A Computational Study of Flexible Routing Strategies for the VRP with Stochastic Demands

by

Kirby Ledvina

S.B. Economics
S.B. Management Science
Massachusetts Institute of Technology, 2017

Submitted to the Department of Civil and Environmental Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Civil and Environmental Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2021

Author . . . .
Department of Civil and Environmental Engineering
January 12, 2021

Certified by . . . .
David Simchi-Levi
Professor of Civil and Environmental Engineering
Thesis Supervisor

Accepted by . . . .
Colette L. Heald
Professor of Civil and Environmental Engineering
Chair, Graduate Program Committee

A Computational Study of Flexible Routing Strategies for the VRP with Stochastic Demands

by

Kirby Ledvina

Submitted to the Department of Civil and Environmental Engineering
on January 12, 2021, in partial fulfillment of the
requirements for the degree of
Master of Science in Civil and Environmental Engineering

Abstract

We develop and numerically test a new strategy for the vehicle routing problem with stochastic customer demands. In our proposed approach, drivers are assigned to predetermined delivery routes in which adjacent routes share some customers. This overlapping assignment structure, which is inspired by the open chain design from the field of manufacturing process flexibility, enables drivers to adapt to variable customer demands while still maintaining largely consistent routes. Through an extensive computational study and scenario analysis, we show that relative to a system without customer sharing, such flexible routing strategies partly mitigate the transportation costs of filling unexpected customer demands, and the relative savings grow with the number of customers in the network. We also find that much of the cost savings is gained with just the first customer that is shared between adjacent routes. Thus, the overlapped routing model forms the basis for a practical and efficient strategy to manage costs from demand uncertainty.

Thesis Supervisor: David Simchi-Levi
Title: Professor of Civil and Environmental Engineering

Acknowledgments

I would foremost like to thank my advisor David for welcoming me into his research group and entrusting me with a new idea to integrate process flexibility and transportation. I am also deeply grateful to Dr. Yehua Wei and Hanzhang Qin, who were crucial collaborators in developing this new research direction. Thank you also to Blue Yonder for its early sponsorship of our work. Finally, I will forever appreciate the consistent encouragement, patience, and support I have received from my family and my friends within the MIT community.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Organizations operate amidst several economic, political, and environmental uncertainties. Thus, decision-makers in these organizations often adopt risk management strategies to navigate uncertainty when planning day-to-day operations and developing longer-term strategies. In the context of operations, one strategy to enable efficient responses to uncertainty is to incorporate *flexibility*, which Simchi-Levi (2010) defines as the ability to respond to change at minimal cost.

Flexibility in practice looks different across organizations and functions. Figure 1 illustrates three forms of operational flexibility in the context of manufacturing: system, process, and product design flexibility (Simchi-Levi 2010). System flexibility is achieved through coordinated manufacturing and distribution networks. For example, manufacturers can enable their factories to build multiple product types; then if there is unforeseen demand or a disruption in manufacturing for a particular product, production capacity elsewhere in the network can be utilized. Process flexibility refers to adaptable operations within individual product lines and network locations. For example, worker cross-training ensures workers have the skills to perform multiple tasks and fill staffing gaps as needed. Finally, product design flexibility ensures that new products are developed with responsive supply chains. For example, a modular product structure facilitates last-minute customization in response to consumer demands. The modular structure also allows for individual product parts to be reused or replaced. In all of these examples, organizations designed and installed flexible systems and procedures in advance to make their operations more responsive to change.

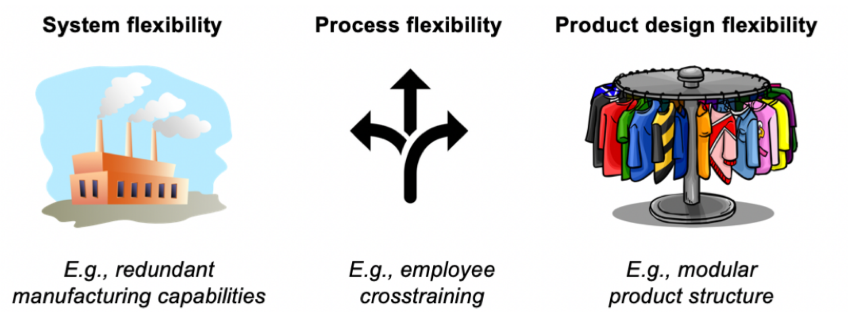


Figure 1.1: Types of operational flexibility

For this thesis, we propose a strategy for incorporating flexibility into transportation and last mile distribution. We explore a setting in which a transportation services provider (distributor) operates a vehicle fleet that makes daily deliveries of a single product type to retail customers. Each day, the distributor executes *a priori* delivery routes designed in advance around expected order quantities, vehicle capacity, and various logistical constraints. The consistency of these fixed routes allows the distributor to enter longer term and lower cost contracts with carrier companies (fleet owners), support driver familiarity with the route, and promote positive customer relationships (Bertsimas 1992, Erera et al. 2009). However, customers may unexpectedly update their orders, and it can be costly for the distributor to accommodate these changes. For example, the distributor may face procurement costs from acquiring last-minute vehicles and drivers for additional deliveries. There may also be customer service costs if drivers change from day to day or if some orders cannot be filled. Additionally, while the existing literature in operations research provides reoptimization approaches to update delivery routes based on new customer information, reassembling routes may not be logistically feasible in a short planning window.

In the setting we consider, if the actual quantities demanded by a route's customers exceed the vehicle's capacity, then the vehicle experiences a *route failure*, and the driver must return to the product depot to restock, increasing transportation costs. Performing a refill trip as described here is an example of a *recourse policy*, which describes how drivers should respond upon experiencing a route failure. Typically, under what we call a *dedicated routing* strategy, routes consist of exclusive sets of customer so that each driver is individually responsible for its assigned customers' orders. However, transportation costs from necessary refill trips grow quickly with the number of customers. Therefore, we explore an alternative, flexible strategy called *overlapped routing* that mitigates the costs accrued through route failures. Under overlapped routing, the a priori routes are designed with overlap such that neighboring routes share some customers. Then, with coordination from the distributor or other central planner, drivers visit a narrowed down subset of customers within their predesigned routes in response to realized customer demands. This overlapping design, inspired by the existing literature in manufacturing process flexibility, maintains much of the route consistency of the dedicated routing strategy while allowing the distributor some flexibility to adjust to changing demands.

In their working paper, Ledvina et al. (2020) first proposed the overlapped routing strategy and provided a theoretical guarantee on its cost in problems with a infinitely large number of customers. The researchers also explored the strategy's non-asymptotic performance through a computational study. Here, we expand on their work with additional scenarios and analysis made possible through revamped simulation code. Through this work, we find that when vehicle capacity is an active constraint, trips to and from the depot drive increasing transportation costs as the number of customers grows. However, overlapped routing harnesses surplus capacity within the vehicle fleet to mitigate these costs. More specifically, through customer sharing, vehicles with excess capacity can assist with deliveries along a neighboring route and sometimes prevent the need for a refill trip elsewhere in the network.

The next chapter provides an overview of the related literature in vehicle routing and process flexibility. Chapter 3 formally defines the routing models and walks through the proposed recourse policies. Chapter 4 describes the computational study on the cost-saving potential of flexible vehicle routing and provides simulation outputs and findings. Chapter 5 concludes.

Chapter 2

Related Literature

In this chapter we review the research that motivates or overlaps with our proposed flexible routing strategy and customer sharing scheme. First, we identify our position within the vehicle routing problem (VRP) with stochastic customer demands. Then we introduce key concepts from the literature on manufacturing process flexibility, which inspires the design for our model's fixed, yet flexible, routes. Finally, we touch on the field of collaborative vehicle routing – a subarea of freight logistics – which explores customer sharing in mostly non-stochastic settings.

2.1 VRP with Stochastic Demands (VRPSD)

The VRP with stochastic demands (VRPSD) describes a class of routing problems in which customer demands are uncertain (Dror et al. 1989, Gendreau et al. 2014). In contrast, the deterministic VRP addresses settings in which customer demands – as well as travel times, costs, and other possible sources of randomness – are fixed and known. Typically, the objective in the VRPSD is to design vehicle routes that minimize the cost of filling the unknown demands subject to a set of unique, setting-specific constraints, e.g., related to fleet size, vehicle capacity, service level targets, delivery time windows, etc. However, solution methodologies depend on each problem's specific characteristics as well as any assumptions on the likely distribution of customer demands.

Gendreau et al. (2016) review notable models and methods for the VRPSD and separate the problem into the *a priori paradigm* and the *reoptimization paradigm.* In the a priori model, routes are fixed in advance before customer demands are realized. Under reoptimization, on the other hand, routes are updated gradually as new demand information becomes available. In some cases these updates occur dynamically while the vehicle executes its route. Note that if demands are learned sufficiently early, organizations may have the information, time, and resources to fully optimize their routes, at which point the VRPSD can be solved as a deterministic VRP.

Our proposed flexible routing model uses a priori routing, or fixed routing. In this setting, a route failure occurs if the vehicle exhausts its capacity prior to completing its assigned deliveries. Recourse policies describe how a driver should respond upon experiencing a route failure. Under a classical recourse policy, for instance, the prescribed response is that the vehicle must detour to the depot to replenish its inventory before proceeding with its route at the point of failure (Gendreau et al. 2016). This response increases travel costs but is necessary if the driver needs to fully serve its route. Therefore, an important research area is in developing initial routes – or for some approaches, larger tours and customer sequences that can then be split into routes – that yield the minimum travel cost in expectation over variable customer demands.

With a priori routing, typically drivers learn the specific customer demands only upon arriving at the customer location – see, e.g., Secomandi and Margot (2009) and Gendreau et al. (2014). In contrast, our routing strategy assumes that drivers learn customer demands before executing their routes. Other researchers such as Bartholdi III et al. (1983), Bertsimas (1992), and Jaillet (1988) have adopted this assumption as well. Bertsimas (1992), for example, evaluates a routing strategy for the VRPSD in which drivers can bypass customers with zero demand and thus decrease transportation costs. Receiving demand information in advance allows for upfront adjustments to the routes that drivers execute. In this way, planners can incorporate some of the flexibility of reoptimization while still preserving some consistency in the initial routes.

Another feature of our proposed flexible routing strategy is overlapped routes, in which delivery vehicles serving adjacent areas share customers. To our knowledge, within the VRPSD literature, only a few other papers consider some version of fixed routing with customer sharing. Erera et al. (2009), for example, propose a fixed routing system and route construction method to accommodate a variable customer list with delivery time windows. In their approach, customers are assigned to both primary and backup routes. Planners then have the option to bump customers from one route to the other to maintain overall feasibility or to lower transportation costs without re-solving the routing problem from scratch.

Ak and Erera (2007) propose a paired vehicle strategy in which vehicles are assigned a priori routes, matched together in exclusive pairs, and then assigned a Type I or Type II designation within their pairs. If a Type I vehicle experiences a route failure, it terminates its route, and the Type II vehicle extends its own route to serve the remaining customers from the Type I route, detouring to the depot to refill as needed.

Lei et al. (2012) also consider a recourse policy with exclusively paired vehicles. In their setup, paired routes can split demands at a single shared customer. The authors derive expected costs of a non-cooperative case in which vehicles fill an optimally predetermined percentage of the shared customer's demand. The authors also discuss a cooperative case in which a vehicle fills as much demand as possible in certain situations to reduce the chance of the other vehicle's experiencing a route failure and performing a costly refill trip.

Finally, the working paper from Ledvina et al. (2020) introduces a route chaining design in which non-exclusive pairs of neighboring routes share customers. The chaining design is inspired by the literature on manufacturing process flexibility, further discussed below. Ledvina et al. (2020) derive theoretical guarantees on the asymptotic performance of their strategy, i.e., as the number of customers approaches infinity. Additionally, like Ak and Erera (2007), Erera et al. (2009), and Lei et al. (2012), they present computational results for smaller problems. This thesis builds on Ledvina et al. (2020) with an expanded computational study and sensitivity analysis on the non-asymptotic costs of the proposed flexible routing strategy.

2.2 Process Flexibility

We employ a fixed route design inspired by manufacturing process flexibility. Jordan and Graves (1995) produced the seminal paper on the principles and benefits of manufacturing process flexibility, which they define as the ability "to build different types of products in the same plant or production facility at the same time" (Jordan and Graves 1995).[1] Factories are assigned to manufacture specific product types within an overall production capacity. However, demands for each product type are variable, and the system's flexibility to meet unexpected spikes in demand increases when more than one plant can produce the requested product type. By simulating random demands for different product types, Jordan and Graves (1995) show that a factory-product assignment structure called the *long chain* achieves comparable service levels (measured as the share of product demand that can be met) as a fully flexible network but with much less investment in redundant production capability. In a *fully flexible* network, every factory can produce every product. Under the long chain, however, each factory can produce only two product types such that the structure of the factory-product assignment bipartite graph forms a closed chain throughout the network. Figure 2.1 from Jordan and Graves (1995) illustrates this finding on the long chain's performance.

Simchi-Levi and Wei (2012) later proved that in adding flexibility to an inflexible, dedicated manufacturing system (in which each factory can produce only one product type), the marginal gains in expected service level increase with each additional flexible edge, or redundant factory-product assignment. The final link which transforms an *open chain* graph into a closed, long chain graph yields the greatest incremental benefit of all. Figure 2.2 from Simchi-Levi and Wei (2012) illustrates the process and marginal gains of incrementally adding flexibility to a manufacturing system. In this

[1] In the literature, "process flexibility" is often used in place of "system flexibility" as defined by Simchi-Levi (2010). Therefore, we will follow Jordan and Graves (1995) in using the term process flexibility to refer to manufacturing networks with coordinated production capabilities.

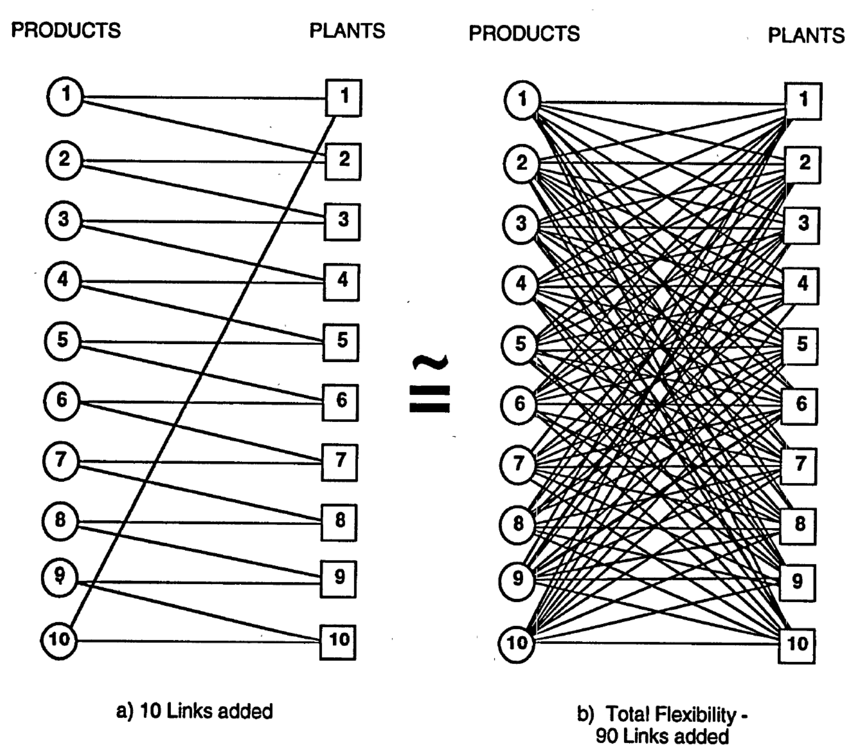
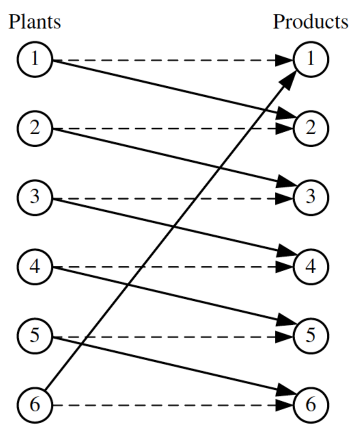


Figure 2.1: Network structures in manufacturing process flexibility. The long chain assignment structure (a) achieves comparable service levels as a fully flexible network (b). Figure from Jordan and Graves (1995).

example, adding edges (1,2),...,(5,6) forms an open chain network while adding edge (6,1) creates the long chain network.

(a) Flexibility structure



(b) Incremental benefits of creating long chain

Structure	Performance	Incr. benefit
Dedicated	5.6	
Added arc (1, 2)	5.622	0.022
Added arc (2, 3)	5.652	0.030
Added arc (3, 4)	5.686	0.035
Added arc (4, 5)	5.724	0.0379
Added arc (5, 6)	5.765	0.0403
Added arc (6, 1)	5.842	0.077
Full flexibility	5.842	

Figure 2.2: Example of the marginal gains in performance (expected sales) as a manufacturing assignment network adds flexible edges one-by-one. Figure from Simchi-Levi and Wei (2012).

Ledvina et al. (2020) embedded process flexibility into the VRPSD by adapting the open chain design from manufacturing process flexibility. They find that an open chain structure for assigning vehicles to customers yields considerable savings in transportation costs relative to a dedicated system. Additionally, in their model, these savings grow as the number of customers increases (Ledvina et al. 2020).

Lyu et al. (2019), Asadpour et al. (2020), and Xu et al. (2020) also add process flexibility into delivery applications but consider different settings and performance metrics than do Ledvina et al. (2020). Specifically, Lyu et al. (2019) presents a case study on assigning workers to parcel delivery zones with a long chain structure such that workers serve overlapping zones and can share any high delivery burden. Asadpour et al. (2020) apply the long chain flexibility design to an online order-fulfillment system with multiple warehouse locations and limited resource inventories. Finally, Xu et al. (2020) extend Asadpour et al. (2020) by considering unbalanced systems with different numbers of warehouses versus product types. These three papers all measure network performance in terms of service level for customer demands while our work focuses on the routing cost of filling all demands. Additionally, Asadpour et al. (2020) and Xu et al. (2020) limit warehouse inventories while our model assumes sufficient inventory to fill all demand.

2.3 Collaborative Routing

Finally, we touch on the literature in collaborative routing in freight logistics. Gansterer and Hartl (2018) define collaborative vehicle routing as "all kinds of cooperations which are intended to increase the efficiency of vehicle fleet operations." Carriers, distributors, or other logistics service providers may share vehicles, facilities, customers, or other resources to decrease costs, increase service level, or even decrease emissions among other objectives (Cleophas et al. 2019, Gansterer and Hartl 2018).

Generally, research in collaborative routing focuses on multi-agent environments. Typically, in non-collaborative settings, individual organizations maintain their own information with an eye towards maximizing their individual profits. In contrast, in collaboration with *decentralized planning*, participants agree on mechanisms such as auctions or hierarchical decision-making to exchange limited information for collaboration. Even more, with *centralized planning*, all participant information needed for collaboration is known, and the objective is to maximize the joint utility (e.g., profit) of all collaborators. See Gansterer and Hartl (2018) for an overview of the recent literature on models and methods for centralized collaborative planning specifically.

As in the VRPSD literature, a major area of research in collaborative routing is in optimizing vehicle routes and customer assignments, with additional insights into the impact of collaboration size, benefits relative to non-cooperative scenarios, and the design of compensation schemes – see e.g., Sanchez et al. (2016), Quintero-Araujo et al. (2016), Defryn et al. (2016), and Guajardo and Rönnqvist (2016). For example, Fernández et al. (2018) introduce the Shared Customer Collaboration Vehicle Routing Problem (SCC-VRP) in which customer orders can be transferred among a predetermined set of carrier companies. Each carrier operates its own fleet of delivery vehicles, and the demand filled by any vehicle cannot exceed the vehicle's capacity. The objective of the SCC-VRP is to minimize the total routing cost across carriers. From their computational studies, Fernández et al. (2018) observe that collaboration yields the most savings relative to a non-collaborative scenario when a larger number of customers are shared over a greater region. Similarly, Sanchez et al. (2016) explore gains from the collaboration of different subsets of carriers and find that the most gains occur with a complete pooling of resources. In their literature survey, Gansterer and Hartl (2018) also observe that the best savings are achieved with complete cooperation.

Our work in flexible vehicle routing is similar to collaborative routing in that we are also designing routes with customer sharing and quantifying the resulting benefits. However, much of the collaborative routing literature assumes deterministic customer demands or order requests (as well as a multi-agent setting), which allows participants to redesign their collaborations for each realized instance. Quintero-Araujo et al. (2016) produced one of the few collaborative routing studies that does consider stochastic demands. In their model, delivery vehicles are penalized for route failures; then the overall transportation cost for a set of routes is calculated as the sum of the deterministic VRP cost and the average route failure cost. Still, our flexible routing strategy is most grounded in VRPSD methodologies. Looking now to the next chapter, we will finally describe our proposed strategy in detail.

Chapter 3

Flexible Vehicle Routing

In this chapter, we establish model notation, define two types of vehicles routes, and introduce the dedicated and overlapped recourse policies. Recall that a recourse policy is a planned response to a route failure, i.e., actions to take when a vehicle exhausts its capacity prior to completing its route. For this reason, we often refer to recourse policies as routing *strategies*. Of the dedicated and overlapped routing strategies, overlapped routing is our proposed flexible approach for settings with a priori routes and stochastic customer demands. For comparison purposes, we also define a fully flexible policy, in which vehicles follow a fixed route but are not restricted to specific customer subsets, as well as reoptimization as alternative routing strategies.

After describing the model, we walk through a concrete example of the routing strategies in a small-scale problem with six customers. Finally, though this thesis focuses on computational results in the non-asymptotic case, we share some key findings from Ledvina et al. (2020) on the performance of the overlapped routing strategy as the number of customers approaches infinity.

3.1 Model Setup

Below we formally define the routing models. We establish the relevant notation and describe the routing strategies that we evaluate.

3.1.1 Notation

A fleet of M homogeneous delivery vehicles must fill the stochastic daily demands of a set of N customers. Vehicles each have a capacity of Q units, meaning a vehicle can serve Q units of customer demand before the vehicle exhausts its capacity and must detour to the depot to reload. We assume a single product type. For computational convenience, we also assume the number of customers N is divisible by fleet size M. This way, we can assign vehicles to routes of equal length in terms of the number of customers they may need to visit.

Next we represent customer locations and demands. Customer i is located at position $\mathbf{x_i} \in \mathbb{R}^2$ on a bounded plane for $1 \leq i \leq N$. Delivery vehicles depart from a single depot located at $\mathbf{x_0} \in \mathbb{R}^2$. Each customer i is then the Euclidean distance $\mathbf{c_{0i}} = \|\mathbf{x_0} - \mathbf{x_i}\|_2$ from the depot and distance $\mathbf{c_{ji}} = \|\mathbf{x_j} - \mathbf{x_i}\|_2$ from any other customer j. We calculate transportation cost as the sum of all costs $\mathbf{c_{0i}}$ and $\mathbf{c_{ji}}$ accrued on the realized (executed) route. All customer demands are independently and identically distributed such that customer i exhibits random demand $D_i \sim D$ where D covers some subset of non-negative integers $0, 1, 2, ..., d_{max}$. Customer i's realized demand for a given day is denoted d_i, and we denote the set of all realized customers demands as $\mathcal{D} = \{d_1, ..., d_N\}$.

Vehicles are assigned to a priori routes with a subset of customers according to some predetermined flexibility design. In our model, flexibility refers to the extent of route overlap (customer sharing) between adjacent vehicle routes. Specifically, we identify two types of a priori routes: primary routes and extended routes. Primary routes are defined as disjoint routes with no customer sharing such that each vehicle $m = 1, .., M$ has an a priori route with $N' = N/M$ customers. More formally, define π as a tour through all N customer locations beginning and ending at the depot, and assign labels $i = 1, 2, ..., N$ sequentially to the customers in the tour π. Then, the primary route for vehicle $m = 1, ..., M$ is the customer sequence $\mathcal{P}_m = [(m-1)N' + 1, (m-1)N' + 2, ..., mN']$. To illustrate, this setup means vehicle 1's primary route is the customer sequence $[1, 2, ..., N']$, vehicle 2's primary route is the customer sequence $[N' + 1, ..., 2N']$, and so on, until we reach vehicle M's route $\mathcal{P}_M = [N - N' + 1, ..., N]$.

Extended routes, on the other hand, are designed with some overlap such that adjacent routes share k customers. Define the extended route for vehicle m as the customer sequence $\mathcal{E}_m = [(m-1)N' + 1, (m-1)N' + 2, ..., mN' + k]$ for vehicles $m = 1, ..., M - 1$. The extended route for vehicle $m = M$ is just $\mathcal{E}_M = [(M - 1)N' + 1, (M-1)N' + 2, ..., MN']$. Put differently, the extended route for vehicle $m = 1, ..., M-1$ is vehicle m's primary routes plus the first k customers from vehicle $m+1$'s primary route while vehicle $m = M$'s extended route is identical to its primary route. Drivers must visit the customers with non-zero demand sequentially within these routes.

Figure 3.1 shows bipartite graphs with vehicle-customer assignments for (i) primary routes, (ii) extended routes with one shared customer ($k = 1$), and (iii) extended routes with overlap size equal to the full primary route size ($k = N'$). Note that these route assignments parallel the flexibility networks from the manufacturing process flexibility literature. Specifically, the primary routes follow a dedicated network design, and the extended routes resemble the open chain design.

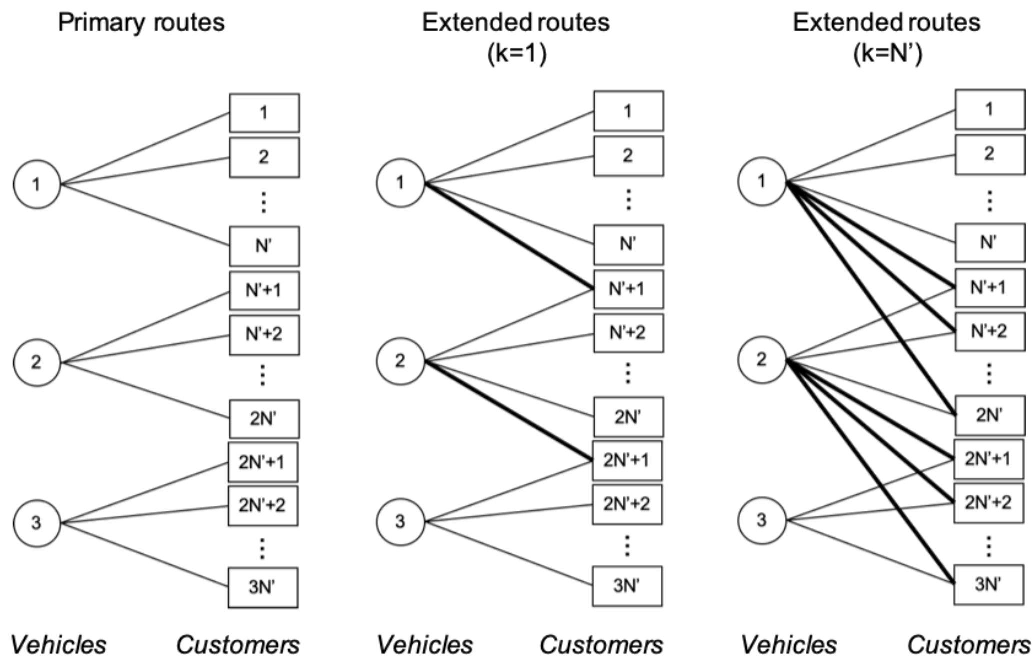


Figure 3.1: Route assignments for a network with $M = 3$ vehicles. **Bold** lines indicate vehicle assignments to the k shared customers for the extended routes.

3.1.2 Recourse Policies

Below we describe the recourse policies for dedicated routing and overlapped routing. We also describe the full flexibility strategy and a separate reoptimization strategy for comparison purposes. All strategies except reoptimization are examples of fixed routing strategies. However, they involve different a priori routes and prescribe different responses to a route failure.

In dedicated routing, vehicles are assigned to their primary routes with disjoint customers subsets. Each day vehicles first receive information on their assigned customers' realized demands and identify which specific customers require deliveries. Each vehicle then departs the depot at full capacity Q and sequentially visits the customers in its primary route $\mathcal{P}_m$, bypassing the customers that have zero demand. If the vehicle exhausts its capacity, the driver detours to the depot to refill to full capacity and resumes its route wherever it left off. Upon filling all customer demands in its primary route, the vehicle has completed its route for the day and returns to the depot.

Under the overlapped routing strategy, vehicles instead are assigned to their extended routes, which include the N' customers in a vehicle's primary route plus some k additional customers (for vehicles $m < M$). As in the dedicated strategy, each day vehicles first receive information on realized customers demands. Upon learning demands, vehicles depart from the depot at full capacity and sequentially visits customers in their primary routes, bypassing customers with zero unfilled demand. Each vehicle m detours to the depot to reload as necessary to serve all unfilled demand in primary route $\mathcal{P}_m$. Upon filling demand of the final primary route customer, if all Q units of capacity are exhausted, the vehicle returns to the depot and concludes its route. However, if vehicle m has any remaining capacity, it sequentially visits and serves any additional customers in its extended route $\mathcal{E}_m$ until the vehicle's surplus capacity is exhausted. At this point, vehicle m returns to the depot for the day. Then vehicle $m+1$ begins its primary route $\mathcal{P}_{m+1}$ wherever vehicle m ended its service. In the case that vehicle m for some $m = 1, ..., M-1$ satisfies the demand of all customers in vehicle $m+1$'s primary route, then vehicle $m+1$ does not need to be deployed. We assume that a central planner assesses the realized customer demands and coordinates each vehicle's starting and ending customers (realized routes) within the extended routes prior to the vehicles' departing the depot. This way, vehicles can execute their routes simultaneously.

While we focus on overlapped routing as our proposed flexible policy, we can also define a strategy with full flexibility in which any vehicle can serve any customer as long as the drivers followed the predetermined customer sequence π. This vehicle-customer assignment structure is inspired by the full flexibility design from the manufacturing process flexibility literature. In executing this strategy, vehicle 1 serves the first Q units of demand at which point vehicle 1 experiences a route failure. Vehicle 2 continues where vehicle 1 ended and serves an additional Q units of demand, and so on, until all customers have been served. Depending on the magnitude of total customer demand, the final vehicle M may need to make multiple trips to serve all remaining customers. Alternatively, the final vehicles may not be needed at all on a given day if the previous vehicles were able to fill all demand. This strategy closely aligns with the classical recourse policy analyzed by Bertsimas (1992) and others. The main difference is that our full flexibility model includes multiple vehicles – though it can be simplified to the single vehicle case – which in practice allows for a divided workload with concurrent route execution under the guidance of a central planner.

Finally, reoptimization generates the lowest-cost vehicle routes for each new set of demands. Unlike the fixed route strategies above, reoptimization does not constrain which vehicles can visit which customers or in which order. In the most general case, reoptimization in our setting is solved as a VRPSD with splittable demands in which a customer can be served through multiple visits.

To help elucidate the policies above, we developed a companion Jupyter notebook file in which users can generate random customer and demand instances and see the resulting routes and costs under the different strategies – please refer to Appendix A for more information. We also used this tool to create the example presented below.

3.2 Example Problem

To illustrate the routing models and recourse policies, consider an example with six customers. Each customer has demand randomly selected from 0, 1,..., 8. We have three vehicles in our fleet, so we decide to create three primary routes with two customers each. Each vehicle's capacity Q is 8, which is the expected combined demand of two customers on a primary route.

To generate the a priori routes, let's first create a giant traveling salesman tour through all customers. Select an arbitrary first customer, and label the tour's customers sequentially as 1, 2,..., and 6. For the primary routes, divide the tour into three sub-sequences of size 2. Define primary route A with customers $\{1, 2\}$, primary route B with customers $\{3, 4\}$, and primary route C with customers $\{5, 6\}$. For the extended routes, let overlap size k be 1, and define extended routes A, B, and C to have customers $\{1, 2, 3\}$, $\{3, 4, 5\}$, and $\{5, 6\}$, respectively. Under this design, adjacent extended routes A and B share one customer (customer 3) and adjacent extended routes B and C share one customer (customer 5). Figure 3.2 shows the giant tour, set of primary routes, and set of extended routes as arranged through customer locations in a Cartesian plane.

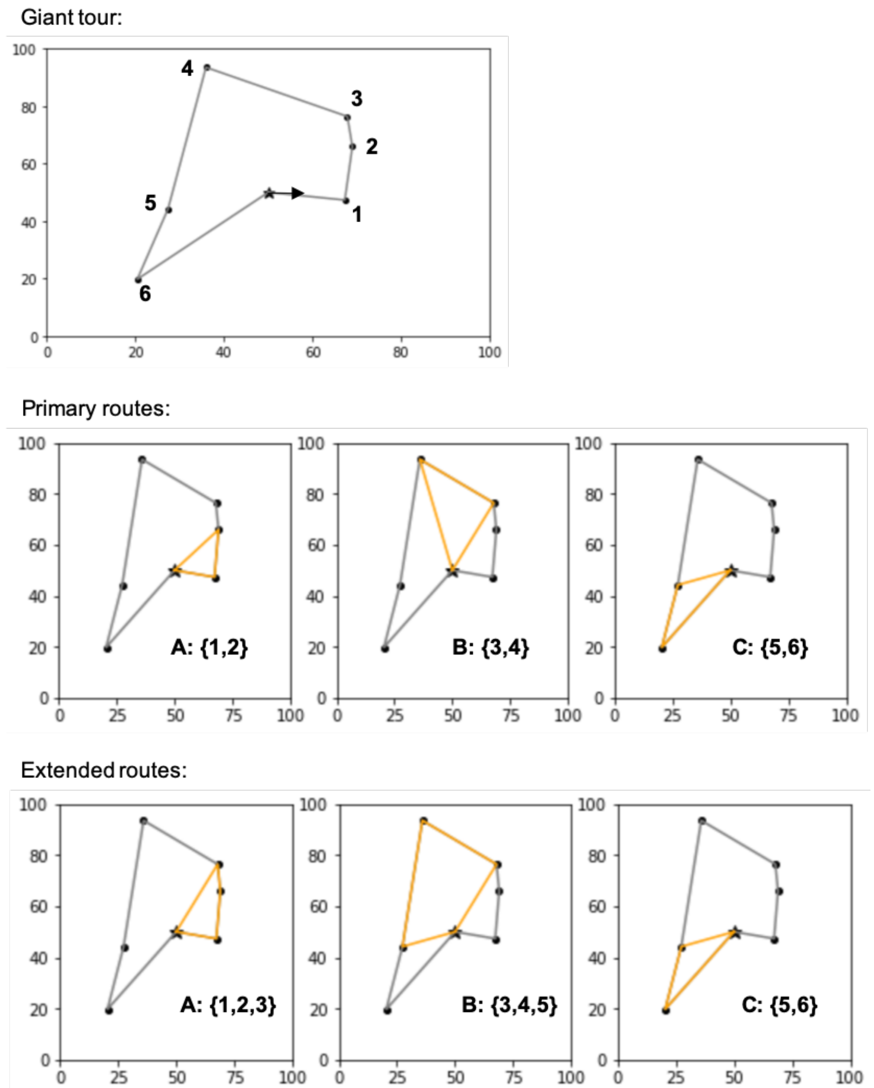


Figure 3.2: Giant tour (grey) and resulting primary and extended routes (orange) in the example problem. Customers were randomly placed on a 100x100 grid. The single centrally located depot is marked with a star.

After creating our routes, we learn customers' actual demands, which may change from day to day. Table 3.1 presents one particular day's demand for each customer in our example. We will illustrate the dedicated routing, overlapped routing, and reoptimization strategies on this demand instance.

	Cust. 1	Cust. 2	Cust. 3	Cust. 4	Cust. 5	Cust. 6	Total
Demand	7	4	1	3	7	2	24

Table 3.1: Realized customer demands in the example problem

Dedicated Routing

Recall that in the dedicated routing strategy, each vehicle is independently responsible for its primary route, either A, B, or C. Figure 3.3 shows how each vehicle navigates its primary route to fill its customers' demands. Here, Vehicle A must fill a total of 11 units of demand on its route, which exceeds the vehicle's capacity of 8. Because Vehicle A exhausts its capacity while serving Customer 2, the driver detours to the depot to restock before completing its delivery. Vehicle B, on the other hand, faces only 4 units of demand and can serve its customers in one trip. Finally, Vehicle C must fill 9 units of demand on its route, and like Vehicle A, requires two trips from the depot to serve fully serve the final two customers. Table 3.2 summarizes each vehicle's realized trip count, customers served, and demand filled. Ultimately, this dedicated routing strategy costs 400.1 units, measured as the Euclidean travel distance in filling all customer demands.

Vehicle	Number of Trips	Customers Served	Demand Filled
A	2	$\{1, 2\}$	11
B	1	$\{3, 4\}$	4
C	2	$\{5, 6\}$	9
Total	5	All	24

Routing Cost: 400.1

Table 3.2: Summary of **dedicated** routing in the example problem

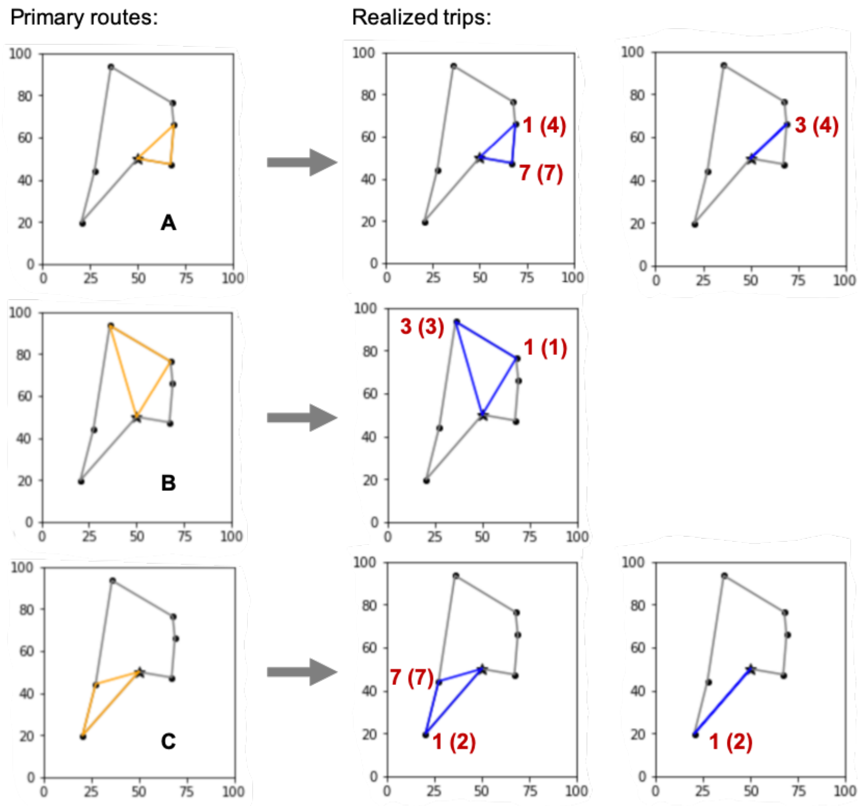


Figure 3.3: Primary routes and realized trips under **dedicated** routing in the example problem. Red labels state the demand filled (total demand) at each customer.

Overlapped Routing

We now walk through an overlapped routing strategy in which adjacent routes share one customer. Figure 3.4 illustrates the a priori extended routes as well as the realized trips for each vehicle under this strategy. In calculating workloads, we iterate sequentially through routes A, B, and C; note, however, that drivers are informed of their updated workloads prior to departure and thus can simultaneously execute their routes.

Vehicle A first executes its primary route as in the dedicated routing strategy. Upon serving Customer 2 in its second trip, the vehicle has 7 units of surplus capacity and thus proceeds to fully serve the 1 unit of demand at Customer 3, the shared customer in Vehicle A's extended route. Upon completing its extended route, Vehicle A returns to the depot. Vehicle B then begins with the first unserved customer in its primary route, which is Customer 4. Upon completing its primary route, Vehicle

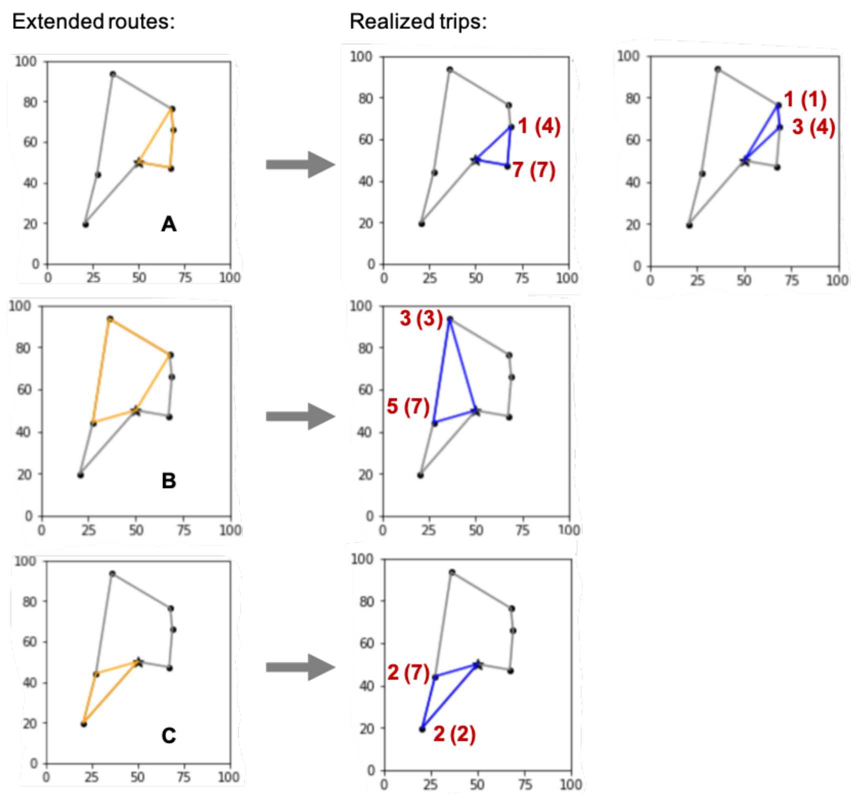


Figure 3.4: Extended routes and realized trips under **overlapped** routing in the example problem. Red labels state the demand filled (total demand) at each customer.

B has 5 units of surplus demand, which it uses to partly serve Customer 5 in its extended route before returning to the depot. Finally, Vehicle C fills the remaining demand at Customer 5 and all demand at Customer 6.

This strategy eliminates the need for the second trip that Vehicle C performed under dedicated routing. Ultimately, as summarized in Table 3.3, the overlapped strategy costs 338.4, a 15% savings over dedicated routing.

Vehicle	Number of Trips	Customers Served	Demand Filled
A	2	$\{1, 2, 3\}$	12
B	1	$\{4, 5\}$	8
C	1	$\{5, 6\}$	4
Total	4	All	24

Routing Cost: 338.4

Table 3.3: Summary of **overlapped** routing in the example problem

Full Flexibility

In routing with full flexibility, any vehicle can serve any customer. As illustrated in Figure 3.5, the structure of the corresponding fully flexible assignment network differs from that of dedicated and overlapped routing, in which customers are restricted to certain customer subsets. However, the construction of realized vehicles routes must still follow the customer sequence defined by our giant tour.

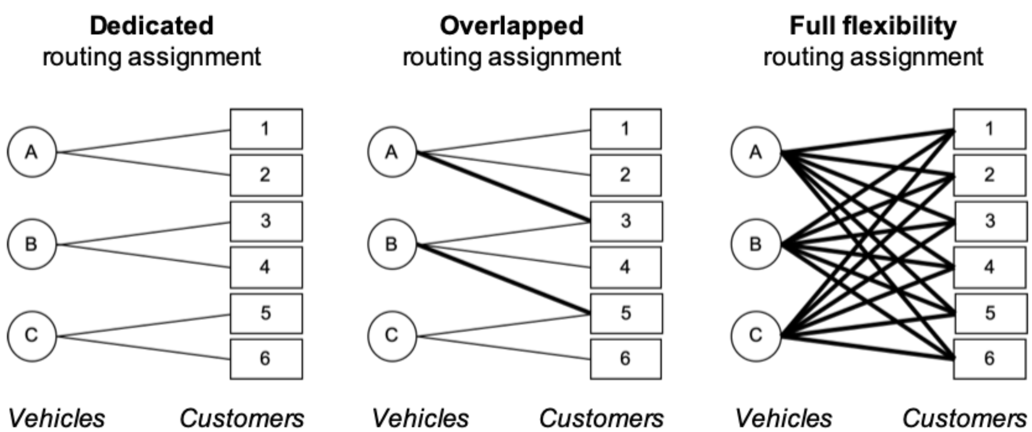


Figure 3.5: Vehicle-customer assignment networks for dedicated, overlapped, and fully flexible routing in the example problem. **Bold** lines indicate assignments to shared customers.

To determine the realized routes, we assign Vehicle A to first serve as much demand as possible. It fully serves Customer 1 but only partly serves Customer 2, at which point Vehicle A experiences a route failure and returns to the depot. Vehicle B then completes the delivery at Customer 2 and manages to fully serve Customers 3 and 4 and partly serve Customer 5 before the vehicle exhausts its capacity. Finally, Vehicle C visits both Customer 5 and Customer 6. As in overlapped routing, this algorithm is used simply to determine the day's routes; upon initial coordination, drivers can then execute their finalized routes simultaneously.

Figure 3.6 illustrates the realized vehicle routes. The key difference from the overlapped routing solution is that Vehicle B is able to serve Customer 2 which is outside of Vehicle B's extended route. This ability is especially valuable since it saves Vehicle A a refill trip and, in this example, has no repercussions further along in the customer sequence. Table 3.4 summarizes realized routes and costs under full flexibility. The total cost of 297.0 is a 12% savings over overlapped routing and a 26% savings over dedicated routing.

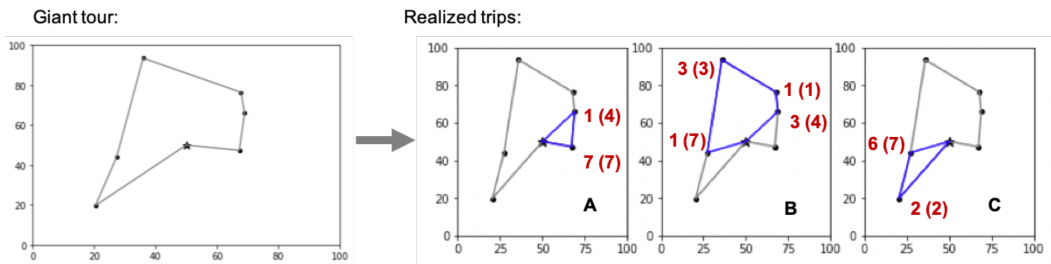


Figure 3.6: Realized trips for each vehicle under routing with **full flexibility**. Red labels state the demand filled (total demand) at each customer.

Vehicle	Number of Trips	Customers Served	Demand Filled
A	1	$\{1, 2\}$	8
B	1	$\{2, 3, 4, 5\}$	8
C	1	$\{5, 6\}$	8
Total	3	All	24

Routing Cost: 297.0

Table 3.4: Summary of **full flexibility** in the example problem

Reoptimization

Unlike the three strategies above, reoptimization does not restrict the vehicles to certain customers or sequences. Figure 3.7 shows the cost-minimizing routes that solves the VRP with splittable demands. In this case, each vehicle fills exactly 8 units of demand.

To summarize, Table 3.5 presents the outcomes for all routing strategies. In this table, *radial cost* is the total distance traveled to and from the depot while *circular cost* is the distance traveled between customers. Looking at the total cost (which equals the sum of the radial and circular costs), full flexibility and reoptimization are tied as the lowest cost strategies while dedicated routing is the most expensive. We also see that total cost increases with trip count, which drives the radial cost component. Here, radial cost makes up 53% of total cost for full flexibility and reoptimization, 69%

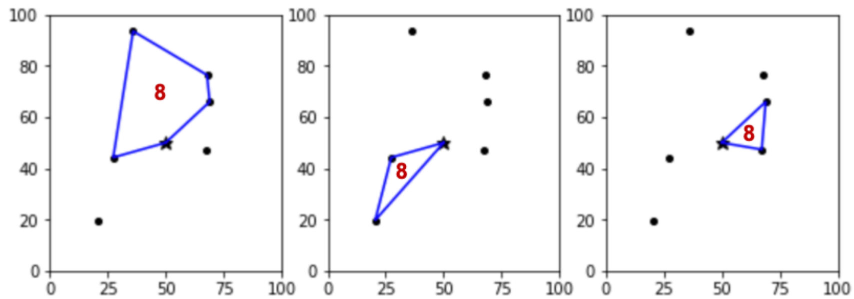


Figure 3.7: **Reoptimized** routes once demands are known in the example problem. Red labels state the demand filled with each trip.

for overlapped routing, and 80% for dedicated routing. The value of both overlapped and fully flexible routing over dedicated routing is that drivers can harness surplus capacity in the fleet to potentially eliminate the need for a refill trip elsewhere in the system. Additionally, even with a fixed sequence of customers, full flexibility matches the cost of reoptimization while still allowing for some consistency and early preparation. Therefore, fixed yet flexible routing can be a valuable strategy in settings where reoptimization is not logistically or computationally practical.

Finally, in comparing the flexible strategies, full flexibility outperforms overlapped routing but likely requires additional investment. For example, each driver must be prepared to travel over any part of the giant tour, and each customer must be willing to receive deliveries from different vehicles each day. Thus, overlapped routing can be a practical middle ground strategy with substantial gains – in this example, through a 15% savings – over the dedicated approach. Even more, we will see in Section 4 that overlapped routing actually performs comparably to full flexibility on average across several location and demand instances.

Routing Strategy	Number of Trips	Radial Cost	Circular Cost	Total Cost
Dedicated	5	319.7	80.3	400.1
Overlapped	4	233.7	104.7	338.4
Full Flexibility	3	156.1	140.9	297.0
Reoptimization	3	156.1	140.9	297.0

Table 3.5: Comparison of all routing strategies in the example problem

3.3 Asympototic Performance

Finally, though this thesis focuses on smaller scale problems, we briefly comment on the asymptotic characteristics of overlapped routing. Ledvina et al. (2020) use probabilistic analysis to derive a theoretical guarantee on the relative performance of overlapped routing as the number of customers approaches infinity. Specifically, they find that given a demand distribution D_i with mean μ, then

$$\lim_{N\to\infty} \frac{Z(O)}{Z^*} = \lim_{N\to\infty} \frac{Qr_{avg}}{N'\mu} \tag{3.1}$$

where Z^* is the optimal travel distance in expectation, $Z(O)$ is the expected travel distance under overlapped routing, and r_{avg} is the expected number of trips per vehicle under overlapped routing. Additionally, if capacity Q is no more than a route's expected demand $N'\mu$, then

$$\lim_{N\to\infty} \frac{Z(O)}{Z^*} \leq 1 + \frac{\sigma}{2\mu\sqrt{N'}} \tag{3.2}$$

where σ is the standard deviation of demand. Excitingly, Equation 3.2 states that as the number of customers is scaled to infinity, the cost of overlapped routing relative to the optimal cost is bounded above by some constant. In other words, there is a cap on how much more costly overlapped routing will be in large scale problems. Even more, as either (i) the coefficient of variation of demand decreases and/or (ii) the size of a vehicle's route increases, the bound becomes tighter and overlapped routing approaches reoptimization in cost. These theoretical guarantees are distributionally robust, meaning they hold for any demand or customer location distribution, assuming the distributions are independent and identical across customers.

Chapter 4

Computational Study

We use numerical simulation to assess the cost-saving potential of the overlapped routing strategy. This chapter describes the simulation setup and presents the results for a baseline scenario along with some variations on the baseline as a sensitivity analysis. Though we focus on the relative performance of overlapped routing, dedicated routing, and reoptimization, we also provide results for full flexibility, which we find closely aligns with overlapped routing.

4.1 Simulation Setup

In this study, we compare costs of the routing strategies from Chapter 3 under various network designs. Cost is measured as the total Euclidean distance traveled by the vehicle fleet in serving all customer demands. For each instance and routing strategy, the simulation program returns the total cost as well its radial and circular cost components. Recall that radial cost is the cumulative distance traveled to and from the depot (at the beginning or end of a route or when conducting a refill trip) while circular cost is the cumulative distance traveled between customers.

We simulate problems with $N = 5$, 10, 20, 40, and 80 customers. For each problem size, we randomly generate 30 customer location instances and 200 demand instances. Customer demands are independently and identically distributed according to the demand scenario as defined in the following sections.

To create the a priori routes for each customer location instance, we first generate a traveling salesman tour through all customer locations. We then create extended routes beginning with the first customer in the tour's sequence and calculate the total cost of overlapped routing over all demand instances. We rotate the tour to test each customer as the first customer in the tour's sequence, and ultimately, keep the sequence that yields the lowest average overlapped routing cost. We use this sequence to generate the a priori routes used in the dedicated and overlapped strategy for all demand instances and that particular customer instance.

Simulations are run in Python 3.7. We use Google Optimization Tools (OR-Tools) (Perron and Furnon 2019) to (i) generate the traveling salesman tour used to create the a priori routes and (ii) solve the VRP with splittable demands for each demand instance. More specifically, for the VRP in each demand instance, we transform the integer demand problem into an equivalent problem with smaller customers each with unit demand and then find the optimal vehicle routes using the OR-Tools VRP solver.

In the next section, we define our main scenario and describe the baseline simulation results. Unless specified otherwise, results for each problem size are presented as the average over all customer and demand instances.

4.2 Baseline Results

The main results in this study are for the baseline scenario, which has the parameters below:

- **Demand** $D_i \sim \text{Uniform}\{0, 8\}$, meaning that realized demand d_i for each customer i equals 0, 1, 2, ..., or 8 with equal probability.
- **Route size** $N' = 5$, meaning that the a priori primary route for each vehicle j has 5 customers.
- **Overlap size** $k = 5$ for each vehicle j, meaning that the a priori extended route for each vehicle $j \neq M$ has 10 customers.[1]

[1] Recall the extended route for the final vehicle $j = M$ is identical to its primary route.

- **Vehicle capacity** $Q = 20$ so that in expectation, vehicle j can fully serve its primary route in a single trip.

Table 4.1 presents the total cost for each routing strategy in the baseline scenario while Figure 4.1 illustrates these results. We include both total cost and the cost relative to reoptimization. As the number of customers grows from $N = 5$ to $N = 80$, dedicated routing increases from 14% more expensive than reoptimization up to 26% more expensive. Overlapped routing, on the other hand, decreases from 14% more expensive than reoptimization at 5 customers down to just 6% more expensive at 80 customers. Ultimately, for 80 customers, we find that overlapped routing yields a 16% savings over dedicated routing on average.

Cust.	Total				Relative to Reoptimization		
	Dedic.	Overlap.	Full Flex.	Reopt.	Dedic.	Overlap.	Full Flex.
5	228.7	228.7	228.7	200.3	1.142	1.142	1.142
10	393.4	387.1	387.1	331.8	1.186	1.167	1.167
20	656.0	619.7	621.7	549.4	1.194	1.128	1.132
40	1,148.1	1,027.9	1,028.0	939.4	1.222	1.094	1.094
80	2,069.9	1,746.2	1,734.5	1,646.6	1.257	1.060	1.053

Table 4.1: Average cost in the baseline scenario

Full flexibility performs almost identically to overlapped routing, though interestingly, overlapped routing actually slightly outperforms full flexibility in problems with 10 customers. While intuitively more flexibility should enable lower costs, our fixed giant tour combined with randomness in the relatively few customer location instances means that full flexibility is not guaranteed to outperform overlapped routing. Unsurprisingly, however, the random cost differences even out as customer size grows so that by 80 customers full flexibility does ultimately yield the lower average cost.

To look beyond average performance, Figure 4.2 shows the distribution of costs over all demand and customer instances. The figure includes one graph for each routing strategy, with each graph plotting a histogram of costs for problems with 5,

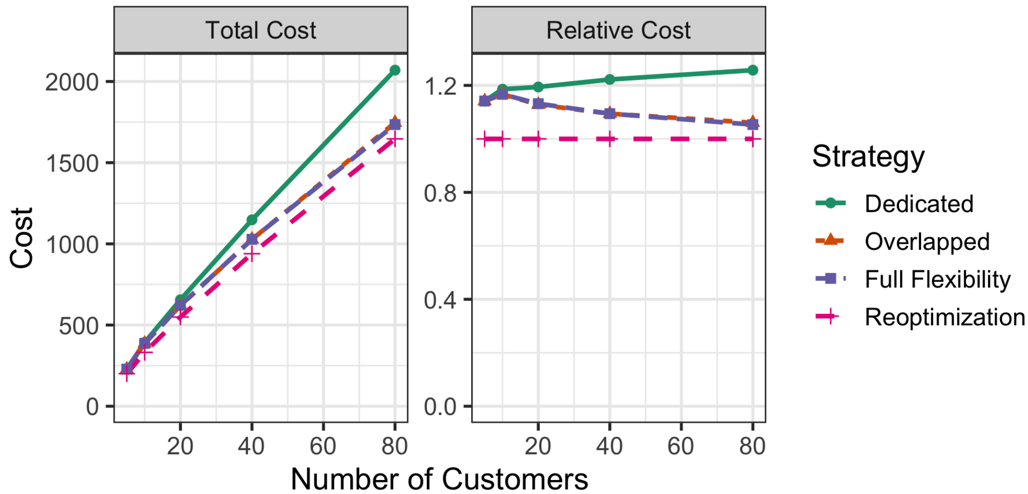


Figure 4.1: Average cost in the baseline scenario presented as (a) total cost and (b) cost relative to reoptimization

20, and 80 customers. We observe that the histograms roughly resemble a normal distribution. For simulations with 80 customers, dedicated routing exhibits a median of 2,060 and standard deviation of 175 while overlapped routing yields a lower median and standard deviation of 1,741 and 117, respectively. These results suggest that overlapped routing can decrease both the expected cost and the cost volatility of serving customers with stochastic demands.

We can also compare the performance of overlapped and dedicated routing for a given instance. Table 4.2 lists the percent of instances in which overlapped routing costs (i) more than, (ii) less than, and (iii) the same as dedicated routing. For instances with 5 customers, costs for dedicated routing and overlapped routing are always equal since the single extended route is identical to the primary route in the baseline scenario. However, for instances with 10 customers, overlapped routing is lower cost or higher cost with about equal probability. To explain the cases with higher cost, a vehicle $j = 1$ may travel extra distance to support customers shared with vehicle $j = 2$, but in a network with only two primary routes, this extra travel might not offset any depot trips. Put differently, in small flexible networks, savings in radial cost may not offset any additional circular cost. Finally, with a problem size of 20 or larger, overlapped routing is very likely to yield a lower cost than dedicated

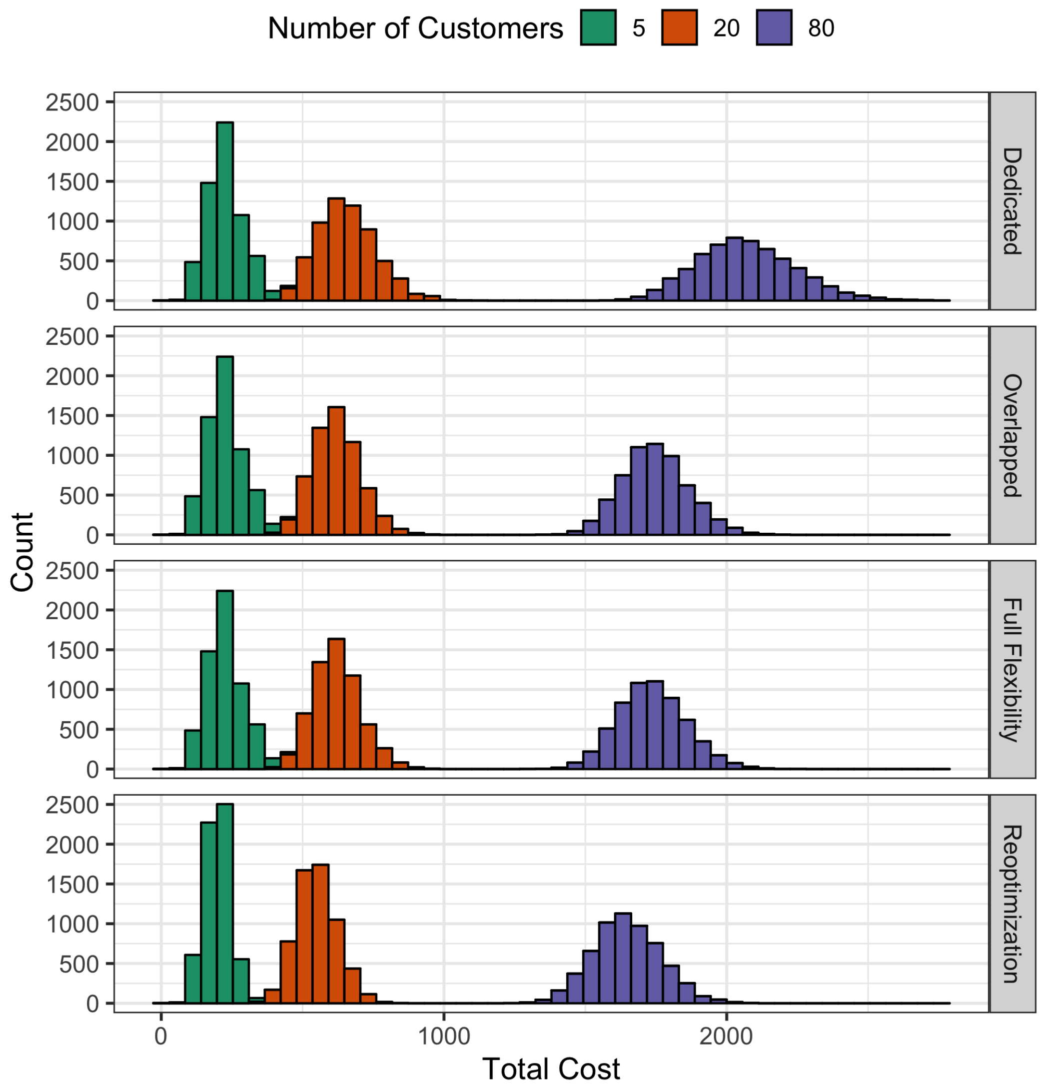


Figure 4.2: Histogram of routing costs in the baseline scenario

routing for any given instance, reaching near certain savings for networks with 80 customers.

To better understand the overlapped strategy's cost-saving mechanism, we separate total cost into its radial and circular components, illustrated in Figure 4.3. Table 4.3 states the corresponding total costs as well as radial cost's share of the total by routing strategy and problem size. We see that in the three strategies presented, the radial share of total cost increases with the number of customers. Under dedicated routing, for example, the radial cost share increases from 40% for 5 customers to

Customers	Overlapped Routing (Relative to Dedicated Routing)		
	Higher Cost	Equal Cost	Lower Cost
5	0%	100%	0%
10	46.2%	6.92%	46.9%
20	28.9%	0.03%	71.0%
40	7.95%	0%	92.0%
80	0.08%	0%	99.9%

Table 4.2: Percent of instances in which overlapped routing yields higher cost, equal cost, or lower cost compared to dedicated routing in the baseline scenario. Note: Rows may not sum to 100% due to rounding.

75% for 80 customers. The radial share for overlapped routing, on the other hand, increases from 40% to 64%, which with a lower total cost equates to a 28% radial cost savings relative to dedicated routing. Overall, the increasing radial shares suggest that trips to and from the depot drive transportation costs as network size grows.

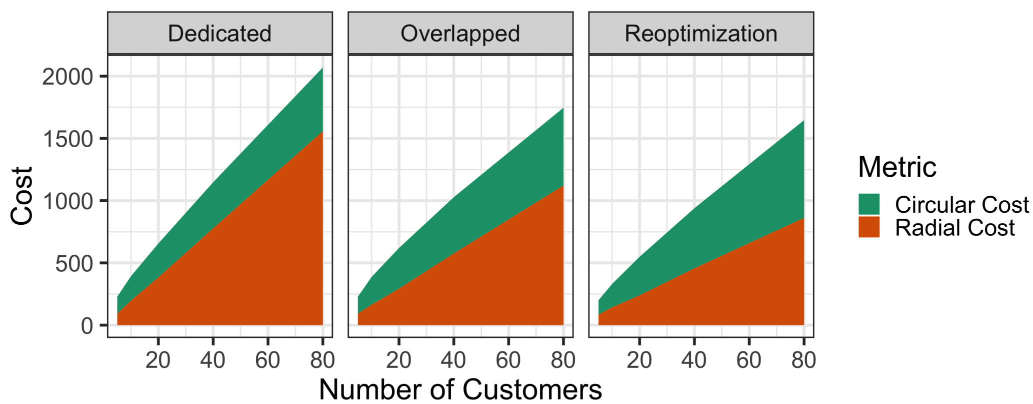


Figure 4.3: Average circular and radial costs in the baseline scenario

We can alternatively can capture the radial cost savings through differences in the number of trips needed for the fleet to meet all customer demands. Table 4.4 presents the average trip counts for each problem size and strategy, including full flexibility. The trip count includes vehicles' initial departures from the depot plus any refill trips. For problems with 5 and 10 customers, the fleet performs similar numbers of trips in all routing strategies. However, with 20 customers, dedicated routing requires roughly

Customers	Dedicated		Overlapped		Reoptimization	
	Total	Radial Share	Total	Radial Share	Total	Radial Share
5	229	40%	229	40%	200	43%
10	393	50%	387	42%	332	43%
20	656	58%	620	47%	549	44%
40	1,148	68%	1,028	56%	940	49%
80	2,070	75%	1,746	64%	1,647	52%

Table 4.3: Average radial share of total cost in the baseline scenario

one more trip than does reoptimization. Then, dedicated routing requires about 3 additional trips for 40 customers and ultimately 6 additional trips for 80 customers. Meanwhile, overlapped routing almost matches full flexibility and reoptimization in trip count across all problem sizes. Figure 4.4 illustrates dedicated routing's dramatic divergence from the other three strategies. Also, note that while full flexibility yields slightly fewer trips than does reoptimization (14.49 versus 14.52, respectively), full flexibility is still the higher cost strategy as reported in Table 4.1 above.

Routing Strategy	Number of Customers				
	5	10	20	40	80
Dedicated	1.28	2.55	5.13	10.27	20.57
Overlapped	1.28	2.20	4.02	7.65	14.90
Full Flexibility	1.28	2.20	3.97	7.48	14.49
Reoptimization	1.28	2.20	3.97	7.48	14.52

Table 4.4: Average number of trips in the baseline scenario

Understandably, all strategies require increasing total costs to meet the demands of growing numbers of customers. However, the baseline analysis reveals that trips to and from the depot contribute to a growing share of total costs as the number of customers increases. We observe that both the overlapped routing and full flexibility strategies can partly mitigate the increasing radial cost since the customer sharing strategy sometimes prevent a refill trip elsewhere in the network. Even more, overlapped routing performs almost equally to full flexibility though their relative per-

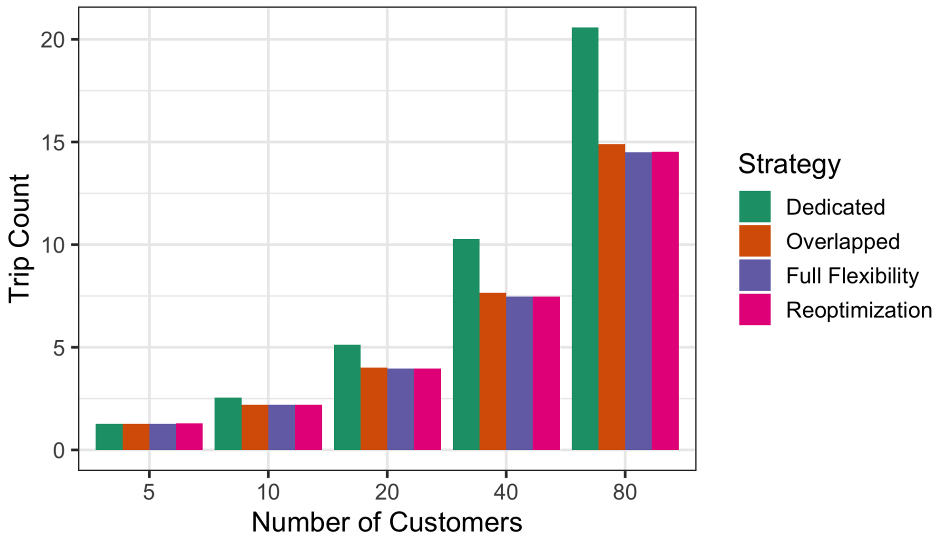


Figure 4.4: Average number of trips in the baseline scenario

formance may vary with the scenario design. Finally, in comparing the fixed routing strategies to reoptimization, we see that dedicated routing becomes relatively more expensive while overlapped routing and full flexibility grow closer to reoptimization in cost. In fact, as discussed in Section 3.3, we know that for infinitely large problems, the ratio between overlapped routing and reoptimization does ultimately converge to a constant lower bound.

4.3 Scenario Analysis

To understand routing cost's sensitivity to different network parameters, we also run simulations for the scenarios summarized in Table 4.5. These additional scenarios are defined as variations on the baseline scenario with changes to overlap size, vehicle capacity, route length, or demand distribution. In this section, we analyze outcomes for each type of parameter change and compare the additional scenario results to the baseline.

Scenario	Route Size, N'	Overlap Size, k	Vehicle Capacity, Q	Demand Distribution, D
Baseline	5	5	20	$D_i \sim \text{Uniform}\{0, 8\}$
Medium Overlap	–	3	–	–
Small Overlap	–	1	–	–
High Capacity	–	–	25	–
Low Capacity	–	–	15	–
Short Route	2	2	8	–
Long Route	10	10	40	–
Binomial Demand	–	–	–	$D_i \sim \text{Binomial}(8, 0.5)$
Stochastic Customers	–	–	–	$\mathrm{D}_i = \begin{cases} 0 & \text{w.p. } 0.5 \\ 8 & \text{w.p. } 0.5 \end{cases}$

Table 4.5: List of scenarios. Cells with dashes indicate the parameter is the same as in the baseline scenario.

Varying Overlap Size

We first analyze the effect of varying overlap size k in the overlapped routing strategy. Recall that the overlap size refers to the number of customers shared between two adjacent vehicle routes. In the baseline, overlap size equals primary route size for all vehicles, which means $k = N' = 5$, Then, the extended route for vehicle $j < M$ under overlapped routing contains the 5 customers in primary routes j and the 5 customers in adjacent primary route $j+1$. However, in two new scenarios we consider a smaller overlap size, specifically $k = 3$ (three shared customers out of 5) in the medium overlap scenario and $k = 1$ (1 shared customer out of 5) in the small overlap scenario. The baseline, medium overlap, and small overlap scenarios differ only in their extended routes, so they should exhibit different overlapped routing costs but identical dedicated routing costs.

Table 4.6 presents the ratio of the average overlapped routing cost to the average dedicated routing cost. For 80 customers, overlapped routing achieves 8% savings over dedicated routing in the small overlap scenario. Savings increase to 12% with medium overlap and 16% with the full baseline overlap. In comparing these scenarios, we find that much of the baseline savings is achieved with just one overlapped customer. In fact, for all problem sizes, the marginal savings decrease as overlap size increases from $k = 1$ to 3 to 5. Intuitively, increasing the number of shared customers among vehicles likely increases coordination time and cost, so our finding that much of the gains from flexible routing can be achieved with minimal investment could be very important in practice.

Scenario	Number of Customers				
	5	10	20	40	80
Small Overlap ($k = 1$)	1.00	0.99	0.98	0.95	0.92
Medium Overlap ($k = 3$)	1.00	0.99	0.96	0.91	0.88
Baseline ($k = 5$)	1.00	0.98	0.94	0.90	0.84

Table 4.6: Average cost of overlapped routing relative to dedicated routing in the small overlap, medium overlap, and baseline scenarios. Note: Ratios for all scenarios use the baseline scenario's computed dedicated cost. Dedicated costs in the medium and small overlap scenarios show minor variation from randomness.

Varying Capacity

We next analyze the impact of different vehicle capacities. In the baseline scenario, we set vehicle capacity Q equal to the primary route's total expected demand. Therefore, for primary routes with 5 customers each with demand uniformly distributed between 0 and 8 and equal to 4 in expectation, all vehicles have a capacity of 20 units. Now, for a low capacity scenario, we decrease capacity by 25% to 15 units and for a high capacity scenario, we increase capacity by 25% to 25 units. All other network parameters remain unchanged from the baseline scenario, so we can think of these scenarios as simply using smaller or larger trucks, respectively, to execute the same routes.

Table 4.7 presents the total routing costs for the low capacity and high capacity scenarios relative to the baseline. As the number of customers grows, relative costs for all routing strategies increase for the low capacity scenario but decrease for the high capacity scenario.[2] This result is expected since a smaller vehicle requires more

[2]We suspect that the cost ratio above 1.00 for $N = 5$ in the high capacity scenario is due to

trips from the depot to serve the same amount of demand as a larger vehicle.

We also observe that overlapped routing almost matches the relative costs under full flexibility, and both flexible strategies moderate the effect of the low capacity and high capacity scenarios. For example, for most of the problem sizes, the cost of dedicated routing in the low capacity scenario relative to the baseline is higher than the cost of overlapped routing in the low capacity scenario relative to the baseline. Meanwhile, the cost of dedicated routing in the high capacity scenario relative to the baseline is *lower than* overlapped routing's relative cost. These numbers suggest the value of flexibility is higher in settings where more trips are needed (e.g., when vehicles are low capacity), a finding consistent with our discussion in section 4.2 on customer sharing as a strategy to prevent refill trips and decrease radial cost.

Scenario	Routing Strategy	Number of Customers				
		5	10	20	40	80
Low Capacity	Dedicated	1.12	1.13	1.16	1.19	1.21
	Overlapped	1.12	1.11	1.14	1.16	1.19
	Full Flexibility	1.12	1.11	1.14	1.16	1.20
	Reoptimization	1.07	1.11	1.16	1.19	1.22
High Capacity	Dedicated	1.01	0.93	0.91	0.88	0.87
	Overlapped	1.01	0.95	0.91	0.90	0.89
	Full Flexibility	1.01	0.95	0.92	0.90	0.88
	Reoptimization	0.98	0.94	0.91	0.88	0.87

Table 4.7: Average cost in the high capacity and low capacity scenarios relative to the baseline scenario

Varying Route Size

We also test different numbers of customers for the primary routes. In the baseline scenario, each primary route has $N' = 5$ customers. We now create a short route scenario in which each primary route has $N' = 2$ customers as well as a long route scenario in which each primary route has $N' = 10$ customers. We also adjust other

randomness in the simulated demands across scenarios.

network parameters to maintain relationships consistent with the baseline. Specifically, for the short route we decrease overlap size to $k = 2$ to equal the new primary route length, and we decrease vehicle capacity to $Q = 8$ to equal the new expected primary route demand. Conversely, for the long route scenario we increase overlap size to $k = 10$ and capacity to $Q = 40$.

Figure 4.5 illustrates the radial and circular components of total cost for each scenario under our three main routing strategies as the number of customers grows. Across scenarios, total costs decrease as route length increases, with much of the savings coming from a decreasing radial cost since fewer vehicle trips are needed to serve the same number of customers. See Table 4.8 for average trip counts, including for full flexibility. Comparing strategies within scenarios, overlapped routing shows a lower radial cost but higher circular cost for a net decrease in total cost relative to dedicated routing. Reoptimization slightly outperforms overlapped routing and remains the lowest cost strategy in all scenarios.

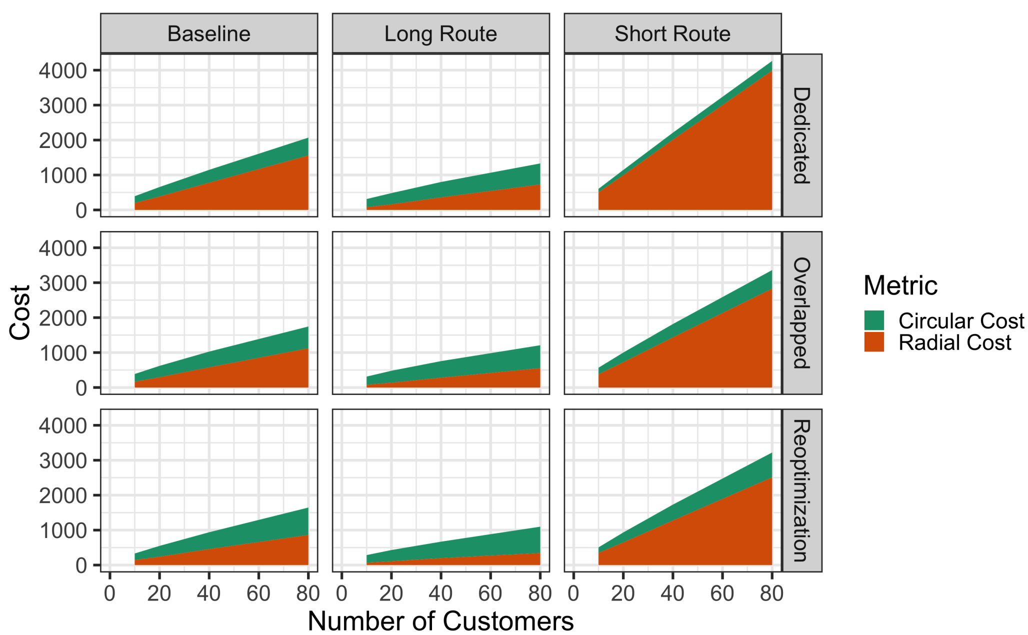


Figure 4.5: Circular and radial costs by routing strategy in the long route, short route, and baseline scenarios

Scenario	Routing Strategy	Number of Customers			
		10	20	40	80
Baseline	Dedicated	2.55	5.13	10.27	20.57
	Overlapped	2.20	4.02	7.65	14.90
	Full Flexibility	2.20	3.97	7.48	14.49
	Reoptimization	2.20	3.97	7.48	14.52
Long Route	Dedicated	1.22	2.45	4.90	9.84
	Overlapped	1.22	2.15	4.01	7.60
	Full Flexibility	1.22	2.15	3.99	7.50
	Reoptimization	1.22	2.15	3.99	7.51
Short Route	Dedicated	6.54	13.10	26.24	52.54
	Overlapped	4.95	9.56	18.80	37.32
	Full Flexibility	4.81	9.18	17.94	35.49
	Reoptimization	4.82	9.19	17.95	35.53

Table 4.8: Average number of trips in the long route, short route, and baseline scenarios

Other Demand Distributions

Lastly, we consider scenarios in which customers face alternate demand distributions. In the baseline scenario, customer i's integer demands are uniformly distributed between 0 and 8, that is $D_i \sim \text{Uniform}\{0, 8\}$. In a second scenario, customers instead exhibit binomially distributed demand with $D_i \sim \text{Binomial}(8, 0.5)$. Finally, we define a stochastic customer scenario in which customers exhibit demand of 0 or 8 with equal probability. Note that all three of these scenarios assume customer demands are independently and identically distributed with expected demand $\mathbb{E}[D_i] = 4$. The demand variance is 5.33 in the baseline scenario, 2 in the binomial demand scenario, and 16 in the stochastic customers scenario.

Table 4.9 presents the total cost in each strategy, scenario, and problem size. For all routing strategies, we observe that the stochastic customer scenario is generally the lowest cost scenario, followed by the baseline scenario, and then the binomial demand scenario. Table 4.10 simply rearranges the costs from Table 4.9 to highlight the relative performance of overlapped routing within each scenario. In all scenarios, overlapped routing almost perfectly matches the cost of full flexibility. We also observe that overlapped routing approaches reoptimization in cost most rapidly in the stochastic customer scenario. Specifically, overlapped routing costs in the stochastic customer scenario decrease from 19% above reoptimization for 5 customers to 3% above for 80 customers, yielding a 16 point change. In comparison, relative costs decrease by 13 points under binomial demand and by only 8 points in the baseline.

Routing Strategy	Scenario	Number of Customers				
		5	10	20	40	80
Dedicated	Baseline	229	393	656	1,148	2,070
	Binomial Demand	266	436	745	1,308	2,330
	Stochastic Customers	180	334	594	1,104	2,070
Overlapped	Baseline	229	387	620	1,028	1,746
	Binomial Demand	266	422	690	1,132	1,935
	Stochastic Customers	180	324	546	958	1,697
Full Flexibility	Baseline	229	387	622	1,028	1,735
	Binomial Demand	266	422	691	1,133	1,934
	Stochastic Customers	180	325	548	956	1,686
Reoptimization	Baseline	200	332	549	939	1,647
	Binomial Demand	226	365	611	1,052	1,845
	Stochastic Customers	152	289	512	911	1,648

Table 4.9: Average cost in the binomial demand, stochastic customer, and baseline scenarios

Finally, for a more granular understanding of the demand scenarios, we can also compare performance for individual instances. Figure 4.6 shows the percent of demand and customer location instances in which overlapped routing costs the same, more, or less than dedicated routing. In all scenarios, overlapped routing is more expensive less than half of the time and more consistently becomes the less expensive strategy as the number of customers grows. Additionally, a greater share of instances see a higher cost for flexible operations when customer demands are uniform as in the baseline scenario. This observation aligns with the baseline's higher relative costs as presented in Table 4.10.

Scenario	Overlapped Cost	Number of Customers				
		5	10	20	40	80
Baseline	Total	229	387	620	1,028	1,746
	Rel. to Dedicated	1.00	0.98	0.94	0.90	0.84
	Rel. to Full Flexibility	1.00	1.00	1.00	1.00	1.01
	Rel. to Reoptimization	1.14	1.17	1.13	1.09	1.06
Binomial Demand	Total	266	422	690	1,132	1,935
	Rel. to Dedicated	1.00	0.97	0.93	0.87	0.83
	Rel. to Full Flexibility	1.00	1.00	1.00	1.00	1.00
	Rel. to Reoptimization	1.18	1.16	1.13	1.08	1.05
Stochastic Customers	Total	180	324	546	958	1,697
	Rel. to dedicated	1.00	0.97	0.92	0.87	0.82
	Rel. to Full Flexibility	1.00	1.00	1.00	1.00	1.01
	Rel. to Reoptimization	1.19	1.12	1.07	1.05	1.03

Table 4.10: Average total and relative costs of overlapped routing in the baseline, binomial demand, and stochastic customer scenarios

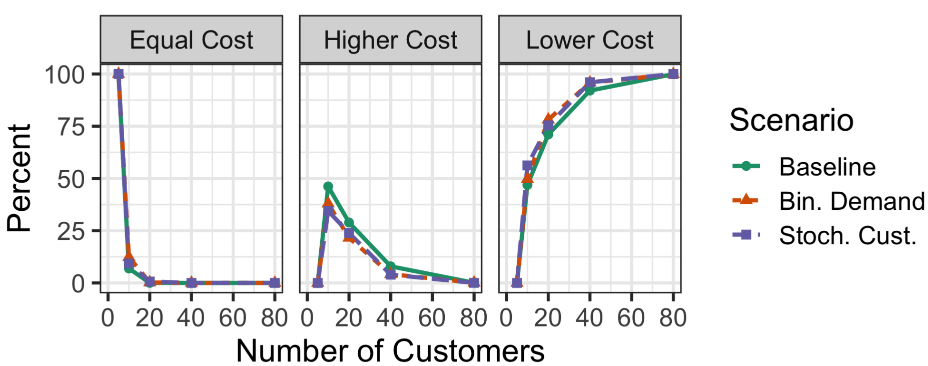


Figure 4.6: Percent of instances in which the cost of overlapped routing is equal to, higher than, or lower than the cost of dedicated routing in the binomial demand, stochastic customer, and baseline scenarios

Chapter 5

Conclusion

For this thesis, we modeled and computationally evaluated a flexible routing strategy for the VRPSD. We considered a setting in which capacitated delivery vehicles execute daily routes to deliver a single product type to retail customers. Routes are designed in advance around expected or forecasted order quantities. However, customer demands change from day to day, so the a priori routes may not be cost efficient if drivers need to return to the depot to restock to serve new demand. To mitigate increases in transportation cost, we proposed a customer sharing scheme – the overlapped routing strategy – in which neighboring routes are designed with some overlap in assigned customers. The route assignment structure was inspired by the open chain bipartite graph analyzed by Jordan and Graves (1995), Simchi-Levi and Wei (2012), and others in the manufacturing process flexibility literature. In executing daily delivery routes, the built-in redundancy in customer assignments provides a central planner with the flexibility to assign drivers to subsets of their overlapped fixed routes in response to realized customer demands.

We also defined alternative routing models with which we could compare the overlapped routing strategy. Dedicated routing is an inflexible policy in which drivers individually execute routes with exclusive sets of customers. Routing with full flexibility allows any driver to serve any customer as long as the executed routes align with a predetermined customer sequence. Finally, under reoptimization, drivers are not constrained to a priori routes and instead are freely assigned to customers to minimize transportation cost for each given set of customer locations and demands.

After defining our models and their corresponding delivery recourse policies, we constructed a baseline scenario and simulated the costs of filling randomly drawn customer demands at randomly distributed locations under the four alternative strategies. We specifically considered problems with between 5 and 80 customers. For the baseline scenario, we found that dedicated routing steeply increased in cost from 14% more costly than reoptimization for 5 customers to 26% more costly with 80 customers. Overlapped routing, on the other hand, decreased from 14% above reoptimization for 5 customers to just 6% above with 80 customers. In addition, despite its restrictions on shared customers, overlapped routing nearly matched full flexibility in cost. We also separated transportation cost into its radial and circular components and found that in our setting trips to and from the depot drive transportation costs as the problem size grows. Importantly, overlapped routing harnesses surplus capacity within the fleet to mitigate the growing radial costs from necessary refill trips.

To understand the sensitivity of our results to different network parameters, we also considered variations on the baseline scenario with various degrees of customer sharing, route lengths, vehicle capacities, and customer demand distributions. This scenario analysis confirmed that the baseline trends generally hold under other network designs. We also found that much of overlapped routing's cost savings comes from the first customer that is shared between each pair of neighboring routes. Put differently, a little bit of flexibility goes a long way. This observation aligns with the consensus in the manufacturing process flexibility literature that a long chain design (some flexibility) in the plant-product bipartite assignment network performs comparatively to a fully connected network (full flexibility). This finding also has practical significance since implementing flexible operations – whether coordinating a customer sharing scheme or installing redundant manufacturing capabilities – likely requires upfront cost and investment.

In future work, the overlapped routing model could be adapted to several other problem settings. For example, the model could be extended to enable flexibility across time such that customers are shifted between routes executed on different days. Alternatively, the model could be adapted to spatial divisions with overlapping service areas rather than overlapping customer sequences. Researchers could also consider the VRP with other sources of stochasticity (e.g., uncertain travel times) and greater levels of complexity (e.g., delivery time windows, heterogeneous vehicles, multiple product types, or a multi-echelon network). Even more, as our study uses a simplified distribution model with simulated data, future research could include an empirical study (see, e.g., Erera et al. (2009)) or possibly a pilot study with an industry partner to assess real-world performance as well as the upfront costs of enabling flexibility in practice. Finally, we refer the reader to Ledvina et al. (2020) for analysis of overlapped routing's asymptotic performance along with a discussion of possible research extensions on the theory behind flexible routing.

In summary, and to reiterate the main takeaways from our computational study, overlapped routing mitigates the costs of meeting unexpected customer demands while still preserving much of the consistency of dedicated routing and other fixed routing strategies. Additionally, designing routes with even a little customer sharing harnesses much of the potential savings from flexible routing schemes. Finally, in many settings, frequent route reoptimization may be undesired or unrealistic. Therefore, predesigning routes with some flexibility through customer sharing could be immensely practical and beneficial in real-world distribution systems.

Bibliography

Ak, A. and Erera, A. L. (2007). A paired-vehicle recourse strategy for the vehicle-routing problem with stochastic demands. *Transportation science*, 41(2):222–237.

Asadpour, A., Wang, X., and Zhang, J. (2020). Online resource allocation with limited flexibility. *Management Science*, 66(2):642–666.

Bartholdi III, J. J., Platzman, L. K., Collins, R. L., and Warden III, W. H. (1983). A minimal technology routing system for meals on wheels. *Interfaces*, 13(3):1–8.

Bertsimas, D. J. (1992). A vehicle routing problem with stochastic demand. *Operations Research*, 40(3):574–585.

Cleophas, C., Cottrill, C., Ehmke, J. F., and Tierney, K. (2019). Collaborative urban transportation: Recent advances in theory and practice. *European Journal of Operational Research*, 273(3):801 – 816.

Defryn, C., Sörensen, K., and Cornelissens, T. (2016). The selective vehicle routing problem in a collaborative environment. *European Journal of Operational Research*, 250(2):400 – 411.

Dror, M., Laporte, G., and Trudeau, P. (1989). Vehicle routing with stochastic demands: Properties and solution frameworks. *Transportation Science*, 23(3):166–176.

Erera, A. L., Savelsbergh, M., and Uyar, E. (2009). Fixed routes with backup vehicles for stochastic vehicle routing problems with time constraints. *Networks: An International Journal*, 54(4):270–283.

Fernández, E., Roca-Riu, M., and Speranza, M. G. (2018). The shared customer collaboration vehicle routing problem. *European Journal of Operational Research*, 265(3):1078 – 1093.

Gansterer, M. and Hartl, R. F. (2018). Collaborative vehicle routing: A survey. *European Journal of Operational Research*, 268(1):1 – 12.

Gendreau, M., Jabali, O., and Rei, W. (2014). Chapter 8: Stochastic vehicle routing problems. In *Vehicle Routing: Problems, Methods, and Applications, Second Edition*, pages 213–239. SIAM.

Gendreau, M., Jabali, O., and Rei, W. (2016). 50th anniversary invited article—future research directions in stochastic vehicle routing. *Transportation Science*, 50(4):1163–1173.

Guajardo, M. and Rönnqvist, M. (2016). A review on cost allocation methods in collaborative transportation. *International Transactions in Operational Research*, 23(3):371–392.

Jaillet, P. (1988). A priori solution of a traveling salesman problem in which a random subset of the customers are visited. *Operations research*, 36(6):929–936.

Jordan, W. and Graves, S. (1995). Principles on the benefits of manufacturing process flexibility. *Management Science*, 41(4):577–594.

Ledvina, K., Qin, H., Simchi-Levi, D., , and Wei, Y. (2020). A new approach for vehicle routing with stochastic demand: Combining route assignment with process flexibility. *SSRN*, page 40.

Lei, H., Laporte, G., and Guo, B. (2012). The vehicle routing problem with stochastic demands and split deliveries. *INFOR: Information Systems and Operational Research*, 50(2):59–71.

Lyu, G., Cheung, W.-C., Chou, M. C., Teo, C.-P., Zheng, Z., and Zhong, Y. (2019). Capacity allocation in flexible production networks: Theory and applications. *Management Science*, 65(11):5091–5109.

Perron, L. and Furnon, V. (2019). Or-tools. https://developers.google.com/optimization/.

Quintero-Araujo, C., Gruler, A., and Juan, A. (2016). Quantifying potential benefits of horizontal cooperation in urban transportation under uncertainty: A simheuristic approach. volume 9868, pages 280–289.

Sanchez, M., Pradenas, L., Deschamps, J.-C., and Parada, V. (2016). Reducing the carbon footprint in a vehicle routing problem by pooling resources from different companies. *Netnomics*, 17(1):29–45.

Secomandi, N. and Margot, F. (2009). Reoptimization approaches for the vehicle-routing problem with stochastic demands. *Operations research*, 57(1):214–230.

Simchi-Levi, D. (2010). *Operations Rules: Delivering Customer Value through Flexible Operations*. The MIT Press.

Simchi-Levi, D. and Wei, Y. (2012). Understanding the performance of the long chain and sparse designs in process flexibility. *Operations Research*, 60(5):1125–1141.

Xu, Z., Zhang, H., Zhang, J., and Zhang, R. Q. (2020). Online demand fulfillment under limited flexibility. *Management Science*.

Appendix A

Simulation Code

As explained in Section 4.1, we created a Python program to simulate and evaluate the costs of dedicated routing, overlapped routing, full flexibility, and reoptimization. This appendix presents our simulation code. Section A.1 contains code to define new scenarios and run the simulations while Section A.2 includes all supporting code including functions to execute routing algorithms and calculate transportation costs for realized routes.

In addition, the GitHub repository flexible-routing[1] contains all project files including the simulation code, output data, and an R script for creating summary figures from the data. The project's readme file explains how to update and run the simulation code. The repository also includes a Jupyter notebook that lets the user specify network parameters (number of customers, route size, vehicle capacity, etc.) and walks through and illustrates the routing strategies for a randomly generated customer and demand instance. This notebook was used to create the example presented in Section 3.2.

A.1 Main Function

```
import pandas as pd
import time
from supporting import *

```

[1]https://github.com/kledvina/flexible-routing

```python
# GLOBAL VARIABLES
field_width = 100 # Customer location has x-coordinate in (0, field_width)
field_height = 100 # Customer location has y-coordinate in (0, field_height)
depot_x = 50 # Depot x-coordinate
depot_y = 50 # Depot y-coordinate


def create_report(inst, scenario, strategy, segments):
    """Gets costs and creates new entry for simulation results"""
    trips = [scenario, inst.size, strategy, 'trip count', get_trip_count(segments)]
    radial = [scenario, inst.size, strategy, 'radial cost', sum([get_radial_cost(inst, seg) for seg in segments])]
    circular = [scenario, inst.size, strategy, 'circular cost', sum([get_circular_cost(inst, seg) for seg in segments])]
    total = [scenario, inst.size, strategy, 'total cost', sum([get_total_cost(inst, seg) for seg in segments])]
    return pd.DataFrame(data=[trips, radial, circular, total],
                        columns=['Scenario', 'Number of Customers', 'Routing Strategy', 'Metric', 'Value'])


def simulate(scenario, problem_sizes, capacity, route_size, overlap_size, cust_sims, dem_sims):

    # Start timers
    start = time.time()
    pt, dt, ot, ft, rt, st = 0, 0, 0, 0, 0, 0

    # Create timestamp for backup outputs
    timestamp = time.strftime("%Y-%m-%d_%H-%M-%S")

    # Print simulation parameters
    print('--- SIMULATION PARAMETERS ---')
    print('Start time:', timestamp)
    print('Scenario Name:', scenario)
    print('Problem sizes:', problem_sizes)
    print('Vehicle capacity:', capacity)
    print('Primary route size:', route_size)
    print('Overlap size:', overlap_size)
    print('Customer instances:', cust_sims)
    print('Demand instances:', dem_sims)
    print()

    # Initialize arrays to store results
    sim_results = pd.DataFrame(columns=['Scenario', 'Number of Customers', 'Routing Strategy', 'Metric', 'Value'])

    # Loop through each problem size
    for num_cust in problem_sizes:

        print('Starting problems of size {}'.format(num_cust))

        new_pt = time.time()
        # Create all customer and demand instances for this problem size
        print('Creating customer instances')
        instances = create_instances(scenario, num_cust, cust_sims, dem_sims)

        # Find cost minimizing starting customer / tour sequence for each set of customer locations
        print('Finding best tour across demand sets')
        for row in instances:
            inst = row[0]  # customer instance
```

```python
        primary_routes = get_primary_routes(inst, route_size)
        extended_routes = get_extended_routes(inst, route_size, overlap_size)
        # set average cost-minimizing tour
        set_best_tours(row, primary_routes, extended_routes, capacity, route_size, overlap_size)
    pt += time.time() - new_pt

    # Loop through instances and find instance cost for different strategies

    for i in range(cust_sims):
        for j in range(dem_sims):

            # Get instance from array
            inst = instances[i][j]

            try:
                # Solve dedicated routing
                new_dt = time.time()
                primary_routes = get_primary_routes(inst, route_size)
                segments = create_full_trips(inst, primary_routes, capacity)
                new_rows = create_report(inst, scenario, 'dedicated', segments)
                sim_results = sim_results.append(new_rows, ignore_index=True)
                dt += time.time() - new_dt

                # Solve overlapped routing
                new_ot = time.time()
                primary_routes = get_primary_routes(inst, route_size)
                extended_routes = get_extended_routes(inst, route_size, overlap_size)
                segments = implement_k_overlapped_alg(inst, primary_routes, extended_routes, capacity, route_size, overlap_size)
                new_rows = create_report(inst, scenario, 'overlapped', segments)
                sim_results = sim_results.append(new_rows, ignore_index=True)
                ot += time.time() - new_ot

                # Solve fully flexible routing
                new_ft = time.time()
                segments = create_full_trips(inst, [inst.tour[1:]], capacity)
                new_rows = create_report(inst, scenario, 'fully flexible', segments)
                sim_results = sim_results.append(new_rows, ignore_index=True)
                ft += time.time() - new_ft

                # Solve reoptimization
                new_rt = time.time()
                segments = solve_SDVRP(inst, capacity)
                new_rows = create_report(inst, scenario, 'reoptimization', segments)
                sim_results = sim_results.append(new_rows, ignore_index = True)
                rt += time.time() - new_rt

            except Exception as e:
                print('ERROR: {}'.format(e))
                print('WARNING: Simulation failed to complete. Printing info for last Instance and returning Instance object.')
                print(inst.demands)
                print(inst.xlocs)
                print(inst.ylocs)
                print(inst.tour)
                return inst

        # Save backup of data
        new_st = time.time()
        sim_results.to_csv('temp/backup_{}.csv'.format(timestamp))
```

```
            st += time.time() - new_st

            end = time.time()
            print('Customer instance {} complete. Time elapsed: {:.2f} min'.format(i + 1, (end-start)/60))

        print('Problems of size {} complete'.format(num_cust))

    print('Simulation complete.')
    print()
    print('---- RUNTIME BREAKDOWN ----')
    print('Setup: {:.2f} min'.format(pt/60))
    print('Dedicated: {:.2f} min'.format(dt/60))
    print('Overlapped: {:.2f} min'.format(ot/60))
    print('Full Flex.: {:.2f} min'.format(ft/60))
    print('Reoptimization: {:.2f} min'.format(rt/60))
    print('Saving: {:.2f} min'.format(st/60))

    return sim_results


if __name__ == "__main__":

    # --- Baseline simulation ---
    # Demand uniformly distributed in [0,8]
    # Route size: 5
    # Overlap size: 5
    results = simulate(scenario = 'baseline', problem_sizes = [5,10,20,40,80], capacity = 20, route_size = 5, overlap_size = 5, cust_sims = 30, dem_sims = 200)

    # Calculate summary statistics over instances
    means = results.groupby(['Scenario', 'Number of Customers', 'Routing Strategy', 'Metric'])['Value'].mean()
    sds = results.groupby(['Scenario', 'Number of Customers', 'Routing Strategy', 'Metric'])['Value'].std()
    ci_low = results.groupby(['Scenario', 'Number of Customers', 'Routing Strategy', 'Metric'])['Value'].quantile(0.025)
    ci_high = results.groupby(['Scenario', 'Number of Customers', 'Routing Strategy', 'Metric'])['Value'].quantile(0.975)

    timestamp = time.strftime("%Y-%m-%d_%H-%M-%S")
    outfile = 'output/results_{}.xlsx'.format(timestamp)

    with pd.ExcelWriter(outfile) as writer:
        results.to_excel(writer, sheet_name = 'baseline')
        means.to_excel(writer, sheet_name = 'summary_mean')
        sds.to_excel(writer, sheet_name = 'summary_sds')
        ci_low.to_excel(writer, sheet_name = 'summary_ci_low')
        ci_high.to_excel(writer, sheet_name = 'summary_ci_high')
```

A.2 Supporting Functions

```
import numpy as np
import math
from ortools.constraint_solver import routing_enums_pb2
from ortools.constraint_solver import pywrapcp

```

```
# GLOBAL VARIABLES
field_width = 100 # Customer location has x-coordinate in (0, field_width)
field_height = 100 # Customer location has y-coordinate in (0, field_height)
depot_x = 50 # Depot x-coordinate
depot_y = 50 # Depot y-coordinate


class Instance():
    """A realized set of node locations and demands and the resulting routing characteristics."""

    def __init__(self, xlocs, ylocs, demands, solve_TSP=True):

        self.size = len(demands) - 1
        self.demands = demands
        self.xlocs = xlocs
        self.ylocs = ylocs
        self.distances = self.calc_distance_matrix()
        self.optimal_routes = 'None'
        self.tour = 'None'
        if solve_TSP:
            # self.tour = self.solve_TSP()
            # self.tour = self.TSP_heuristic()
            self.tour = self.solve_TSP()

    def calc_distance_matrix(self):
        """Returns a matrix with pairwise node distances"""
        distances = np.zeros((self.size + 1, self.size + 1), dtype=float)
        for i in range(self.size + 1):
            for j in range(self.size + 1):
                new_dist = math.sqrt((self.xlocs[i] - self.xlocs[j]) ** 2 + (self.ylocs[i] - self.ylocs[j]) ** 2)
                distances[i, j] = new_dist
        return distances

    def update_demands(self, demands):
        self.demands = demands

    def update_tour(self, tour):
        self.tour = tour

    def get_lowerbound(self, capacity):
        """Returns a theoretical lowerbound on the optimal routing cost"""
        return (2 / capacity) * sum([self.demands[i] * self.distances[0, i]
                                     for i in range(len(self.demands))])

    def get_fleet_size(self, route_size):
        """Returns the number of vehicles needed to visit all nodes given a fixed route size"""
        assert self.size % route_size == 0, "Number of customers must be evenly divisible by the route size."
        return int(self.size / route_size)

    def save_optimal_routes(self, route_list):
        self.optimal_routes = route_list

    def solve_TSP(self):

        def create_data_model():
            data = {}
            data['distance_matrix'] = self.distances
            data['num_vehicles'] = 1
            data['depot'] = 0
```

```python
            return data

        def get_tour(manager, routing, solution):
            index = routing.Start(0)
            plan_output = ''
            while not routing.IsEnd(index):
                plan_output += '{},'.format(manager.IndexToNode(index))
                previous_index = index
                index = solution.Value(routing.NextVar(index))
            plan_output += '{}'.format(manager.IndexToNode(index))
            as_list = plan_output.split(',')
            return [int(i) for i in as_list][:-1]  # excludes return to depot in tour

        # --- RUN PROGRAM ---

        # Instantiate the data problem.
        data = create_data_model()

        # Create the routing index manager.
        manager = pywrapcp.RoutingIndexManager(len(data['distance_matrix']),
                                               data['num_vehicles'], data['depot'])

        # Create Routing Model.
        routing = pywrapcp.RoutingModel(manager)

        def distance_callback(from_index, to_index):
            """Returns the distance between the two nodes."""
            # Convert from routing variable Index to distance matrix NodeIndex.
            from_node = manager.IndexToNode(from_index)
            to_node = manager.IndexToNode(to_index)
            return data['distance_matrix'][from_node][to_node]

        transit_callback_index = routing.RegisterTransitCallback(distance_callback)

        # Define cost of each arc.
        routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

        # Setting first solution heuristic.
        search_parameters = pywrapcp.DefaultRoutingSearchParameters()
        search_parameters.first_solution_strategy = (
            routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)

        # Solve the problem.
        solution = routing.SolveWithParameters(search_parameters)
        return get_tour(manager, routing, solution)


def get_trip_count(route_list):
    """Returns number of trips in route list"""
    assert type(route_list[0]) == list, "route_list must be a list of lists (routes)"
    count = 0
    for route in route_list:
        if route != []:
            count += 1
    return count


def get_circular_cost(inst, segment):
    """Returns the total distance of moving from node to node within the given segment"""
    if len(segment) == 0:
        return 0
```

```python
    else:
        return sum([inst.distances[segment[i], segment[i + 1]] for i in range(len(segment) - 1)])


def get_radial_cost(inst, segment):
    """Returns the total distance of trips to/from the depot at segment endpoints."""
    if len(segment) == 0:
        return 0
    else:
        return inst.distances[0, segment[0]] + inst.distances[0, segment[-1]]


def get_total_cost(inst, segment):
    """Returns sum of circular and radial costs for the given segment"""
    return get_circular_cost(inst, segment) + get_radial_cost(inst, segment)


def optimize(inst, capacity):
    def create_data_model(inst, capacity):
        data = {}
        data['distance_matrix'] = inst.distances
        data['demands'] = inst.demands
        data['vehicle_capacities'] = [capacity] * inst.size
        data['num_vehicles'] = sum(inst.demands)
        data['depot'] = 0
        return data

    def get_routes(solution, routing, manager):
        """Get vehicle routes from a solution and store them in an array."""
        # Get vehicle routes and store them in a two dimensional array whose
        # i,j entry is the jth location visited by vehicle i along its route.
        routes = []
        for route_nbr in range(routing.vehicles()):
            index = routing.Start(route_nbr)
            route = [manager.IndexToNode(index)]
            while not routing.IsEnd(index):
                index = solution.Value(routing.NextVar(index))
                route.append(manager.IndexToNode(index))
            routes.append(route)
        return routes

    def distance_callback(from_index, to_index):
        """Returns the distance between the two nodes."""
        # Convert from routing variable Index to distance matrix NodeIndex.
        from_node = manager.IndexToNode(from_index)
        to_node = manager.IndexToNode(to_index)
        return data['distance_matrix'][from_node][to_node]

    def demand_callback(from_index):
        """Returns the demand of the node."""
        # Convert from routing variable Index to demands NodeIndex.
        from_node = manager.IndexToNode(from_index)
        return data['demands'][from_node]

    # ---- RUN PROGRAM ----

    # Zero cost if no demands
    if all(dem == 0 for dem in inst.demands):
        return [[]]

    # Set up data model
```

```
    data = create_data_model(inst, capacity)

    # Create the routing index manager
    manager = pywrapcp.RoutingIndexManager(len(data['distance_matrix']), data['num_vehicles'], data['depot'])

    # Create routing model
    routing = pywrapcp.RoutingModel(manager)

    # Create and register a transit callback
    transit_callback_index = routing.RegisterTransitCallback(distance_callback)

    # Define cost of each arc
    routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

    # Add capacity constraint
    demand_callback_index = routing.RegisterUnaryTransitCallback(demand_callback)
    routing.AddDimensionWithVehicleCapacity(
        demand_callback_index,
        0,  # null capacity slack
        data['vehicle_capacities'],  # vehicle maximum capacities
        True,  # start cumul to zero
        'Capacity')

    # Setting first solution heuristic
    search_parameters = pywrapcp.DefaultRoutingSearchParameters()
    search_parameters.first_solution_strategy = (
        routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)

    # Solve the problem
    solution = routing.SolveWithParameters(search_parameters)
    all_routes = get_routes(solution, routing, manager)
    nonempty_routes = [route for route in all_routes if not all(i == 0 for i in route)]

    # Remove the depot from the optimal routes
    parsed_routes = [route[1:-1] for route in nonempty_routes]

    return parsed_routes


def solve_SDVRP(inst, capacity):
    """Creates equivalent demand/location instance with unit demand and solves the VRP with splittable demands"""
    # Create equivalent instance with unit demand customers
    split_xlocs = [[depot_x]] + [[inst.xlocs[i]] * inst.demands[i] for i in range(1, len(inst.demands))]
    split_xlocs = [v for sublist in split_xlocs for v in sublist]

    split_ylocs = [[depot_y]] + [[inst.ylocs[i]] * inst.demands[i] for i in range(1, len(inst.demands))]
    split_ylocs = [v for sublist in split_ylocs for v in sublist]

    split_demands = [[0]] + [[1] * inst.demands[i] for i in range(1, len(inst.demands))]
    split_demands = [v for sublist in split_demands for v in sublist]

    split_inst = Instance(split_xlocs, split_ylocs, split_demands, solve_TSP=False)

    # Solve VRP with unit demand customers
    vrp = optimize(split_inst, capacity)

    # Convert back to non-unit demand problem
```

```
    # to get SDVRP solution for original instance
    ids = [[i] * inst.demands[i] for i in range(1, len(inst.demands))]
    ids = [v for sublist in ids for v in sublist]
    if ids == []:
        return [[]]  # No routes
    else:
        sdvrp = [[ids[c - 1] for c in route] for route in vrp]

    return sdvrp


def get_primary_routes(inst, route_size):
    """Splits customer sequence into segments of 'route_size' number of customers.
    Requires that number of customers is evenly divisible by route_size."""
    assert inst.size%route_size == 0, "The number of customers must be evenly divisible by route_size."
    tour = inst.tour[1:]  # Exclude depot
    routes = []
    for i in range(0, len(tour), route_size):
        new_route = tour[i:i + route_size]
        routes.append(new_route)
    return routes


def get_extended_routes(inst, route_size, overlap_size):
    """Splits customer sequnce into segments of 'route_size + overlap_size' number of customers, where adjacent
    segments SHARE overlap_size number of customers. Requires that (i) number of customers is evenly divisible by route_size
    and (ii) overlap size is less than or equal to route_size."""
    assert inst.size % route_size == 0, "The number of customers must be evenly divisible by primary route size."
    assert overlap_size <= route_size, "Overlap size must be less than or equal to primary route size."
    tour = inst.tour[1:]
    routes = []
    for i in range(0, len(tour), route_size):
        new_route = tour[i:i + route_size + overlap_size]
        routes.append(new_route)
    return routes


def create_full_trips(inst, route_list, capacity, demand_filled=None):
    """Splits a sequence of customers into individual trips. Returns a list of lists."""

    assert type(route_list[0]) == list, "route_list must be a list of lists (routes)"

    # Dictionary for tracking remaining demand filled at all customers
    remaining_demand = dict([(inst.tour[i], inst.demands[inst.tour[i]]) for i in range(1, len(inst.tour))])

    segments = []
    for m in range(len(route_list)):
        i = 0
        seg_dict = {}  # demand filled on current trip
        vehicle_dict = dict(
            [(inst.tour[i], 0) for i in range(1, len(inst.tour))])  # total demand filled by vehicle on this route
        while i < len(route_list[m]):
            cust = route_list[m][i]
            for d in range(inst.demands[cust]):
```

```
                if demand_filled != None and sum(vehicle_dict.values()) == demand_filled[m]:
                    # Route's vehicle achieved its predetermined workload (if applicable)
                    # Force to end this route and move to next
                    i = len(route_list[m])
                    break

                elif sum(remaining_demand[c] for c in route_list[m]) == 0:
                    # Route is completed
                    # Force to end this route and move to next
                    i = len(route_list[m])
                    break

                elif sum(seg_dict.values()) == capacity:
                    # Vehicle is at capacity
                    # End current trip, and begin a new trip within this route
                    segments.append(list(seg_dict))
                    seg_dict = {cust: 1}
                    vehicle_dict[cust] += 1
                    remaining_demand[cust] -= 1

                elif remaining_demand[cust] > 0:
                    if cust not in seg_dict:
                        # Begin service
                        seg_dict[cust] = 1
                    else:
                        # Continue service
                        seg_dict[cust] += 1
                    vehicle_dict[cust] += 1
                    remaining_demand[cust] -= 1

            i += 1  # Moves to next customer

        # Append route's last segment
        segments.append(list(seg_dict))

    return segments


def implement_k_overlapped_alg(inst, primary_routes, extended_routes, capacity, route_size, overlap_size):
    """Implement's general k-overlapped routing algorithm. Returns list of realized vehicle routes."""
    assert type(primary_routes[0]) == list, "primary_routes must be a list of lists (routes)"
    assert type(extended_routes[0]) == list, "extended_routes must be a list of lists (routes)"

    # Get overlapped segments (note that last route does not have any shared customers at the route's end)
    overlapped_segments = []
    for j in range(len(primary_routes) - 1):
        new_segment = [c for c in extended_routes[j] if c not in primary_routes[j]]
        overlapped_segments.append(new_segment)

    # Initialize arrays
    primary_demands = np.asarray([sum(inst.demands[cust] for cust in route) for route in
                                  primary_routes])  # a priori primary route demand for each vehicle
    extended_demands = np.asarray([sum(inst.demands[cust] for cust in route) for route in
                                   extended_routes])  # a priori extended route demand for each vehicle
    overlap_demands = extended_demands - primary_demands  # demands of customers in k-overlapped
```

```
regions for each vehicle

    first = np.asarray([route[0] for route in primary_routes])  # first customer in route for each vehicle
    last = np.asarray([route[-1] for route in overlapped_segments] + [inst.tour[-1]])

    excess = np.zeros(len(primary_routes))  # surplus capacity for each vehicle (updated below)
    workload = np.zeros(len(primary_routes))  # demand ultimately filled by each vehicle (updated below)

    realized_routes = []

    # Loop through vehicles
    for j in range(len(primary_routes)):

        if j == 0:
            workload[j] = primary_demands[j]
        else:
            workload[j] = max(0, primary_demands[j] - excess[j - 1])

        excess[j] = min(capacity * np.ceil(float(workload[j]) / capacity) - workload[j], overlap_demands[j])
        remaining_surplus = excess[j]

        i = 0
        while remaining_surplus > 0:
            if i < len(overlapped_segments[j]):
                # fill demand of next shared customer
                # override default first and last customer if appropriate
                remaining_surplus -= inst.demands[overlapped_segments[j][i]]

                if remaining_surplus == 0:
                    # set last customer
                    last[j] = overlapped_segments[j][i]

                    # set first customer for next route
                    if i == len(overlapped_segments[j]) - 1 and overlap_size == route_size:
                        # next vehicle does not need to leave depot
                        first[j + 1] = 0  # next vehicle does not need to leave depot
                    else:
                        #
                        first[j + 1] = primary_routes[j + 1][i + 1]

                elif remaining_surplus < 0:
                    # vehicles will split this customer
                    last[j] = overlapped_segments[j][i]
                    first[j + 1] = overlapped_segments[j][i]
            i += 1

    # Determine realized routes based on updated first and last customers
    realized_routes = []
    for j in range(len(primary_routes)):

        # Create vehicle route
        if first[j] == 0:
            route = []  # vehicle doesn't leave depot
        else:
            first_index = inst.tour.index(first[j])
            last_index = inst.tour.index(last[j])
            route = inst.tour[first_index:last_index + 1]

```

```
            # Append to realized routes
            realized_routes.append(route)

    # Create full trips (i.e., segments) from realized routes
    demand_filled = [workload[j] + excess[j] for j in range(len(primary_routes))]
    segments = create_full_trips(inst, realized_routes, capacity, demand_filled)

    return segments


def create_instances(scenario, num_cust, cust_sims, dem_sims):
    """Returns cust_sims by dem_sims array of Instances"""

    np.random.seed(1)

    def gen_new_instance(num_cust, scenario):

        # Generate customer locations
        new_xlocs = field_width * np.random.random(num_cust)  # x coordinates of all customers
        new_ylocs = field_height * np.random.random(num_cust)  # y coordinates of all customers

        # Generate demands depending on scenario
        if scenario == 'stochastic_customers':
            # Equal probability of selecting 0 or 8
            new_dems = list(np.random.choice([0,8], num_cust))
        else:
            # Uniformly distributed between 0 and 8
            new_dems = list(np.random.randint(0, 8, num_cust))


        # Return new instance
        new_xlocs = list(np.append([depot_x], new_xlocs))  # include depot in customer x-coords
        new_ylocs = list(np.append([depot_y], new_ylocs))  # include depot in customer y-coords
        new_dems = list(np.append([0], new_dems))  # include depot in customer demands
        return Instance(new_xlocs, new_ylocs, new_dems)

    def update_demands(inst, scenario):
        # Creates copy of instance with updated demands depending on scenario
        if scenario == 'stochastic_customers':
            # Equal probability of selecting 0 or 8
            new_dems = list(np.random.choice([0, 8], num_cust))
        else:
            # Uniformly distributed between 0 and 8
            new_dems = list(np.random.randint(0, 8, num_cust))

        new_dems = list(np.append([0], new_dems))  # include depot in customer demands
        new_inst = Instance(inst.xlocs, inst.ylocs, new_dems, solve_TSP=False)
        new_inst.tour = inst.tour
        return new_inst

    # Create instance array with new customer instances
    instances = [[None for j in range(dem_sims)] for i in range(cust_sims)]
    customer_instances = [gen_new_instance(num_cust, scenario) for i in range(cust_sims)]

    # Create demand instances for each customer instance
    for i in range(cust_sims):
        for j in range(dem_sims):
            instances[i][j] = update_demands(customer_instances[i], scenario)

    return instances

```

```
def set_best_tours(demand_instances, primary_routes, extended_routes, capacity, route_size, overlap_size):
    """Updates the tour of all instances to the sequence that minimizes the average cost of the routes over all demand instances.
    Assumes all instances in list demand_instances have identical customer locations."""

    # Get any customer instance
    inst = demand_instances[0]
    # Set current tour and cumulative cost over all demand instances as best so far
    # Note: cumulative cost yields same tour ranking as average cost across demand instances
    best_tour = inst.tour
    segments = implement_k_overlapped_alg(inst, primary_routes, extended_routes, capacity, route_size, overlap_size)
    lowest_cumul_cost = sum([get_total_cost(inst, seg) for seg in segments for inst in demand_instances])

    # Copy of tour (for rotating below)
    tour = inst.tour

    # Loop over all customers
    for c in range(inst.size):

        # Rotate tour by one customer (keeps depot at very first spot)
        tour = tour[0:1] + tour[2:] + tour[1:2]
        inst.update_tour(tour)
        tour_cost = 0

        # Get cumulative cost over all demand instances
        for inst in demand_instances:
            segments = implement_k_overlapped_alg(inst, primary_routes, extended_routes, capacity, route_size,
                                                   overlap_size)
            for seg in segments:
                tour_cost += get_total_cost(inst, seg)

        # Set new best tour if lower cost
        if tour_cost < lowest_cumul_cost:
            best_tour = tour
            lowest_cumul_cost = tour_cost

    # Update tour for all demand instances in this customer row
    for inst in demand_instances:
        inst.update_tour(best_tour)
    return
```