver to at most two walking paths starting and ending at node i , i.e., $\sum_{k \in \mathcal{M}} y_{i,i,k} \leq 2$. In our experiments, five instances are affected by this limitation. These five instances are indicated by “*” markers in Table 6. Based on this set of instances, only allowing at most two walking paths that start and end at a node can lead to suboptimal solutions, while it may not impact most instances.

In short, when driving might be slower than walking, IP1 provides satisfactory performance with smaller capacities ($q = 1, 2, 3$) but is not a viable option with larger capacities ($q = 4, 5$). On the other hand, BP provides robust performance across all capacities and especially excels with larger capacities.

5.3. A Closer Look at the Branch-and-Price Approach

Across both instance sets, BP has an average run time of 149.5 seconds and an average optimality gap of 0.22% based on the 426 instances in which we can obtain optimal solutions. Furthermore, it finds optimal solutions for 281 out of the 426 instances. In this section, we provide more information regarding BP.

	Both Instance Sets	Set 1	Set 2
Overall	14.21	12.31	20.87
$q = 1$	8.48	5.73	18.07
$q = 2$	14.85	13.02	21.28
$q = 3$	15.80	14.11	21.71
$q = 4$	16.01	14.42	21.60
$q = 5$	15.92	14.27	21.70

Table 8 Average Optimality Gaps (%) of BP Based on the Lower Bounds Obtained by BP.

5.3.1. Optimality Gaps of BP: Given that we are able to obtain optimal solutions for most of the instances, we present the optimality gaps of BP earlier based on the optimal solutions. However, another way to assess the solution quality is to compare the best feasible solution to the lower bound obtained by BP itself. Table 8 shows the average optimality gaps based on the lower bounds obtained by BP. Overall, on average, the optimality gap is 14.21%. It is smaller (12.31%) for instance-set 1 but larger (20.87%) for instance-set 2. Another observation is that the average gap is significantly smaller when $q = 1$, but it is about the same for instances with other capacities. Even though we know that BP finds solutions of high quality, it might still be interesting to improve the lower bound.

	Both Instance Sets	Set 1	Set 2
Overall	49.84	53.12	38.34
$q = 1$	37.22	36.52	39.69
$q = 2$	44.80	46.39	39.22
$q = 3$	49.24	53.31	34.98
$q = 4$	58.91	64.06	40.88
$q = 5$	59.01	65.32	36.92

Table 9 Average Percentages (%) of Pricing Time in the Run Time.

5.3.2. Efficiency of the Pricing Procedure: It is well-known that the pricing procedure can be time-consuming for any approach based on the column-generation idea. Table 9 shows the average percentages of the pricing time in the run time. Overall, on average, the pricing procedure takes 49.84% of the run time. It is higher (53.12%) for instance-set 1 but lower (38.34%) for instance-set 2. Another observation is that the pricing time takes a larger portion as the capacity increases. However, the behavior is different among the two instance sets. When the capacity increases, the portion of the pricing time increases for instance-set 1 but does not have a clear pattern for instance-set 2. We note that our implementation is in Python, which is not famous for its efficiency. It is possible to switch to a lower-level language if one wishes to reduce the time spent on the pricing procedure.

	Run Time			y Variables		
	Both Instance Sets	Set 1	Set 2	Both Instance Sets	Set 1	Set 2
Overall	32.41	29.78	41.63	51.06	49.24	57.43
$q = 1$	37.44	36.66	40.18	7.92	0.00	35.64
$q = 2$	33.20	32.75	34.76	39.43	34.75	55.79
$q = 3$	30.11	27.41	39.58	58.71	58.43	59.71
$q = 4$	28.88	25.03	42.39	71.79	73.34	66.35
$q = 5$	32.42	27.04	51.24	77.45	79.67	69.67

Table 10 Average Percentages (%) of Run Time and y Variables Due to Path Enumeration.

5.3.3. Impact of Path Enumeration: Another important component in the BP setting is path enumeration. Table 10 shows the average percentages of run times and y variables due to path enumeration. Path enumeration takes about one-third of the run time, and it introduces a significant port of y variables (51.06%), on average. While the run time does not change much, the percentage of y variables increases as the capacity increases. We note that when $q = 1$, even though path enumeration does not introduce new y variables on instance-set 1, we still include the run time of path enumeration.

	# of Improvements	Avg	Max
Both Instance Sets	120	0.29%	4.39%
Set 1	78	0.08%	0.63%
Set 2	42	0.67%	4.39%

Table 11 Impact on Solution Quality Due to Path Enumeration.

Regarding the impact of path enumeration on solution quality, we provide summary results in Table 11. Out of the 450 instances, path enumeration improves the solutions of 120 instances: 78 are in instance-set 1, and 42 are in instance-set 2. The average improvement is 0.29%, and the maximum improvement is 4.39%. For the instances in instance-set 1, the improvement is smaller, with an average of 0.08%. However, path enumeration is more effective on the instances in instance-set 2. The average improvement is 0.67%, and 42 out of 100 instances benefit from it. Consequently, it seems that path enumeration does provide some positive impact. It leads to small yet important improvements in the solutions obtained by BP, especially for instance-set 2.

6. Conclusions

Recent years have seen exponential growth in package volumes, with last-mile delivery significantly contributing to logistics costs. Companies are now exploring autonomous assisted delivery to reduce expenses while meeting service and delivery commitments. This paper focuses on the Capacitated Autonomous Vehicle Assisted Delivery Problem. We introduce an IP formulation and a branch-and-price approach. Enhancements, valid inequalities, and preprocessing steps are meticulously designed by examining the structural properties of optimal solutions. Our computational experiments on real-world data instances highlight the impact of these components, showing that our approach produces optimal or near-optimal solutions for most cases in the literature. Our results provide further evidence in favor of autonomous vehicle assisted delivery.

As for future work, one natural direction is to improve the dual bound further, as we mentioned earlier. One could study the underlying polytope to derive more effective valid inequalities. This would involve exploring the structure of the problem and identifying additional constraints or relationships that can be incorporated into the model. By doing so, we can tighten the bounds

and potentially improve the efficiency of the solution algorithms. One could also investigate the branching rule because the current one does not improve the dual bound significantly in our experiments.

Another potential direction is to explore the development of efficient metaheuristics, which are usually preferred by companies in practice. Metaheuristic algorithms, such as genetic algorithms, simulated annealing, or particle swarm optimization, can provide good approximate solutions. By designing tailored metaheuristics specifically for the CAVADP, we can potentially achieve high-quality solutions in a reasonable amount of time.

Additionally, one could generalize the model to allow for multiple delivery persons and/or multiple autonomous vehicles. In the future, it is envisioned that a delivery person may be paired with multiple autonomous vehicles and vice versa, with the idea that a delivery person dropped off by one autonomous vehicle may then be picked up by another. This would enable more flexibility and efficiency in the delivery process. However, such a scenario introduces new challenges related to synchronization and coordination among multiple delivery persons and/or multiple autonomous vehicles. This generalization to multiple delivery persons and/or multiple autonomous vehicles presents an exciting avenue for further research. It not only expands the scope of the problem but also opens up opportunities to tackle the additional modeling and computational challenges that arise.

Acknowledgments

The author thanks the area editor Dr. David Alderson, the associate editor, and two referees for their insightful comments and suggestions. The associate editor has been extremely helpful. The author also thanks Mengting Chao and Chengyi Lyu for their useful feedback. Moreover, the author thanks Dr. Sara Reed for sharing detailed results from their paper.

References

- Allen, J., Piecyk, M., Piotrowska, M., McLeod, F., Cherrett, T., Ghali, K., Nguyen, T., Bektas, T., Bates, O., Friday, A., Wise, S., and Austwick, M. (2018). Understanding the impact of e-commerce on last-mile light goods vehicle activity in urban areas: The case of London. *Transportation Research Part D: Transport and Environment*, 61:325–338.
- 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*, 115:351–385.
- 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.
- Chiara, G. D., Cheah, L., Azevedo, C. L., and Ben-Akiva, M. E. (2020). A policy-sensitive model of parking choice for commercial vehicles in urban areas. *Transportation Science*, 54(3):606–630.
- Chung, S. H., Sah, B., and Lee, J. (2020). Optimization for drone and drone-truck combined operations: A review of the state of the art and future directions. *Computers & Operations Research*, 123:105004.
- Corredor-Montenegro, D., Cabrera, N., Akhavan-Tabatabaei, R., and Medaglia, A. L. (2021). On the shortest α -reliable path problem. *Top*, 29(1):287–318.

- Costa, L., Contardo, C., and Desaulniers, G. (2019). Exact branch-price-and-cut algorithms for vehicle routing. *Transportation Science*, 53(4):946–985.
- Dantzig, G., Fulkerson, R., and Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410.
- Deffenbaugh, R. and Eisenpress, C. (2021). Inside the city’s wild west of package deliveries. *Crain’s New York Business*.
- Desaulniers, G., Desrosiers, J., and Solomon, M. M. (2006). *Column Generation*, volume 5. Springer Science & Business Media, Berlin/Heidelberg, Germany.
- Dror, M. (1994). Note on the complexity of the shortest path models for column generation in VRPTW. *Operations Research*, 42(5):977–978.
- Florio, A. M., Hartl, R. F., and Minner, S. (2020). New exact algorithm for the vehicle routing problem with stochastic demands. *Transportation Science*, 54(4):1073–1090.
- Gavish, B. and Graves, S. C. (1978). The travelling salesman problem and related problems. Working Paper, Massachusetts Institute of Technology, Operations Research Center.
- Google, Inc (2024). Google Maps. maps.google.com. Last access date: 15-March-2024.
- Haag, M. and Hu, W. (2019). 1.5 million packages a day: The Internet brings chaos to NY streets. *The New York Times*, 28.
- INRIX Research (2017). Searching for parking costs Americans \$73 billion a year. <https://inrix.com/press-releases/parking-pain-us/>. Last access date: 15-May-2023.
- Irnich, S. and Desaulniers, G. (2005). Shortest path problems with resource constraints. In *Column generation*, pages 33–65. Springer.
- Lozano, L., Duque, D., and Medaglia, A. L. (2016). An exact algorithm for the elementary shortest path problem with resource constraints. *Transportation Science*, 50(1):348–357.
- Lozano, L. and Smith, J. C. (2017). A backward sampling framework for interdiction problems with fortification. *INFORMS Journal on Computing*, 29(1):123–139.
- Miller, C. E., Tucker, A. W., and Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329.
- Mims, C. (2019). Why your ice cream will ride in a self-driving car before you do. *Wall Street Journal (Online)*.
- Otto, A., Agatz, N., Campbell, J., Golden, B., and Pesch, E. (2018). Optimization approaches for civil applications of unmanned aerial vehicles (UAVs) or aerial drones: A survey. *Networks*, 72(4):411–458.
- Pessoa, A., Sadykov, R., Uchoa, E., and Vanderbeck, F. (2020). A generic exact solver for vehicle routing and related problems. *Mathematical Programming*, 183:483–523.
- Poikonen, S. and Golden, B. (2020a). The mothership and drone routing problem. *INFORMS Journal on Computing*, 32(2):249–262.
- Poikonen, S. and Golden, B. (2020b). Multi-visit drone routing problem. *Computers & Operations Research*, 113:104802.
- Pugliese, L. D. P. and Guerriero, F. (2013). A survey of resource constrained shortest path problems: Exact solution approaches. *Networks*, 62(3):183–200.
- Reed, S., Campbell, A. M., and Thomas, B. W. (2022a). Impact of autonomous vehicle assisted last-mile delivery in urban to rural settings. *Transportation Science*, 56(6):1530–1548.
- Reed, S., Campbell, A. M., and Thomas, B. W. (2022b). The value of autonomous vehicles for last-mile deliveries in urban environments. *Management Science*, 68(1):280–299.
- Romo, V. and Domonoske, C. (2019). US postal service tests self-driving trucks. *National Public Radio*.
- Rosenberger, S. (2022). Last mile delivery landscape in the transportation sector. <https://www2.deloitte.com/global/en/pages/consumer-business/articles/last-mile-delivery-landscape-transportation-sector.html>. Last access date: 16-Feb-2023.

-
- Rostami, B., Desaulniers, G., Errico, F., and Lodi, A. (2021). Branch-price-and-cut algorithms for the vehicle routing problem with stochastic and correlated travel times. *Operations Research*, 69(2):436–455.
- Schrotenboer, A. H., Ursavas, E., and Vis, I. F. (2019). A branch-and-price-and-cut algorithm for resource-constrained pickup and delivery problems. *Transportation Science*, 53(4):1001–1022.
- Simon, S. (2022). New York City is drowning in packages. *MIT Technology Review*.
- United Nations (2021). Global e-commerce jumps to \$26.7 trillion, fuelled by COVID-19. <https://news.un.org/en/story/2021/05/1091182>. Last access date: 16-Feb-2023.
- U.S. Census Bureau (2022). Latest quarterly e-commerce report. <https://www.census.gov/retail/index.html>. Last access date: 16-Feb-2023.
- U.S. Postal Service (2017). Autonomous vehicles for the postal service. Technical report, Office of Inspector General U.S. Postal Service, Washington, DC. <https://www.uspsaig.gov/sites/default/files/reports/2023-01/RARC-WP-18-001.pdf/>. Last access date: 15-May-2023.
- Yamín, D., Medaglia, A. L., and Prakash, A. A. (2022). Exact bidirectional algorithm for the least expected travel-time path problem on stochastic and time-dependent networks. *Computers & Operations Research*, 141:105671.
- Zhang, R. (2024). Instances and Code in Branch-and-Price for the Capacitated Autonomous Vehicle Assisted Delivery Problem. Available for download at <https://github.com/INFORMSJoC/2023.0177>.

Electronic Companion

EC.1. The All-Pair Pulse Algorithm

We first introduce a metric called semi-reduced cost, which is calculated as $\hat{w}_{i,j,k} - \sum_{l \in \sigma_k} (\alpha_l^* + \delta_l^*) - \mathbb{1}(i=j)\beta_i^* - \mathbb{1}(i \neq j)\gamma_{i,j}^*$ for a path (i, j, k) . It is later used in the bounds pruning strategy.

Algorithm 1 The All-Pair Pulse Algorithm

```

1: Input:  $G$  graph, driving time  $T$ , walking time  $W$ , capacity  $q$ , loading time  $f$ , and  $(\alpha^*, \beta^*, \gamma^*, \delta^*)$ .
2: for all sink in  $\mathcal{C}$  do
3:   for all source_served_flag in  $\{\text{True}, \text{False}\}$  do
4:     Pulse_with_Fixed_Sink(sink, source_served_flag, sink_served_flag = False)
5:   end for
6: end for
```

Algorithm 1 shows the pseudocode for the all-pair pulse algorithm. We adopt the convention terminology, *source* and *sink*, from the shortest path literature. Each node in $\mathcal{C}$ is considered as the sink. Then, for each sink, we invoke the Pulse_with_Fixed_Sink() for two cases: one serves the source, and the other does not. Note that we set `sink_served_flag` as False and allow the sink not to be served. There are four cases in total by changing the values of `source_served_flag` and `sink_served_flag`. However, our algorithm design only needs to run two cases rather than four because we let the algorithm decide whether the sink should be served.

Algorithm 2 Initialize_Path

```

1: Initialize source, initial_capacity, load,  $\hat{w}$ ,  $\pi$  as 0s,  $r$  and  $sr$  as  $\infty$  unless stated otherwise
2: Initialize walking_path as a list containing source
3: return a path object: p
```

Before presenting the function Pulse_with_Fixed_Sink(), we first introduce the function Initialize_Path(), which is used to create a path object called `p`. This object contains the following attributes. `source`: the source of the path. `initial_capacity`: the initial capacity of the path. `load`: the load so far. `walking_path`: the walking path. $\hat{w}$: the walking time of the walking path. π : the dual cost associated with the walking path. r : the reduced cost of the path. sr : the semi-reduced cost of the path. Except for `walking_path`, r , and sr , all other attributes are initialized as 0s unless specified otherwise. `walking_path` is initialized as a list with the `source`, sr , and r is initialized as ∞ .

In the Pulse_with_Fixed_Sink() function, we use `primal_bounds[i, sink]` to store the path with the smallest reduced cost from node i to the `sink`. The pulse algorithm in Lozano et al. (2016) creates buckets by a parameter known as step size. In our pricing problem, the discrete value of the

Algorithm 3 Pulse_with_Fixed_Sink

```

1: Input: sink, source_served_flag, sink_served_flag
2: for all  $i$  in  $\mathcal{C}$  do:
3:   primal_bounds[ $i$ , sink]  $\leftarrow$  Initialize_Path()
4:   for all cap in  $\{0, 1, \dots, q\}$  do:
5:     semiCost_lbs[ $i$ , cap, sink]  $\leftarrow$  Initialize_Path()
6:     dualCost_lbs[ $i$ , cap, sink]  $\leftarrow$  Initialize_Path()
7:   end for
8: end for
9: if source_served_flag = True then
10:   Set max_remaining_capacity  $\leftarrow q - 1$ 
11: else
12:   Set max_remaining_capacity  $\leftarrow q$ 
13: end if
14: for all cap in  $\{0, 1, \dots, \text{max\_remaining\_capacity}\}$  do
15:   for all  $i$  in  $\mathcal{C}$  do
16:     Create a path object:  $p \leftarrow$  Initialize_Path(source =  $i$ , initial_capacity = cap)
17:     Set  $p.\text{load} \leftarrow q - \text{cap}$ 
18:     if source_served_flag = True then
19:       Set  $p.\pi \leftarrow \alpha_i^* + \delta_i^*$ 
20:     end if
21:     if  $i = \text{sink}$  then
22:       Set  $p.\pi \leftarrow p.\pi + \beta_i^*$ 
23:     else
24:       Set  $p.\pi \leftarrow p.\pi + \gamma_{i,\text{sink}}^*$ 
25:     end if
26:     Pulse_Procedure( $p$ )
27:   end for
28: end for

```

capacity allows us to create buckets, called `semiCost_lbs` and `dualCost_lbs`, naturally (lines 2–8). Thus, `semiCost_lbs[ $i$ ,  $k$ , sink]` and `dualCost_lbs[ $i$ ,  $k$ , sink]` are used to store the path with the smallest semi-reduced cost and the path with the largest dual cost from node i to the sink with capacity k , respectively. After that, we set up the `max_remaining_capacity`, depending on whether the source is served (lines 9–13). Finally, we start the pulse algorithm for each capacity level from each node in $\mathcal{C}$ with the corresponding load and dual cost (π) (lines 14–28).

In the `Pulse_Procedure()` function, we first check whether the path reaches the sink. If so, we calculate the path's reduced cost and update the corresponding `primal_bounds`, `semiCost_lbs`, and `dualCost_lbs` if needed (lines 3–15). Then, following Lozano et al. (2016), we have three pruning strategies. The `Is_Bounded()` function is for bounds pruning. The `Should_Rollback()` function is for rollback pruning. Line 24 in Algorithm 4 is for infeasibility pruning, which ensures the elementarity of the path. If the load is smaller than the full capacity, we extend the path to a node adjacent to the path's last node while ensuring that the path remains an elementary one. Then, we update the relevant attributes, including `walking_path`, `load`, $\hat{w}$, and π , and call the `Pulse_Procedure()` function again (lines 22–29). If the load is equal to the full capacity, we could still walk back to the

Algorithm 4 Pulse.Procedure

```

1: Input: a current path  $p$ 
2: Set  $i$  as the first node of  $p.walking\_path$ , Set  $j$  as the last node of  $p.walking\_path$ 
3: if  $j = \text{sink}$  then
4:    $p.r = f + \max\{t_{i,j}, p.\hat{w}\} - p.\pi$ , and  $p.sr = p.\hat{w} - p.\pi$ 
5:   if  $p.r < \text{primal\_bounds}[i, \text{sink}].r$  then
6:     Set  $\text{primal\_bounds}[i, \text{sink}] \leftarrow p$ 
7:   end if
8:   if  $p.sr < \text{semiCost\_lbs}[i, p.\text{initial\_capacity}, \text{sink}].sr$  then
9:     Set  $\text{semiCost\_lbs}[i, p.\text{initial\_capacity}, \text{sink}] \leftarrow p$ 
10:  end if
11:  if  $p.\pi > \text{dualCost\_lbs}[i, p.\text{initial\_capacity}, \text{sink}].\pi$  then
12:    Set  $\text{dualCost\_lbs}[i, p.\text{initial\_capacity}, \text{sink}] \leftarrow p$ 
13:  end if
14:  return
15: end if
16: if  $\text{Is\_Bounded}(p)$  then
17:   return
18: end if
19: if  $\text{Should\_Rollback}(p)$  then
20:   return
21: end if
22: if  $p.\text{load} < q$  then
23:   for all each node  $j'$  in  $n(j)^+$  do
24:     if node  $j'$  is not visited in  $p.walking\_path$  then
25:       Append  $j'$  to  $p.walking\_path$ 
26:       Set  $p.\text{load} \leftarrow p.\text{load} + 1$ ,  $p.\hat{w} \leftarrow p.\hat{w} + w_{j,j'}$ ,  $p.\pi \leftarrow p.\pi + \alpha_{j'}^* + \delta_i^*$ 
27:       Call Pulse.Procedure( $p$ )
28:     end if
29:   end for
30: else if  $\text{source} = \text{sink}$  or  $\text{sink\_served\_flag} = \text{False}$  then
31:   Append  $\text{sink}$  to  $p.walking\_path$  and set  $p.\hat{w} \leftarrow p.\hat{w} + w_{j,\text{sink}}$ 
32:   Call Pulse.Procedure( $p$ )
33: end if

```

sink when **source** is equal to **sink** or the sink is not necessarily to be served (**sink_served_flag** is False). After that, we also need to update the relevant attributes and call the Pulse.Procedure() function (lines 30–33).

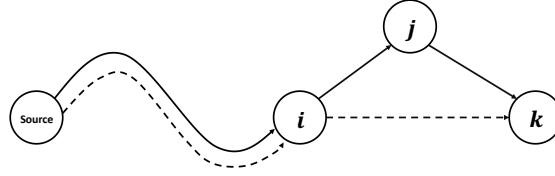
Both the Is.Bounded() and Should.Rollback() functions follow the idea in Lozano et al. (2016). For the bounds pruning, we maintain a primal bound from the source to the sink and lower bounds from a node to the sink with a capacity level. Lozano et al. (2016) use reduced costs for lower bounds. However, a crucial difference is that we cannot use that because of the nonlinear objective function in our pricing problem. Instead of reduced costs, we maintain two lower bounds: the semi-reduced cost and the dual cost from a node to the sink with a capacity level. All of the primal and lower bounds are constantly updated with the value of the best one found during the exploration. Before pruning, we calculate the **offset** to avoid double counting the dual cost (lines 4–7). The bounds pruning strategy uses primal and lower bounds to determine whether the partial path is

Algorithm 5 Is_Bounded

```

1: Input: a current path  $p$ 
2: Set  $i$  and  $j$  as the first and last node of  $p.walking\_path$ , respectively. Set  $cap \leftarrow q - p.load$ 
3: if  $semiCost\_lbs[j, cap, sink].r < \infty$  then
4:   Set  $offset \leftarrow \gamma_{j,sink}^* + semiCost\_lbs[j, cap, sink].load \times (\delta_j - \delta_i)$ 
5:   if  $source\_served\_flag = \text{True}$  then
6:     Set  $offset \leftarrow offset + \alpha_j^*$ 
7:   end if
8:   if  $p.\hat{w} + semiCost\_lbs[j, cap, sink].\hat{w} \geq t_{i,sink}$  then
9:     return  $f + p.\hat{w} + semiCost\_lbs[j, cap, sink].\hat{w}$ 
10:     $-(p.\pi + semiCost\_lbs[j, cap, sink].\pi - offset) > primal\_bounds[i, sink].r$ 
11:   else
12:     return  $f + t_{i,sink}$ 
13:     $-(p.\pi + dualCost\_lbs[j, cap, sink].\pi - offset) > primal\_bounds[i, sink].r$ 
14:   end if
15: end if

```

**Figure EC.1** Rollback Pruning Illustration (Adapted from Lozano et al. (2016)).

promising (lines 8 – 15 in Algorithm 5). The correctness of the bounds pruning strategy is shown in the following proposition.

PROPOSITION EC.1. *The bounds pruning strategy in Algorithm 5 is valid.*

Algorithm 6 Should_Rollback

```

1: Input: a current path  $p$ 
2: Extract the last three nodes from  $p.walking\_path$  as  $i, j$ , and  $k$ , respectively
3: return  $w_{i,k} - \alpha_k^* \leq w_{i,j} + w_{j,k} - \alpha_j^* - \alpha_k^* - \delta_{p.source}$ 

```

For the rollback pruning, this pruning strategy seeks to validate the most recent decision made, and if needed, it rolls back this decision to avoid further exploration of suboptimal paths. For the last three nodes, denoted as i, j , and k , of the walking path, as shown in Figure EC.1, we compare the reduced costs of two possibilities: one with walking from node i to node k (the dashed one in Figure EC.1) and the other one with walking from node i to node j , then from node j to node k (the solid one in Figure EC.1). If the reduced cost of the former one is not worse than that of the latter one, we roll back (line 3 in Algorithm 6).

EC.2. The Branch-and-Price Approach for the CAVADP

Algorithm 7 The Branch-and-Price Approach for the CAVADP

```

1: Initialize UB (upper bound) as  $\infty$  and LB (lower bound) as  $-\infty$  and NodeCount as 0.
2: if Driving is always faster than walking then
3:   Initialize (IP)  $\leftarrow$  (IP2R) and (LP)  $\leftarrow$  (LP2R) with initial paths ( $y$  variables)
4: else
5:   Initialize (IP)  $\leftarrow$  (IP1R) and (LP)  $\leftarrow$  (LP1R) with initial paths ( $y$  variables)
6: end if
7: Let RunningTime track the running time, and TimeLimit, NodeLimit, and ST (stopping tolerance) be
   user-specified input parameters
8: Initialize a list nextLevel with the root node
9: while (UB - LB)/UB > ST and NodeCount < NodeLimit and RunningTime  $\leq$  TimeLimit do
10:   Set currentLevel  $\leftarrow$  nextLevel
11:   for each node in currentLevel do
12:     Set foundPathFlag  $\leftarrow$  1
13:     while foundPathFlag = 1 do
14:       Solve (LP) at the node to obtain a solution  $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$  and set foundPathFlag  $\leftarrow$  0
15:       Run the Column Generation procedure based on the All-Pair Pulse Algorithm
16:       if promising  $y$  variables are found then
17:         Include newly found  $y$  variables to (LP) and (IP) and set foundPathFlag  $\leftarrow$  1
18:       end if
19:     end while
20:     if Current node is the root node then
21:       Run the Path Enumeration procedure
22:     end if
23:     Apply branching rule to create two nodes and include the two nodes in list nextLevel
24:   end for
25:   Update LB, UB, and NodeCount accordingly
26:   Set currentLevel  $\leftarrow$  nextLevel
27: end while

```

EC.3. Proofs

EC.3.1. Proof for Proposition EC.1

Proof: We want to see if the current partial path p is promising. The basic idea is completing the current partial path p (referred to as the 1st-half path) by using a path that starts from node j to the sink (referred to as the 2nd-half path). Then, we compare the completed path's reduced cost to the current primal bound.

To do that, we need a way to evaluate the completed path's reduced cost. Given the way of calculating reduced cost (5), the walking time and the associated dual cost of the 2nd-half path are relevant, which is calculated as the semi-reduced cost. We store the 2nd-half path with the smallest semi-reduced cost so far in `semiCost_lbs[j, cap, sink]`. However, we cannot enforce the elementary requirement in this way. Thus, we can only obtain a lower bound on the potential reduced cost by relaxing the elementary requirement.

On the one hand, when the walking time of the delivery person of the resulting path is not smaller than the driving time from node i to sink, $(\mathbf{p}.\hat{w} + \text{semiCost_lbs}[j, \text{cap}, \text{sink}]).\hat{w} \geq$

$t_{i,sink}$), the cost function becomes linear in this case after combining the 1st-half and the 2nd-half paths. Therefore, lines 9-10 in Algorithm 5 provide a lower bound of the reduced cost of the true elementary path.

On the other hand, when the walking time of the delivery person of the resulting path is smaller than the driving time from node i to sink, $(p.\hat{w} + \text{semiCost_lbs}[j, \text{cap}, \text{sink}]).\hat{w} < t_{i,sink}$, the semi-reduced cost of the 2nd-half path is not valid anymore. However, we can use the largest dual cost instead. Lines 12-13 in Algorithm 5 use $t_{i,sink}$ and the current largest dual cost stored in $\text{dualCost_lbs}[j, \text{cap}, \text{sink}]$ to obtain a lower bound of the reduced cost of the true elementary path. $\square$

EC.3.2. Proof for Proposition 1

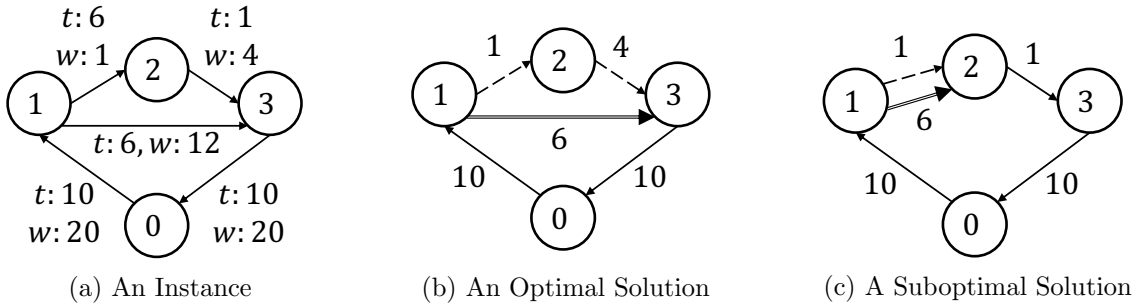


Figure EC.2 The Counter-Example in Proposition 1.

Proof: First, we show that we need driving not to be slower than walking to construct the restricted walking graph. This is demonstrated by a counter-example.

In this example, we show that it is possible to have the delivery person walk on an arc (i, j) with $t_{i,j} + f \leq w_{i,j}$ when driving is slower than walking. The counter-example is in Figure EC.2. The instance is in Figure EC.2(a) with the driving time (t) and walking time (w) beside each arc. Given $f = 2.8$, the arc $(2, 3)$ is the one with $t_{2,3} + f \leq w_{2,3}$. With $q = 3$, Figure EC.2(b) shows an optimal solution where the delivery person walks on arc $(2, 3)$. The vehicle and the delivery person travel together to node 1. Then, the delivery person loads at node 1 and serves nodes 1, 2, and 3. The delivery person and the vehicle meet at node 3 and travel together back to node 0. Its objective value is 28.8.

On the other hand, if the delivery person is not allowed to walk on arc $(2, 3)$, a suboptimal solution is shown in Figure EC.2(c). In this solution, after traveling to node 1, the delivery person loads at node 1 and serves nodes 1 and 2. The delivery person and the vehicle meet at node 2 and travel together to node 3. Then, the delivery person loads and serves node 3 before returning to node 0. The objective value is 32.6.

Next, when driving is not slower than walking, the $c_{i,j,k}$ value of a path (i, j, k) is decided only by the walking time of the delivery person. Given a solution, if a delivery person walks on an arc (i, j) with $t_{i,j} + f \leq w_{i,j}$, a solution can be constructed with a better or not-worse objective value such that the delivery person does not walk on this arc. Let arc (i, j) be on a path starting at node i' and ending at j' . The modified solution has the vehicle meet with the delivery person at node i . Thus, the vehicle travels from nodes i' to i . After picking up the delivery person, they travel together to node j . Then, the delivery person loads at node j and continues serving to node j' . The vehicle moves from nodes j to j' . The remaining part of the original solution stays the same.

□

EC.3.3. Proof for Proposition 4

Proof: First, given $\mathcal{S} \subseteq \mathcal{C}$ and $k \in \mathcal{S}$, we add $|\mathcal{S}| - 1$ to both sides of $\sum_{i,j \in \mathcal{S}: i \neq j} (x_{i,j} + z_{i,j}) - \sum_{i \in \mathcal{S} \setminus \{k\}} (\sum_{j \in \mathcal{V}: i \neq j} x_{j,i} + \sum_{j \in \mathcal{C}: i \neq j} z_{j,i}) \leq 0$ to obtain $\sum_{i,j \in \mathcal{S}: i \neq j} (x_{i,j} + z_{i,j}) + \sum_{i \in \mathcal{S} \setminus \{k\}} (1 - (\sum_{j \in \mathcal{V}: i \neq j} x_{j,i} + \sum_{j \in \mathcal{C}: i \neq j} z_{j,i})) \leq |\mathcal{S}| - 1$.

For a feasible solution of the CAVADP, there should not be any cycles formed by the nodes in $\mathcal{C}$ that are visited by the vehicle. If node i is visited by the vehicle, we have $\sum_{j \in \mathcal{V}: i \neq j} x_{j,i} + \sum_{j \in \mathcal{C}: i \neq j} z_{j,i} = 1$ and $(\sum_{j \in \mathcal{V}: i \neq j} x_{j,i} + \sum_{j \in \mathcal{C}: i \neq j} z_{j,i}) = (\sum_{j \in \mathcal{V}: i \neq j} x_{i,j} + \sum_{j \in \mathcal{C}: i \neq j} z_{i,j}) = 1$ because of constraints (3e). Thus, node i should be included to prevent a subtour. If a node i is not visited by the vehicle, we have this node included in a path by the corresponding y variable. Thus, we have $\sum_{j \in \mathcal{V}: i \neq j} x_{j,i} + \sum_{j \in \mathcal{C}: i \neq j} z_{j,i} = 0$ and $(\sum_{j \in \mathcal{V}: i \neq j} x_{j,i} + \sum_{j \in \mathcal{C}: i \neq j} z_{j,i}) = (\sum_{j \in \mathcal{V}: i \neq j} x_{i,j} + \sum_{j \in \mathcal{C}: i \neq j} z_{i,j}) = 0$ because of constraints (3e). Then, we should not worry about node i being part of a subtour. We can reduce the right-hand side by 1 as $(\sum_{j \in \mathcal{V}: i \neq j} x_{j,i} + \sum_{j \in \mathcal{C}: i \neq j} z_{j,i}) = (\sum_{j \in \mathcal{V}: i \neq j} x_{i,j} + \sum_{j \in \mathcal{C}: i \neq j} z_{i,j}) = 0$.

Effectively, we only focus on the nodes in $\mathcal{S}$ that are visited by the vehicle. Given that node k is removed from $\mathcal{S}$, the largest value of $\sum_{i \in \mathcal{S} \setminus \{k\}} (1 - (\sum_{j \in \mathcal{V}: i \neq j} x_{j,i} + \sum_{j \in \mathcal{C}: i \neq j} z_{j,i}))$ is $|\mathcal{S}| - 1$, which does not exceed the right-hand side value. Overall, it ensures that for a subset of nodes in $\mathcal{C}$, the graph induced by the x and z variables does not have any cycles. Thus, any feasible solution of the CAVADP satisfies $\sum_{i,j \in \mathcal{S}: i \neq j} (x_{i,j} + z_{i,j}) - \sum_{i \in \mathcal{S} \setminus \{k\}} (\sum_{j \in \mathcal{V}: i \neq j} x_{j,i} + \sum_{j \in \mathcal{C}: i \neq j} z_{j,i}) \leq 0, \forall \mathcal{S} \subseteq \mathcal{C}, |\mathcal{S}| \geq 2, k \in \mathcal{S}$.

Next, notice that combining constraint (3d) and subtour elimination constraints gives $\sum_{j \in \mathcal{V}: i \neq j} x_{j,i} + \sum_{j \in \mathcal{C}: i \neq j} z_{j,i} = 1 - \sum_{(i,j,k) \in \mathcal{R}_i} y_{i,j,k} \leq 1$, as node i is included in at most one path, $\sum_{p \in \mathcal{R}_i} y_p \leq 1$. Thus, we have $\sum_{i,j \in \mathcal{S}: i \neq j} (x_{i,j} + z_{i,j}) \leq \sum_{i,j \in \mathcal{S}: i \neq j} (x_{i,j} + z_{i,j}) + \sum_{i \in \mathcal{S} \setminus \{k\}} (1 - (\sum_{j \in \mathcal{V}: i \neq j} x_{j,i} + \sum_{j \in \mathcal{C}: i \neq j} z_{j,i})) \leq |\mathcal{S}| - 1$ as $1 - (\sum_{j \in \mathcal{V}: i \neq j} x_{j,i} + \sum_{j \in \mathcal{C}: i \neq j} z_{j,i}) \geq 0$. It shows that $\sum_{i,j \in \mathcal{S}: i \neq j} (x_{i,j} + z_{i,j}) - \sum_{i \in \mathcal{S} \setminus \{k\}} (\sum_{j \in \mathcal{V}: i \neq j} x_{j,i} + \sum_{j \in \mathcal{C}: i \neq j} z_{j,i}) \leq 0, \forall \mathcal{S} \subseteq \mathcal{C}, |\mathcal{S}| \geq 2, k \in \mathcal{S}$ dominates $\sum_{i,j \in \mathcal{S}: i \neq j} x_{i,j} \leq |\mathcal{S}| - 1, \forall \mathcal{S} \subseteq \mathcal{C}, |\mathcal{S}| \geq 2$. □

EC.3.4. Proof for Proposition 5

Proof: We know that $\sum_{(l,m,k) \in \mathcal{R}_i \cap \mathcal{R}_j} y_{l,m,k} \leq 1$ because of constraint (3d). When $\sum_{(l,m,k) \in \mathcal{R}_i \cap \mathcal{R}_j} y_{l,m,k} = 0$, we have $x_{i,j} + x_{j,i} \leq 1$ because of the subtour elimination constraints.

When $\sum_{(l,m,k) \in \mathcal{R}_i \cap \mathcal{R}_j} y_{l,m,k} = 1$, we want to show that $x_{i,j} + x_{j,i} = 0$. There are two cases. First, if nodes i and j are the start and end nodes (or the end and start nodes) of path (i, j, k) , respectively, we should have $z_{i,j} = 1$ (or $z_{j,i} = 1$). Given the subtour elimination constraints, we have $x_{i,j} + x_{j,i} = 0$.

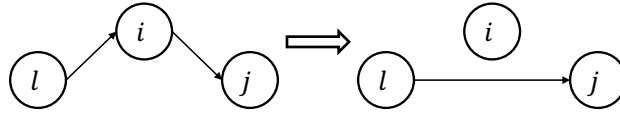


Figure EC.3 Illustration of Solution Modification When Nodes i and j Are Not the Start and End Nodes and $x_{l,i} = 1$ in the Poof of Proposition 5.

Second, nodes i and j are served in a path p , but they are not the start and end nodes. In this case, without loss of generality, assume that $x_{i,j} = 1$. Given flow balance constraint (3e), we have $x_{l,i} + z_{l,i} = 1$ for some node l . When $x_{l,i} = 1$, we can modify the solution by setting $x_{l,j} = 1$ and $x_{i,j} = x_{l,i} = 0$, as illustrated in Figure EC.3. The objective value is not worse because the driving time satisfies the triangular inequality.

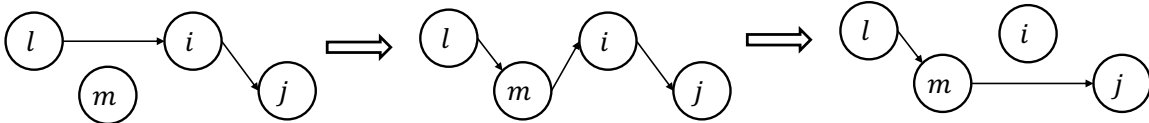


Figure EC.4 Illustration of Solution Modification When Nodes i and j Are Not the Start and End Nodes and $z_{l,i} = 1$ in the Poof of Proposition 5.

When $z_{l,i} = 1$, we know that the corresponding path p' starting from node l and ending at node i does not serve node i because node i is served in path p as given and constraint (3d). Let node m be the node visited on the path p' before node i . We can modify the solution in the following way: Let $z_{l,m} = 1$ and $x_{m,i} = 1$, as illustrated in the middle of Figure EC.3. This modified solution has a not-worse objective value because driving is not slower than walking. Furthermore, we can set $x_{m,j} = 1$ and $x_{m,i} = x_{i,j} = 0$, as illustrated on the right of Figure EC.3. This new solution has a not-worse objective value because the driving time satisfies the triangular inequality. This completes the proof. $\square$

EC.3.5. Proof for Proposition 6

Proof: We know that $\sum_{(i,j,k) \in \mathcal{O}_i} y_{i,j,k} \leq 1$ because of constraint (3d) and $\mathcal{O}_i \subset \mathcal{R}_i$. When $\sum_{(l,j,k) \in \mathcal{O}_i} y_{l,j,k} = 0$, we have $\sum_{j \in \mathcal{V}: i \neq j} x_{j,i} \leq 1$ because of the subtour elimination constraints. When $\sum_{(l,j,k) \in \mathcal{O}_i} y_{l,j,k} = 1$, we want to show that $\sum_{j \in \mathcal{V}: i \neq j} x_{j,i} = 0$. We assume that it is not true. Due to the subtour elimination constraints, $x_{j,i} = 1$ for exactly one node j . Then, we can use the argument in Proposition 5 to construct a not-worse solution with $\sum_{j \in \mathcal{V}: i \neq j} x_{j,i} = 0$. This completes the proof.

□