Iowa Research Online
The University of Iowa's Institutional Repository

University of Iowa
Iowa Research Online

Theses and Dissertations

Summer 2010

Solution methodologies for vehicle routing problems with stochastic demand

Justin Christopher Goodson
University of Iowa

Copyright 2010 Justin Christopher Goodson

This dissertation is available at Iowa Research Online: http://ir.uiowa.edu/etd/675

Recommended Citation
Goodson, Justin Christopher. "Solution methodologies for vehicle routing problems with stochastic demand." PhD (Doctor of Philosophy) thesis, University of Iowa, 2010.
http://ir.uiowa.edu/etd/675.

Follow this and additional works at: http://ir.uiowa.edu/etd

Part of the Business Administration, Management, and Operations Commons

SOLUTION METHODOLOGIES FOR VEHICLE ROUTING PROBLEMS WITH STOCHASTIC DEMAND

by

Justin Christopher Goodson

An Abstract

Of a thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy
degree in Business Administration
in the Graduate College of
The University of Iowa

July 2010

Thesis Supervisors: Associate Professor Jeffrey Ohlmann
Associate Professor Barrett Thomas

ABSTRACT

We present solution methodologies for vehicle routing problems (VRPs) with stochastic demand, with a specific focus on the *vehicle routing problem with stochastic demand* (VRPSD) and the *vehicle routing problem with stochastic demand and duration limits* (VRPSDL). The VRPSD and the VRPSDL are fundamental problems underlying many operational challenges in the fields of logistics and supply chain management.

We model the VRPSD and the VRPSDL as large-scale Markov decision processes. We develop *cyclic-order neighborhoods*, a general methodology for solving a broad class of VRPs, and use this technique to obtain static, fixed route policies for the VRPSD. We develop *pre-decision*, *post-decision*, and *hybrid rollout policies* for approximate dynamic programming (ADP). We provide analytical results that position these policies within the rollout literature and identify conditions under which our proposed rollout methods are equivalent to the traditional form. Our rollout policies lay a methodological foundation for solving large-scale sequential decision problems and provide a framework for developing dynamic routing policies.

Our dynamic rollout policies for the VRPSDL significantly improve upon benchmark fixed route policies frequently implemented in practice. We also identify circumstances in which our rollout policies appear to offer little or no benefit compared to this benchmark. These observations can guide managerial decision making regarding when the use of our procedures is justifiable. We also demonstrate that our methodology lends itself to real-time implementation, thereby providing a mechanism to make high-quality, dynamic routing decisions for large-scale operations.

Finally, we consider a more traditional ADP approach to the VRPSDL by developing a parameterized linear function to approximate the value functions corresponding to our problem formulation. We estimate parameters via a simulation-based algorithm and show that initializing parameter values via our rollout policies leads to significant improvements. However, we conclude that additional research is required to develop a parametric ADP methodology comparable or superior to our rollout policies.

Abstract Approved: ______________________________
Thesis Supervisor

____
Title and Department

____
Date

____
Thesis Supervisor

____
Title and Department

____
Date

SOLUTION METHODOLOGIES FOR VEHICLE ROUTING PROBLEMS WITH STOCHASTIC DEMAND

by

Justin Christopher Goodson

A thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy
degree in Business Administration
in the Graduate College of
The University of Iowa

July 2010

Thesis Supervisors: Associate Professor Jeffrey Ohlmann
Associate Professor Barrett Thomas

Copyright by
JUSTIN CHRISTOPHER GOODSON
2010
All Rights Reserved

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

PH.D. THESIS

This is to certify that the Ph.D. thesis of

Justin Christopher Goodson

has been approved by the Examining Committee for the thesis requirement for the Doctor of Philosophy degree in Business Administration at the July 2010 graduation.

Thesis committee: ______________________________
Jeffrey Ohlmann, Thesis Supervisor

____
Barrett Thomas, Thesis Supervisor

____
Ann Campbell

____
Timothy Lowe

____
Pavlo Krokhmal

To Heidi, Dallin, Christian, Bethany, and Abigail

My God hath been my support.

2 Nephi 4:20

ACKNOWLEDGEMENTS

A candid remark from one of my children seems appropriate: “You’re still working on the truck problem? Wow, that’s a really long homework assignment!” With this in mind, I express my deepest appreciation to my wife, Heidi, and to my children, Dallin, Christian, Bethany, and Abigail. Their patience and support have enabled the completion of this thesis.

I am indebted to my advisors, Jeff Ohlmann and Barry Thomas. On many occasions, I felt as though I had encountered insurmountable obstacles in my research. Each time, their insight and direction led to a solution. I will always value the friendship that grew out of our weekly meetings.

During the first two years of my doctoral program, Thaddeus Sim and Kaan Ataman were my mentors. I appreciate their frequent advice and words of encouragement. Throughout my studies I have enjoyed the camaraderie of fellow doctoral students Brent Hickman, Adam Bradford, James Lambert, Jieqiu Chen, and Nick Leifker. Their willingness to discuss the complexities of graduate student life put my mind at ease on more than one occasion.

I express my appreciation to various faculty members in my department and on my committee: Nick Street, Kurt Anstreicher, Sam Burer, Ann Campbell, Ray de Matta, and Pavlo Krokhmal. During my time at the University of Iowa, they have offered support and timely advice. I would also like to thank my department secretary, Barb Carr, and the Ph.D. program coordinator, Renea Jay – their assistance with administrative tasks was very much appreciated.

Regarding the “nuts and bolts” of my thesis, I sincerely appreciate the time and efforts of Robert Hansen. He taught me the nuances of programming in C++, something I never could have learned from a textbook or instruction manual. I also gratefully acknowledge the University of Iowa Information Technology Services for allowing me access to a computing cluster. Without their assistance, I literally would have been unable to carry out the computational experiments necessary to complete my thesis. I also express my appreciation for a grant from the Executive Council for Graduate and Professional Students at the University of Iowa. The grant allowed for the purchase of a computer on which I conducted many preliminary computational experiments.

ABSTRACT

We present solution methodologies for vehicle routing problems (VRPs) with stochastic demand, with a specific focus on the *vehicle routing problem with stochastic demand* (VRPSD) and the *vehicle routing problem with stochastic demand and duration limits* (VRPSDL). The VRPSD and the VRPSDL are fundamental problems underlying many operational challenges in the fields of logistics and supply chain management.

We model the VRPSD and the VRPSDL as large-scale Markov decision processes. We develop *cyclic-order neighborhoods*, a general methodology for solving a broad class of VRPs, and use this technique to obtain static, fixed route policies for the VRPSD. We develop *pre-decision*, *post-decision*, and *hybrid rollout policies* for approximate dynamic programming (ADP). We provide analytical results that position these policies within the rollout literature and identify conditions under which our proposed rollout methods are equivalent to the traditional form. Our rollout policies lay a methodological foundation for solving large-scale sequential decision problems and provide a framework for developing dynamic routing policies.

Our dynamic rollout policies for the VRPSDL significantly improve upon benchmark fixed route policies frequently implemented in practice. We also identify circumstances in which our rollout policies appear to offer little or no benefit compared to this benchmark. These observations can guide managerial decision making regarding when the use of our procedures is justifiable. We also demonstrate that our methodology lends itself to real-time implementation, thereby providing a mechanism to make high-quality, dynamic routing decisions for large-scale operations.

Finally, we consider a more traditional ADP approach to the VRPSDL by developing a parameterized linear function to approximate the value functions corresponding to our problem formulation. We estimate parameters via a simulation-based algorithm and show that initializing parameter values via our rollout policies leads to significant improvements. However, we conclude that additional research is required to develop a parametric ADP methodology comparable or superior to our rollout policies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

CHAPTER 1 INTRODUCTION

An important aspect of supply chain management is the coordination of logistical operations for the transport of products within the supply chain. The task of designing delivery (or pickup) routes to service customers in a company's supply chain is known in the literature as a *vehicle routing problem* (VRP). In practice, customer demands may not be known with certainty prior to arrival at customer locations. In such situations, vehicle capacity may be insufficient to fulfill the demand actually encountered, thereby necessitating a return trip to a central depot to replenish capacity before continuing to service customers. Designing routes that effectively manage these potential *route failures* is an important consideration and a difficult task.

In this thesis, we present solution methodologies to address vehicle routing in the presence of stochastic demand. The two problem variants we consider share the following base problem description. Let $G = (\mathcal{N}, \mathcal{E})$ be a complete graph where $\mathcal{N} = \{0, 1, \ldots, n\}$ is a set of $n + 1$ nodes and $\mathcal{E} = \{(i, j) : i, j \in \mathcal{N}\}$ is the set of edges connecting the nodes. Node 0 represents a depot and nodes $1, \ldots, n$ represent customer locations. Vehicles routes begin and end at the depot. Travel times $t(i, j)$ associated with each edge $(i, j) \in \mathcal{E}$ are known and assumed deterministic. Let $\mathcal{M} = \{1, \ldots, m\}$ be a set of m identical vehicles initially located at the depot. Let Q denote vehicle capacity. Customer demands are random variables that follow a known joint probability distribution f with a support restricted to be a subset of $[0, \infty)^n$. Prior to arrival at customer locations, customer demands are known only in distribution. Upon arrival, customer demands are observed and served to the maximum extent, given available vehicle capacity. When its capacity is reached or exceeded, a vehicle must return to the depot to restore capacity up to Q.

The first variant we consider requires demand at each customer to be fully served and employs a traditional objective of minimizing expected costs. We refer to this variant as the *vehicle routing problem with stochastic demand* (VRPSD). Historically, the VRPSD serves as the fundamental problem underlying many operational challenges in the field of logistics. For example, in less-than-truckload operations, an estimate of customer demands may be available at the time vehicle routes are planned, but actual demands often differ from the initial estimate. Additionally, in many vendor-managed inventory systems, actual customer inventory levels are first observed upon arrival to the customer. In such cases, vehicle capacity may be inadequate to fully serve customer demand, resulting in a route failure requiring the vehicle to return to a central depot to replenish capacity before continuing to service customers.

In the second variant, we do not require demand to be fully served. Instead, we employ an objective of maximizing expected demand served subject to a route duration limit, L, by which time, e.g., end of a working day, all vehicles must return to the depot. Demand that remains unserved after an initial vehicle visit may possibly be satisfied on subsequent visits by the same vehicle, or by another. We refer to this variant as the *vehicle routing problem with stochastic demand and duration limits* (VRPSDL). In §2, we use the language of Markov decision processes (MDPs) to formulate the VRPSD and VRPSDL as stochastic shortest path problems.

The objective of the VRPSDL differs from much of the vehicle routing literature, which typically seeks to minimize some measure of expected travel cost while requiring that all customer demand be served. Because customer demands are not known until arrival at customer locations, and because vehicle routes must adhere to a route duration limit, there is a positive probability that some customer demands may be unserved, regardless of the fleet size and routing. Thus, employing a traditional cost-minimization objective while requiring all customer demands be served results in an ill-defined problem. Furthermore, the modification reflects an increased emphasis on customer service levels (van de Klundert and Wormer, 2010). It also addresses the fact that employee salaries and benefits dominate vehicle operating costs – this is especially true for the large number of unionized drivers in the trucking industry (ATA Economics & Statistical Analysis Department, 1999). Thus, companies are motivated to maximize the use of drivers' time, which occurs when drivers serve as much demand as possible. When vehicle capacity and route duration constraints are binding, this objective implicitly minimizes travel costs because efficient routes are necessary to maximize demand served.

A common practice to handle uncertainty in customer demands is to design a set of static, or fixed, routes for vehicles to follow. Often, fixed route policies implement simple recourse strategies to accommodate route failures, such as requiring a vehicle to return to the central depot, replenish capacity, and continue serving customers in the order specified by the fixed route. An optimal fixed route policy optimizes the expected routing cost subject to these restrictions. While these static policies can perform well on average, additional improvements may be obtained by dynamically modifying the routing plan as customer demands become known. Recent advances in communication technologies make such modifications possible, but the body of literature addressing these dynamic and stochastic VRPs is limited.

In this thesis, we develop methodologies to obtain both fixed and dynamic routing policies for VRPs with stochastic demand. In §3, we develop a general methodology for solving a broad class of VRPs and demonstrate its effectiveness by obtaining high-quality fixed route policies for the VRPSD. The methodology employs a *cyclic order* solution representation for VRPs. A cyclic order encoding can actually correspond to a very large number of VRP solutions, but the best of these VRP solutions can be identified via a polynomial time algorithm. The primary contribution of §3 is the design and implementation of neighborhoods to search cyclic orders. We demonstrate the potential of cyclic-order neighborhoods to facilitate the discovery of high quality solutions by embedding them within a simulated annealing framework to solve the classical VRP and the VRPSD. We propose an updating procedure and demonstrate its ability to reduce the computational expense of our approach. Our results underscore the potential for the use of cyclic-order neighborhoods as a general solution method for a variety of routing problems. Without tailoring our solution procedure to a specific routing problem, we are able to obtain solutions to the classical VRP within 0.60 percent of best known solutions and are able to match 16 of 19 known optimal VRPSD solutions.

The remainder of the thesis focuses on obtaining dynamic routing policies for the VRPSDL. In §4, we lay the foundation for the dynamic policies of §5 and §6 by developing new *rollout policies* for *approximate dynamic programming* (ADP). ADP is a general solution methodology for dynamic and stochastic sequential decision problems and seeks to overcome the well-known curses of dimensionality associated with dynamic programs. Traditional rollout policies employ heuristic optimization to approximate value functions in dynamic programs, but can be difficult to obtain for problems with large action spaces. By partitioning the state transition of an MDP (i.e., a stochastic dynamic program) into two parts, we introduce two extensions which we refer to as *pre-decision* and *post-decision* rollout policies. Furthermore, we discuss how these two new types of rollout policy may be used in combination, resulting in a *hybrid* rollout policy that, in our experience, achieves high quality solutions and results in a computational reduction sufficient for real-time implementation in large-scale settings. We also provide analytical results that position our extensions within the rollout literature and identify conditions under which our proposed rollout methods are equivalent to the traditional form.

In §5 and §6, we use the MDP formulation of §2 and the rollout framework of §4 to develop hybrid and pre-decision rollout policies for the VRPSDL. Similar to the literature on dynamic solution methodologies for single-vehicle VRPs with stochastic demand, a fixed route heuristic forms the basis of our rollout policy. We demonstrate the effectiveness of our methods by solving large-scale instances with as many as 100 customers and 19 vehicles. By testing our procedure across a broad range of problem parameters, we empirically establish conditions under which the demand served by our rollout policies is significantly higher than the demand served by benchmark fixed route policies. We also identify circumstances in which our rollout policies appear to offer little or no benefit compared to this benchmark. These observations can guide managerial decision making regarding when the use of our procedures is justifiable. Finally, we demonstrate that our methodology lends itself to real-time implementation, thereby providing a mechanism to make high-quality, dynamic routing decisions for large-scale operations.

In §7, we depart from our rollout framework and make first steps toward developing a parametric ADP method for obtaining dynamic VRPSDL policies. Unlike our rollout procedures, which approximate the value functions via fixed route policies, this chapter considers a parameterized linear function to approximate the value functions. We estimate the parameters via a simulation-based algorithm. The computational results we present indicate that initializing parameter values via the rollout methods of §5 and §6 leads to significant improvements. However, we conclude that additional research is required to develop a parametric ADP methodology comparable or superior to the rollout policies of §5 and §6.

We conclude the thesis in §8 by summarizing our contributions and by outlining directions for future research.

CHAPTER 2
PROBLEM FORMULATIONS

In this chapter, we use the language of Markov decision processes (MDPs) to formulate the VRPSD and the VRPSDL as stochastic shortest path problems. We refer to Puterman (1994) and Bertsekas (2000) for an introduction to MDPs and stochastic shortest path problems. Contrary to standard modeling techniques for MDPs, we split the transition from one state to another into two parts: a deterministic transition to a post-decision state followed by a random transition to a pre-decision state. Modeling transitions in this fashion is advocated by Powell (2007) and is essential when employing the rollout policies we develop in §4.

2.1 VRPSD Formulation

We first summarize the sequence of events that gives rise to our problem formulation. Figure 2.1 provides a timeline that illustrates the sequence of events and shows how we incorporate them into our model. We elaborate on terms and notation referenced below and in Figure 2.1 in subsequent sections. A decision epoch is triggered by the arrival of one or more vehicles at customer locations or at the depot (multiple vehicles may arrive simultaneously). The random time of decision epoch k, T_k, marks the end of period $k-1$ and the beginning of period k. Upon vehicle arrival at customer locations, actual demand is observed. In addition, vehicle capacities are replenished for vehicles arriving at the depot. These events are captured in the random transition from post-decision state s^a_{k-1} to pre-decision state s_k (the transition is random because customer demands may not be known prior to T_k). For vehicles at customer locations or at the depot (i.e., vehicles not en route), an action is selected to indicate where vehicles will travel next. Customer demand at current vehicle locations is then served to the fullest extent given available vehicle capacity and the selected action is executed incurring a cost of $C_k(s_k, a)$. These events are captured in the deterministic transition from pre-decision state s_k to post-decision state s_k^a. The next period begins at time T_{k+1}, when one or more vehicles arrive at their respective destinations. The process then repeats until the pre-decision state is in the set of absorbing states, $\mathcal{S}_K$.

2.1.1 States

The state of the system captures all relevant information to make routing decisions and includes vehicle destinations, arrival times at vehicle destinations, remaining vehicle capacities, unserved customer demand, and the history of observed demand. We represent attributes of vehicle $c \in \mathcal{M}$ by the tuple (l_c, t_c, q_c), where $l_c \in \mathcal{N}$ is the destination of vehicle c (or its current location if the vehicle has arrived), $t_c \in [0, \infty)$ is the time at which vehicle c will arrive at l_c, and $q_c \in [0, Q]$ is the available capacity of vehicle c. Let $(l, t, q) = (l_c, t_c, q_c)_{c \in \mathcal{M}}$ denote the vector of vehicle attributes. We characterize demand at customer $i \in \mathcal{N} \backslash \{0\}$ by the pair (d_i, x_i), where $d_i \in \{\{?\} \cup [0, \infty)\}$ is the unserved demand at customer i and $x_i \in \{\{?\} \cup [0, \infty)\}$ is the observed demand at customer i. We include x in the state because transition probabilities are conditional upon observed demands. Note that $(d_i, x_i) = (?, ?)$ in-

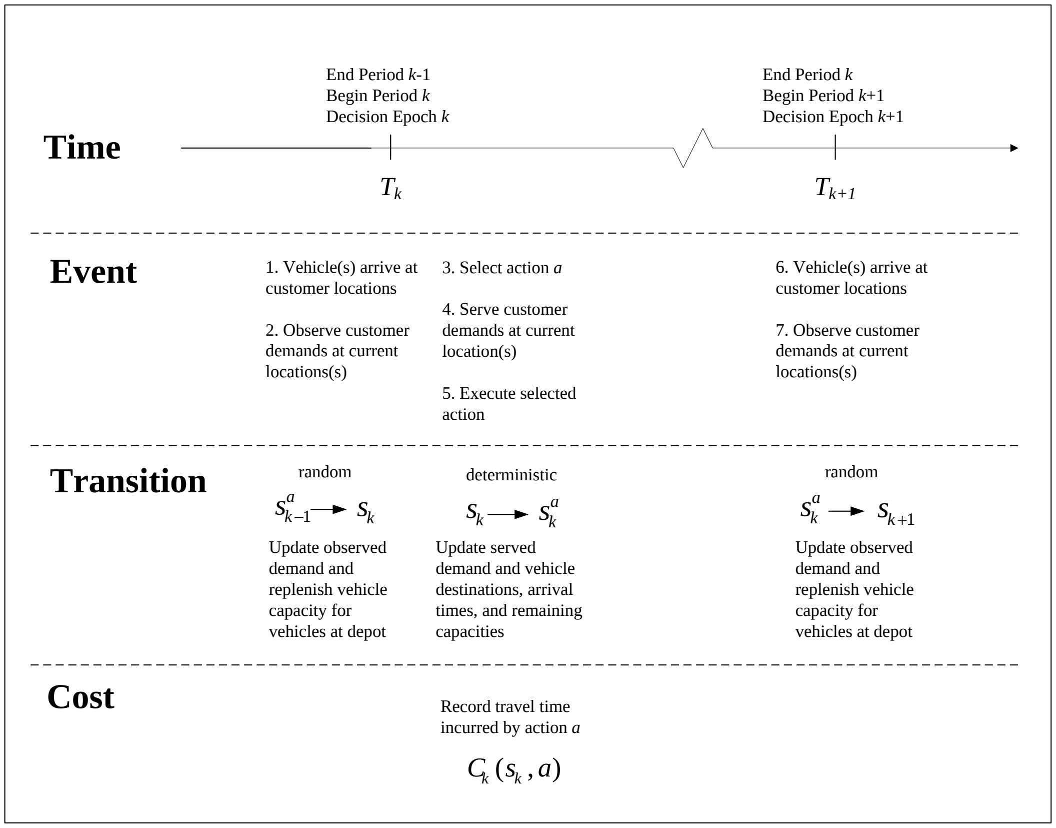
Time
End Period k-1
Begin Period k
Decision Epoch k
End Period k
Begin Period k+1
Decision Epoch k+1
T_k
T_{k+1}
Event
1. Vehicle(s) arrive at customer locations
2. Observe customer demands at current locations(s)
3. Select action a
4. Serve customer demands at current location(s)
5. Execute selected action
6. Vehicle(s) arrive at customer locations
7. Observe customer demands at current locations(s)
Transition
random
$s_{k-1}^a \rightarrow s_k$
Update observed demand and replenish vehicle capacity for vehicles at depot
deterministic
$s_k \rightarrow s_k^a$
Update served demand and vehicle destinations, arrival times, and remaining capacities
random
$s_k^a \rightarrow s_{k+1}$
Update observed demand and replenish vehicle capacity for vehicles at depot
Cost
Record travel time incurred by action a
$C_k(s_k, a)$

Figure 2.1: VRPSD Timeline

dicates that the demand at customer i is currently unknown. Furthermore, $d_i = ?$ if $x_i = ?$ and the demand served at customer i is $x_i - d_i$. Let $(d, x) = (d_i, x_i)_{i \in \mathcal{N}}$ denote the vector of unserved and observed demand. At decision epoch k, a state s_k in state space $\mathcal{S} = \mathcal{N}^m \times [0, \infty)^m \times [0, Q]^m \times \{\{?\} \cup [0, \infty)\}^n \times \{\{?\} \cup [0, \infty)\}^n$ takes the form $((l, t, q), (d, x))$. The initial state is $s_0 = ((0, 0, Q)^m, (?, ?)^n)$. The set of absorbing states is $\mathcal{S}_K = \{((0, t, Q)^m, (0, 0)) : t \in [0, \infty)^m\}$, where K denotes the final decision epoch.

We note that the inclusion of arrival time in the state variable only serves to identify the time of the next decision epoch. An alternative is to model the problem with a constant amount of time between epochs, e.g., 10 minutes or one hour. We choose to include time in the state variable to be consistent with our VRPSDL model in §2.2, where arrival times play a more prominent role.

Although Yang et al. (2000) demonstrate that an optimal single-vehicle policy with and equivalent cost exists for any multi-vehicle policy, this result does not hold in the presence of side constraints. Therefore, our formulation explicitly accounts for multiple vehicles as we impose a constraint (elaborated upon in §3) that requires the expected demand served by a vehicle to be less than or equal to vehicle capacity.

2.1.2 Actions

An action, taken at each decision epoch, is an assignment of the vehicles in $\mathcal{M}$ to locations in $\mathcal{N}$. Given state s_k, decision epoch k occurs at time $T_k = \min_{c \in \mathcal{M}} t_c(k)$. At decision epoch k, let $\mathcal{M}'_k = \arg\min_{c \in \mathcal{M}} t_c(k)$ be the set of vehicles currently at a customer location or at the depot (i.e., they are no longer en route). For brevity, we suppress notational reliance on k. A vehicle in $\mathcal{M}'$ may be assigned to the depot to replenish or may be assigned to a customer whose demand is either unknown or will be pending after customer demands are served in the current period. Vehicles in $\{\mathcal{M} \setminus \mathcal{M}'\}$ remain en route to their destinations, i.e., we do not consider diverting them.

We place the following restrictions on the available actions. If a vehicle's capacity will be depleted by serving customer demand at its current location, then the vehicle must return to the depot to replenish. We prohibit actions assigning more than one vehicle to a customer, but allow multiple vehicles to return to the depot.

Action a is an m-dimensional vector where the c^{th} element, a_c, is the action directing vehicle $c \in \mathcal{M}$. The set of actions available in state s_k is

$$\begin{aligned}
\mathcal{A}(s_k) = \{\, & a \in \mathcal{N}^m : \\
& a_c = l_c \,\forall\, c \in \{\mathcal{M} \setminus \mathcal{M}'\}, && (2.1)\\
& a_c \neq l_c \,\forall\, c \in \mathcal{M}', && (2.2)\\
& a_c = 0 \,\forall\, \{c \in \mathcal{M}' : q_c \leq d_{l_c}, l_c \neq 0\}, && (2.3)\\
& a_c \neq a_j \,\forall\, \{c, j \in \mathcal{M} : c \neq j, a_c \neq 0, a_j \neq 0\}, && (2.4)\\
& a_c \notin \{j \in \{\cup_{h \in \mathcal{M}'} l_h \setminus \{0\}\} : d_j \leq q_{\text{veh}(j)}\}, && (2.5)\\
& a_c \notin \{j \in \{\mathcal{N} \setminus \{0\}\} : d_j = 0\}. && (2.6)
\end{aligned}$$

Condition (2.1) requires that vehicles en route continue to their current destination.

For vehicles in $\mathcal{M}'$, condition (2.2) disallows assignment to the current location (i.e., waiting is not permitted). Condition (2.3) requires vehicles in $\mathcal{M}'$ to return to the depot if capacity will be depleted by serving customer demands at current locations. Condition (2.4) prevents assignment of multiple vehicles to customer locations, except in the case of assignments to the depot. Conditions (2.5) and (2.6) disallow assignment of vehicles to locations with zero demand (veh(j) denotes the vehicle at location j).

2.1.3 Transition to Post-decision State

Given that the state is currently s_k and that action $a \in \mathcal{A}(s_k)$ is selected, a deterministic transition is made to post-decision state $s_k^a = ((l^a, t^a, q^a), (d^a, x^a))$ by updating vehicle destinations, vehicle arrival times, remaining vehicle capacities, and served demand. Vehicle destinations in state s_k^a are $l_c^a = a_c$ for all $c \in \mathcal{M}$. Arrival times are set to

$$t_c^a = \begin{cases} t_c + t(l_c, l_c^a), & \forall\, c \in \mathcal{M}' \\ t_c, & \forall\, c \in \{\mathcal{M} \setminus \mathcal{M}'\} \end{cases} . \tag{2.7}$$

We update remaining vehicle capacities to reflect the demand served at each location visited at the current epoch:

$$q_c^a = \begin{cases} \max\{q_c - d_{l_c}, 0\}, & \forall\, c \in \{j \in \mathcal{M}' : l_j \neq 0\} \\ q_c, & \text{otherwise} \end{cases} . \tag{2.8}$$

In addition, we update unserved customer demands:

$$d_j^a = \begin{cases} \max\{d_j - q_{\text{veh}(j)}, 0\}, & \forall\, j \in \{\cup_{c \in \mathcal{M}'} l_c \setminus \{0\}\} \\ \\ d_j, & \text{otherwise} \end{cases} . \tag{2.9}$$

Observed demands remain unchanged, thus $x^a = x$.

2.1.4 Costs

A transition from pre-decision state s_k to post-decision state s_k^a results in a cost $C_k(s_k, a) = \sum_{c \in \mathcal{M}} C_{k,c}(s_k, a)$ equal to the travel time incurred by action a, where

$$C_{k,c}(s_k, a) = \begin{cases} t(l_c, a_c), & \forall\, c \in \mathcal{M}' \\ \\ 0, & \text{otherwise} \end{cases} . \tag{2.10}$$

2.1.5 Transition to Pre-decision State

At decision epoch $k+1$, a transition is made from post-decision state s_k^a to pre-decision state $s_{k+1} = ((l, t, q), (d, x))$ by observing customer demand and replenishing the capacity of vehicles at the depot. The joint density function f governs state transitions conditional on observed demand x. While it would also be correct to model capacity replenishment in the transition to the post-decision state (as a consequence of assigning a vehicle to the depot), we choose to model it in the transition to the pre-decision state because capacity is not actually replenished until a vehicle arrives at the depot. In this transition, vehicle destinations and arrival times remain unchanged, thus $l = l^a$ and $t = t^a$. We update capacities by

$$q_c = \begin{cases} Q, & \forall\, c \in \{j \in \mathcal{M}' : l_j = 0\} \\ \\ q_c^a, & \text{otherwise} \end{cases}. \tag{2.11}$$

Let $\mathcal{N}' = \{l_c^a : c \in \mathcal{M}', l_c^a \neq 0, x_{l_c}^a = ?\}$ be the set of customers with unknown demand at which a vehicle has arrived at the current decision epoch. If $\hat{x}$ is the observed demand at decision epoch $k+1$, then set

$$x_i = \begin{cases} \hat{x}_i, & i \in \mathcal{N}' \\ \\ x_i^a, & \text{otherwise} \end{cases} \tag{2.12}$$

and

$$d_i = \begin{cases} \hat{x}_i, & i \in \mathcal{N}' \\ \\ d_i^a, & \text{otherwise} \end{cases}. \tag{2.13}$$

2.1.6 Criterion and Objective

We seek a policy that minimizes the expected travel time. Because all information required to make a decision is contained in the current state, it is straightforward to show that the optimal policy is Markovian and deterministic (Puterman, 1994). Let Π be the set of all Markovian deterministic policies. A policy $\pi \in \Pi$ defines a decision function $\delta^\pi(s) : s \mapsto \mathcal{A}(s)$ that maps each state to an action. The criterion is

$$V_0^\pi = \mathbb{E}^\pi \left\{ \sum_{k=0}^{K} C_k(s_k, \delta^\pi(s_k)) \Big| s_0 \right\}. \tag{2.14}$$

We seek a policy $\pi^\star$ such that $V_0^{\pi^\star} \leq V_0^\pi$ for all $\pi \in \Pi$.

2.2 VRPSDL Formulation

Although our VRPSDL and VRPSD formulations are similar, there is a fundamental difference. As noted in §1, the objective of the VRPSDL is to maximize expected demand served subject to a route duration limit. This difference changes the set of absorbing states, the feasible action set, the criterion, and the objective; other elements of the formulation are the same. For completeness, we provide the VRPSDL formulation in its entirety.

We first summarize the sequence of events that gives rise to our problem formulation. Figure 2.2 provides a timeline that illustrates the sequence of events and shows how we incorporate them into our model. A decision epoch is triggered by the arrival of one or more vehicles at customer locations or at the depot (multiple vehicles may arrive simultaneously). The random time of decision epoch k, T_k, marks the end of period $k-1$ and the beginning of period k. Upon vehicle arrival at customer locations, actual demand is observed. In addition, vehicle capacities are replenished for vehicles arriving at the depot. These events are captured in the random transition from post-decision state s_{k-1}^a to pre-decision state s_k (the transition is random because customer demands may not be known prior to T_k). For vehicles at customer locations or at the depot (i.e., vehicles not en route), an action is selected to indicate where vehicles will travel next. Customer demand at current vehicle locations is then served to the fullest extent given available vehicle capacity (recorded as reward $R_k(s_k, a)$)

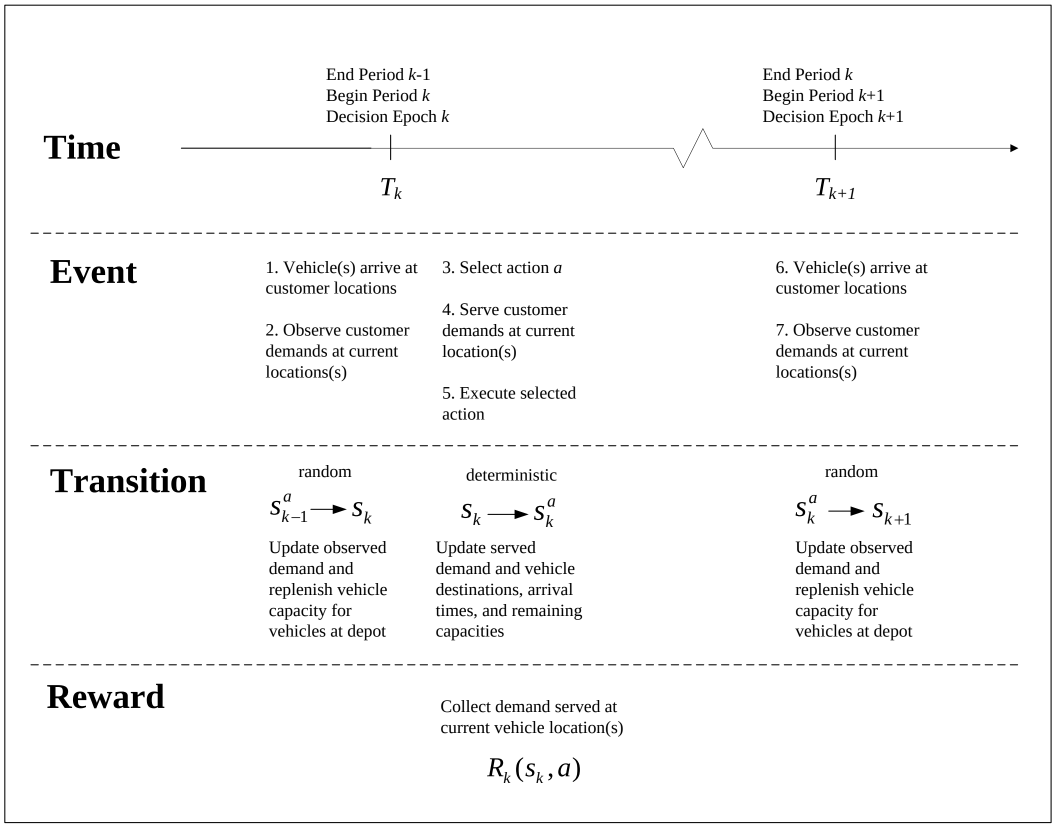
End Period k-1
Begin Period k
Decision Epoch k
End Period k
Begin Period k+1
Decision Epoch k+1
Time
T_k
T_{k+1}
Event
1. Vehicle(s) arrive at customer locations
2. Observe customer demands at current locations(s)
3. Select action a
4. Serve customer demands at current location(s)
5. Execute selected action
6. Vehicle(s) arrive at customer locations
7. Observe customer demands at current locations(s)
Transition
random
$s_{k-1}^a \rightarrow s_k$
Update observed demand and replenish vehicle capacity for vehicles at depot
deterministic
$s_k \rightarrow s_k^a$
Update served demand and vehicle destinations, arrival times, and remaining capacities
random
$s_k^a \rightarrow s_{k+1}$
Update observed demand and replenish vehicle capacity for vehicles at depot
Reward
Collect demand served at current vehicle location(s)
$R_k(s_k, a)$

Figure 2.2: VRPSDL Timeline

and the selected action is executed. These events are captured in the deterministic transition from pre-decision state s_k to post-decision state s_k^a. The next period begins at time T_{k+1}, when one or more vehicles arrive at their respective destinations. The process then repeats until the pre-decision state is in the set of absorbing states, $\mathcal{S}_K$.

2.2.1 States

The state of the system captures all relevant information to make routing decisions and includes vehicle destinations, arrival times at vehicle destinations, remaining vehicle capacities, unserved customer demand, and the history of observed demand. We represent attributes of vehicle $c \in \mathcal{M}$ by the tuple (l_c, t_c, q_c), where $l_c \in \mathcal{N}$ is the destination of vehicle c (or its current location if the vehicle has arrived), $t_c \in [0, L]$ is the time at which vehicle c will arrive at l_c, and $q_c \in [0, Q]$ is the available capacity of vehicle c. Let $(l, t, q) = (l_c, t_c, q_c)_{c \in \mathcal{M}}$ denote the vector of vehicle attributes. We characterize demand at customer $i \in \mathcal{N} \setminus \{0\}$ is characterized by the pair (d_i, x_i), where $d_i \in \{\{?\} \cup [0, \infty)\}$ is the unserved demand at customer i and $x_i \in \{\{?\} \cup [0, \infty)\}$ is the observed demand at customer i. We include x in the state because transition probabilities are conditional upon observed demands. Note that $(d_i, x_i) = (?, ?)$ indicates that the demand at customer i is currently unknown. Furthermore, $d_i = ?$ if $x_i = ?$ and the demand served at customer i is $x_i - d_i$. Let $(d, x) = (d_i, x_i)_{i \in \mathcal{N}}$ denote the vector of unserved and observed demand. At decision epoch k, a state s_k in state space $\mathcal{S} = \mathcal{N}^m \times [0, L]^m \times [0, Q]^m \times \{\{?\} \cup [0, \infty)\}^n \times \{\{?\} \cup [0, \infty)\}^n$ takes the form $((l, t, q), (d, x))$. The initial state is $s_0 = ((0, 0, Q)^m, (?, ?)^n)$. As we require all vehicles to return to the depot by the duration limit, the set of absorbing states is $\mathcal{S}_K = \{((0, L, Q)^m, (d, x)) : d_i = ? \text{ if } x_i = ? \text{ and } d_i \leq x_i \text{ if } x_i \neq ?\}$, where K denotes the final decision epoch.

2.2.2 Actions

An action, taken at each decision epoch, is an assignment of the vehicles in $\mathcal{M}$ to locations in $\mathcal{N}$. Given state s_k, decision epoch k occurs at time $T_k = \min_{c \in \mathcal{M}} t_c(k)$. At decision epoch k, let $\mathcal{M}'_k = \arg\min_{c \in \mathcal{M}} t_c(k)$ be the set of vehicles currently at a customer location or at the depot (i.e., they are no longer en route). For brevity, we suppress notational reliance on k. A vehicle in $\mathcal{M}'$ may be assigned to the depot to replenish or may be assigned to a customer whose demand is either unknown or will be pending after customer demands are served in the current period. Vehicles in $\{\mathcal{M} \setminus \mathcal{M}'\}$ remain en route to their destinations, i.e., we do not consider diverting them.

We place the following restrictions on the available actions. If a vehicle's capacity will be depleted by serving customer demand at its current location, then the vehicle must return to the depot to replenish. We prohibit actions assigning more than one vehicle to a customer, but allow multiple vehicles to return to the depot. Actions requiring a vehicle to return to the depot at a time greater than L are also prohibited; it is possible to identify actions violating L since travel times are known and deterministic.

Action a is an m-dimensional vector where the c^{th} element, a_c, is the action directing vehicle $c \in \mathcal{M}$. The set of actions available in state s_k is

$$\mathcal{A}(s_k) = \{\, a \in \mathcal{N}^m :$$

$$a_c = l_c \,\forall\, c \in \{\mathcal{M} \setminus \mathcal{M}'\}, \tag{2.15}$$

$$a_c \neq l_c \,\forall\, c \in \mathcal{M}', \tag{2.16}$$

$$a_c = 0 \,\forall\, \{c \in \mathcal{M}' : q_c \leq d_{l_c}, l_c \neq 0\}, \tag{2.17}$$

$$a_c \neq a_j \,\forall\, \{c, j \in \mathcal{M} : c \neq j, a_c \neq 0, a_j \neq 0\}, \tag{2.18}$$

$$a_c \notin \{j \in \{\cup_{h \in \mathcal{M}'} l_h \setminus \{0\}\} : d_j \leq q_{\text{veh}(j)}\}, \tag{2.19}$$

$$a_c \notin \{j \in \{\mathcal{N} \setminus \{0\}\} : d_j = 0\}, \tag{2.20}$$

$$a_c \notin \{j \in \mathcal{N} : T_k + t(l_c, j) + t(j, 0) > L\}\}. \tag{2.21}$$

Condition (2.15) requires that vehicles en route continue to their current destination. For vehicles in $\mathcal{M}'$, condition (2.16) disallows assignment to the current location (i.e., waiting is not permitted). Condition (2.17) requires vehicles in $\mathcal{M}'$ to return to the depot if capacity will be depleted by serving customer demands at current locations. Condition (2.18) prevents assignment of multiple vehicles to customer locations, except in the case of assignments to the depot. Conditions (2.19) and (2.20) disallow assignment of vehicles to locations with zero demand (veh(j) denotes the vehicle at location j). Condition (2.21) prohibits the assignment of vehicles to locations that will result in violations of the route duration limit.

It is instructive to compare the action space described by (2.15)-(2.21) to the action space of the single-vehicle problems considered in the literature (see §5.2). In a single-vehicle problem, an action is a scalar representing the location to which the vehicle will travel next (e.g., a customer with unknown/pending demand or the depot). In our multi-vehicle formulation, an action is a vector with one element for each of the m vehicles. Generally speaking, vector-valued actions can result in very large action spaces (Powell, 2010b). At a given decision epoch, the VRPSDL action space is most likely to experience a combinatorial explosion when the number of vehicles in $\mathcal{M}'$ is greater than one (actions for vehicles not in $\mathcal{M}'$ are fixed). Provided that these vehicles have positive capacity and enough time to visit additional locations, then the number of feasible actions can be exponential in size, an upper bound being $|\mathcal{M}'|^{|\mathcal{N}|}$. The difficulty imposed by this potentially large action space is the original motivation for the methodology we develop in §4.

2.2.3 Transition to Post-decision State

Given that the state is currently s_k and that action $a \in \mathcal{A}(s_k)$ is selected, a deterministic transition is made to post-decision state $s_k^a = ((l^a, t^a, q^a), (d^a, x^a))$ by updating vehicle destinations, vehicle arrival times, remaining vehicle capacities, and served demand. Vehicle destinations in state s_k^a are $l_c^a = a_c$ for all $c \in \mathcal{M}$. Arrival times are set to

$$t_c^a = \begin{cases} t_c + t(l_c, l_c^a), & \forall\, c \in \mathcal{M}' \\ \\ t_c, & \forall\, c \in \{\mathcal{M} \setminus \mathcal{M}'\} \end{cases} . \tag{2.22}$$

We update remaining vehicle capacities to reflect the demand served at each location visited at the current epoch:

$$q_c^a = \begin{cases} \max\{q_c - d_{l_c}, 0\}, & \forall\, c \in \{j \in \mathcal{M}' : l_j \neq 0\} \\ \\ q_c, & \text{otherwise} \end{cases} . \tag{2.23}$$

In addition, we update unserved customer demands:

$$d_j^a = \begin{cases} \max\{d_j - q_{\text{veh}(j)}, 0\}, & \forall\, j \in \{\cup_{c\in\mathcal{M}'} l_c \setminus \{0\}\} \\ d_j, & \text{otherwise} \end{cases}. \tag{2.24}$$

Observed demands remain unchanged, thus $x^a = x$.

2.2.4 Rewards

A transition from pre-decision state s_k to post-decision state s_k^a results in a reward $R_k(s_k, a) = \sum_{c\in\mathcal{M}} R_{k,c}(s_k, a)$ equal to the served demand, where

$$R_{k,c}(s_k, a) = \begin{cases} \min\{d_{l_c}, q_c\}, & \forall\, c \in \{j \in \mathcal{M}' : l_j \neq 0\} \\ 0, & \text{otherwise} \end{cases}. \tag{2.25}$$

To be consistent with standard MDP modeling techniques, we denote the reward as a function of the action selected. Note, however, that the reward is a function of demand served at current vehicle locations and is therefore independent of the selected action.

2.2.5 Transition to Pre-decision State

At decision epoch $k+1$, a transition is made from post-decision state s_k^a to pre-decision state $s_{k+1} = ((l, t, q), (d, x))$ by observing customer demand and replenishing the capacity of vehicles at the depot. The joint density function f governs state transitions conditional on observed demand x. While it would also be correct to model capacity replenishment in the transition to the post-decision state (as a consequence of assigning a vehicle to the depot), we choose to model it in the transition to the pre-decision state because capacity is not actually replenished until a vehicle arrives at the depot. In this transition, vehicle destinations and arrival times remain unchanged, thus $l = l^a$ and $t = t^a$. We update capacities by

$$q_c = \begin{cases} Q, & \forall\, c \in \{j \in \mathcal{M}' : l_j = 0\} \\ q_c^a, & \text{otherwise} \end{cases}. \tag{2.26}$$

Let $\mathcal{N}' = \{l_c^a : c \in \mathcal{M}', l_c^a \neq 0, x_{l_c}^a = ?\}$ be the set of customers with unknown demand at which a vehicle has arrived at the current decision epoch. If $\hat{x}$ is the observed demand at decision epoch $k+1$, then set

$$x_i = \begin{cases} \hat{x}_i, & i \in \mathcal{N}' \\ x_i^a, & \text{otherwise} \end{cases} \tag{2.27}$$

and

$$d_i = \begin{cases} \hat{x}_i, & i \in \mathcal{N}' \\ d_i^a, & \text{otherwise} \end{cases}. \tag{2.28}$$

2.2.6 Criterion and Objective

We seek a policy that maximizes the expected demand served. Because all information required to make a decision is contained in the current state, it is straightforward to show that the optimal policy is Markovian and deterministic (Puterman, 1994). Let Π be the set of all Markovian deterministic policies. A policy $\pi \in \Pi$ defines a decision function $\delta^\pi(s) : s \mapsto \mathcal{A}(s)$ that maps each state to an action. The criterion is

$$V_0^\pi = \mathbb{E}^\pi \left\{ \sum_{k=0}^{K} R_k(s_k, \delta^\pi(s_k)) \middle| s_0 \right\}. \tag{2.29}$$

We seek a policy $\pi^\star$ such that $V_0^{\pi^\star} \geq V_0^\pi$ for all $\pi \in \Pi$.

CHAPTER 3
CYCLIC-ORDER NEIGHBORHOODS FOR VEHICLE ROUTING PROBLEMS

3.1 Introduction

The problem of designing a minimum cost set of routes to serve a collection of customers with a fleet of vehicles is a fundamental challenge in the field of logistics. Because of the complexity of these vehicle routing problems (VRPs), vast literature is devoted to the development of heuristic search methods that seek to achieve optimal or near-optimal solutions with a reasonable amount of computational effort.

In this chapter, we examine a local search scheme utilizing the representation of VRP solutions as a single permutation of customers, called a *cyclic order* (not to be confused with *cyclic transfers* described in Thompson and Psaraftis (1993)). A cyclic order is a permutation of the set of customers, $\{1, \ldots, n\}$, for which the first customer in the order is also treated as the $(n+1)^{st}$, the second customer is also the $(n+2)^{st}$, and so forth. By iteratively sweeping across the permutation of customers to generate a collection of candidate routes, Ryan et al. (1993) show that a cyclic-order encoding corresponds to a very large number of VRP solutions from which the best solution can be identified via a polynomial time algorithm. Furthermore, Ryan et al. (1993) show that there exists a family of cyclic orders that correspond to an optimal VRP solution. Motivated by these results, we investigate the application of local search on the space of cyclic-order representations to seek optimal VRP solutions.

The main contribution of this chapter is the design and implementation of local search neighborhoods for the cyclic-order representation of VRPs. The cyclic-order neighborhood search in this chapter applies to the class of VRP problems in which a homogeneous fleet of capacitated vehicles serves a set of customers such that: (i) each customer is assigned to exactly one vehicle route, (ii) the objective is to find a feasible set of routes $x^\star$ such that $f(x^\star) \leq f(x)$ for all feasible route sets x, where f is cost separable in the routes in x, and (iii) any constraints pertain to restrictions on individual routes rather than collections of routes. We demonstrate the potential of cyclic-order neighborhoods to facilitate the discovery of high quality solutions by embedding them within a simulated annealing framework to solve the classical VRP and the VRP with stochastic demand (VRPSD). Throughout this chapter, when we refer to the VRPSD, we refer to the problem of obtaining an optimal fixed route policy. Fixed route policies meet conditions (i), (ii), and (iii) because each customer appears exactly once on a fixed route (although multiple visits may be required to fully serve demand), calculating the cost of a fixed route policy (i.e., the criterion (2.14)) is separable by route, and all constraints pertain to individual routes. When generating a neighbor cyclic order from a current cyclic order, we demonstrate that substantial computational savings result from employing our procedure for updating the corresponding set of candidate routes.

In §3.2, we explain the generation of a set of candidate routes from a specified cyclic order. We define neighborhoods to search cyclic orders in §3.3. In §3.4, we describe methods to update the set of candidate routes in local search schemes. We apply cyclic-order neighborhood search to the VRP and the VRPSD and present computational results in §3.5. We summarize the chapter and suggest directions for future research in §3.6.

3.2 Generation of Candidate Routes from a Cyclic Order

In this section, we detail the generation of the collection of candidate routes corresponding to a cyclic order π. As in Gillett and Miller (1974), we use a sweep procedure to generate $\mathcal{R}(\pi)$, the set of candidate routes consisting of contiguous elements of π. Denoting the customer at the i^{th} position of π by $\pi(i)$, $r = \{\pi(i), \pi(i+1), \ldots, \pi(j)\}$ is a route that visits the subset of customers corresponding to positions i through j of π. We uniquely identify each route $r \in \mathcal{R}(\pi)$ by the first and last customers of the subset of customers it visits. In this example, r can be uniquely identified by the ordered pair $\langle \text{first}(r), \text{last}(r) \rangle = \langle \pi(i), \pi(j) \rangle$.

Algorithm 3.1 describes a generic sweep procedure that we use throughout the chapter. For a cyclic order π, a route set $\mathcal{S}$, and indices $i, j \in \{1, \ldots, n\}$, an arbitrary call to SWEEP$(\pi, \mathcal{S}, i, j)$ begins by attempting to create a route r that visits the subset of customers $\{\pi(i), \pi(i+1), \ldots, \pi(j)\}$; as before, we denote such a route as $\langle \pi(i), \pi(j) \rangle$. If r is feasible, we insert it into $\mathcal{S}$, increment j, and repeat the procedure. Otherwise, the procedure terminates and returns the updated $\mathcal{S}$. As we discuss in §3.1, π is cyclic. Thus, is $j < i$, then the corresponding route visits the subset of customers $\{\pi(i), \pi(i+1), \ldots, \pi(n), \pi(1), \ldots, \pi(j)\}$. Similarly, in line 2 of Algorithm 3.1, if index $p = n$, then incrementing sets $p = 1$.

To explain the use of Algorithm 3.1 to construct $\mathcal{R}(\pi)$, let $\mathcal{R}_{\pi(i)}$ denote the set of routes consisting of contiguous elements of π such that $\text{first}(r) = \pi(i)$. We

Algorithm 3.1 Sweep Procedure for Customer $\pi(i)$ Beginning at Customer $\pi(j)$

```
1: procedure SWEEP($\pi, \mathcal{S}, i, j$)
2:     for $p = j$ to $i - 1$ step $+1$ do
3:         $r \leftarrow \langle \pi(i), \pi(p) \rangle$
4:         if $r$ is feasible then
5:             $\mathcal{S} \leftarrow \mathcal{S} \cup r$
6:         else
7:             break
8:     return $\mathcal{S}$
```

generate $\mathcal{R}(\pi) = \cup_{i=1}^{n} \mathcal{R}_{\pi(i)}$ by iteratively calling SWEEP$(\pi, \emptyset, i, i)$ for $i = 1, \ldots, n$ and storing the respective output in $\mathcal{R}_{\pi(i)}$. For example, consider the cyclic order $\pi =$ (7, 8, 9, 10, 11, 12, 13, 1, 2, 3, 4, 5, 6) for the 13-customer problem instance given in Ryan et al. (1993). We construct $\mathcal{R}_{\pi(1)}$ by executing SWEEP$(\pi, \emptyset, 1, 1)$ which first creates the singleton route $\langle \pi(1), \pi(1) \rangle = \langle 7, 7 \rangle$, followed by routes $\langle \pi(1), \pi(2) \rangle = \langle 7, 8 \rangle$ and $\langle \pi(1), \pi(3) \rangle = \langle 7, 9 \rangle$. In this example, $\langle \pi(1), \pi(4) \rangle = \langle 7, 10 \rangle$ is infeasible with respect to vehicle capacity, so the procedure terminates and returns $\mathcal{R}_{\pi(1)} = \{\langle 7, 7 \rangle, \langle 7, 8 \rangle, \langle 7, 9 \rangle\}$. Repeating the call to SWEEP$(\pi, \emptyset, i, i)$ for each remaining position i results in the complete construction of $\mathcal{R}(\pi) = \cup_{i=1}^{13} \mathcal{R}_{\pi(i)}$ as the left half of Table 3.1 illustrates (the right-half of Table 3.1 will be used in a subsequent illustration).

Provided that a singleton route to each customer is feasible, $\mathcal{R}(\pi)$ will always contain a subset of routes that corresponds to a feasible VRP solution (otherwise, the problem instance is infeasible). In general, the total number of feasible VRP solutions that can be assembled from $\mathcal{R}(\pi)$ cannot be determined a priori. However, for a

Table 3.1: Example of Updating $\mathcal{R}(\pi)$ to Obtain $\mathcal{R}(\pi')$ for a 3-shift Neighbor with $i = 2$ and $j = 8$

i	$\pi(i)$	$\mathcal{R}_{\pi(i)}$				$\pi'(i)$	$\mathcal{R}_{\pi'(i)}$		
1	7	$\langle 7,7\rangle$	$\langle 7,8\rangle^{\ddagger}$	$\langle 7,9\rangle^{\ddagger}$		8	$\langle 8,8\rangle$	$\langle 8,9\rangle$	
2	8	$\langle 8,8\rangle$	$\langle 8,9\rangle$			9	$\langle 9,9\rangle$	$\langle 9,10\rangle$	
3	9	$\langle 9,9\rangle$	$\langle 9,10\rangle$			10	$\langle 10,10\rangle$	$\langle 10,1\rangle^{*}$	$\langle 10,2\rangle^{*}$
4	10	$\langle 10,10\rangle$	$\langle 10,11\rangle^{\ddagger}$			1	$\langle 1,1\rangle$	$\langle 1,2\rangle$	$\langle 1,3\rangle$
5	11	$\langle 11,11\rangle$	$\langle 11,12\rangle$	$\langle 11,13\rangle$		2	$\langle 2,2\rangle$	$\langle 2,3\rangle$	$\langle 2,4\rangle$
6	12	$\langle 12,12\rangle$	$\langle 12,13\rangle$	$\langle 12,1\rangle^{\ddagger}$		3	$\langle 3,3\rangle$	$\langle 3,4\rangle$	
7	13	$\langle 13,13\rangle$	$\langle 13,1\rangle^{\ddagger}$	$\langle 13,2\rangle^{\ddagger}$	$\langle 13,3\rangle^{\ddagger}$	4	$\langle 4,4\rangle$	$\langle 4,5\rangle$	
8	1	$\langle 1,1\rangle$	$\langle 1,2\rangle$	$\langle 1,3\rangle$		5	$\langle 5,5\rangle$	$\langle 5,6\rangle$	
9	2	$\langle 2,2\rangle$	$\langle 2,3\rangle$	$\langle 2,4\rangle$		6	$\langle 6,6\rangle$	$\langle 6,7\rangle$	$\langle 6,11\rangle^{*}$
10	3	$\langle 3,3\rangle$	$\langle 3,4\rangle$			7	$\langle 7,7\rangle$	$\langle 7,11\rangle^{*}$	$\langle 7,12\rangle^{*}$
11	4	$\langle 4,4\rangle$	$\langle 4,5\rangle$			11	$\langle 11,11\rangle$	$\langle 11,12\rangle$	$\langle 11,13\rangle$
12	5	$\langle 5,5\rangle$	$\langle 5,6\rangle$			12	$\langle 12,12\rangle$	$\langle 12,13\rangle$	
13	6	$\langle 6,6\rangle$	$\langle 6,7\rangle$	$\langle 6,8\rangle^{\ddagger}$		13	$\langle 13,13\rangle$	$\langle 13,8\rangle^{*}$	$\langle 13,9\rangle^{*}$

solution with m vehicles, an upper bound on the number of feasible VRP solutions is $O\left(\binom{|\mathcal{R}(\pi)|}{m}\right)$.

The concept of generating a VRP solution by *sweeping* a permutation of customers was first proposed by Gillett and Miller (1974). Foster and Ryan (1976) consider radial orderings of customers and show that by solving a linear program we can find the feasible solution $x^{\star}_{\pi}$ such that $f(x^{\star}_{\pi}) \leq f(x_{\pi})$ for every feasible $x_{\pi} \subseteq \mathcal{R}(\pi)$. Ryan et al. (1993) show that $x^{\star}_{\pi}$ can be found by solving a set partitioning problem (SPP) where a column exists for each route in $\mathcal{R}(\pi)$ and each row represents one of the n customers. In the SPP, the cost of each column is the cost of the corresponding route (which can include a large fixed cost to reflect the cost of a vehicle). Additionally, both Ryan et al. (1993) and Boctor and Renaud (2000) show that the special structure of the routes produced by the sweep procedure allows such SPPs to be solved in polynomial time by solving a series of shortest path problems. Renaud et al. (1996) and Renaud and Boctor (2002) employ this sweep method as a construction heuristic for vehicle routing problems. In contrast, we iteratively employ the sweep procedure in a local search procedure. To the best of the authors' knowledge, this is the first work to examine local search for vehicle routing problems using the cyclic-order representation.

3.3 Cyclic-Order Neighborhoods

While Ryan et al. (1993) establish that there exists at least one cyclic order $\pi^\star$ for which $\mathcal{R}(\pi^\star)$ contains the routes composing an optimal VRP solution, effective means to obtain $\pi^\star$ have not been explored in the literature. We propose a collection of permutation-based neighborhoods to facilitate local search on the space of cyclic orders.

In the following descriptions, $\text{dist}(i, j)$ is the number of times index i of π must be incremented such that it equals index j of π. That is, $\text{dist}(i, j) = j - i$ if $i \leq j$ and $n - i + j$ otherwise. We consider the following neighborhoods:

- ***k*-shift**. Select $i, j \in \{1, \ldots, n\}$ and shift $\pi(i), \ldots, \pi(i + k - 1)$, as a group, to the positions immediately prior to $\pi(j)$. For a positive integer $k < n - 1$, a k-shift neighborhood consists of the cyclic orders that can be constructed from the set $\{i, j : \text{dist}(i, j) > k\}$.
- **Scramble**. Select $i, j \in \{1, \ldots, n\}$ and randomly shuffle $\pi(i), \ldots, \pi(j)$. A scramble neighborhood consists of the cyclic orders that can be constructed from all i, j pairs. If $i = j$, then the entire cyclic order is scrambled.
- **Reverse**. Select $i, j \in \{1, \ldots, n\}$ and reverse the order of $\pi(i), \ldots, \pi(j)$. A

reverse neighborhood consists of the cyclic orders that can be constructed from all i, j pairs. If $i = j$, then the entire cyclic order is reversed. Note that a reverse neighborhood is a special case of the scramble neighborhood.

- **Exchange**. Select $i, j \in \{1, \ldots, n\}$ and exchange $\pi(i)$ and $\pi(j)$. An exchange neighborhood consists of the cyclic orders that can be constructed from the set $\{i, j : i < j\}$.

With respect to the number of cyclic orders, the size of each of the above neighborhoods is $O(n^2)$. Because a cyclic order corresponds to a potentially large number of VRP solutions, however, one potential advantage of operating on cyclic orders rather than directly on VRP solutions is that a small change in the cyclic order may result in significant change in the structure of the resulting VRP solution. To illustrate the diversity of solutions that a cyclic-order neighborhood can generate, we compare Figures 3.1 and 3.2, which display the optimal VRP solutions corresponding to $\pi = (7, 8, 9, 10, 11, 12, 13, 1, 2, 3, 4, 5, 6)$ and $\pi' = (7, 8, 9, 10, 12, 11, 13, 1, 2, 3, 4, 5, 6)$, respectively, for the 13-customer problem from Ryan et al. (1993). We obtain π' from π by applying the exchange operation with $i = 5$ and $j = 6$. Comparing the route structures in Figure 3.1 to those in Figure 3.2 reveals that even small changes to the cyclic order (i.e., exchanging two consecutive elements) can result in very different solutions. Furthermore, obtaining the route structure of Figure 3.2 from Figure 3.1 requires 10 customer insertions/removals if applying local search directly on the VRP solution rather than the cyclic order.

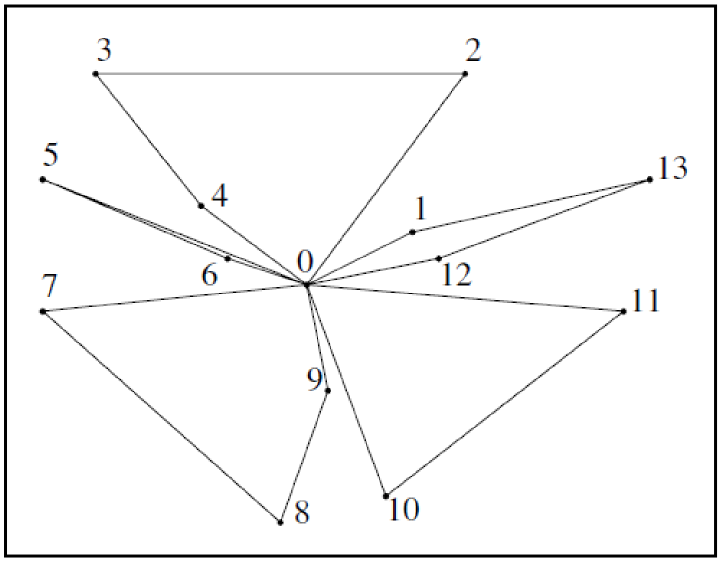

Figure 3.1: Optimal VRP Solution Corresponding to π – We Note that this Solution Corrects a Minor Oversight in Ryan et al. (1993)

3.4 Updating Procedure for Neighboring Cyclic Orders

As discussed in §3.2, the evaluation of a cyclic order π is a two-step process. First, we generate the corresponding set of candidate routes, $\mathcal{R}(\pi)$. Second, we identify $x^{\star}_{\pi}$, the optimal VRP solution with respect to $\mathcal{R}(\pi)$. The optimization is achieved in polynomial time by solving a series of shortest path problems (Ryan et al., 1993; Boctor and Renaud, 2000). In a local search procedure, a naive approach to evaluate π', a neighbor of π, would be to construct $\mathcal{R}(\pi')$ in its entirety by iteratively

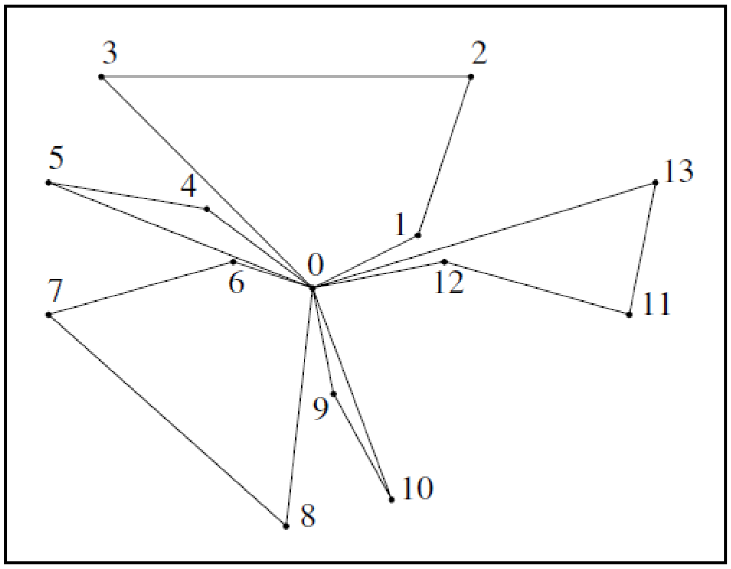

Figure 3.2: Optimal VRP Solution Corresponding to π'

calling SWEEP$(\pi', \emptyset, i, i)$ for $i = 1, \ldots, n$. However, many of the candidate routes in $\mathcal{R}(\pi')$ may also be in $\mathcal{R}(\pi)$. In this section, we show how to obtain $\mathcal{R}(\pi')$ by updating $\mathcal{R}(\pi)$ and demonstrate in §3.5 that this updating can significantly reduce computation time.

To motivate our updating procedures, we again consider the example presented in Table 3.1. As before, the left-hand portion of the table displays a cyclic order π and the corresponding set of candidate routes for the 13-customer problem in Ryan et al. (1993). The fourth column of Table 3.1 displays a cyclic order π' generated by applying the 3-shift operator on π with $i = 2$ and $j = 8$. The fifth column contains the corresponding set of candidate routes, $\mathcal{R}(\pi')$. To obtain $\mathcal{R}(\pi')$ by updating $\mathcal{R}(\pi)$, we must remove from $\mathcal{R}(\pi)$ the eight routes superscripted by ‡ and add the seven routes superscripted by $*$ (all other routes remain unchanged). In comparison, totally reconstructing $\mathcal{R}(\pi')$ requires the creation of 33 routes, 26 of which already exist in $\mathcal{R}(\pi)$.

By considering the construction of $\mathcal{R}(\pi)$ (see §3.2), it is possible to systematically identify routes to retain, remove, and add to obtain $\mathcal{R}(\pi')$. Recall from §3.2 that each route in $\mathcal{R}(\pi)$ visits a subset of customers consisting of contiguous elements of the cyclic order π. Thus, segments of π and π' that are identical generate the same subset of routes. These routes exist both in $\mathcal{R}(\pi)$ and $\mathcal{R}(\pi')$. To illustrate the process of removing and adding routes to obtain $\mathcal{R}(\pi')$ by updating $\mathcal{R}(\pi)$, consider $\mathcal{R}_{\pi(1)} = \{\langle 7, 7\rangle, \langle 7, 8\rangle, \langle 7, 9\rangle\}$ in the left-hand portion of Table 3.1. Recall that the sweep procedure creates these routes because customers 7, 8, and 9 are positioned

Algorithm 3.2 Update $\mathcal{R}(\pi)$ to Obtain $\mathcal{R}(\pi')$ for a k-shift Neighbor

```
1: Input: cyclic order $\pi$, candidate route set $\mathcal{R}(\pi)$, indices $i$ and $j$, positive integer
   $k < n - 1$
2: Output: neighbor cyclic order $\pi'$, candidate route set $\mathcal{R}(\pi')$
3: $\pi' \leftarrow (\pi(i), \pi(i+1), \ldots, \pi(i+k-1), \pi(j), \pi(j+1), \ldots,$
   $\pi(i-1), \pi(i+k), \pi(i+k+1), \ldots, \pi(j-1))$
4: $\mathcal{R}(\pi') \leftarrow \mathcal{R}(\pi)$
5: for $p = k$ to $1$ step $-1$ do
6:     if $|\mathcal{R}_{\pi'(p)}| > \text{dist}(p, k+1)$ then
7:         $\mathcal{R}_{\pi'(p)} \leftarrow \mathcal{R}_{\pi'(p)} \setminus \mathcal{R}_{\pi'(p),\pi'(k+1)}$
8:         $\mathcal{R}_{\pi'(p)} \leftarrow \text{SWEEP}(\pi', \mathcal{R}_{\pi'(p)}, p, k+1)$
9:     else if $|\mathcal{R}_{\pi'(p)}| = \text{dist}(p, k+1)$ then
10:        $\mathcal{R}_{\pi'(p)} \leftarrow \text{SWEEP}(\pi', \mathcal{R}_{\pi'(p)}, p, k+1)$
11:    else
12:        break
13: for $p = k + \text{dist}(j,i)$ to $k+1$ step $-1$ do
14:    if $|\mathcal{R}_{\pi'(p)}| > \text{dist}(p, k + \text{dist}(j,i) + 1)$ then
15:        $\mathcal{R}_{\pi'(p)} \leftarrow \mathcal{R}_{\pi'(p)} \setminus \mathcal{R}_{\pi'(p),\pi'(k+\text{dist}(j,i)+1)}$
16:        $\mathcal{R}_{\pi'(p)} \leftarrow \text{SWEEP}(\pi', \mathcal{R}_{\pi'(p)}, p, k + \text{dist}(j,i) + 1)$
17:    else if $|\mathcal{R}_{\pi'(p)}| = \text{dist}(p, k + \text{dist}(j,i) + 1)$ then
18:        $\mathcal{R}_{\pi'(p)} \leftarrow \text{SWEEP}(\pi', \mathcal{R}_{\pi'(p)}, p, k + \text{dist}(j,i) + 1)$
19:    else
20:        break
21: for $p = n$ to $k + \text{dist}(j,i) + 1$ step $-1$ do
22:    if $|\mathcal{R}_{\pi'(p)}| > \text{dist}(p, 1)$ then
23:        $\mathcal{R}_{\pi'(p)} \leftarrow \mathcal{R}_{\pi'(p)} \setminus \mathcal{R}_{\pi'(p),\pi'(1)}$
24:        $\mathcal{R}_{\pi'(p)} \leftarrow \text{SWEEP}(\pi', \mathcal{R}_{\pi'(p)}, p, 1)$
25:    else if $|\mathcal{R}_{\pi'(p)}| = \text{dist}(p, 1)$ then
26:        $\mathcal{R}_{\pi'(p)} \leftarrow \text{SWEEP}(\pi', \mathcal{R}_{\pi'(p)}, p, 1)$
27:    else
28:        break
```

contiguously in π and can feasibly exist on the same route. In π', however, this is not the case. Customers 8 and 9 are adjacent, but customer 7 precedes customer 11. Thus, $\mathcal{R}(\pi')$ does not contain routes $\langle 7, 8\rangle$ and $\langle 7, 9\rangle$ and the updating procedure must remove them. To generate the new routes beginning with customer 7 (due to its new position 10), we execute SWEEP$(\pi', \mathcal{R}_{\pi'(10)}, 10, 11)$ via Algorithm 3.1 to obtain the routes $\langle 7, 11\rangle$ and $\langle 7, 12\rangle$. The result is $\mathcal{R}_{\pi'(10)} = \{\langle 7, 7\rangle, \langle 7, 11\rangle, \langle 7, 12\rangle\}$.

In the above example, to identify routes in $\mathcal{R}_{\pi(1)}$ that do not exist in $\mathcal{R}(\pi')$, it is sufficient to determine which routes visit customers 7 and 8. Because customer 7 no longer precedes customer 8 in π', we remove all routes in $\mathcal{R}_{\pi(1)}$ visiting these two customers to update $\mathcal{R}(\pi')$. It is possible that other sets $\mathcal{R}_{\pi(i)}, i = 2, \ldots, n$, also contain routes visiting customers 7 and 8 (e.g., route $\langle 6, 8\rangle$ in $\mathcal{R}_{\pi(13)}$); to identify such sets, it is sufficient to examine their cardinality. Denote by $\text{pos}(x)$ the position of customer x in π. If $\mathcal{R}_{\pi(i)}$ contains any routes visiting customers 7 and 8, then $|\mathcal{R}_{\pi(i)}| > \text{dist}(i, \text{pos}(8))$, where $\text{pos}(8) = 2$. Because $|\mathcal{R}_{\pi(13)}| = 3 > \text{dist}(13, 2) = 2$, we know that $\mathcal{R}_{\pi(13)}$ contains $3 - 2 = 1$ route that visits customers 7 and 8 (route $\langle 6, 8\rangle$). Because $|\mathcal{R}_{\pi(12)}| = 2 \not> \text{dist}(12, 2) = 3$, $\mathcal{R}_{\pi(12)}$ does not contain any routes visiting customers 7 and 8. Moreover, elements in the set $\{\mathcal{R}_{\pi(p)} : p \in (12, 11, \ldots, 2)\}$ do not contain any routes visiting customers 7 and 8. We formally state these relationships in Proposition 3.1, where $\mathcal{R}_{\pi(i),\pi(j)} = \{r \in \mathcal{R}_{\pi(i)} : \text{dist}(i, \text{pos}(\text{last}(r))) \geq \text{dist}(i, j)\}$ is the set of routes in $\mathcal{R}_{\pi(i)}$ that visit customers $\pi(i)$ and $\pi(j)$.

Proposition 3.1. *For all $i, j \in \{1, \ldots, n\}$, if $|\mathcal{R}_{\pi(i)}| > dist(i, j)$, then routes $\mathcal{R}_{\pi(i),\pi(j)}$ visit customers $\pi(i)$ and $\pi(j)$ and $|\mathcal{R}_{\pi(i),\pi(j)}| = |\mathcal{R}_{\pi(i)}| - dist(i, j)$. Otherwise, $\{\mathcal{R}_{\pi(p)}$:*

$p \in (i, i-1, \ldots, j+1)\}$ *do not contain any routes that visit customers* $\pi(i)$ *and* $\pi(j)$.

We note that the sweep procedure in Algorithm 3.1 creates the routes of $\mathcal{R}_{\pi(i)}$ for $i = 1, \ldots, n$, in order of increasing pos(last($\cdot$)) (we present the route sets in Table 3.1 in this order), thus facilitating the identification of $\mathcal{R}_{\pi(i),\pi(j)}$. To update $\mathcal{R}(\pi')$, we simply remove the last $|\mathcal{R}_{\pi(i)}| - \text{dist}(i,j)$ routes from $\mathcal{R}_{\pi(i)}$, an $O(1)$ operation. Furthermore, upon the insertion of new routes to update $\mathcal{R}(\pi')$, maintaining this ordering requires no additional sorting as we create the new routes for insertion via the sweep procedure in Algorithm 3.1.

For a cyclic order π and a neighbor cyclic order π' generated from one of the neighborhoods in §3.3, we employ Proposition 3.1 to obtain $\mathcal{R}(\pi')$ by updating $\mathcal{R}(\pi)$. Algorithm 3.2 details this procedure for the k-shift neighborhood. We explain the procedure by stepping through the example in Table 3.1, where π' is in the k-shift neighborhood of π with $k = 3$, $i = 2$, and $j = 8$.

Line 3 of Algorithm 3.2 constructs π' by shifting customers 8, 9, and 10 to the position immediately before customer 1. In line 3, $\pi(i) = 8$, $\pi(i+1) = 9$, $\pi(i+k-1) = 10$, $\pi(j) = 1$, $\pi(j+1) = 2$, $\pi(i-1) = 7$, $\pi(i+k) = 11$, $\pi(i+k+1) = 12$, and $\pi(j-1) = 13$. The loop beginning on line 5 iterates over indices 3, 2, and 1 to remove all routes containing customers 10 and 11 and to insert all routes containing customers 10 and 1. Line 6 employs Proposition 3.1 to identify routes for removal. When $p = 3$, $\mathcal{R}_{\pi'(3)}$ is initially $\{\langle 10, 10\rangle, \langle 10, 11\rangle\}$, thus $|\mathcal{R}_{\pi'(3)}| = 2 > \text{dist}(3,4) = 1$. Line 7 removes route $\langle 10, 11\rangle$ and line 8 adds routes $\langle 10, 1\rangle$ and $\langle 10, 2\rangle$

Algorithm 3.3 Update $\mathcal{R}(\pi)$ to Obtain $\mathcal{R}(\pi')$ for a Scramble or Reverse Neighbor

```
1: **Input**: cyclic order $\pi$, candidate route set $\mathcal{R}(\pi)$, indices $i$ and $j$
2: **Output**: neighbor cyclic order $\pi'$, candidate route set $\mathcal{R}(\pi')$
3: construct $\pi'$ from $i$, $j$, and $\pi$
4: $\mathcal{R}(\pi') \leftarrow \mathcal{R}(\pi)$
5: for $p = i$ to $j$ step $+1$ do
6:     if $|\mathcal{R}_{\pi'(p)}| > \text{dist}(p, p+1)$ then
7:         $\mathcal{R}_{\pi'(p)} \leftarrow \mathcal{R}_{\pi'(p)} \setminus \mathcal{R}_{\pi'(p),\pi'(p+1)}$
8:         $\mathcal{R}_{\pi'(p)} \leftarrow \text{SWEEP}(\pi', \mathcal{R}_{\pi'(p)}, p, p+1)$
9:     else
10:        $\mathcal{R}_{\pi'(p)} \leftarrow \text{SWEEP}(\pi', \mathcal{R}_{\pi'(p)}, p, p+1)$
11: for $p = i-1$ to $j+1$ step $-1$ do
12:     if $|\mathcal{R}_{\pi'(p)}| > \text{dist}(p, i)$ then
13:         $\mathcal{R}_{\pi'(p)} \leftarrow \mathcal{R}_{\pi'(p)} \setminus \mathcal{R}_{\pi'(p),\pi'(i)}$
14:         $\mathcal{R}_{\pi'(p)} \leftarrow \text{SWEEP}(\pi', \mathcal{R}_{\pi'(p)}, p, i)$
15:     else if $|\mathcal{R}_{\pi'(p)}| = \text{dist}(p, i)$ then
16:         $\mathcal{R}_{\pi'(p)} \leftarrow \text{SWEEP}(\pi', \mathcal{R}_{\pi'(p)}, p, i)$
17:     else
18:         break
```

by executing SWEEP$(\pi', \mathcal{R}_{\pi'(3)}, 3, 4)$ via Algorithm 3.1. When $p = 2$, $\mathcal{R}_{\pi'(2)}$ is initially $\{\langle 9,9\rangle, \langle 9,10\rangle\}$, thus $|\mathcal{R}_{\pi'(2)}| = 2 \not> \text{dist}(2,4) = 2$ and the inequality in line 6 is not satisfied. The equality in line 9 is satisfied, however, and an attempt to add routes containing customers 9 and 1 is made by executing SWEEP$(\pi', \mathcal{R}_{\pi'(2)}, 2, 4)$. No such routes are added to $\mathcal{R}_{\pi'(2)}$ because they are infeasible with respect to vehicle capacity. When $p = 1$, neither of the conditions in lines 6 or 9 are met and the loop terminates.

In a similar manner, the loop beginning on line 13 removes all routes containing customers 7 and 8 and adds all routes containing customers 7 and 11. Finally, the loop beginning on line 21 removes all routes containing customers 13 and 1 and adds all routes containing customers 13 and 8. Table 3.1 denotes the routes that we remove and insert during the procedure by superscripting them with ‡ and $*$, respectively. We note that if neither of the conditions for removing and adding routes (lines 6 and 9, lines 14 and 17, lines 22 and 25, respectively) are met for an index p, then we terminate the respective loop immediately as it is not necessary to check the conditions for the smaller indices of the loop.

In the worst case, where we examine each route set and create n routes, the complexity of Algorithm 3.2 is $O(n^2)$. While this is the same complexity as that of constructing $\mathcal{R}(\pi')$ in its entirety, we note that the worst-case complexity for this updating is only approached if vehicle capacity is large enough to allow all customers to be serviced on one route. As our results in §3.5 demonstrate, the updating procedure provides significant computational advantages. Algorithms 3.3 and 3.4 detail similar updating procedures for the scramble, reverse, and exchange cyclic-order neighborhoods. These procedures also have a worst case complexity of $O(n^2)$.

3.5 Application of Cyclic-Order Neighborhood Search

To demonstrate the effectiveness of cyclic-order neighborhood search, we apply the methods developed in this chapter to two problems, the classical VRP and the VRP with stochastic demand (VRPSD). As discussed in §3.1, we address the problem of obtaining optimal fixed route policies for the VRPSD. In §3.5.1, we define these problems and provide a brief literature review, and §3.5.2 provides details relevant to our implementations. In §3.5.3, we discuss the computational benefit of employing the updating procedures outlined in §3.4. In §3.5.4, we report our experience with benchmark problem instances for the VRP. Finally, in §3.5.5, we report our experience with benchmark problem instances for the VRPSD.

3.5.1 Test Problems

The classical VRP can be formally defined as follows. Let $G = (V, E)$ be a connected digraph where $V = \{0, 1, \ldots, n\}$ is a set of $n + 1$ nodes and E is a set of edges. Node 0 is a depot at which m identical vehicles of capacity Q are based, while the remaining nodes are customers. A deterministic symmetric travel cost matrix is defined on E. With each customer is associated a nonnegative deterministic demand to be collected or delivered, but not both. The demand of each route cannot exceed Q. The objective consists of determining a set of routes starting and ending at the depot that minimize the total travel cost. The VRP has been studied extensively in

Algorithm 3.4 Update $\mathcal{R}(\pi)$ to Obtain $\mathcal{R}(\pi')$ for an Exchange Neighbor

```
1: **Input**: cyclic order $\pi$, candidate route set $\mathcal{R}(\pi)$, indices $i$ and $j$
2: **Output**: neighbor cyclic order $\pi'$, candidate route set $\mathcal{R}(\pi')$
3: construct $\pi'$ from $i$, $j$, and $\pi$
4: $\mathcal{R}(\pi') \leftarrow \mathcal{R}(\pi)$
5: **for** all $p \in \{i, j\}$ **do**
6:     **if** $|\mathcal{R}_{\pi'(p)}| > \text{dist}(p, p+1)$ **then**
7:         $\mathcal{R}_{\pi'(p)} \leftarrow \mathcal{R}_{\pi'(p)} \setminus \mathcal{R}_{\pi'(p),\pi'(p+1)}$
8:         $\mathcal{R}_{\pi'(p)} \leftarrow \text{SWEEP}(\pi', \mathcal{R}_{\pi'(p)}, p, p+1)$
9:     **else**
10:        $\mathcal{R}_{\pi'(p)} \leftarrow \text{SWEEP}(\pi', \mathcal{R}_{\pi'(p)}, p, p+1)$
11: **for** $p = i - 1$ **to** $j + 1$ **step** $-1$ **do**
12:     **if** $|\mathcal{R}_{\pi'(p)}| > \text{dist}(p, i)$ **then**
13:         $\mathcal{R}_{\pi'(p)} \leftarrow \mathcal{R}_{\pi'(p)} \setminus \mathcal{R}_{\pi'(p),\pi'(i)}$
14:         $\mathcal{R}_{\pi'(p)} \leftarrow \text{SWEEP}(\pi', \mathcal{R}_{\pi'(p)}, p, i)$
15:     **else if** $|\mathcal{R}_{\pi'(p)}| = \text{dist}(p, i)$ **then**
16:         $\mathcal{R}_{\pi'(p)} \leftarrow \text{SWEEP}(\pi', \mathcal{R}_{\pi'(p)}, p, i)$
17:     **else**
18:         break
19: **for** $p = j - 1$ **to** $i + 1$ **step** $-1$ **do**
20:     **if** $|\mathcal{R}_{\pi'(p)}| > \text{dist}(p, j)$ **then**
21:         $\mathcal{R}_{\pi'(p)} \leftarrow \mathcal{R}_{\pi'(p)} \setminus \mathcal{R}_{\pi'(p),\pi'(j)}$
22:         $\mathcal{R}_{\pi'(p)} \leftarrow \text{SWEEP}(\pi', \mathcal{R}_{\pi'(p)}, p, j)$
23:     **else if** $|\mathcal{R}_{\pi'(p)}| = \text{dist}(p, j)$ **then**
24:         $\mathcal{R}_{\pi'(p)} \leftarrow \text{SWEEP}(\pi', \mathcal{R}_{\pi'(p)}, p, j)$
25:     **else**
26:         break
```

the literature, and we do not attempt to provide a complete review here. For further details on solution methodologies, we refer the reader to the categorized bibliography of Gendreau et al. (2008).

In contrast to the VRP, where customer demand is deterministic, the VRPSD associates with each customer a nonnegative stochastic demand, as described in §2.1. We assume that customer demands are independent and that demand becomes known only when the vehicle arrives at the customer location. As in Laporte et al. (2002) and Christiansen and Lysgaard (2007), we require the expected demand of a route to be less than or equal to vehicle capacity. A direct consequence of stochastic demand is that a planned vehicle route may fail at a given customer location whenever the accumulated demand exceeds Q. In such a case, a *failure* is said to occur and a *recourse action* generating extra costs must be implemented. Thus, the objective is to design a set of a priori, or fixed, routes such that the expected travel cost is minimized. To be consistent with the majority of the literature, we restrict the recourse action to return trips to the depot. As demonstrated by (Teodorović and Pavković, 1992), it is straightforward to compute the criterion (2.14) for a fixed route policy.

Compared to the classical VRP, the VRPSD has received relatively little attention in the literature despite its applicability to many distribution systems. Exact algorithms for solving the VRPSD include the methods of Gendreau et al. (1995), Hjorring and Holt (1999), Laporte et al. (2002), and Christiansen and Lysgaard (2007). The integer L-shaped algorithm of Laporte et al. (2002) has been successful in solving to optimality problem instances with up to 100 customers and two vehicles. The branch-and-price procedure of Christiansen and Lysgaard (2007) has solved to optimality problem instances with many more vehicles, but fewer customers. Recent heuristic procedures for the VRPSD include the methods of Gendreau et al. (1996), Yang et al. (2000), Chepuri and Homem-de Mello (2005), Novoa et al. (2006), and Ak and Erera (2007). Many of these methods report solutions for problem instances including up to 100 customers and several vehicles. Due to a lack of standard benchmark problems for the VRPSD, however, it is difficult to compare the effectiveness of different heuristic methods.

3.5.2 Implementation Details

For both test problems, we implement the cyclic-order neighborhood search within a simulated annealing algorithm. Simulated annealing (Kirkpatrick et al., 1983; Johnson et al., 1989, 1991) is a local search algorithm in which non-improving moves are probabilistically accepted in an attempt to avoid becoming trapped in a low-quality, locally optimal solution. We choose simulated annealing to govern the search process for two reasons. First, simulated annealing has proven successful on a variety of difficult combinatorial problems such as the set covering problem (Brusco et al., 1999), the vehicle routing problem with time windows (Bent and Van Hentenryck, 2004), and the orthogonal stock-cutting problem (Burke et al., 2009). Second, the simple logic of simulated annealing (as opposed to a more complex search procedure) allows us to more easily distill the impact of the cyclic-order neighborhoods.

At each iteration of our simulated annealing implementation, we randomly generate a neighbor cyclic order from a randomly-selected neighborhood (1-shift, 2-shift, 3-shift, reverse, or exchange) of the current cyclic order. We accept the neighbor solution with probability $\exp\left(-(g(x_{\pi'}) - g(x_\pi))^+/\tau\right)$, where $(y)^+ = y$ if $y > 0$ and 0 otherwise, τ is the current temperature (a control parameter in simulated annealing), and $g(x)$ is the total travel cost of routing plan x.

Our simulated annealing implementation begins with a radial ordering of the customers about the depot (i.e., customers are ordered by their polar angle with respect to a horizontal axis through the depot). For problems where customer locations are unknown, a randomly generated cyclic order can serve as the initial solution. The algorithm employs a geometric cooling schedule with a multiplier of 0.97 and an initial temperature of 10. At each temperature, we generate 50,000 neighbor cyclic orders via the method described above. The procedure terminates after performing at least 100 temperature decrements and not updating the best-found cyclic order for 75 successive temperature decrements. These parameter settings, established with computational experiments, appear to be robust for the VRP instances considered in this chapter.

For the VRPSD, we employ a two-phase procedure. In the first phase, we apply simulated annealing to solve the deterministic version, i.e., the classical VRP, in which customer demand is fixed at the average demand. The second phase seeks to further improve the solution by explicitly accounting for the cost of recourse actions resulting from potential route failures. In the second phase, we again employ simulated annealing, but account for stochastic demand and use the VRPSD objective function, i.e., minimizing expected travel cost (Teodorović and Pavković, 1992). We observe that, for a variety of VRPSD instances, a high-quality VRP solution is also a high-quality VRPSD solution, thus motivating our two-phase approach (Savelsbergh and Goetschalchx (1995) propose a similar two-phase procedure). The primary benefit of the two-phase procedure is reduction in computation time; we obtain good VRPSD solutions employing only the second phase, but with increased computational effort. In both phases, we generate 7,000 neighbor cyclic orders at each temperature, employ a temperature multiplier of 0.97, and begin with an initial temperature of 10. Phase one begins with a radial ordering of customers and phase two begins with the best cyclic order obtained in phase one. Both phases terminate after at least 100 temperature decrements and not updating the best-found cyclic order for 75 successive temperature decrements.

As discussed in §3.2, other authors also utilize sweep procedures to solve VRPs. These construction approaches typically seek to improve the sequence of customers in each sweep-generated route via a traveling salesman heuristic. In contrast to this practice, we allow the cyclic order to define the sequence of customers in a route. This offers two computational advantages. First, we can store a route r in memory as the pair $\langle \text{first}(r), \text{last}(r) \rangle$, as opposed to an array containing the sequence of customers. Second, we can calculate the cost of a route more efficiently by updating the cost from knowledge of a subroute rather than by adding the costs of all the edges in the route. For example, we can compute the cost of a VRP route $\langle \pi(i), \pi(p+1) \rangle$ easily from the cost of route $\langle \pi(i), \pi(p) \rangle$ as $\langle \pi(i), \pi(p+1) \rangle$ simply inserts the customer $\pi(p+1)$ at the end of route $\langle \pi(i), \pi(p) \rangle$. Note that this simple updating is not possible if the sequence of customers on a route is not strictly defined by the sweep procedure applied to a cyclic order. Similarly, the additional cost of recourse generated by inserting a customer on a VRPSD route depends only on the preceding customers and can be calculated efficiently from knowledge of a subroute. For more details on the route cost calculations for the VRPSD, see Teodorović and Pavković (1992).

The drawback of our approach is that it may be possible to improve the VRP solution corresponding to a given cyclic order by changing the sequence of the customers on one or more routes contained in $\mathcal{R}(\pi)$. Because the size of $\mathcal{R}(\pi)$ can be quite large, however, this significantly increases computation time. As we evaluate many cyclic orders during our search process, rather than performing the costly optimization of individual routes in $\mathcal{R}(\pi)$, we aim to find the appropriate routes by quickly evaluating more cyclic orders during our search.

We implement the cyclic-order simulated annealing procedures for the VRP and VRPSD in C++. We execute all computational experiments on 2.66GHz Intel Core2 Quad processors with 4GB of RAM and openSUSE Linux version 10.3 (we do not utilize parallel processing).

3.5.3 Benefit of Updating Procedures

We demonstrate the computational savings of our proposed updating schemes on 10 VRP instances proposed in Augerat et al. (1995). Christiansen and Lysgaard

(2007) adapt these instances to the VRPSD by assuming that customer demands are Poisson random variables with the mean demand for each customer equal to the deterministic value of demand given in the underlying VRP problem instance. The name of each problem instance indicates the number of customers (including the depot) and the minimum number of vehicles. For example, instance A-n38-k5 requires 37 customers be serviced by at least 5 vehicles. The instances are publicly available (Ralphs, 2008). Table 3.2 contains the percentage decrease in CPU time when $\mathcal{R}(\pi')$ is constructed by updating $\mathcal{R}(\pi)$ for a cyclic order π' in the neighborhood of π (as opposed to totally reconstructing $\mathcal{R}(\pi')$). We compute each entry of Table 3.2 by observing the CPU time required to evaluate each neighboring cyclic order in the neighborhood of π, where π is the radial ordering of customers.

Overall, the updating procedures provide substantial computational savings. Because the calculation of expected route cost is significantly more expensive than the computation of deterministic route cost in the classical VRP, the savings are particularly large for the VRPSD. Other factors affecting the magnitude of computational savings include problem size (in terms of the number of customers), type of cyclic-order neighborhood, and the required fleet size. By comparing results for instances with the similar numbers of vehicles but different numbers of customers, we observe that as the number of customers increases, the computational savings also tends to increase (although this is not always true depending on the location and demand of the additional customers).

The type of cyclic-order neighborhood also affects the magnitude of computational savings. The savings for the scramble and reverse neighborhoods is, on average, 33.4 percent less than the savings afforded by the updating schemes for the k-shift and exchange neighborhoods. When updating $\mathcal{R}(\pi')$ from $\mathcal{R}(\pi)$, the scramble and reverse neighborhoods require the removal and insertion of more routes than the k-shift and exchange neighborhoods. For a similar reason, the computational savings for the k-shift neighborhood decreases as k increases. We conclude that the computational savings for a neighborhood is negatively correlated with the degree of change it causes in the set of routes corresponding to the cyclic order.

We achieve larger computational savings by updating rather than reconstructing $\mathcal{R}(\pi')$ for instances in which more vehicles are required to meet customer demand, i.e., vehicle capacity is more constraining. By examining problem instances in Table 3.2 with similar numbers of customers but different numbers of vehicles, we can observe this effect. For example, comparing instances P-n21-k2 and P-n22-k8 across all six neighborhood structures suggests that the updating procedures provide greater computational savings when eight vehicles are required instead of only two. We can explain this observation by examining typical sets of candidate routes for each problem instance. The size of each $\mathcal{R}_{\pi(i)}$ in P-n22-k8 is generally smaller than the size of each $\mathcal{R}_{\pi(i)}$ in P-n21-k2. In other words, because vehicle capacity is more restrictive in P-n22-k8, each call to the SWEEP procedure in Algorithm 3.1 generates fewer routes than for problem instance P-n21-k2. Consequently, the number of routes in $\mathcal{R}(\pi)$ is larger for P-n21-k2, thus requiring the removal and insertion of more routes when updating $\mathcal{R}(\pi)$ for a neighboring cyclic order. Thus, as vehicle capacity decreases, the procedures outlined in Algorithms 3.2, 3.3, and 3.4 provide more of an advantage over totally reconstructing $\mathcal{R}(\pi')$.

3.5.4 VRP Results

In this section, we examine the performance of cyclic-order neighborhood search governed by simulated annealing using the 14 benchmark problem instances (labeled C1 through C14) for the classical VRP proposed by Christofides et al. (1979). Problems C6 through C10, C13, and C14 constrain both vehicle capacity and route duration. The remaining problems only constrain vehicle capacity. We compare the solutions obtained by cyclic-order simulated annealing to the best known solutions, as well as the solutions obtained by the algorithmic predecessors of cyclic order neighborhoods (discussed in §3.2). Column 1 of Table 3.3 denotes the problem instance with the number of customers in parentheses. Column 2 indicates the best known solutions for each problem instance. Best known solutions are obtained by Rochat and Taillard (1995) for all instances except C5, which is obtained by Nagata and Bräysy (2008). Columns 3-5, 6-8, and 9-11 respectively display the results obtained by the *sweep* method of Gillett and Miller (1974), the *petal* method of Foster and Ryan (1976), and the *improved petal* method of Renaud et al. (1996). The columns labeled "Cost" display the solution values obtained by each method. The columns labeled "CPU" display the number of CPU seconds (on a Sun Sparc 2 workstation) required for each method to obtain a solution. The solution values and CPU times reported in Table 3.3 for the sweep, petal, and improved petal methods are taken

Table 3.2: Percentage Decreases in CPU Time When $\mathcal{R}(\pi')$ is Obtained by Updating $\mathcal{R}(\pi)$

	1-shift		**5-shift**		**10-shift**		**Reverse**		**Scramble**		**Exchange**	
Problem	**VRP**	**VRPSD**	**VRP**	**VRPSD**	**VRP**	**VRPSD**	**VRP**	**VRPSD**	**VRP**	**VRPSD**	**VRP**	**VRPSD**
A-n38-k5	57.4	73.0	52.4	63.6	48.6	60.7	20.0	31.6	17.6	32.5	57.1	71.1
A-n44-k6	63.5	76.8	56.1	68.6	54.4	68.2	20.8	35.6	21.5	33.9	61.0	74.1
A-n45-k6	62.8	76.7	56.3	68.5	54.1	66.7	20.7	36.5	21.4	36.5	62.2	73.8
A-n53-k7	65.2	76.0	60.0	70.2	58.3	69.3	23.8	36.9	23.0	37.0	64.9	75.2
P-n20-k2	32.7	39.4	19.7	24.4	23.0	26.3	8.0	17.2	7.3	17.0	30.5	34.1
P-n21-k2	33.2	38.6	20.0	23.7	20.0	20.5	8.0	17.1	8.0	16.8	28.8	33.8
P-n22-k8	51.3	77.8	46.7	76.2	48.8	78.6	15.1	34.5	13.8	33.3	46.2	71.4
P-n23-k8	54.1	78.9	51.5	73.3	50.0	70.0	15.8	33.3	15.3	38.1	53.3	70.0
P-n50-k10	65.8	84.2	62.5	80.0	60.9	80.6	24.4	40.0	23.0	38.5	65.4	82.1
P-n50-k8	65.9	80.4	61.2	76.0	60.8	75.6	22.1	38.6	22.4	36.8	65.3	78.6
Average	55.2	70.2	48.6	62.5	47.9	61.7	17.9	32.1	17.3	32.0	53.5	66.4

Table 3.3: Computational Results for the VRP

	Best	**Sweep**			**Petal**			**Improved Petal**			**Simulated Annealing with Cyclic-Order Neighborhoods**				
Problem	**Cost**	**Cost**	**CPU**	**Gap**	**Cost**	**CPU**	**Gap**	**Cost**	**CPU**	**Gap**	**Best Cost**	**Avg. Cost**	**St. Dev.**	**Avg. CPU**	**Gap**
C1 (50)	524.61	531.90	7.2	1.39	531.90	6.0	1.39	524.61	45.6	0.00	524.61	525.45	2.65	345.1	0.00
C2 (75)	835.26	884.20	10.2	5.86	885.02	4.2	5.96	854.09	31.2	2.25	838.04	846.06	4.90	430.2	0.33
C3 (100)	826.14	846.34	70.8	2.45	836.34	19.2	1.23	830.40	230.4	0.52	826.14	831.57	4.19	1415.5	0.00
C4 (150)	1028.42	1075.38	151.8	4.57	1070.50	24.6	4.09	1054.62	355.8	2.55	1039.83	1046.53	4.99	2369.5	1.11
C5 (199)	1291.29	1396.05	216.0	8.11	1406.84	24.6	8.95	1354.23	372.6	4.87	1320.69	1334.01	7.64	3599.8	2.28
C6 (50)	553.43	560.08	9.6	1.20	560.08	5.4	1.20	560.08	33.6	1.20	555.43	556.02	1.47	218.8	0.36
C7 (75)	909.68	965.51	11.4	6.14	968.89	4.2	6.51	922.75	25.8	1.44	909.68	914.37	3.54	232.2	0.00
C8 (100)	865.94	883.56	88.2	2.03	877.80	15.0	1.37	877.29	174.6	1.31	865.95	872.21	4.79	791.2	0.00
C9 (150)	1162.55	1220.71	180.0	5.00	1220.20	15.6	4.96	1194.51	214.8	2.75	1171.43	1192.99	14.50	1097.4	0.76
C10 (199)	1395.85	1526.64	294.6	9.37	1515.95	21.0	8.60	1470.31	311.4	5.33	1424.01	1441.40	10.30	1709.4	2.02
C11 (120)	1042.11	1265.65	211.2	21.45	1252.84	36.6	20.22	1109.14	702.0	6.43	1042.12	1043.34	1.82	2987.5	0.00
C12 (100)	819.56	919.51	38.4	12.20	824.77	12.6	0.64	824.77	126.6	0.64	819.56	819.63	0.15	1029.0	0.00
C13 (120)	1541.14	1785.30	134.4	15.84	1773.69	15.6	15.09	1585.20	198.6	2.86	1565.60	1567.76	2.33	1190.9	1.59
C14 (100)	866.37	911.81	51.0	5.24	894.77	10.2	3.28	885.87	101.4	2.25	866.37	866.60	0.25	931.8	0.00
Average	975.88	1055.19	105.3	7.20	1044.26	15.3	5.96	1003.42	208.9	2.46	983.53	989.85	4.54	1310.6	0.60

from Renaud et al. (1996). The columns labeled "Gap" display the percentage above the best known solution values. For example, if the best known solution value is x and the sweep method returns a solution value of y, then the gap is $(y - x) \times 100/x$. Columns 12-16 display the results of cyclic-order simulated annealing. These columns report, over 10 runs, the best solution values, the average solution values, the standard deviations of solution values, the average CPU seconds (on a 2.66GHz Intel Core2 Quad processor), and the percentage above the best known solutions.

As Table 3.3 illustrates, our approach obtains best known solutions on 7 of the 14 benchmark problems. Moreover, our solutions are, on average, 0.60 percent above the best known solutions, a substantial improvement over the sweep (7.20 percent), petal (5.96 percent), and improved petal (2.46 percent) methods. The computation time required by cyclic-order simulated annealing is longer than that required by the sweep, petal, and improved petal methods, especially considering the difference in computer architectures. As discussed in §3.5.2, the results reported in Table 3.3 are obtained by generating 50,000 neighboring cyclic orders at each temperature. If we decrease the number of neighbors generated to 5,000, the computation time decreases from an overall average of 1,310.6 CPU seconds to 173.2 CPU seconds, while the average gap increases from 0.60 to 1.73 percent. Thus, when computation time is an important consideration, cyclic-order simulated annealing is still able to deliver solutions, but of slightly less quality.

3.5.5 VRPSD Results

In this section, we compare the performance of cyclic-order neighborhood search embedded within simulated annealing to benchmark results for the VRPSD. Following the protocol in the literature, we consider VRPSD instances in which the customer demand is a Poisson random variable with the mean demand for each customer equal to the deterministic value of demand given in the underlying VRP problem instance. Table 3.4 compares the solutions obtained by our two-phase procedure for the VRPSD to the optimal solutions of Christiansen and Lysgaard (2007) and to the expected cost of best known routing plans for the (deterministic) VRP. In the columns displaying Christiansen and Lysgaard's results, when a value is reported for both the number of vehicles and the cost, this corresponds to the optimal routing cost. If only a cost is reported, then this is the cost of the best integer solution obtained within 1200 CPU seconds on a Pentium Centrino 1500 MHz processor (Christiansen and Lysgaard (2007) do not report the number of vehicles in these cases). If neither the number of vehicles or a cost is reported (such entries are denoted by "–"), then an integer solution was not obtained within 1200 CPU seconds. The only exception to this is instance P-n60-k15, where Christiansen and Lysgaard report a CPU time of 1348 CPU seconds, which is the time required to solve the root node in their branch and price procedure. In this case, the root node yielded an optimal integer solution which they report.

As in Table 3.3, we compute the entries of Table 3.4 corresponding to our approach over 10 runs. The column labeled "Gap" is the percentage above the minimum

Table 3.4: Computational Results for the VRPSD

	Christiansen and Lysgaard			Best Deterministic		Simulated Annealing with Cyclic-Order Neighborhoods					
Problem	# Veh.	Cost	CPU	# Veh.	Cost	Best # Veh.	Best Cost	Avg. Cost	St. Dev.	Avg. CPU	Gap
A-n32-k5	5	853.60	282	5	890.13	5	853.60	853.60	0.00	453.3	0.00
A-n33-k5	5	704.20	8	5	722.99	5	704.20	705.91	2.79	442.4	0.00
A-n33-k6	6	793.90	49	6	816.58	6	793.90	793.95	0.10	349.1	0.00
A-n34-k5	–	827.87	1200	5	839.95	6	826.87	827.26	1.23	450.8	-0.12
A-n36-k5	–	–	–	5	907.55	5	858.71	859.48	1.42	507.5	-5.38
A-n37-k5	–	708.34	1200	5	709.83	5	708.34	709.67	4.18	737.7	0.00
A-n37-k6	–	1030.75	1200	6	1069.32	7	1030.73	1031.74	3.20	400.5	0.00
A-n38-k5	–	778.09	1200	5	831.99	6	775.14	775.25	0.35	644.6	-0.38
A-n39-k5	6	869.18	3	5	903.26	6	869.18	869.58	0.46	552.8	0.00
A-n39-k6	6	876.60	279	6	960.81	6	876.60	883.44	7.29	573.3	0.00
A-n44-k6	–	1025.48	1200	6	1047.18	7	1025.48	1029.72	6.10	687.4	0.00
A-n45-k6	–	–	–	6	1096.19	7	1026.73	1027.92	1.36	561.2	-6.34
A-n45-k7	7	1264.83	882	7	1302.20	7	1264.99	1288.70	17.13	503.3	0.01
A-n46-k7	–	1002.41	1200	7	1069.66	7	1002.22	1003.19	1.07	643.1	-0.02
A-n48-k7	–	–	–	7	1248.27	7	1187.14	1188.06	1.96	705.3	-4.90
A-n53-k7	–	–	–	7	1180.10	8	1124.27	1129.36	4.06	986.9	-4.73
A-n54-k7	–	–	–	7	1342.87	8	1287.07	1292.29	8.87	834.8	-4.16
A-n55-k9	–	–	–	9	1264.18	10	1179.11	1184.77	5.70	604.9	-6.73
A-n60-k9	–	–	–	9	1608.40	10	1529.82	1535.35	7.38	824.9	-4.89
E-n22-k4	4	411.57	1	4	411.73	4	411.57	411.57	0.00	186.3	0.00
E-n33-k4	4	850.27	86	4	850.27	4	850.27	851.24	3.07	763.3	0.00
E-n51-k5	–	–	–	5	553.26	6	552.26	552.81	0.79	1531.7	-0.18
P-n16-k8	8	512.82	0	8	512.82	8	512.82	512.82	0.00	18.5	0.00
P-n19-k2	3	224.06	153	2	229.68	3	224.06	224.06	0.00	479.8	0.00
P-n20-k2	2	233.05	352	2	233.05	2	233.05	233.05	0.00	530.8	0.00
P-n21-k2	2	218.96	5	2	218.96	2	218.96	218.96	0.00	650.6	0.00
P-n22-k2	2	231.26	219	2	231.26	2	231.26	231.26	0.00	715.4	0.00
P-n22-k8	9	681.06	0	8	707.80	9	681.06	681.06	0.00	52.5	0.00
P-n23-k8	9	619.52	1	8	662.31	9	619.53	619.57	0.12	50.0	0.00
P-n40-k5	5	472.50	6	5	475.45	5	472.50	472.50	0.00	802.2	0.00
P-n45-k5	–	–	–	5	546.05	5	533.52	537.13	4.19	1141.8	-2.29
P-n50-k10	–	–	–	10	792.20	11	760.94	764.12	3.90	374.5	-3.95
P-n50-k7	–	–	–	7	606.41	7	582.37	584.95	2.33	710.2	-3.96
P-n50-k8	–	–	–	8	724.69	9	669.81	674.11	4.54	443.0	-7.57
P-n51-k10	11	809.70	430	10	859.24	11	812.74	816.95	4.27	356.8	0.37
P-n55-k10	–	–	–	10	797.21	10	745.70	752.36	3.94	619.4	-6.46
P-n55-k15	18	1068.05	792	15	1191.34	18	1068.05	1072.17	3.71	227.1	0.00
P-n55-k7	–	–	–	7	616.44	7	588.56	593.10	3.15	1179.6	-4.52
P-n60-k10	–	–	–	10	831.24	11	804.24	810.22	2.55	668.0	-3.25
P-n60-k15	16	1085.49	1348	15	1133.30	16	1087.41	1098.11	6.91	290.3	0.18

of the solution values given in columns 3 and 6. A negative number in this column indicates that the solution value obtained by our two-phase procedure improves upon the solution value of Christiansen and Lysgaard, the value of the best deterministic solution, or both.

Of the 19 problem instances in which Christiansen and Lysgaard (2007) obtain optimal solutions, our two-phase procedure obtains 16 optimal solutions. The percentages above the optimal solution values are small for the three instances where we obtain suboptimal solutions: A-n45-k7 (gap of 0.01), P-n51-k10 (gap of 0.37), and P-n60-k15 (gap of 0.18). Of the 21 problem instances in which Christiansen and Lysgaard (2007) do not obtain provably optimal solutions, our two-phase procedure matches or improves upon the best integer solution returned by Christiansen and Lysgaard's method or the expected value of the best known deterministic solution. The average gap for these 21 instances is -3.33. The difference in computer architectures makes it difficult to compare run times, but we find the average run time of 581.4 CPU seconds for our two-phase method to be acceptable given that practitioners typically employ VRPSD solutions as a set of fixed routes to be used over extended periods.

3.6 Conclusion and Future Research

This chapter examines cyclic-order neighborhoods for VRPs and proposes procedures to facilitate the efficient search of such neighborhoods in local search schemes. We demonstrate that, when embedded in a simulated annealing framework, cyclic-order neighborhood search is capable of obtaining high quality solutions for the classical VRP and the VRPSD. These results underscore the potential for the use of cyclic-order neighborhoods as a general solution method for a variety of routing problems. Without tailoring our solution procedure to a specific routing problem, we are able to obtain solutions to the classical VRP within 0.60 percent of best known solutions and are able to match 16 of 19 known optimal VRPSD solutions.

Due to the computational effort required to execute cyclic-order neighborhood search, one direction for future research is to investigate techniques to reduce the size of the cyclic-order search space. This could perhaps be accomplished by exploiting information pertaining to customer location. Specifically, cyclic orders may be removed from a neighborhood that would generate candidate routes unlikely to be observed in optimal solutions. For example, customers on opposite sides of the depot are unlikely to be serviced on the same route in an optimal solution. Thus, including such routes in the set of candidate routes is unnecessary. Such reductions in neighborhood size may be especially advantageous when it is necessary to evaluate all cyclic orders in a given neighborhood, such as in best-improving search, variable neighborhood search, or tabu search. How to use memory in the search procedure is another direction for future research. This might be accomplished by exploiting information contained in a history of candidate route sets ($\mathcal{R}(\pi)$), or by identifying common sequences of customers in a history of cyclic orders.

CHAPTER 4
ROLLOUT POLICIES FOR APPROXIMATE DYNAMIC PROGRAMMING

4.1 Introduction

The static routing policies we consider in §3 require vehicles to visit customers according to fixed, or a priori, routes. In this chapter, we develop a methodological framework to relax this restriction. In §5 and §6, we employ the techniques of this chapter to develop fully dynamic routing policies, i.e., routes that can be changed to account for new information. To generate these dynamic policies, we employ the solution method generally known as *approximate dynamic programming* (ADP).

ADP seeks to overcome the well-known curses of dimensionality (Powell, 2007) associated with dynamic programs. Traditional ADP methods, which often rely on parametric functional approximations of the value functions (Bellman, 1957), have proven effective on a variety of problems, e.g., Topaloglu and Powell (2006) or Patrick et al. (2008). In contrast to these parametric ADP methods, *rollout policies* (Bertsekas et al., 1997; Bertsekas, 2000) employ heuristic optimization techniques to approximate the value functions, thereby removing the dependency on a particular functional class. Rollout policies have also demonstrated success on a variety of problems, e.g., Bertsimas and Popescu (2003) or Secomandi (2001). However, due to the computation required to evaluate actions, practical applications of rollout policies are often limited to problems with relatively small action spaces. In this chapter, we suggest extensions of traditional rollout policies that more readily accommodate dynamic programs with large action spaces, thereby providing a general framework for using heuristic optimization techniques to address large-scale dynamic programs.

By partitioning the state transition of a Markov decision process (MDP) (i.e., a stochastic dynamic program) into two parts, we introduce two extensions which we refer to as *pre-decision* and *post-decision rollout policies.* Furthermore, we discuss how these two new types of rollout policy may be used in combination, resulting in a *hybrid rollout policy* that, in our experience, achieves high quality solutions and results in a computational reduction sufficient for real-time implementation in large-scale settings. Underscoring the importance of our work is the recent conversation of Powell (2010b), Tsitsiklis (2010), Ruszczyński (2010), and Powell (2010a), which calls for further development of ADP methods to solve large-scale MDPs.

In §4.2, we introduce the notation used throughout the chapter. In §4.3, we discuss the computational challenge associated with traditional rollout policies. In §4.4, we propose pre- and post-decision rollout policies aimed at reducing this computational burden. In §4.5, we show how pre- and post-decision rollout policies can be used in combination in a hybrid rollout policy. Finally, in §4.6 and §4.7, we discuss special cases where the performance of our rollout policies can be bounded. We also provide analytical results that position our extensions within the rollout literature and identify conditions under which our proposed rollout methods are equivalent to the traditional form.

4.2 Notation

Throughout this chapter, we discuss rollout policies in the context of a stochastic dynamic program with state space $\mathcal{S}$ and action space $\mathcal{A}(s)$ for each $s \in \mathcal{S}$. The set of absorbing states is $\mathcal{S}_K \subseteq \mathcal{S}$, where K denotes the final decision epoch. A state transition from the current state s_k to s_{k+1} is a function of the selected action a and the set of random variables W_{k+1} representing the random information arriving between decision epochs k and $k+1$: $s_{k+1} = S^M(s_k, a, W_{k+1})$. We adopt the notation of Powell (2007), where the superscript M in the state transition function $S^M(\cdot)$ denotes the transition with respect to a specific *model*, M. As advocated by Powell (2007), we split the transition into two parts – a transition from pre-decision state s_k to post-decision state $s_k^a = S^{M,a}(s_k, a)$ and a transition from s_k^a to pre-decision state $s_{k+1} = S^{M,W}(s_k^a, W_{k+1})$. The function $S^{M,a}(s_k, a)$ captures the deterministic aspects of the transition that occur as a result of selecting action a when the process is in state s_k. The superscript a indicates that function $S^{M,a}(\cdot)$ carries out all aspects of the state transition that occur as a result of selecting an action. The function $S^{M,W}(s_k^a, W_{k+1})$ captures all random aspects of the transition that occur when the process is in state s_k^a and random information W_{k+1} is observed. The superscript W indicates that function $S^{M,W}(\cdot)$ carries out all aspects of the transition that occur as a result of observing random information. Thus, $s_{k+1} = S^M(s_k, a, W_{k+1}) = S^{M,W}(S^{M,a}(s_k, a), W_{k+1})$. We assume known state transition probabilities $\mathbb{P}\{\cdot\}$. Selecting action a while the process is in state s_k at the kth decision epoch incurs reward $R_k(s_k, a)$. Let Π be the set of all Markovian deterministic policies. A policy $\pi \in \Pi$ defines a set of decision functions $\delta^\pi(s) : \mathcal{S} \mapsto \mathcal{A}(s)$ that specify an action for each state. We seek a policy $\pi \in \Pi$ that maximizes $\mathbb{E}^\pi\{\sum_{k=0}^{K} \gamma^k R_k(s_k, \delta^\pi(s_k))|s_0\}$, where $\mathbb{E}^\pi\{\cdot\}$ is the expectation operator with respect to policy π and $\gamma \in (0, 1]$ is a discount factor. For notational convenience, we denote the expected reward accrued by a policy π in decision epochs i through $j - 1$ as $H(\pi, i, j) \equiv \mathbb{E}^\pi\{\sum_{k=i}^{j-1} \gamma^{k-i} R_k(s_k, \delta^\pi(s_k))|s_i\}$. Similarly, we denote the expected reward accrued by a policy π from decision epoch i onward as $H(\pi, i) \equiv \mathbb{E}^\pi\{\sum_{k=i}^{K} \gamma^{k-i} R_k(s_k, \delta^\pi(s_k))|s_i\}$.

To clarify the concepts we discuss in this chapter, we present Figure 4.1, which depicts an MDP as a decision tree. A decision tree illustrates all possible sequences of action selection and realizations of random information in an MDP. Square nodes are *decision nodes* and round nodes are *state of nature*, or *chance*, nodes. The left-most decision node represents an initial pre-decision state, s_0. The three arcs originating from s_0 represent the three actions that comprise action space $\mathcal{A}(s_0)$. Selecting an action $a \in \mathcal{A}(s_0)$ earns a reward $R_0(s_0, a)$ (not displayed in Figure 4.1) and results in a transition to a post-decision state $s_0^a = S^{M,a}(s_0, a)$. There are three possible post-decision states, each represented by a chance node at the tail ends of the arcs emanating from the initial decision node. The dashed arcs originating from these chance nodes represent realizations of random information W_1. Observing a particular realization of W_1 results in a transition to pre-decision state $s_1 = S^{M,W}(s_0^a, W_1)$. In Figure 4.1, there are four possible pre-decision states s_1, each corresponding to a particular realization of W_1. The process of selecting actions and observing random information continues until an absorbing state is reached. In the context of a decision

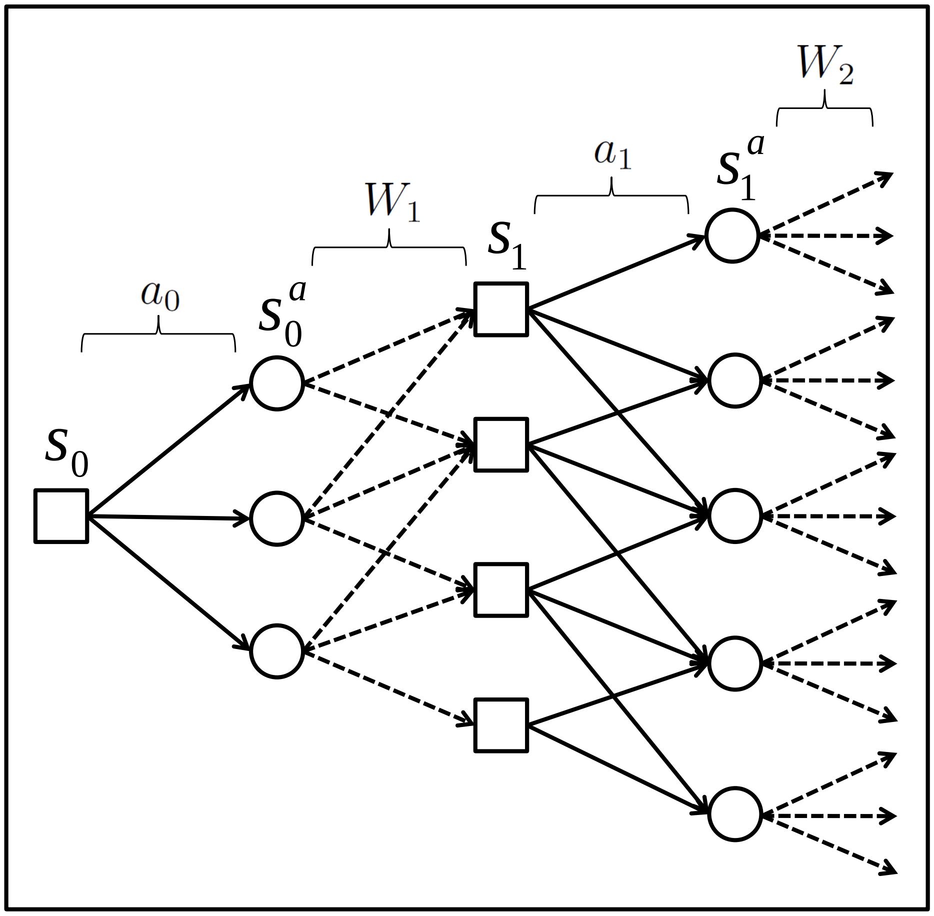

Figure 4.1: Markov Decision Process Depicted as a Decision Tree

tree, a policy is a set of rules that dictates the action to be selected at each decision node. The value of a policy is the expected reward earned by following the policy from the initial state to an absorbing state. An optimal policy is a policy that obtains the maximum possible value.

4.3 Traditional Rollout Policies

Let $\mathcal{H}(s)$ be a heuristic that, when applied in state $s \in \mathcal{S}$, generates a policy $\pi_{\mathcal{H}(s)}$. Heuristic $\mathcal{H}(\cdot)$ may be a simple set of rules that generate a policy, or it may be a more elaborate procedure, such as the local search heuristic we develop in §5. The only requirements we place on $\mathcal{H}(\cdot)$ are that it takes as input a state s and that it returns a policy capable of selecting actions in all future decision epochs. For example, applying a heuristic $\mathcal{H}(\cdot)$ to one of the four possible states s_1 in Figure 4.1 results in a policy to select actions in decision epoch one and in each subsequent decision epoch. It is instructive to think of applying $\mathcal{H}(\cdot)$ from a state s_k as a method to heuristically solve the problem of finding an optimal policy from state s_k on: $\max_{\pi \in \Pi} H(\pi, k)$.

Rollout policies aim to increase the effectiveness of a heuristic $\mathcal{H}(\cdot)$ by iteratively applying it, or rolling it out, at each decision epoch. In a *traditional rollout policy*, from a current state s_k, heuristic $\mathcal{H}(\cdot)$ is applied from each possible s_{k+1} that may result from selecting an action in $\mathcal{A}(s_k)$ and observing random information W_{k+1}. This process is shown in Figure 4.2a, which depicts heuristic $\mathcal{H}(\cdot)$ being applied to each of six possible states at the next decision epoch $k+1$, thus resulting in six heuristic policies $\pi_{\mathcal{H}(s_{k+1})}$. For a given action a, denote the set of reachable states as $\mathcal{S}(s_k, a) = \{s_{k+1} : \mathbb{P}\{s_{k+1}|s_k, a\} > 0\}$. In a traditional rollout policy, the future expected reward for selecting action a is approximated as the expected value of the heuristic policies $\{\pi_{\mathcal{H}(s_{k+1})} : s_{k+1} \in \mathcal{S}(s_k, a)\}$. Using the notation of §4.2, this expected value is denoted as $\mathbb{E}\{H(\pi_{\mathcal{H}(s_{k+1})}, k+1)|s_k, a\}$. A traditional rollout policy selects the action that maximizes the sum of the current-state reward, $R_k(s_k, a)$, and the approximate expected future reward. More formally, a traditional rollout policy, $\pi_{\mathcal{RH}}$, assigns the action for state s_k according to:

$$\delta^{\pi_{\mathcal{RH}}}(s_k) = \arg \max_{a \in \mathcal{A}(s_k)} \left\{ R_k(s_k, a) + \gamma \mathbb{E} \left\{ H(\pi_{\mathcal{H}(s_{k+1})}, k+1) \middle| s_k, a \right\} \right\}. \tag{4.1}$$

For a given action a, Algorithm 4.1 details the calculation of the expected future reward in a traditional rollout policy. The local variable λ, initialized on line 1, is used to update the calculation. The loop beginning on line 2 loops through all states reachable from state s_k given that action a is selected. For a given future state s_{k+1}, line 3 executes the heuristic $\mathcal{H}(\cdot)$ from state s_{k+1} to obtain heuristic policy $\pi_{\mathcal{H}(s_{k+1})}$, which is then evaluated as $H(\pi_{\mathcal{H}(s_{k+1})}, k+1)$ on line 4. The evaluation may be estimated via simulation, or, if possible, may be obtained via an exact calculation. Algorithm 4.1 concludes on line 5 by returning the expected future reward of the traditional rollout policy given that action a is selected.

Algorithm 4.1 Calculation of Expected Future Reward in Traditional Rollout Policies

```
1: $\lambda \leftarrow 0$
2: **for** $s_{k+1} \in \mathcal{S}(s_k, a)$ **do**
3:     execute $\mathcal{H}(s_{k+1})$ to obtain $\pi_{\mathcal{H}(s_{k+1})}$
4:     $\lambda \leftarrow \lambda + H(\pi_{\mathcal{H}(s_{k+1})}, k+1) \times \mathbb{P}\{s_{k+1}|s_k, a\}$
5: $\mathbb{E}\{H(\pi_{\mathcal{H}(s_{k+1})}, k+1)|s_k, a\} \leftarrow \lambda$
```

Algorithm 4.1 emphasizes the potential computational challenge associated with computing a traditional rollout policy. At each decision epoch k, Algorithm 4.1 is executed for each feasible action, resulting in $\sum_{a \in \mathcal{A}(s_k)} |\mathcal{S}(s_k, a)|$ executions of $\mathcal{H}(\cdot)$

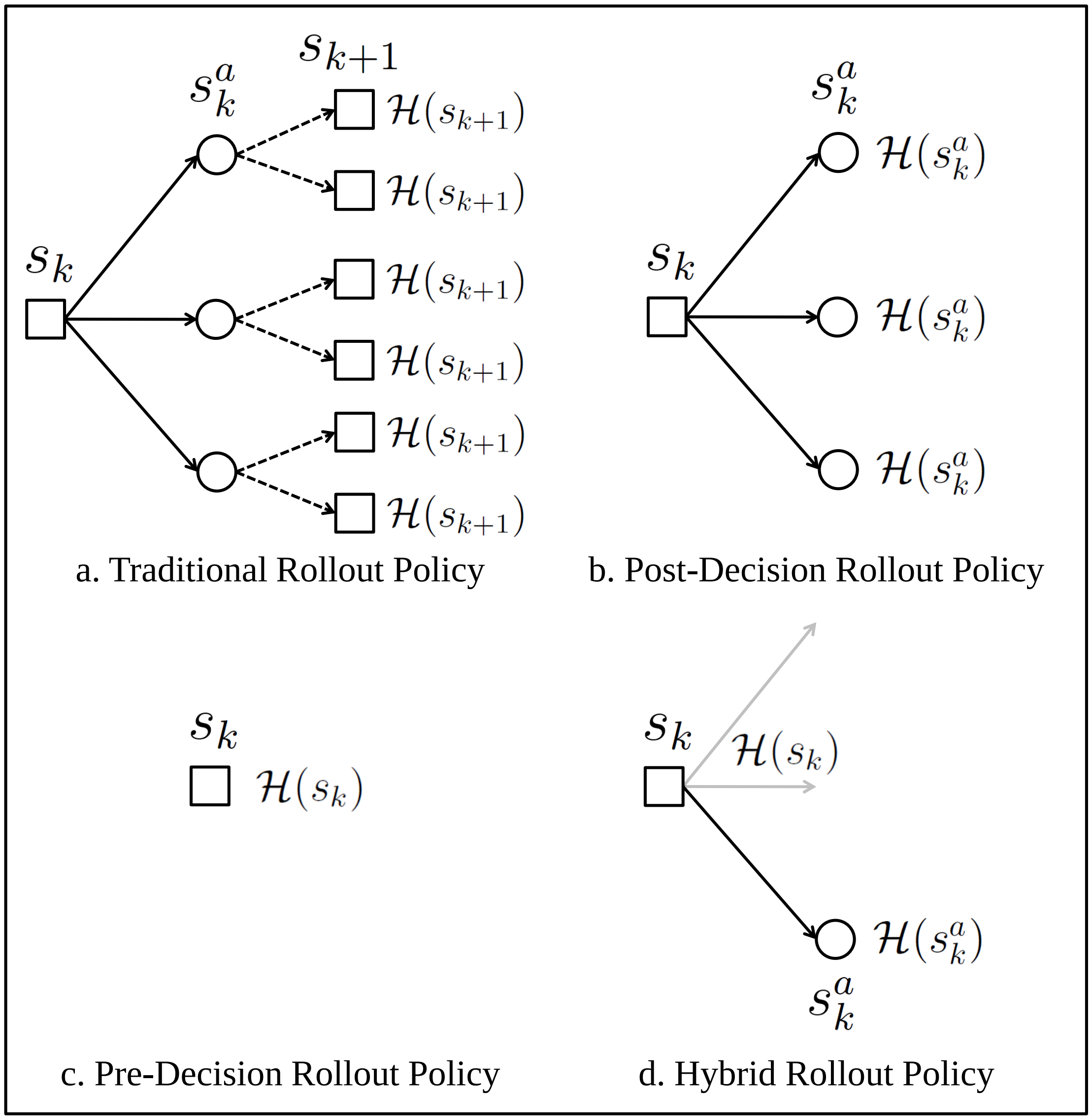

Figure 4.2: Traditional, Post-Decision, Pre-Decision, and Hybrid Rollout Policies

and evaluations of $\pi_{\mathcal{H}(\cdot)}$. As $|\mathcal{A}(s_k)|$ and $|\mathcal{S}(s_k, a)|$ increase, evaluating (4.1) becomes a computational challenge even when heuristic $\mathcal{H}(\cdot)$ is a simple procedure.

Guerriero et al. (2002) offer two ideas for reducing this computational burden by restricting $\mathcal{A}(s_k)$ in (4.1) to a set of "promising" actions, $\tilde{\mathcal{A}}(s_k) \subseteq \mathcal{A}(s_k)$. In their first method, Guerriero et al. (2002) suggest scoring each action in $\mathcal{A}(s_k)$ via an easily computable parametric function in order to identify $\tilde{\mathcal{A}}(s_k)$. Actions meeting a certain threshold, as determined by the scoring function, are assigned to $\tilde{\mathcal{A}}(s_k)$ and further investigated via heuristic $\mathcal{H}(\cdot)$ in the evaluation of (4.1). Guerriero et al. (2002) demonstrate the effectiveness of this method by obtaining competitive solutions for the classical traveling salesman problem. Although the method shows promise, identifying a good scoring function may be a difficult task.

Guerriero et al.'s second method solves a relaxation of the original problem and uses information from the corresponding solution to identify $\tilde{\mathcal{A}}(s_k)$ at each decision epoch. The method is shown to be effective on the classical traveling salesman problem by relaxing subtour constraints and using information related to these relaxed constraints to identify promising actions. However, the nature of the relaxation and how to extract useful information from it are problem dependent.

4.4 Pre- and Post-Decision Rollout Policies

We propose a *post-decision rollout policy*, denoted $\pi_{\mathcal{RH}'}$, as a method of reducing the computational burden of (4.1). The post-decision rollout policy utilizes post-decision state s_k^a:

$$\delta^{\pi_{\mathcal{RH}'}}(s_k) = \arg\max_{a \in \mathcal{A}(s_k)} \left\{ R_k(s_k, a) + \gamma \mathbb{E}\left\{ H(\pi_{\mathcal{H}(s_k^a)}, k+1) \middle| s_k, a \right\} \right\}. \qquad (4.2)$$

The key computational difference between (4.1) and (4.2) is the application of $\mathcal{H}(\cdot)$ to the unique post-decision state s_k^a instead of each possible pre-decision state s_{k+1}. For a given action a, state s_k^a is unique because the transition from s_k to $s_k^a = S^{M,a}(s_k, a)$ is deterministic by definition of the transition function. Algorithm 4.2 details the calculation of expected future reward in a post-decision rollout policy. Because post-decision state s_k^a is unique for a given $a \in \mathcal{A}(s_k)$, heuristic $\mathcal{H}(\cdot)$ needs be executed only once for each action, as shown in line 2 of Algorithm 4.2. Thus, a post-decision rollout policy executes heuristic $\mathcal{H}(\cdot)$ only $|\mathcal{A}(s_k)|$ times each decision epoch instead of $\sum_{a \in \mathcal{A}(s_k)} |\mathcal{S}(s_k, a)|$ times, as in a traditional rollout policy. Similar to a traditional rollout policy, the calculation of future expected reward requires policy $\pi_{\mathcal{H}(s_k^a)}$ be evaluated for each $s_{k+1} \in \mathcal{S}(s_k, a)$ (see line 4 of Algorithm 4.2).

Figure 4.2b depicts a post-decision rollout policy, where heuristic $\mathcal{H}(\cdot)$ is applied in each of three post-decision states. The post-decision rollout policy executes heuristic $\mathcal{H}(\cdot)$ three fewer times than the traditional rollout policy of Figure 4.2a.

Although post-decision rollout policies offer some computational benefits, they can still be challenging to evaluate for dynamic programs with large action spaces, even when $\mathcal{H}(\cdot)$ is a simple procedure. Motivated by this issue, we also propose a *pre-decision rollout policy*, denoted $\pi_{\mathcal{RH}''}$, that executes heuristic $\mathcal{H}(\cdot)$ only once from the current, pre-decision state:

Algorithm 4.2 Calculation of Expected Future Reward in Post-Decision Rollout Policies

```
1: $\lambda \leftarrow 0$
2: execute $\mathcal{H}(s_k^a)$ to obtain $\pi_{\mathcal{H}(s_k^a)}$
3: for $s_{k+1} \in \mathcal{S}(s_k, a)$ do
4:     $\lambda \leftarrow \lambda + H(\pi_{\mathcal{H}(s_k^a)}, k+1) \times \mathbb{P}\{s_{k+1}|s_k, a\}$
5: $\mathbb{E}\{H(\pi_{\mathcal{H}(s_k^a)}, k+1)|s_k, a\} \leftarrow \lambda$
```

$$\delta^{\pi_{\mathcal{RH}''}}(s_k) = \delta^{\pi_{\mathcal{H}(s_k)}}(s_k). \tag{4.3}$$

The key difference between a pre-decision rollout policy and both post-decision and traditional rollout policies is the use of $\mathcal{H}(\cdot)$. In (4.1) and (4.2), the feasible action set $\mathcal{A}(s_k)$ is enumerated and $\mathcal{H}(\cdot)$ generates policies that are in turn used to approximate the expected future reward for each feasible action. Thus, in traditional and post-decision rollout policies, $\mathcal{H}(\cdot)$ is used solely as a tool to evaluate actions. In (4.3), $\mathcal{H}(\cdot)$ is executed once from current state s_k and the resulting policy $\pi_{\mathcal{H}(s_k)}$ is used to select an action. Thus, instead of enumerating $\mathcal{A}(s_k)$, a pre-decision rollout policy implicitly evaluates feasible actions via $\mathcal{H}(\cdot)$. Hence, in contrast to (4.1) and (4.2), pre-decision rollout policies use $\mathcal{H}(\cdot)$ as both an evaluation tool for future actions and to generate a policy that selects an action. This is depicted in the pre-decision rollout policy of Figure 4.2c, where heuristic $\mathcal{H}(\cdot)$ is applied only once in pre-decision state s_k. The action selected in decision epoch k is the action recommended by policy $\pi_{\mathcal{H}(s_k)}$.

4.5 Hybrid Rollout Policies

Despite the computational benefits afforded by pre-decision rollout policies, a potential shortcoming of these policies provides motivation to use post-decision rollout policies when computationally tractable. In particular, heuristic $\mathcal{H}(\cdot)$ may not be capable of generating a policy that returns certain feasible actions, even when such actions belong to an optimal policy. Therefore, because all possible actions at a decision epoch are not enumerated when applying a pre-decision rollout policy (it directly recommends an action via $\pi_{\mathcal{H}(\cdot)}$), it may be impossible to select optimal, or even near-optimal, actions.

To illustrate this shortcoming and to facilitate subsequent discussion, we describe a simple heuristic for the single-vehicle VRPSDL. The heuristic, which we denote by $\bar{\mathcal{H}}(\cdot)$, generates a fixed route, which in turn defines a policy $\pi_{\bar{\mathcal{H}}(\cdot)}$ requiring the vehicle to visit customers in the sequence given on the fixed route, returning to the depot only when: (a) all customer demand has been served, (b) vehicle capacity is exhausted, or (c) if continuing on the fixed route will violate the route duration limit. Heuristic $\bar{\mathcal{H}}(\cdot)$ consists of the following three steps. First, identify a subset of customers $\mathcal{V}$ (e.g., three customers) with the largest expected or known demand. Second, for each customer $v \in \mathcal{V}$, construct a fixed route beginning at the vehicle's current location, then traveling to customer v, then visiting remaining customers with unknown/pending demand in descending order according to expected/known demand. Third, let $v^{\star}$ be the customer in $\mathcal{V}$ whose immediate visit results in the fixed route with the largest expected demand served and return this fixed route.

When using $\bar{\mathcal{H}}(\cdot)$ in a pre-decision rollout policy, the only actions that can potentially be returned by $\pi_{\bar{\mathcal{H}}(\cdot)}$ are the customers in $\mathcal{V}$ (or the depot if a route failure has occurred, thereby requiring replenishment before visiting $v^\star$). Thus, if the optimal action is to next visit a location not in $\mathcal{V}$, then it is impossible to select the optimal action. In contrast to pre-decision rollout policies, post-decision and traditional rollout policies for this single-vehicle VRPSDL use $\pi_{\bar{\mathcal{H}}(\cdot)}$ to approximate the expected future reward for *each* feasible action, thereby making it at least possible to select an optimal action.

Motivated by this issue, and the fact that post-decision rollout policies pose computational challenges, we propose a *hybrid rollout policy*, denoted $\pi_{\mathcal{HY}}$, that combines pre- and post-decision rollout policies. Let $\mathcal{A}'(s) \subseteq \mathcal{A}(s)$ be the set of actions possible for recommendation from state s by applying heuristic $\mathcal{H}(s)$ to obtain policy $\pi_{\mathcal{H}(s)}$. In the example above, when applying $\bar{\mathcal{H}}(s)$ to obtain $\pi_{\bar{\mathcal{H}}(s)}$, $\mathcal{A}'(s)$ consists of the customers in $\mathcal{V}$. The actions in $\mathcal{A}'(s)$ can be thought of as decision variables considered by heuristic $\mathcal{H}(s)$ and a pre-decision rollout policy as a method to select an action from $\mathcal{A}'(s)$. If we evaluate remaining actions $\mathcal{A}(s) \setminus \mathcal{A}'(s)$ in a post-decision rollout fashion, then, at decision epoch k, a hybrid rollout policy $\pi_{\mathcal{HY}}$ selects the action that maximizes

$$\max\left\{H(\pi_{\mathcal{H}(s_k)}, k), \left\{R_k(s_k, a) + \gamma \mathbb{E}\left\{H(\pi_{\mathcal{H}(s_k^a)}, k+1)\middle| s_k, a\right\} : a \in \mathcal{A}(s_k) \setminus \mathcal{A}'(s_k)\right\}\right\}. \tag{4.4}$$

Following the hybrid rollout policy of (4.4) overcomes the potential shortcoming of a pre-decision rollout policy by considering the entire set of feasible actions, $\mathcal{A}(s)$. Additionally, assuming $\mathcal{A}'(s)$ is nonempty, heuristic $\mathcal{H}(\cdot)$ is applied fewer times than in a post-decision rollout policy, thereby retaining some of the computational benefit afforded by a pre-decision rollout policy.

Figure 4.2d depicts a hybrid rollout policy. Applying heuristic $\mathcal{H}(\cdot)$ in pre-decision state s_k implicitly evaluates actions $\mathcal{A}'(s_k)$, comprised by the two gray arcs originating from s_k. The third action, represented by the black arc originating from s_k, comprises $\mathcal{A}(s_k) \setminus \mathcal{A}'(s_k)$, and is evaluated by heuristic $\mathcal{H}(\cdot)$ in the post-decision state. Thus, in this case, the heuristic is applied only twice, rather than three times as in the post-decision rollout policy.

4.6 Sequentially Consistent Heuristics

Generally speaking, the primary contribution of pre-decision, post-decision, and hybrid rollout policies is a reduction in computation over traditional rollout policies. Unless special structure is imposed on heuristic $\mathcal{H}(\cdot)$, however, we must rely on empirical methods to compare the performance of different types of rollout policies for a given problem. In this section, we discuss *sequentially consistent heuristics*, which provide a theoretical basis for bounding the performance of rollout policies.

Bertsekas et al. (1997) propose the concept of a sequentially consistent heuristic for deterministic dynamic programs and Secomandi (2003) extends the definition to stochastic dynamic programs. Let w_k be a realization of W_k, the random information arriving between decision epoch $k-1$ and k. Let $w = (w_k)_{k=1}^{K}$ be a sequence of realizations from decision epoch one to K and let W be the set of all such sequences. For a given $w \in W$, a *sample path* is the sequence of states visited by following a given policy π. Denote a sample path from a state s_k to an absorbing state $s_K \in \mathcal{S}_K$ by $p(s_k, \pi, w) = (s_k, s_{k+1}, \ldots, s_K)$. Denote the i^{th} element of a sample path, s_{k+i-1}, by $p_i(s_k, \pi, w)$. Denote the set of all possible sample paths generated by following a policy π from a state s_k by $\mathcal{P}(s_k, \pi) = \{p(s_k, \pi, w) : w \in W\}$.

A sequentially consistent heuristic $\mathcal{H}(\cdot)$ is characterized by the set of sample paths it generates from a current state s_k to absorbing states $\mathcal{S}_K$. If $\mathcal{H}(\cdot)$ is then applied to any s_{k+1} on any of these sample paths, then the sample paths resulting from $\mathcal{H}(s_{k+1})$ are identical to those resulting from $\mathcal{H}(s_k)$, except they begin at s_{k+1} instead of at s_k. In other words, if policy $\pi_{\mathcal{H}(s_k)}$ is used to select an action in decision epoch k, then following policy $\pi_{\mathcal{H}(s_k)}$ from thereon is equivalent to following policy $\pi_{\mathcal{H}(s_{k+1})}$. Similar to Secomandi (2003), Definition 4.1 formalizes the concept of a sequentially consistent heuristic, where $\oplus$ is the state concatenation operator, i.e., $s_k \oplus p(s_{k+1}, \pi, w) = (s_k, s_{k+1}, \ldots, s_K)$. Bertsekas et al. (1997) and Secomandi (2003) provide examples of sequentially consistent heuristics for deterministic and stochastic problems.

Definition 4.1. *Heuristic $\mathcal{H}(\cdot)$ is sequentially consistent if for all pairs of states s_k and s_{k+1}, $k = 0, 1, \ldots, K-1$, the following holds:*

$$\left\{p(s_k, \pi_{\mathcal{H}(s_k)}, w) : p(s_k, \pi_{\mathcal{H}(s_k)}, w) \in \mathcal{P}(s_k, \pi), w \in W, p_w(s_k, \pi_{\mathcal{H}(s_k)}, w) = s_{k+1}\right\}$$

$$= \left\{ s_k \oplus p(s_{k+1}, \pi_{\mathcal{H}(s_{k+1})}, w) : p(s_{k+1}, \pi_{\mathcal{H}(s_{k+1})}, w) \in \mathcal{P}(s_{k+1}, w), w \in W \right\}.$$

Sequentially consistent heuristics yield traditional rollout policies that possess an *improvement property.* As shown in Proposition 2 of Secomandi (2003), a traditional rollout policy $\pi_{\mathcal{RH}}$ performs no worse than the heuristic policy $\pi_{\mathcal{H}(\cdot)}$ obtained via a sequentially consistent $\mathcal{H}(\cdot)$. In Proposition 4.1, we state the same result for post-decision, pre-decision, and hybrid rollout policies. We also state that the improvement property holds at equality for pre-decision rollout policies. Similar to Secomandi (2003), we denote the expected reward of following a policy π from initial state s_0 to an intermediate state s_k and policy $\pi_{\mathcal{H}(s_k)}$ thereafter as $\mathbb{E}\{H(\pi, 0, k) + \gamma^k H(\pi_{\mathcal{H}(s_k)}, k)\}$. The expectation is conditional on a given initial state s_0. As in Proposition 2 of Secomandi (2003), we assume policy $\pi_{\mathcal{H}(\cdot)}$ and the rollout policies *terminate*, meaning they reach an absorbing state. Satisfaction of this condition must be verified for a given problem and heuristic $\mathcal{H}(\cdot)$, but it is generally not difficult to ensure (see Chapter 6 of Bertsekas (2000) for a more detailed discussion of termination).

Proposition 4.1. *Assume heuristic $\mathcal{H}(\cdot)$ is sequentially consistent and policy $\pi_{\mathcal{H}(\cdot)}$ terminates from any state. Further, assume post-decision rollout policy $\pi_{\mathcal{RH}'}$, hybrid rollout policy $\pi_{\mathcal{HY}}$, and pre-decision rollout policy $\pi_{\mathcal{RH}''}$ all terminate. Then, for all $\pi \in \{\pi_{\mathcal{RH}'}, \pi_{\mathcal{HY}}, \pi_{\mathcal{RH}''}\}$, the following sequence of inequalities holds for $k = 1, \ldots, K$:*

$$\begin{aligned} H(\pi_{\mathcal{H}(s_0)}, 0) &\leq \mathbb{E}\left\{H(\pi, 0, 1) + \gamma H(\pi_{\mathcal{H}(s_1)}, 1)\right\} \\ &\cdots \\ &\leq \mathbb{E}\left\{H(\pi, 0, k) + \gamma^k H(\pi_{\mathcal{H}(s_k)}, k)\right\} \\ &\cdots \\ &\leq \mathbb{E}\left\{H(\pi, 0)\right\}. \end{aligned} \tag{4.5}$$

Further, when $\pi = \pi_{\mathcal{RH}''}$, *the inequalities hold at equality.*

Proof. We first prove the property holds when $\pi = \pi_{\mathcal{RH}'}$. The proof is by induction and mirrors the proof of Proposition 2 in Secomandi (2003), which proves the same result for traditional rollout policies. Because $\mathcal{H}(\cdot)$ is assumed to be sequentially consistent, $H(\pi_{\mathcal{H}(s_0)}, 0) = \mathbb{E}\{H(\pi_{\mathcal{H}(s_0)}, 0, 1) + \gamma H(\pi_{\mathcal{H}(s_0^{\bar{a}})}, 1)\}$, where $\bar{a} = \delta^{\pi_{\mathcal{H}(s_0)}}(s_0)$ is the action selected by $\pi_{\mathcal{H}(s_0)}$ when the process is in state s_0. Since $\delta^{\pi_{\mathcal{H}(s_0)}}(s_0) \in \mathcal{A}(s_0)$, it follows from (4.2) that

$$\begin{aligned} H(\pi_{\mathcal{H}(s_0)}, 0) &\leq \max_{a \in \mathcal{A}(s_0)} \left\{R_0(s_0, a) + \gamma \mathbb{E}\left\{H(\pi_{\mathcal{H}(s_0^a)}, 1) \middle| s_0, a\right\}\right\} \\ &\equiv \mathbb{E}\left\{H(\pi_{\mathcal{RH}'}, 0, 1) + \gamma H(\pi_{\mathcal{H}(s_1)}, 1)\right\}. \end{aligned} \tag{4.6}$$

Hence, the property holds for $k = 1$. Assume the property holds for $k > 1$: $H(\pi_{\mathcal{H}(s_0)}, 0) \leq \cdots \leq \mathbb{E}\left\{H(\pi_{\mathcal{RH}'}, 0, k) + \gamma^k H(\pi_{\mathcal{H}(s_k)}, k)\right\}$. By conditioning on the random reward accrued to reach any intermediate state s_l from state s_0 before arriving at state s_k, the expected reward accrued to reach an absorbing state when $\pi_{\mathcal{RH}'}$ is used to take the first k decisions can be decomposed as

$$\begin{aligned} &\mathbb{E}\left\{H(\pi_{\mathcal{RH}'}, 0, k) + \gamma^k H(\pi_{\mathcal{H}(s_k)}, k)\right\} \\ &\qquad = \mathbb{E}\left\{H(\pi_{\mathcal{RH}'}, 0, l) + \mathbb{E}\left\{\gamma^l H(\pi_{\mathcal{RH}'}, l, k) + \gamma^k H(\pi_{\mathcal{H}(s_k)}, k)\right\}\right\}. \end{aligned} \tag{4.7}$$

Proceeding as in (4.6), it is straightforward to show that $\gamma^k H(\pi_{\mathcal{H}(s_k)}, k) \leq \mathbb{E}\{\gamma^k H(\pi_{\mathcal{RH}'}, k, k+1) + \gamma^{k+1} H(\pi_{\mathcal{H}(s_{k+1})}, k+1)\}$. Then, this inequality, the induction hypothesis, and (4.7) imply

$$\begin{aligned} H(\pi_{\mathcal{H}(s_0)}, 0) &\leq \mathbb{E}\left\{H(\pi_{\mathcal{RH}'}, 0, k) + \gamma^k H(\pi_{\mathcal{H}(s_k)}, k)\right\} \\ &\cdots \\ &\leq \mathbb{E}\{H(\pi_{\mathcal{RH}'}, 0, k) + \mathbb{E}\{\gamma^k H(\pi_{\mathcal{RH}'}, k, k+1) \\ &\qquad + \gamma^{k+1} H(\pi_{\mathcal{H}(s_{k+1})}, k+1)\}\} \\ &= \mathbb{E}\left\{H(\pi_{\mathcal{RH}'}, 0, k+1) + \gamma^{k+1} H(\pi_{\mathcal{H}(s_{k+1})}, k+1)\right\}. \end{aligned}$$

Therefore, the property holds for $k+1$. Since $\pi_{\mathcal{RH}'}$ terminates by assumption, the result holds by induction when $\pi = \pi_{\mathcal{RH}'}$.

The proof for $\pi = \pi_{\mathcal{HY}}$ follows the same structure. Similar to (4.6),

$$\begin{aligned} H(\pi_{\mathcal{H}(s_0)}, 0) &\leq \max\left\{H(\pi_{\mathcal{H}(s_0)}, 0), \left\{R_0(s_0, a) + \gamma \mathbb{E}\left\{H(\pi_{\mathcal{H}(s_0^a)}, 1) \middle| s_0, a\right\}\right.\right. && (4.8) \\ &\qquad \left.\left. : a \in \mathcal{A}(s_0) \setminus \mathcal{A}'(s_0)\right\}\right\} \\ &\equiv \mathbb{E}\left\{H(\pi_{\mathcal{HY}}, 0, 1) + \gamma H(\pi_{\mathcal{H}(s_1)}, 1)\right\}. && (4.9) \end{aligned}$$

Hence, the property holds for $k = 1$. Assume the property holds for $k > 1$: $H(\pi_{\mathcal{H}(s_0)}, 0) \leq \cdots \leq \mathbb{E}\left\{H(\pi_{\mathcal{HY}}, 0, k) + \gamma^k H(\pi_{\mathcal{H}(s_k)}, k)\right\}$. Similar to (4.7),

$$\begin{aligned} &\mathbb{E}\left\{H(\pi_{\mathcal{HY}}, 0, k) + \gamma^k H(\pi_{\mathcal{H}(s_k)}, k)\right\} \\ &\qquad = \mathbb{E}\left\{H(\pi_{\mathcal{HY}}, 0, l) + \mathbb{E}\left\{\gamma^l H(\pi_{\mathcal{HY}}, l, k) + \gamma^k H(\pi_{\mathcal{H}(s_k)}, k)\right\}\right\}. \quad (4.10) \end{aligned}$$

Proceeding as in (4.9), it is straightforward to show that $\gamma^k H(\pi_{\mathcal{H}(s_k)}, k) \leq \mathbb{E}\{\gamma^k H(\pi_{\mathcal{H}\mathcal{Y}}, k, k+1) + \gamma^{k+1} H(\pi_{\mathcal{H}(s_{k+1})}, k+1)\}$. Then, this inequality, the induction hypothesis, and (4.10) imply

$$\begin{aligned} H(\pi_{\mathcal{H}(s_0)}, 0) &\leq \mathbb{E}\left\{H(\pi_{\mathcal{H}\mathcal{Y}}, 0, k) + \gamma^k H(\pi_{\mathcal{H}(s_k)}, k)\right\} \\ &\cdots \\ &\leq \mathbb{E}\{H(\pi_{\mathcal{H}\mathcal{Y}}, 0, k) + \mathbb{E}\{\gamma^k H(\pi_{\mathcal{H}\mathcal{Y}}, k, k+1) \\ &\qquad + \gamma^{k+1} H(\pi_{\mathcal{H}(s_{k+1})}, k+1)\}\} \\ &= \mathbb{E}\left\{H(\pi_{\mathcal{H}\mathcal{Y}}, 0, k+1) + \gamma^{k+1} H(\pi_{\mathcal{H}(s_{k+1})}, k+1)\right\}. \end{aligned}$$

Therefore, the property holds for $k+1$. Since $\pi_{\mathcal{H}\mathcal{Y}}$ terminates by assumption, the result holds by induction when $\pi = \pi_{\mathcal{H}\mathcal{Y}}$.

The fact that the inequalities hold at equality when $\pi = \pi_{\mathcal{R}\mathcal{H}''}$ can be seen by noting that, for $k = 1, \ldots, K$,

$$\mathbb{E}\left\{H(\pi_{\mathcal{R}\mathcal{H}''}, 0, k) + \gamma^k H(\pi_{\mathcal{H}(s_k)}, k)\right\} = \mathbb{E}\left\{\sum_{i=0}^{k-1} \gamma^i H(\pi_{\mathcal{H}(s_i)}, i, i+1) + \gamma^k H(\pi_{\mathcal{H}(s_k)}, k)\right\} \tag{4.11}$$

$$= \mathbb{E}\left\{H(\pi_{\mathcal{H}(s_0)}, 0)\right\}. \tag{4.12}$$

Equation (4.11) holds by the definition of a pre-decision rollout policy and (4.12) holds by the sequential consistency of $\mathcal{H}(\cdot)$. $\square$

In addition to establishing the improvement property for our rollout policies, Proposition 4.1 implies $H(\pi_{\mathcal{R}\mathcal{H}''}, 0) = H(\pi_{\mathcal{H}(s_0)}, 0)$. From a computational perspective, this is significant since a pre-decision rollout policy built on a sequentially consistent heuristic need only execute $\mathcal{H}(\cdot)$ once from initial state s_0, instead of once at each decision epoch. For a given heuristic $\mathcal{H}(\cdot)$, Proposition 4.1 also implies that hybrid and post-decision rollout policies perform weakly better than pre-decision rollout policies from initial state s_0: $H(\pi_{\mathcal{RH}''}, 0) \leq H(\pi_{\mathcal{HY}}, 0)$ and $H(\pi_{\mathcal{RH}''}, 0) \leq H(\pi_{\mathcal{RH}'}, 0)$.

Furthermore, Proposition 4.2 states that when $\mathcal{H}(\cdot)$ is sequentially consistent, traditional and post-decision rollout policies are equivalent. Computationally, this is a notable result since post-decision rollout policies significantly reduce the number of times heuristic $\mathcal{H}(\cdot)$ must be executed to select an action.

Proposition 4.2. *If $\mathcal{H}(\cdot)$ is sequentially consistent, then traditional and post-decision rollout policies are equivalent.*

Proof. Consider a sample path generated by $\mathcal{H}(s_k^a)$: $(s_{k+1}, s_{k+2}, \ldots, s_K)$, where $s_K \in \mathcal{S}_K$. For the same action, consider a sample path generated by $\mathcal{H}(s_{k+1})$: $(s'_{k+1}, s'_{k+2}, \ldots, s'_K)$, where $s'_K \in \mathcal{S}_K$. By definition, $s_{k+1} = S^{M,W}(s_k^a, W_{k+1}) = S^M(s_k, a, W_{k+1})$. Thus, for a given action and realization of W_{k+1}, $s_{k+1} = s'_{k+1}$. Then, by the sequential consistency of $\mathcal{H}(\cdot)$, $s_i = s'_i$ for $i = k+2, k+3, \ldots, K$. Repeating this observation for all sample paths generated by action a, $\mathbb{E}\{H(\pi_{\mathcal{H}(s_k^a)}, k+1)|s_k, a\} = \mathbb{E}\{H(\pi_{\mathcal{H}(s_{k+1})}, k+1)|s_k, a\}$, implying that post-decision and traditional rollout policies yield the same evaluation for a given action. Since both policies return the action with the maximum evaluation, the policies are equivalent. $\square$

4.7 Restricted Optimal Heuristics

Although sequentially consistent heuristics facilitate the improvement property, we note along with Bertsekas et al. (1997) and Secomandi (2003) that the improvement property does not provide any guarantee on the overall performance of a rollout policy. In our experience, the performance of a rollout policy is driven primarily by the ability of $\mathcal{H}(\cdot)$ to accurately approximate future rewards. Thus, we do not believe that a sequentially consistent $\mathcal{H}(\cdot)$ is a necessary condition for the success of a rollout policy. This observation is supported by Secomandi (2003), who shows that the performance of two related heuristics (one sequentially consistent and the other not) is similar.

Motivated by this observation, we focus in this section on a special class of sequentially consistent heuristics that we refer to as *restricted optimal heuristics.* A restricted optimal heuristic, denoted $\mathcal{H}^{\star}(\cdot)$, is a heuristic that returns an optimal policy over a subset of policies $\tilde{\Pi} \subseteq \Pi$. By the principle of optimality, $\mathcal{H}^{\star}(\cdot)$ satisfies the definition of sequential consistency. The subset $\tilde{\Pi}$ is determined by the restrictions a heuristic places on the original dynamic program. For example, heuristic $\bar{\mathcal{H}}(\cdot)$ restricts attention to fixed route policies, which restrict the action space by requiring vehicles to follow predefined routes. Thus, for $\bar{\mathcal{H}}(\cdot)$, $\tilde{\Pi}$ is the set of all fixed route policies.

When used in rollout policies, restricted optimal heuristics yield two important consequences. First, because restricted optimal heuristics are sequentially consistent, the resulting rollout policy possesses the improvement property. Second, restricted optimal heuristics result in the equivalence of hybrid and post-decision rollout policies. This result follows directly from Proposition 4.3, which states that the use of a restricted optimal heuristic results in pre-decision and post-decision rollout policies being equivalent over the subset of feasible actions $\mathcal{A}'(\cdot)$. Because a hybrid rollout policy evaluates remaining actions $\mathcal{A}(\cdot) \setminus \mathcal{A}'(\cdot)$ via a post-decision rollout policy, hybrid and post-decision rollout policies are equivalent. In light of Proposition 4.2, this result implies that a hybrid rollout policy built on a restricted optimal heuristic is equivalent to a traditional rollout policy. Computationally, this is a significant result because hybrid rollout policies appreciably reduce the number of times $\mathcal{H}^\star(\cdot)$ must be executed to select an action.

Proposition 4.3. *Assume heuristic $\mathcal{H}^\star(\cdot)$ is a restricted optimal heuristic. Further, assume pre- and post-decision rollout policies each employ the same, state independent rule for resolving ties among actions with the same evaluation. Then, a pre-decision rollout policy is equivalent to a post-decision rollout policy over actions $\mathcal{A}'(\cdot)$.*

Proof. As in §4.5, assume $\mathcal{H}^\star(\cdot)$ can return any action in $\mathcal{A}'(\cdot)$. Assume the policies are not equivalent at the k^{th} decision epoch and let $a' = \delta^{\pi_{\mathcal{RH}'}}(s_k) \neq \delta^{\pi_{\mathcal{RH}''}}(s_k) = a''$. This assumption gives rise to three cases, all of which lead to a contradiction. In the first case, the pre-decision rollout policy is better than the post-decision rollout policy, meaning

$$H(\pi_{\mathcal{H}^\star(s_k)}, k) > R_k(s_k, a') + \gamma \mathbb{E}\left\{ H(\pi_{\mathcal{H}^\star(s_k^{a'})}, k+1) \middle| s_k, a' \right\}. \tag{4.13}$$

Because the post-decision rollout policy selected a', it must be that

$$R_k(s_k, a') + \gamma \mathbb{E}\left\{H(\pi_{\mathcal{H}^\star(s_k^{a'})}, k+1)\middle| s_k, a'\right\}$$
$$> R_k(s_k, a'') + \gamma \mathbb{E}\left\{H(\pi_{\mathcal{H}^\star(s_k^{a''})}, k+1)\middle| s_k, a''\right\}. \quad (4.14)$$

By definition, for a given realization of W_{k+1}, $s_{k+1} = S^{M,W}(s_k^{a''}, W_{k+1}) = S^M(s_k, a'', W_{k+1})$. Thus, by the restricted optimality of $\mathcal{H}^\star(\cdot)$, $H(\pi_{\mathcal{H}^\star(s_k^{a''})}, k+1) = H(\pi_{\mathcal{H}^\star(s_k)}, k+1)$. Then, since

$$H(\pi_{\mathcal{H}^\star(s_k)}, k) = R_k(s_k, a'') + \gamma \mathbb{E}\left\{H(\pi_{\mathcal{H}^\star(s_k^{a''})}, k+1)\middle| s_k, a''\right\}$$
$$> R_k(s_k, a') + \gamma \mathbb{E}\left\{H(\pi_{\mathcal{H}^\star(s_k^{a'})}, k+1)\middle| s_k, a'\right\} \quad (4.15)$$

by assumption, (4.14) cannot be true, thus implying that a' cannot be the action chosen by the post-decision rollout policy.

In the second case, the post-decision rollout policy is better than the pre-decision rollout policy, meaning

$$R_k(s_k, a') + \gamma \mathbb{E}\left\{H(\pi_{\mathcal{H}^\star(s_k^{a'})}, k+1)\middle| s_k, a'\right\} > H(\pi_{\mathcal{H}^\star(s_k)}, k). \quad (4.16)$$

To be clear about the action selected by the pre-decision rollout policy, we augment our notation. If the pre-decision rollout policy selects action a in state s_k, then denote the policy by $\pi_{\mathcal{H}^\star(s_k,a)}$. Because the pre-decision rollout policy returned a'', and because $\mathcal{H}^\star(s_k)$ returns the optimal action in $\mathcal{A}'(s_k)$ by assumption, it must be that

$$R_k(s_k, a'') + \gamma \mathbb{E}\left\{ H(\pi_{\mathcal{H}^\star(s_k,a'')}, k+1) \middle| s_k, a'' \right\} \\ > R_k(s_k, a') + \gamma \mathbb{E}\left\{ H(\pi_{\mathcal{H}(s_k,a')}, k+1) \middle| s_k, a' \right\}. \quad (4.17)$$

As before, $s_{k+1} = S^{M,W}(s_k^{a''}, W_{k+1}) = S^M(s_k, a'', W_{k+1})$ for a given realization of W_{k+1}, and by the restricted optimality of $\mathcal{H}^\star(\cdot)$, $H(\pi_{\mathcal{H}^\star(s_k^{a''})}, k+1) = H(\pi_{\mathcal{H}^\star(s_k)}, k+1)$. Then, since

$$R_k(s_k, a') + \gamma \mathbb{E}\left\{ H(\pi_{\mathcal{H}^\star(s_k^{a'})}, k+1) \middle| s_k, a' \right\} = R_k(s_k, a') \\ + \gamma \mathbb{E}\left\{ H(\pi_{\mathcal{H}^\star(s_k,a')}, k+1) \middle| s_k, a' \right\} > H(\pi_{\mathcal{H}^\star(s_k)}, k) \\ = R_k(s_k, a'') + \gamma \mathbb{E}\left\{ H(\pi_{\mathcal{H}^\star(s_k,a'')}, k+1) \middle| s_k, a'' \right\} \quad (4.18)$$

by assumption, (4.17) cannot be true, thus implying that a'' cannot be the action chosen by the pre-decision rollout policy.

In the third case, the value of the pre-decision rollout policy is equivalent to the value of the post-decision rollout policy, meaning

$$H(\pi_{\mathcal{H}^\star(s_k)}, k) = R_k(s_k, a') + \gamma \mathbb{E}\left\{ H(\pi_{\mathcal{H}^\star(s_k^{a'})}, k+1) \middle| s_k, a' \right\}. \quad (4.19)$$

Let $\mathcal{A}^\star(s_k) \subseteq \mathcal{A}'(s_k)$ be the set of restricted optimal actions. By the restricted optimality of $\mathcal{H}^\star(\cdot)$, $a', a'' \in \mathcal{A}^\star(s_k)$. By assumption, $\pi_{\mathcal{RH}'}$ and $\pi_{\mathcal{RH}''}$ employ the same rule for resolving ties among actions with the same evaluation. Thus, it must be that $a' = a''$, which is a contradiction. □

The improvement property and equivalence of the various rollout policies resulting from restricted optimal heuristics provide strong motivation for the use of mathematical programming methods in the solution of large-scale dynamic programs within a rollout framework. The downside of employing restricted optimal heuristics within a rollout framework is the potentially large computing time required to execute $\mathcal{H}^{\star}(\cdot)$, possibly limiting the ability of a rollout procedure to make dynamic, real-time decisions.

CHAPTER 5
A HYBRID ROLLOUT POLICY FOR THE VEHICLE ROUTING PROBLEM WITH STOCHASTIC DEMAND AND DURATION LIMITS

5.1 Introduction

Traditional rollout policies have been successfully applied to obtain dynamic policies for single-vehicle VRPs with stochastic demand (Secomandi, 2001; Novoa and Storer, 2008). Because the fleet size is limited to only one vehicle, the number of feasible actions at each decision epoch is relatively small. When considering multiple vehicles, however, the size of the action space can increase exponentially, thereby rendering straightforward extensions of single-vehicle methods to the multi-vehicle case computationally daunting. In this chapter, we demonstrate the potential of the rollout framework we propose in §4 by developing a hybrid rollout policy to obtain dynamic solutions for the multi-vehicle routing problem with stochastic demand and duration limits (VRPSDL), an important and difficult problem in supply chain management.

Similar to the literature on dynamic solution methodologies for single-vehicle VRPs with stochastic demand, a *fixed route* heuristic forms the basis of our rollout policy for the VRPSDL. Fixed route policies require vehicles to follow predefined routes and implement simple recourse strategies to accommodate route failures. Using our rollout policy, we solve large-scale instances with as many as 100 customers and 19 vehicles. By testing our procedure across a broad range of problem parameters, we empirically establish conditions under which the demand served by our rollout policy is significantly higher than the demand served by a method frequently implemented in practice. We also identify circumstances in which our rollout policy appears to offer little or no benefit compared to this benchmark. These observations can guide managerial decision making regarding when the use of our procedure is justifiable. Finally, we demonstrate that our methodology lends itself to real-time implementation, thereby providing a mechanism to make high-quality, dynamic routing decisions for large-scale operations.

This chapter is organized as follows. In §5.2, we discuss literature related to the VRPSDL. We discuss our hybrid rollout procedure in §5.3. We present computational results in §5.4. In §5.5, we summarize the chapter and suggest directions for future research.

5.2 VRPSDL-Related Literature

In this section, we review literature addressing dynamic policies for VRPs with stochastic demand. For a review of static, fixed route policies, we refer the reader to the recent review by Campbell and Thomas (2008). The MDP framework underlying much of the dynamic VRP literature is the single-vehicle model proposed by Dror et al. (1989). The objective of Dror et al.'s formulation is to obtain a policy that minimizes expected travel cost subject to a constraint on vehicle capacity. In addition to incorporating a route duration limit, we deviate from this model in two additional ways. First, our model allows for multiple vehicles, which leads to a larger state space and a significantly more complex action space (motivating our rollout framework). Second, as motivated in §1, our objective is to maximize expected demand served, rather than to minimize expected travel costs.

For problem instances of any practical size, obtaining optimal policies for Dror et al.'s single-vehicle MDP is computationally intractable due to the well known "curse of dimensionality" associated with dynamic programs. Thus, much of the literature focuses on heuristic solution methods. Secomandi (2000) investigates an approximate policy iteration procedure that estimates the cost-to-go via a parametric function. Such an approach, which is typical of more traditional approximate dynamic programming, refines parameter estimates via simulation. Secomandi concludes that a traditional rollout policy performs better than using a parametric function to approximate the cost-to-go. Within the traditional rollout framework, Secomandi (2000, 2001) establishes a single fixed route policy and repeatedly uses it to approximate the cost-to-go for a given action by computing the expected cost of cyclically following this fixed route (skipping customers whose demand has been fully served). Novoa and Storer (2008) show that using a higher quality initial fixed route leads to improved policies. They also demonstrate that computation times can be reduced, without a loss in solution quality, by using simulation to estimate the expected cost of following a fixed route.

As demonstrated by Fan et al. (2006), one way to address a multi-vehicle problem is to decompose it into single-vehicle problems by first clustering customers into groups (one for each vehicle) and then applying Secomandi (2001) to each group.

However, extending the method of Secomandi (2001) in this fashion has limitations. Using an initial set of fixed routes to approximate the cost-to-go can only be accomplished if the customers to be served by each vehicle are fixed at the start of the time horizon. Otherwise, it is not clear how to approximate the cost-to-go for actions that assign vehicles to customers outside of the initial fixed routes. In this paper, we overcome this limitation by updating fixed route policies at each decision epoch (instead of reusing an initial policy) that collectively approximate the cost-to-go for each feasible action.

Secomandi and Margot (2009) suggest partial-reoptimization strategies for a constrained version of Dror et al.'s single-vehicle model. The constraint requires each customer demand to be serviced in full before proceeding to another customer, thereby reducing the size of the state space. The state space is further reduced by heuristically selecting a promising set of states over which an optimal policy is obtained. Unfortunately, imposing similar constraints on a multi-vehicle problem does not circumvent the exponential increase in the action space, thereby negating a straightforward extension of Secomandi and Margot (2009) to the problem considered in this paper.

Proper and Tadepalli (2006) consider a combined inventory control and multi-vehicle routing problem with stochastic demand where the objective is to obtain a policy that minimizes a measure of travel cost subject to a constraint on vehicle capacity. Recognizing the computational challenge imposed by a large action space, the authors suggest a simple local search procedure to guide action selection. From a given state, instead of calculating the expected cost-to-go for each feasible action, the heuristic evaluates only a subset of the feasible actions. Using several four-vehicle, five-customer problem instances, the authors show their method to improve upon the performance of a greedy heuristic. Using a heuristic policy to select actions from a current state has the flavor of the pre-decision rollout policies we propose in §4.4. However, we consider much larger problems.

Hvattum et al. (2006, 2007) consider a multi-vehicle routing problem with stochastic demand where customer orders are placed over a given time horizon. The problem is modeled as a multi-stage stochastic programming problem with an objective of minimizing travel cost. Within this framework, actions are selected at pre-defined stages through a sample-based heuristic. In our MDP framework, we heuristically select actions whenever new information is learned. Additionally, Hvattum et al. (2006, 2007) require customer demand to be served completely by one vehicle on one visit. If this is not possible, then the customer is skipped. In contrast, we require customer demand to be served to the fullest extent given available vehicle capacity. Any remaining customer demand may be served during subsequent visits by the same vehicle, by another vehicle, or not at all.

5.3 A Hybrid Rollout Policy

In this section, we describe a hybrid rollout policy for the VRPSDL. As outlined in §4.5, a hybrid rollout policy requires the specification of a heuristic $\mathcal{H}(\cdot)$ and the partition of the feasible actions into two subsets, $\mathcal{A}'(s)$ and $\{\mathcal{A}(s) \setminus \mathcal{A}'(s)\}$. In

§5.3.1, we describe an estimation-based local search heuristic. The policies returned by this heuristic do not explicitly consider replenishment actions, thereby leading to a natural partition of the action space, which we describe in §5.3.2. Algorithm 5.1 summarizes the steps required to follow our hybrid rollout policy for a given VRPSDL instance.

Algorithm 5.1 Hybrid Rollout Policy for the VRPSDL

```
1: $k \leftarrow 0$
2: while $s_k \notin \mathcal{S}_K$ do
3:     From pre-decision state $s_k$, execute $\mathcal{H}(s_k)$ to obtain $\pi_{\mathcal{H}(s_k)}$
4:     For each $a \in \{\mathcal{A}(s_k) \setminus \mathcal{A}'(s_k)\}$, execute $\mathcal{H}(s_k^a)$ to obtain $\pi_{\mathcal{H}(s_k^a)}$
5:     Select action $\bar{a}$ according to a policy that satisfies (4.4)
6:     Make state transitions: $s_k \to s_k^{\bar{a}} \to s_{k+1}$.
7:     $k \leftarrow k + 1$
```

5.3.1 Fixed Route Heuristic

Similar to Secomandi (2001) and Novoa and Storer (2008), the heuristic $\mathcal{H}(\cdot)$ we employ is a *fixed route* heuristic, which returns a *fixed route* policy. Fixed route policies for the VRPSDL require vehicles to visit customers in the order specified by predefined routes, where each customer with pending or unknown demand appears exactly once on exactly one vehicle route. After serving demand at a given customer, a vehicle continues directly to the next customer on its fixed route. If continuing on to the next customer will result in a violation of the route duration limit, a vehicle is required to return to the depot. A return trip to the depot for the purpose of capacity replenishment is required in the event of a route failure and is not permitted otherwise. Although it is possible to consider fixed route policies that permit replenishment prior to route failures, our experience indicates that local search schemes designed around such policies are computationally prohibitive.

We denote a fixed route for vehicle $c \in \mathcal{M}$ by the sequence of locations $v^c = (v_1^c, \ldots, v_b^c)$, where $v_1^c \in \mathcal{N}$ and $v_k^c \in \mathcal{N} \setminus \{0\}$ for $k = 2, \ldots, b$. When in initial state s_0, $v_1^c = 0$ for all $c \in \mathcal{M}$. When $\mathcal{H}(s)$ is executed from some $s \in \mathcal{S}$, v_1^c is initialized by s and may not necessarily be the depot. We denote a collection of fixed routes by $v = (v^c)_{c \in \mathcal{M}}$ and require that each customer with unknown or unserved demand be assigned to exactly one route in v. For individual customers, we drop the superscript c except when doing so leads to ambiguity. In accordance with the literature on fixed route policies, we assume customer demands are independent. This assumption results in separability by route when calculating the value of a policy, thereby improving the computational tractability of searching for fixed route policies.

Our fixed route heuristic $\mathcal{H}(\cdot)$ is a first-improving local search procedure with a relocation neighborhood (Kindervater and Savelsbergh, 2003). The relocation neighborhood of a current solution v consists of all fixed routes that can be obtained by relocating a customer v_i^c after another customer $v_j^{c'}$ such that $i \neq 1$ (current vehicle locations are fixed). If $c = c'$, i.e., the customer is not assigned to a different vehicle, then we also require $j \neq i$ and $j \neq i - 1$, as these moves do not change the solution. Given a current set of fixed routes, we randomly generate a neighbor set of fixed routes from the relocation neighborhood. If the expected demand served by the neighboring fixed routes is greater than the expected demand served by the current fixed routes, then the current solution is updated with the neighboring solution and the process repeats. Otherwise, the search continues in the neighborhood of the current solution and the procedure terminates when a current solution is determined to serve at least as much expected demand as all solutions in its relocation neighborhood.

Denote by $v^\star(k-1)$ the fixed routes returned by the hybrid rollout policy at decision epoch $k-1$. At the next decision epoch, when executing $\mathcal{H}(s_k)$ or $\mathcal{H}(s_k^a)$ for some action $a \in \{\mathcal{A}(s_k) \setminus \mathcal{A}'(s_k)\}$, an initial set of fixed routes is obtained by updating $v^\star(k-1)$ to reflect pre- or post-decision state transitions. Specifically, the first location on each route is synchronized with vehicle destinations. Arrival times and remaining vehicle capacities at the these locations are also adjusted. Our experience indicates that initializing the local search in this fashion accelerates convergence to high-quality, locally-optimal solutions.

In §5.6, we develop an exact, polynomial-time procedure to calculate the expected demand served at a given customer. Applying this method to each customer in a collection of fixed routes and summing the results allows for the exact evaluation of a fixed route policy. Even though the complexity of this procedure can be bounded, calculating the expected demand served by a fixed route policy can be computationally prohibitive, particularly when many policies must be evaluated, as is the case with $\mathcal{H}(\cdot)$.

A computationally attractive alternative is to estimate the value of a policy via simulation. Recently, Birattari et al. (2008) and Balaprakash et al. (2010) have demonstrated the effectiveness of estimation-based local search on fixed route policies for the probabilistic traveling salesman problem. They show that estimating the difference between the objective values of two solutions offers significant computational advantages over exact evaluation methods. Furthermore, because their estimation-based procedures can search many more solutions in a given period of time than exact procedures, the authors are able to find high quality solutions in less computing time. We have observed similar results when estimating the value of fixed route policies in $\mathcal{H}(\cdot)$. Therefore, we only employ our exact evaluation to precisely compute the expected demand served of the solution returned by $\mathcal{H}(\cdot)$.

Denote by R_{v_k} the random amount of demand served at customer v_k. Given M randomly generated demand realizations from f, $\hat{x}^1, \dots, \hat{x}^M$, the expected demand served at customer v_k is estimated to be

$$\widehat{\mathbb{E}}\{R_{v_k}\} = \frac{1}{M}\sum_{j=1}^{M} R_{v_k}(\hat{x}^j), \tag{5.1}$$

where $R_{v_k}(\hat{x}^j)$ is the demand served at customer v_k given demand realization $\hat{x}^j$. It is straightforward to show that (5.1) is an unbiased estimator of the expected demand served at customer v_k. The total value of a fixed route policy, $\pi(v)$, is estimated to be the sum of the estimated demand served at each customer:

$$\widehat{V}^{\pi(v)} = \sum_{c=1}^{m}\sum_{k=1}^{|v^c|} \widehat{\mathbb{E}}\{R_{v_k^c}\}. \tag{5.2}$$

As we show below, R_{v_k} depends on vehicle capacities, arrival times, and demands at customers $v_1, \dots, v_k$. In the local search procedure of $\mathcal{H}(\cdot)$, this allows the values of fixed route policies within the neighborhood of a current solution to be computed efficiently. Let v be a current solution and $\bar{v}$ a solution in the relocation neighborhood of v obtained by relocating customer v_i^c after customer $v_j^{c'}$. To calculate $\widehat{V}^{\pi(\bar{v})}$, it is not necessary to estimate the expected demand served at each customer. If $c \neq c'$, i.e., they are different vehicles, then it is only necessary to calculate $R_{\bar{v}_k^c}$ for $k = i, \dots, |v^c| - 1$ and $R_{\bar{v}_k^{c'}}$ for $k = j+1, \dots, |v^{c'}| + 1$. If $c = c'$, i.e., they are the same vehicle, and $i < j$, then it is only necessary to calculate $R_{\bar{v}_k^c}$ for $k = i, \dots, |v^c|$. If $c = c'$ and $i > j$, then it is only necessary to calculate $R_{\bar{v}_k^c}$ for $k = j+1, \dots, |v^c|$. Calculation of estimated demand served at other customers in $\bar{v}$ remains the same as in v.

R_{v_k} depends on vehicle capacity upon arrival to customer v_k and arrival time at v_k. Denote these quantities as Q_{v_k} and A_{v_k}, respectively, and denote the random amount of demand at customer v_k by x_{v_k}. We separate the calculation of Q_{v_k} and A_{v_k} into three cases:

$$Q_{v_k} = \begin{cases} Q_{v_{k-1}} - x_{v_{k-1}}, & x_{v_{k-1}} < Q_{v_{k-1}} \\ Q, & \frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} = \left\lfloor \frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} \right\rfloor \\ \left\lceil \frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} \right\rceil Q - x_{v_{k-1}} + Q_{v_{k-1}}, & \frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} > \left\lfloor \frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} \right\rfloor \end{cases}, \tag{5.3}$$

and

$$A_{v_k} = \begin{cases} A_{v_{k-1}} + t(v_{k-1}, v_k), & x_{v_{k-1}} < Q_{v_{k-1}} \\ A_{v_{k-1}} + \left(\frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} + 1\right) t(v_{k-1}, 0) & \\ \qquad + \left(\frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q}\right) t(0, v_{k-1}) + t(0, v_k), & \frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} = \left\lfloor \frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} \right\rfloor, \\ A_{v_{k-1}} + t(v_{k-1}, v_k) + \left\lceil \frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} \right\rceil & \\ \qquad \times \left(t(v_{k-1}, 0) + t(0, v_{k-1})\right), & \frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} > \left\lfloor \frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} \right\rfloor \end{cases} \tag{5.4}$$

where $v_1 = l_c$, $Q_{v_1} = q_c$, and $A_{v_1} = t_c$ are given by the current state of vehicle c. In the first case, demand at customer v_{k-1} is less than vehicle capacity upon arrival to v_{k-1}, thus vehicle capacity is simply decremented by the amount of demand and the vehicle travels directly from v_{k-1} to v_k. The second and third cases account for situations where demand at customer v_{k-1} is greater than or equal to vehicle capacity upon arrival to v_{k-1}, thereby requiring return trips to the depot to replenish capacity. The required number of return trips is $\lfloor (x_{v_{k-1}} - Q_{v_{k-1}})/Q \rfloor$. In the second case, satisfying demand at v_{k-1} exactly depletes vehicle capacity, thus requiring the vehicle to replenish at the depot one additional time and travel directly to customer v_k with full capacity. In the third case, there is some capacity remaining after serving demand at customer v_{k-1}. After making the necessary return trips to the depot, the vehicle travels directly from v_{k-1} to v_k with the remaining capacity.

To compute R_{v_k}, we consider the route duration limit, noting that violations of the route duration limit result in zero demand served. Three cases are considered in the calculation:

$$R_{v_k} = \begin{cases} 0, & A_{v_k} > L - t(v_k, 0) \\ \left\lfloor \frac{L - t(v_k,0) - A_{v_k}}{t(v_k,0)+t(0,v_k)} \right\rfloor Q + Q_{v_k}, & \left\lfloor \frac{L - t(v_k,0) - A_{v_k}}{t(v_k,0)+t(0,v_k)} \right\rfloor \leq \left\lfloor \frac{x_{v_k} - Q_{v_k}}{Q} \right\rfloor \\ x_{v_k}, & x_{v_k} \leq Q_{v_k} \text{ and } A_{v_k} \leq L - t(v_k, 0), \\ & \text{or } \left\lfloor \frac{L - t(v_k,0) - A_{v_k}}{t(v_k,0)+t(0,v_k)} \right\rfloor > \left\lfloor \frac{x_{v_k} - Q_{v_k}}{Q} \right\rfloor \end{cases} . \tag{5.5}$$

In the first case, zero demand is served because the route duration limit is violated. In the second case, only a portion of demand is served because the vehicle does not have enough time to make the return trips to the depot necessary to serve demand in full. In the third case, demand is served in full either because the vehicle arrives prior to the route duration limit and with sufficient capacity, or because the vehicle has enough time to make any necessary replenishments.

5.3.2 Action Space Partition

When applied in a pre-decision state $s_k = ((l, t, q), (d, x))$, fixed route heuristic $\mathcal{H}(s_k)$ returns a set of fixed routes v that constitute a fixed route policy $\pi_{\mathcal{H}(s_k)}$. The policy recommends an action a as follows. As required by (2.15), vehicles en route are not diverted. Thus, for a vehicle $c \in \{\mathcal{M} \setminus \mathcal{M}'\}$, $a_c = v_1^c$, the current destination of vehicle c. In accordance with (2.17), vehicles in $\mathcal{M}'$ whose capacity will be depleted after serving demand at current locations must replenish at the depot: $a_c = 0$ for all $c \in \mathcal{M}'$ such that $q_c \leq d_{v_1^c}$. Further, to respect (2.21), vehicles in $\mathcal{M}'$ with positive capacity after serving demand at current locations must return to the depot if visiting the next customer on the fixed route will result in a violation of the route duration limit: $a_c = 0$ for all $c \in \mathcal{M}'$ such that $q_c > d_{v_1^c}$ and $t_c + t(v_1^c, v_2^c) + t(v_2^c, 0) > L$. For remaining vehicles $c \in \mathcal{M}'$, the action recommended by the fixed route policy is to travel directly to the next customer on the fixed route ($a_c = v_2^c$), unless $|v^c| = 1$, then vehicle c returns to the depot ($a_c = 0$) having served all customer demand on its fixed route.

Because a fixed route policy does not explicitly consider capacity replenishment actions (except to maintain feasibility with respect to vehicle capacity and the route duration limit), we consider such actions in a post-decision rollout policy. Let $\{\mathcal{A}(s_k) \setminus \mathcal{A}'(s_k)\} = \{a \in \mathcal{A}(s_k) : a_c = 0$ for some $c \in \mathcal{M}'$ such that $q_c > d_{v_1^c}$ and $t_c + t(v_1^c, v_2^c) + t(v_2^c, 0) \leq L\}$ be the set of actions that allow vehicles to replenish capacity prior to a route failure. An upper bound on the number of actions in this set is $2^{|\mathcal{M}'|} - 1$, which is only obtained if all vehicles in $\mathcal{M}'$ meet the specified criteria. As outlined in (4.4), for each action $a \in \{\mathcal{A}(s_k) \setminus \mathcal{A}'(s_k)\}$, we transition to post-decision state s_k^a and apply heuristic $\mathcal{H}(s_k^a)$. We then compare the best post-decision policy to the pre-decision policy and select the action corresponding to the policy with the largest expected demand served. This process of obtaining pre- and post-decision policies to solve (4.4) is summarized in Algorithm 5.1.

5.4 Computational Experience

In this section, we examine the effectiveness of our hybrid rollout policy for the VRPSDL and discuss the managerial implications of our method. Ideally, our hybrid rollout policy should be compared to an optimal policy. However, a primary motivation for the methodology we develop in this paper is the difficulty of obtaining optimal VRPSDL policies. To emphasize this, consider the work of Secomandi and Margot (2009), which addresses a single-vehicle VRP with stochastic demand. Provably optimal policies are obtained only for problems with 15 or fewer customers. Moreover, the authors are only able to obtain perfect-information bounds on the value of optimal policies for problem instances with 15 or fewer customers. Due to the additional complexity introduced by multiple vehicles, obtaining optimal VRPSDL policies or bounds on the value of optimal VRPSDL policies, is a nontrivial task and is a subject of future research.

Given these limitations, we choose to benchmark our results by comparing the demand collected by our hybrid rollout policy to the demand collected by a stand-alone fixed route policy for the VRPSDL. This comparison allows us to compare dynamic updating to the static routing often implemented in practice (Erera et al., 2010). Our methodology for obtaining benchmark fixed route policies is described in §5.4.1. In §5.4.2, we detail the generation of problem instances used to make the comparison. Implementation details are outlined in §5.4.3. A discussion of the results and their managerial implications is provided in §5.4.4.

5.4.1 Benchmark Fixed Route Policies

To ensure our benchmark fixed route policies are of high quality, we use a simulated annealing procedure that has matched many of the optimal fixed route policies obtained by Christiansen and Lysgaard (2007) for a VRP with stochastic demand. Simulated annealing is a local search algorithm in which non-improving moves are probabilistically accepted in an attempt to avoid becoming trapped in a low-quality, locally-optimal solution (Kirkpatrick et al., 1983; Johnson et al., 1989, 1991).

Given a current set of fixed routes, our simulated annealing procedure generates a neighboring set of fixed routes via one of the five neighborhood structures employed by Bent and Van Hentenryck (2004) in a simulated annealing algorithm for the VRP with time windows: two-exchange, Or-exchange, relocation, exchange, and crossover. A detailed explanation of these neighborhoods is given in Kindervater and Savelsbergh (2003). At each iteration of our simulated annealing implementation, we randomly generate a neighboring set of fixed routes from a randomly selected neighborhood of the current set of fixed routes. Denote the policy induced by the current set of fixed routes by $\pi(v)$ and the policy induced by a neighboring set of fixed routes by $\pi(v')$. Denote the expected demand served by these policies as $V^{\pi(v)}$ and $V^{\pi(v')}$, respectively. We accept the neighboring fixed route policy with probability $\exp(-(V^{\pi(v)} - V^{\pi(v')})^{+}/\tau)$, where $(y)^{+} = y$ if $y > 0$ and 0 otherwise and τ is the current temperature, a control parameter in simulated annealing (Kirkpatrick et al., 1983). In the manner discussed in §5.3.1, to reduce the computational burden of evaluating a fixed route policy, we estimate the expected demand served via simulation.

5.4.2 Problem Instances

To facilitate our experiments, we modify eight problems derived from the instances of Solomon (1987), ignoring the time windows, with the aim of considering a broad range of problem types. The problems include R101 (randomly dispersed customers) and C101 (clustered customers), each with 25, 50, 75, and 100 customers. We vary vehicle capacity, impose route duration limits, and vary customer demand variability to yield a total of 216 problem instances. We set problem parameters as follows. For each of the eight Solomon instances, we consider vehicle capacities of 25, 50, and 75 units, hereafter referred to as *small*, *medium*, and *large*. To determine fleet size and route duration limits, we first solve each of the eight Solomon instances as a classical VRP (i.e., time windows are ignored) with a vehicle capacity of 100. We employ the cyclic-order neighborhood search heuristic of §3 for this purpose. The fleet size m for each problem instance is set to the number of routes in the VRP solution. We then multiply the length of the longest route in each solution by 0.75, 1.25, and 1.75. The resulting route duration limits are hereafter referred to as *short*, *medium*, and *long*. Fleet size, route duration limits, and capacities are displayed in Table 5.1. To the best of our knowledge, only Hvattum et al. (2006, 2007) consider similarly sized dynamic and stochastic VRP instances. In their work, the number of orders revealed dynamically over a given time horizon is as large as 151. However, the fleet size never exceeds six vehicles, whereas we consider as many as 19 vehicles.

We assume customer demands are independent random variables, and we construct discrete, symmetric probability distributions for customer demand with *low*,

Table 5.1: Problem Parameters

Problem	**Vehicles**	**Duration Limits (short, medium, long)**	**Capacities (small, medium, large)**
R101 (25)	4	(85.575, 142.625, 199.675)	(25, 50, 75)
C101 (25)	5	(67.875, 113.125, 158.375)	(25, 50, 75)
R101 (50)	8	(89.475, 149.125, 208.775)	(25, 50, 75)
C101 (50)	9	(67.875, 113.125, 158.375)	(25, 50, 75)
R101 (75)	11	(103.05, 171.75, 240.45)	(25, 50, 75)
C101 (75)	14	(99.45, 165.75, 232.05)	(25, 50, 75)
R101 (100)	15	(94.875, 158.125, 221.375)	(25, 50, 75)
C101 (100)	19	(95.325, 158.875, 222.425)	(25, 50, 75)

moderate, and *high* variability. The expected demand for each customer is set to the deterministic demand given in the original Solomon instances; denote this quantity by $\bar{x}$. To construct a high variability probability distribution for a given customer, we assign uniform probabilities of 0.20 to $\{0, \bar{x}/2, \bar{x}, 3\bar{x}/2, 2\bar{x}\}$. To construct a moderate variability probability distribution for a given customer, we assign probabilities of $\{0.05, 0.15, 0.60, 0.15, 0.05\}$ to $\{0, \bar{x}/2, \bar{x}, 3\bar{x}/2, 2\bar{x}\}$. To construct a low variability probability distribution for a given customer, we assign probabilities of $\{0.05, 0.9, 0.05\}$ to $\{\bar{x}/2, \bar{x}, 3\bar{x}/2\}$.

For each of the 216 problem instances, we randomly generate 500 realizations according to the specified probability distributions for customer demand (a total 108,000 realizations). For each realization, we compare the demand collected by the benchmark fixed route policy to the demand collected by our hybrid rollout policy.

This is accomplished by constructing a confidence interval on the percent increase in demand served by the hybrid rollout policy over the benchmark fixed route policy. Let $\pi_{\mathcal{HY}}$ denote the hybrid rollout policy, $\pi(v)$ the benchmark fixed route policy, and $\hat{x}^i$ the i^{th} demand realization. For each demand realization, we calculate the percent increase in demand served by the hybrid rollout policy over the benchmark fixed route policy as

$$\hat{\Delta}^{\pi_{\mathcal{HY}},\pi(v)}(\hat{x}^i) = \frac{\widehat{V}^{\pi_{\mathcal{HY}}}(\hat{x}^i) - \widehat{V}^{\pi(v)}(\hat{x}^i)}{\widehat{V}^{\pi(v)}(\hat{x}^i)} \times 100.$$

We then calculate the average percent increase across all 500 realizations of the problem instance:

$$\bar{\Delta}^{\pi_{\mathcal{HY}},\pi(v)} = \frac{1}{500}\sum_{i=1}^{500} \hat{\Delta}^{\pi_{\mathcal{HY}},\pi(v)}(\hat{x}^i).$$

Letting S_Δ be the sample standard deviation of $(\hat{\Delta}^{\pi_{\mathcal{HY}},\pi(v)}(\hat{x}^i))_{i=1}^{500}$, we calculate a confidence interval as

$$\left[\bar{\Delta}^{\pi_{\mathcal{HY}},\pi(v)} - z_{0.975}S_\Delta, \bar{\Delta}^{\pi_{\mathcal{HY}},\pi(v)} + z_{0.975}S_\Delta\right].$$

5.4.3 Implementation Details

In the case of our hybrid rollout procedure, for each realization of each problem instance, the action taken in the first decision epoch is obtained by following the benchmark fixed route policy. Actions are selected in subsequent decision epochs via the hybrid rollout policy. This serves as a control in the first decision epoch and it also provides a method to initialize the first application of heuristic $\mathcal{H}(\cdot)$, which occurs in the second decision epoch.

The simulated annealing procedure used to obtain benchmark fixed route policies begins with an initial set of m fixed routes obtained by assigning customers to routes in a radial fashion with respect to the depot. We generate 7,000 neighbor solutions at each temperature, employ a temperature multiplier of 0.95, and begin with an initial temperature of 100. The procedure terminates after at least 100 temperature decrements and not updating the best-found solution for 75 successive temperature decrements. Our experience suggests that these parameters yield high-quality solutions. We run the procedure 10 times and select as the benchmark policy the fixed routes serving the greatest expected demand.

When estimating the expected demand served by fixed routes in the simulated annealing procedure, we use $M = 100$ demand samples. We decrease the number of demand samples to $M = 30$ in fixed route heuristic $\mathcal{H}(\cdot)$. We observe that 30 demand samples provides an acceptable tradeoff between solution quality and computing time. When we apply $\mathcal{H}(\cdot)$ in our hybrid rollout policy, computing time is an important consideration for real-time decision making. In either method, we use the same set of M demand samples throughout the procedure. As in Birattari et al. (2008), using common random numbers in this fashion reduces the variance of the difference between two solution values and also decreases computing time.

We implement our experiments, which required more than two CPU years, in C++ and execute them on a computing cluster provided by the University of Iowa

Information Technology Services. The cluster contains eight nodes with the CentOS 5.3 operating system. Each node consists of dual quad core AMD Opteron Processors 2350 at 2GHz and 16GB of ECC DDR2 667MHz SDRAM.

5.4.4 Results and Discussion

Table 5.2 displays confidence intervals for the percent increase in demand served by our hybrid rollout policies over the corresponding benchmark fixed route policies. Confidence intervals to the right of zero are in bold and indicate a statistically significant increase in demand served by our hybrid rollout policies over the benchmark fixed route policies. Confidence intervals containing zero are in normal typeface and indicate no statistically significant difference in demand collected by the two policies.

Across all 216 problem instances, the lower limit of the confidence intervals is never less than zero, thereby suggesting, with a strong likelihood, that following our hybrid rollout policy weakly improves upon a stand-alone fixed route policy. In some cases the improvement is dramatic, and in others, there is little to no benefit. We note that our initial attempts to outperform the benchmark fixed route policies were unsuccessful, often resulting in a statistically significant decrease in demand served. These attempts consisted primarily of greedy fixed route heuristics, similar to $\bar{\mathcal{H}}(\cdot)$ described in §4.5, implemented in a post-decision rollout policy. Due to the large action space of the VRPSDL, use of more sophisticated heuristics, such as the heuristic described in §5.3.1, was computationally intractable in a post-decision rollout policy. Only after developing the concept of pre-decision and hybrid rollout policies were we able to address the large action space and obtain desirable results.

We observe several notable trends in Table 5.2. First, for a given duration limit, as vehicle capacity decreases, the percent demand served over benchmark fixed route policies almost always increases. Intuition suggests the following explanation. Vehicles with smaller capacities experience more route failures/replenishments than vehicles with larger capacities. Fixed route policies handle route failures by requiring a single vehicle to make return trips to the depot until the customer's demand is satisfied. The hybrid rollout policies allows unserved demands resulting from a route failure to be met by multiple vehicles, thereby reducing the number of time-consuming return trips to the depot and allowing more demand to be served.

One way to support this claim is to compare the total distance traveled by the hybrid rollout policy to the total distance traveled by the benchmark fixed route policy. The following example is illustrative of the trend across other problem instances. For problem R101(50) with high variability in customer demand, a medium duration limit, and a large capacity, the average percent decrease in distance traveled by the hybrid rollout policy compared to the benchmark fixed route policy is –3.39 percent, indicating that the hybrid policy incurred a larger travel distance than the benchmark fixed route policy. In contrast, when the capacity is medium the average percent decrease is 4.41 percent and when the capacity is small the average percent decrease is 17.02 percent, thereby suggesting that the hybrid rollout policy is particularly efficient compared to the benchmark fixed route policy when capacity is medium

Table 5.2: Confidence Intervals for Percent Increase in Demand Served by Hybrid Rollout Policies Over Benchmark Fixed Route Policies

Duration	short			medium			long		
Capacity	small	medium	large	small	medium	large	small	medium	large
				Low Variability					
R101 (25)	**[13.26, 14.45]**	**[1.86, 2.57]**	[0.00, 0.03]	**[1.92, 2.53]**	**[0.77, 1.13]**	**[0.07, 0.12]**	**[2.11, 2.76]**	[0.00, 0.00]	[0.00, 0.00]
C101 (25)	**[6.07, 6.69]**	**[0.01, 0.11]**	**[0.06, 0.14]**	**[1.41, 1.85]**	**[3.00, 4.00]**	[0.00, 0.00]	**[0.70, 0.97]**	**[0.01, 0.07]**	[0.00, 0.00]
R101 (50)	**[3.62, 4.22]**	**[1.98, 2.47]**	**[0.20, 0.32]**	**[1.78, 2.24]**	**[0.56, 0.75]**	**[0.04, 0.14]**	**[1.73, 2.08]**	[0.00, 0.01]	[0.00, 0.00]
C101 (50)	**[1.24, 1.60]**	**[0.83, 1.18]**	**[1.11, 1.28]**	**[1.01, 1.33]**	**[3.71, 4.42]**	**[0.11, 0.22]**	**[1.13, 1.40]**	**[0.05, 0.15]**	[0.00, 0.01]
R101 (75)	**[2.34, 2.87]**	**[1.77, 2.18]**	**[0.66, 0.92]**	**[1.88, 2.22]**	**[0.28, 0.46]**	[0.00, 0.01]	**[1.83, 2.06]**	[0.00, 0.02]	[0.00, 0.03]
C101 (75)	**[2.33, 2.59]**	**[1.26, 1.57]**	**[0.26, 0.37]**	**[1.12, 1.33]**	**[0.07, 0.14]**	[0.00, 0.02]	**[1.17, 1.40]**	**[0.01, 0.04]**	[0.00, 0.00]
R101 (100)	**[2.56, 2.89]**	**[1.67, 2.08]**	**[1.04, 1.31]**	**[2.11, 2.43]**	**[0.10, 0.19]**	**[0.02, 0.07]**	**[2.36, 2.62]**	**[0.02, 0.05]**	**[0.01, 0.04]**
C101 (100)	**[2.89, 3.16]**	**[1.38, 1.66]**	**[0.29, 0.49]**	**[1.58, 1.85]**	**[1.10, 1.33]**	**[0.01, 0.05]**	**[2.24, 2.48]**	**[0.01, 0.05]**	[0.00, 0.01]
				Moderate Variability					
R101 (25)	**[7.61, 8.85]**	**[0.93, 1.62]**	**[0.09, 0.33]**	**[4.54, 5.44]**	**[4.57, 5.74]**	**[0.92, 1.42]**	**[5.78, 6.69]**	**[0.08, 0.25]**	[0.00, 0.00]
C101 (25)	**[12.67, 14.05]**	**[2.08, 2.77]**	**[0.80, 1.13]**	**[5.03, 5.96]**	**[5.93, 7.19]**	**[1.17, 2.01]**	**[4.14, 4.81]**	**[0.71, 1.06]**	**[0.02, 0.06]**
R101 (50)	**[5.47, 6.24]**	**[1.47, 1.85]**	**[1.17, 1.60]**	**[4.54, 5.10]**	**[2.24, 2.83]**	**[0.19, 0.40]**	**[4.69, 5.16]**	**[0.10, 0.26]**	[0.00, 0.03]
C101 (50)	**[5.10, 5.73]**	**[2.19, 2.78]**	**[0.69, 0.90]**	**[5.78, 6.49]**	**[5.71, 6.47]**	**[1.59, 2.13]**	**[4.05, 4.57]**	**[1.27, 1.66]**	**[0.04, 0.13]**
R101 (75)	**[5.90, 6.55]**	**[3.21, 3.81]**	**[1.82, 2.18]**	**[4.66, 5.08]**	**[1.48, 1.89]**	**[0.01, 0.05]**	**[4.40, 4.75]**	**[0.03, 0.07]**	**[0.01, 0.04]**
C101 (75)	**[5.81, 6.29]**	**[2.66, 3.04]**	**[2.29, 2.63]**	**[3.65, 3.99]**	**[1.64, 2.03]**	**[0.04, 0.09]**	**[4.21, 4.54]**	**[0.12, 0.24]**	**[0.01, 0.08]**
R101 (100)	**[6.96, 7.55]**	**[4.67, 5.08]**	**[1.44, 1.65]**	**[4.96, 5.34]**	**[1.99, 2.3]**	**[0.02, 0.07]**	**[4.73, 5.05]**	**[0.1, 0.18]**	**[0.02, 0.08]**
C101 (100)	**[5.64, 6.03]**	**[2.99, 3.4]**	**[2.06, 2.31]**	**[4.75, 5.08]**	**[4.17, 4.6]**	**[0.42, 0.63]**	**[4.88, 5.19]**	**[0.19, 0.3]**	**[0.02, 0.07]**
				High Variability					
R101 (25)	**[6.69, 8.56]**	**[3.57, 4.98]**	**[1.24, 2.07]**	**[7.50, 9.05]**	**[6.92, 8.36]**	**[1.90, 2.39]**	**[7.36, 8.47]**	**[0.65, 0.97]**	[0.00, 0.01]
C101 (25)	**[12.97, 14.47]**	**[5.88, 7.06]**	**[2.26, 2.98]**	**[8.68, 9.88]**	**[6.85, 7.97]**	**[3.17, 3.91]**	**[7.75, 8.81]**	**[2.78, 3.49]**	**[0.15, 0.22]**
R101 (50)	**[3.61, 4.49]**	**[3.21, 3.97]**	**[1.21, 1.81]**	**[6.06, 6.87]**	**[4.32, 4.95]**	**[0.68, 0.92]**	**[6.74, 7.41]**	**[0.59, 0.89]**	**[0.01, 0.03]**
C101 (50)	**[6.24, 7.17]**	**[4.18, 5.03]**	**[1.81, 2.30]**	**[8.03, 8.96]**	**[7.75, 8.56]**	**[2.98, 3.60]**	**[7.38, 8.10]**	**[3.92, 4.49]**	**[0.11, 0.17]**
R101 (75)	**[7.69, 8.46]**	**[4.48, 5.10]**	**[2.43, 2.88]**	**[5.70, 6.26]**	**[3.96, 4.44]**	**[0.15, 0.23]**	**[6.78, 7.29]**	**[0.16, 0.30]**	**[0.03, 0.09]**
C101 (75)	**[8.04, 8.66]**	**[4.47, 5.03]**	**[4.05, 4.61]**	**[7.33, 7.89]**	**[4.31, 4.78]**	**[0.13, 0.22]**	**[7.94, 8.44]**	**[0.51, 0.67]**	**[0.02, 0.04]**
R101 (100)	**[8.31, 9.00]**	**[4.86, 5.39]**	**[2.49, 2.91]**	**[7.27, 7.83]**	**[3.61, 3.99]**	**[0.25, 0.40]**	**[7.08, 7.53]**	**[0.24, 0.36]**	**[0.05, 0.11]**
C101 (100)	**[8.62, 9.22]**	**[4.32, 4.82]**	**[3.28, 3.69]**	**[7.94, 8.49]**	**[6.11, 6.67]**	**[1.92, 2.30]**	**[8.13, 8.61]**	**[1.19, 1.43]**	**[0.06, 0.13]**

Table 5.3: Average and Maximum Per Epoch CPU Seconds for Hybrid Rollout Policies

Duration	short			medium			long		
Capacity	small	medium	large	small	medium	large	small	medium	large
				Low Variability					
R101 (25)	(0.03, 0.51)	(0.02, 0.49)	(0.02, 0.38)	(0.02, 0.60)	(0.02, 0.53)	(0.02, 0.56)	(0.02, 0.80)	(0.01, 0.55)	(0.02, 0.57)
C101 (25)	(0.03, 0.68)	(0.02, 0.52)	(0.02, 0.47)	(0.02, 0.86)	(0.02, 0.67)	(0.01, 0.55)	(0.03, 1.00)	(0.02, 0.97)	(0.01, 0.45)
R101 (50)	(0.28, 1.00)	(0.26, 1.00)	(0.19, 1.00)	(0.25, 1.01)	(0.16, 1.01)	(0.16, 1.06)	(0.23, 1.12)	(0.16, 1.00)	(0.14, 1.00)
C101 (50)	(0.40, 1.11)	(0.27, 1.01)	(0.30, 1.19)	(0.30, 1.02)	(0.22, 1.06)	(0.20, 2.17)	(0.28, 1.12)	(0.32, 4.10)	(0.18, 1.01)
R101 (75)	(5.65, 18.40)	(4.58, 20.16)	(3.65, 23.44)	(4.89, 29.79)	(3.39, 27.34)	(3.09, 20.73)	(3.76, 22.21)	(2.43, 12.10)	(1.88, 9.86)
C101 (75)	(6.85, 28.96)	(5.34, 35.84)	(4.53, 26.21)	(5.35, 23.72)	(3.73, 20.93)	(3.30, 14.36)	(5.19, 35.41)	(3.35, 16.69)	(2.60, 14.28)
R101 (100)	(17.91, 66.72)	(10.48, 45.83)	(10.01, 54.42)	(13.88, 79.17)	(8.36, 67.81)	(7.00, 51.52)	(8.72, 61.18)	(6.84, 35.24)	(5.72, 26.05)
C101 (100)	(16.80, 60.50)	(13.86, 54.05)	(12.05, 59.17)	(14.88, 65.43)	(10.43, 63.57)	(9.53, 68.61)	(13.22, 74.07)	(9.00, 45.97)	(8.24, 41.57)
				Moderate Variability					
R101 (25)	(0.03, 0.55)	(0.02, 0.51)	(0.02, 0.39)	(0.02, 0.75)	(0.02, 0.61)	(0.02, 0.56)	(0.02, 0.90)	(0.03, 0.99)	(0.01, 0.47)
C101 (25)	(0.03, 0.82)	(0.02, 0.61)	(0.02, 0.63)	(0.02, 0.84)	(0.02, 0.73)	(0.02, 0.56)	(0.02, 0.98)	(0.07, 1.60)	(0.11, 2.00)
R101 (50)	(0.32, 1.10)	(0.27, 1.02)	(0.20, 1.00)	(0.29, 1.15)	(0.20, 1.10)	(0.18, 1.05)	(0.34, 2.87)	(0.18, 1.09)	(0.13, 1.00)
C101 (50)	(0.41, 1.23)	(0.30, 1.03)	(0.27, 1.21)	(0.34, 1.55)	(0.26, 1.47)	(0.25, 2.14)	(0.31, 1.47)	(0.46, 7.73)	(0.23, 1.40)
R101 (75)	(4.61, 20.25)	(3.79, 19.49)	(3.40, 19.90)	(4.22, 35.90)	(2.84, 53.19)	(2.34, 19.68)	(3.53, 27.04)	(1.94, 17.02)	(1.57, 8.75)
C101 (75)	(6.37, 29.43)	(5.17, 32.47)	(4.95, 47.11)	(5.40, 27.41)	(3.89, 29.41)	(3.27, 28.68)	(4.57, 33.24)	(3.21, 24.51)	(2.87, 18.53)
R101 (100)	(23.38, 105.11)	(19.94, 114.35)	(17.41, 93.14)	(20.01, 169.02)	(15.35, 124.35)	(13.30, 128.67)	(18.10, 154.78)	(10.60, 72.98)	(9.19, 46.16)
C101 (100)	(26.75, 106.48)	(22.55, 123.23)	(18.38, 106.89)	(23.86, 128.74)	(16.35, 120.05)	(15.62, 140.20)	(20.05, 146.49)	(14.54, 116.92)	(10.89, 57.74)
				High Variability					
R101 (25)	(0.03, 0.57)	(0.02, 0.48)	(0.02, 0.50)	(0.03, 0.79)	(0.02, 0.71)	(0.02, 0.53)	(0.03, 0.97)	(0.02, 0.73)	(0.02, 0.53)
C101 (25)	(0.03, 0.87)	(0.02, 0.64)	(0.02, 0.56)	(0.03, 0.97)	(0.02, 0.78)	(0.02, 0.72)	(0.03, 0.99)	(0.02, 0.94)	(0.11, 2.00)
R101 (50)	(0.30, 1.05)	(0.29, 1.13)	(0.22, 1.01)	(0.34, 2.19)	(0.24, 1.41)	(0.23, 2.47)	(0.30, 1.99)	(0.26, 3.50)	(0.14, 1.00)
C101 (50)	(0.44, 1.21)	(0.33, 1.08)	(0.30, 1.23)	(0.35, 1.73)	(0.30, 1.89)	(0.35, 5.33)	(0.34, 2.22)	(0.23, 1.94)	(0.50, 9.83)
R101 (75)	(6.56, 29.02)	(5.20, 25.06)	(4.81, 25.59)	(5.97, 38.39)	(4.32, 30.18)	(3.80, 29.34)	(5.30, 37.92)	(3.30, 34.25)	(2.30, 15.46)
C101 (75)	(7.76, 39.49)	(5.92, 47.04)	(5.46, 33.36)	(7.15, 45.50)	(4.98, 43.04)	(3.73, 20.67)	(5.99, 42.98)	(3.97, 47.04)	(3.22, 15.12)
R101 (100)	(24.27, 113.89)	(13.06, 83.63)	(12.40, 71.06)	(18.69, 151.67)	(10.89, 89.89)	(10.25, 123.85)	(14.21, 117.68)	(7.80, 64.49)	(6.18, 32.13)
C101 (100)	(19.74, 104.19)	(16.31, 116.57)	(13.49, 73.99)	(18.02, 119.02)	(12.69, 93.53)	(10.63, 80.62)	(14.35, 125.75)	(11.18, 186.01)	(8.97, 53.10)

or small.

Second, when variability in customer demand is high, hybrid rollout policies tend to perform better on clustered problem instances than on randomized problem instances. We suggest the following as a likely explanation for this trend. For a fixed route policy, route failures in a clustered problem instance tend to be more costly than route failures in a randomized problem instance. In a randomized problem instance, because customer locations are scattered about the service area, fixed route policies may be designed so that route failures are most likely to occur near the depot, thereby minimizing the time required to make return trips to the depot and allowing additional time to serve more demand. In a clustered problem instance, because customers are located in groups which may be some distance from the depot, each route failure requires a similar time to make a return trip to the depot, therefore making it difficult to leverage customer location when designing fixed routes. Because hybrid rollout policies allow customer demands to be served by multiple vehicles, some return trips to the depot can be avoided, thus allowing vehicles to use this time to serve more demand. This trend becomes less prominent as variability in customer demand decreases to moderate and low. When customer demands are known with higher accuracy, the likelihood of a well-designed fixed route collecting more demand is higher, thereby decreasing the potential to leverage customer location when designing fixed routes for randomized problem instances.

As before, we can support this claim by comparing the average percent decrease in distance traveled by hybrid rollout policies compared to benchmark fixed route policies for randomized and clustered instances with high demand variability. For problem C101(50) with high variability in customer demand and a medium duration limit, the percent decreases are 10.11, 14.51, and 15.75 percent for large, medium, and small capacities, respectively. Comparing these figures to those given above for R101(50), we see that when capacity is large or medium, the hybrid rollout policy decreases travel distance more for the clustered problem than for the randomized problem. The figures are similar when capacity is small. This example is illustrative of the trend we observe across other problem instances with high demand variability.

Third, the benefit of our hybrid rollout policies over the benchmark fixed route policies generally decreases as variability in customer demand decreases. As before, it appears that when customer demands are known with higher accuracy, the likelihood of well-designed fixed routes yielding high performance is high.

For companies that employ fixed routes, these observations suggest conditions under which operations may be improved by employing our hybrid rollout policy. The largest benefit is most likely to be realized when vehicle capacities are small relative to demand, customers are clustered, and demand variability is high. As vehicle capacities increase and customer demand variability decreases, the benefits diminish. Because fixed routes offer certain managerial advantages over a dynamic routing scheme (Campbell and Thomas, 2008), these considerations are important.

Although solution quality is our primary consideration, the potential for real-time decision making is also important. Table 5.3 displays per epoch computing times. The first number of each pair is the average number of CPU seconds required to select an action across all decision epochs in all 500 realizations. The second number of each pair is the average of the maximum number of CPU seconds required to select an action across all 500 realizations. For the problems we consider, two CPU minutes is the largest average maximum number of CPU seconds required to select an action, thereby suggesting that our method is implementable in real time for the problem instances we consider.

Speaking more generally, per epoch computing times are influenced by three things. First, the number of actions in $\{\mathcal{A}(s) \setminus \mathcal{A}'(s)\}$ determines the number of times heuristic $\mathcal{H}(\cdot)$ is applied. While these applications of $\mathcal{H}(\cdot)$ may be processed in parallel, the CPU seconds reported in Table 5.3 correspond to a serial implementation. Second, the number of demand samples, M, used to estimate expected demand served by a set of fixed routes can significantly influence the computation required to execute $\mathcal{H}(\cdot)$. For the problems we consider, $M = 30$ yields an acceptable trade-off between computation time and solution quality, but different problem instances may necessitate adjustments to this parameter. Third, as demonstrated in Table 5.3, problem size can increase per epoch computing times considerably.

5.5 Conclusion and Future Research

We demonstrate the potential of the rollout framework we propose in §4 by developing a hybrid rollout policy to obtain dynamic solutions for the vehicle routing problem with stochastic demand and duration limits, an important and difficult problem in supply chain management. We develop a fixed route heuristic that forms the basis of our rollout policy. Using our rollout policy, we solve large-scale problem instances and demonstrate that our method can significantly improve upon a method frequently implemented in practice. We also identify circumstances in which our rollout policy appears to offer little or no benefit compared to this benchmark. These observations can guide managerial decision making regarding when the use of our procedure is justifiable. We also demonstrate that our methodology lends itself to real-time implementation, thereby providing a mechanism to make high-quality, dynamic routing decisions for large-scale operations.

A potential drawback of using fixed route policies to approximate future demand served is that the approximation does not account for possible interactions among vehicle routes. This is because fixed route policies restrict vehicles to disjoint sets of customers. Accounting for these interactions may lead to improved policies. A method that may avoid this shortcoming is to approximate future demand served via a parametric function. Such an approach, which is typical of more traditional approximate dynamic programming, often refines parameter estimates via simulation. Preliminary work in this area suggests that seeding initial parameter values based on rollout policies is advantageous (see §7). Another possibility is to use more sophisticated fixed route policies, such as those suggested by Ak and Erera (2007), which coordinate vehicle routes in pairs. However, incorporating these more complex recourse actions may lead to increased computing times, thereby making it difficult to implement such methods in real time.

5.6 Polynomial-Time Procedure

In this appendix, we derive a polynomial-time procedure to calculate the expected demand served by a fixed route policy. Let $\mathcal{N}' = \{l_i^a : i \in \mathcal{M}', l_i^a \neq 0, x_{l_i}^a = ?\}$ be the set of customers with unknown demand at which a vehicle has arrived and let $\mathcal{N}'' = \{i \in \mathcal{N} : l_i^a \neq 0, x_i^a \neq ?\}$ be the set of customers whose demand has been observed. We denote by $x_{\mathcal{I}}$ the vector of customer demands indexed by the set $\mathcal{I}$. Then, $f_{x_{\mathcal{N}'}|x_{\mathcal{N}''}}(\cdot)$ is the joint density function of the demand at customers in $\mathcal{N}'$ conditional on the observed demand at customers in $\mathcal{N}''$.

By conditioning on vehicle capacity, arrival time, and customer demand, expected demand served at customer v_k may be calculated as

$$\mathbb{E}\{R_{v_k}\} = \sum_q \sum_t \mathbb{E}\left\{R_{v_k} | Q_{v_k} = q, A_{v_k} = t\right\} \times \mathbb{P}\left\{Q_{v_k} = q, A_{v_k} = t\right\} \tag{5.6a}$$

$$= \sum_q \sum_t \sum_x \mathbb{E}\left\{R_{v_k} | Q_{v_k} = q, A_{v_k} = t, x_{v_k} = x\right\} \tag{5.6b}$$

$$\times \mathbb{P}\left\{x_{v_k} = x | Q_{v_k} = q, A_{v_k} = t\right\} \times \mathbb{P}\left\{Q_{v_k} = q, A_{v_k} = t\right\} \tag{5.6c}$$

$$= \sum_q \sum_t \sum_x \mathbb{E}\left\{R_{v_k} | Q_{v_k} = q, A_{v_k} = t, x_{v_k} = x\right\} \times f_{x_{v_k}|x_{\mathcal{N}''}}(x|x_{\mathcal{N}''}) \tag{5.6d}$$

$$\times \mathbb{P}\left\{Q_{v_k} = q, A_{v_k} = t\right\}, \tag{5.6e}$$

where we assume a discrete joint probability distribution for computational tractability. The conditional expectation in (5.6d) may be obtained directly from (5.5). The second term in (5.6d) follows from the independence of demand conditional on q and t and is given as the density f. The joint probability term in (5.6e) may be calculated as follows:

$$\mathbb{P}\left\{Q_{v_k} = q, A_{v_k} = t\right\} = \sum_{q'} \sum_{t'} \mathbb{P}\left\{Q_{v_k} = q, A_{v_k} = t | Q_{v_{k-1}} = q', A_{v_{k-1}} = t'\right\} \tag{5.7a}$$

$$\times \mathbb{P}\left\{Q_{v_{k-1}} = q', A_{v_{k-1}} = t'\right\} \tag{5.7b}$$

$$= \sum_{q'} \sum_{t'} \sum_{x'} \mathbb{P}\{Q_{v_k} = q, A_{v_k} = t | Q_{v_{k-1}} = q' \tag{5.7c}$$

$$A_{v_{k-1}} = t', x_{v_{k-1}} = x'\} \times f_{x_{v_{k-1}}|x_{\mathcal{N}''}}(x'|x_{\mathcal{N}''}) \tag{5.7d}$$

$$\times \mathbb{P}\left\{Q_{v_{k-1}} = q', A_{v_{k-1}} = t'\right\}, \tag{5.7e}$$

where $\mathbb{P}\left\{Q_{v_k} = q, A_{v_k} = t | Q_{v_{k-1}} = q', A_{v_{k-1}} = t', x_{v_{k-1}} = x'\right\} = 1$ if (5.3) and (5.4) are satisfied and 0 otherwise.

Using this recursive relationship, Algorithm 5.2 describes a procedure to calculate $\mathbb{E}\{R_{v_k^c}\}$, the expected demand served at customer v_k^c. We define $g(v_k, q, t) = \mathbb{P}\left\{Q_{v_k} = q, A_{v_k} = t\right\}$ and append Q_{v_k}, A_{v_k}, and R_{v_k} with $(\cdot)$ to denote their dependence on other terms. The support of x_{v_k} is denoted $\text{supp}(x_{v_k})$. Assuming integer demand values and travel times, the worst case complexity of Algorithm 5.2 is $O(Q \times L \times \max_{j=1,\ldots,k}\{\text{supp}(x_{v_k})\})$.

Algorithm 5.2 Calculation of Expected Demand Served at Customer v_k for an A Priori Policy

```
 1: $g(v_1, q_c, t_c) \leftarrow 1$
 2: $g(v_1, q, t) \leftarrow 0$ for all $(q, t) \neq (q_c, t_c)$
 3: $g(v_j, q, t) \leftarrow 0$ for $j = 2, 3, \ldots, k$ and for all $(q, t)$
 4: $\mathbb{E}\{R_{v_k}\} \leftarrow 0$
 5: for $j = 2, 3, \ldots, k$ do
 6:     for $g(v_{j-1}, q', t') \neq 0$ do
 7:         for $x' \in \text{supp}(x_{v_{j-1}})$ do
 8:             $g(v_j, Q_{v_j}(x', q'), A_{v_j}(x', q', t')) \leftarrow$
                    $g(v_{j-1}, Q_{v_{j-1}}(x', q'), A_{v_{j-1}}(x', q', t')) \times f_{x_{v_{j-1}}|x_{\mathcal{N}''}}(x'|x_{\mathcal{N}''})$
 9: for $g(v_k, q', t') \neq 0$ do
10:     for $x' \in \text{supp}(x_{v_k})$ do
11:         $\mathbb{E}\{R_{v_k}\} \leftarrow \mathbb{E}\{R_{v_k}\} + R_{v_k}(x', q', t') \times g(v_k, q', t') \times f_{x_{v_k}|x_{\mathcal{N}''}}(x'|x_{\mathcal{N}''})$
```

CHAPTER 6
A RESTOCKING-BASED PRE-DECISION ROLLOUT POLICY FOR THE VEHICLE ROUTING PROBLEM WITH STOCHASTIC DEMAND AND DURATION LIMITS

6.1 Introduction

In this chapter, we develop a pre-decision rollout policy to obtain dynamic solutions for the multi-vehicle routing problem with stochastic demand and duration limits (VRPSDL). The fundamental difference between the methods of this chapter and the hybrid rollout policy of §5 is how capacity replenishment, or restocking, actions are evaluated. In the hybrid rollout policy of §5, restocking actions are enumerated and evaluated in the post-decision state via a fixed route heuristic. In this chapter, we expand the action space of the policy derived from a fixed route to explicitly consider restocking actions prior to route failures. In our fixed route heuristic, we still require vehicles to visit customers in the order specified by a fixed route, but we now consider preemptive returns to the depot as a way to improve the expected demand served by a fixed route. As a result, we are able to consider the full set of feasible actions, including restocking actions, from a pre-decision state via the fixed route heuristic. Computationally, this is important because the fixed route heuristic needs be executed only once at each decision epoch, as opposed to at least twice in the hybrid rollout policy of §5. Computational results for our pre-decision rollout policy reflect this by showing a decrease in per epoch computing times compared to the hybrid rollout policy of §5. In addition, results indicate that the demand served by our pre-decision rollout policy is comparable to the demand served by the hybrid rollout policy of §5. Thus, a contribution of this chapter is a comparable rollout policy to that of §5, but with shorter per epoch computing times.

Expanding the action space of a fixed route policy to explicitly consider restocking actions poses computational challenges. Unlike the more restricted fixed route policy we describe in §5.3.1, we are unable to evaluate a restocking fixed route policy in polynomial time. A second contribution of this chapter is a computationally feasible method to estimate the value of an optimal restocking policy along a given fixed route.

This chapter is organized as follows. In §6.2, we provide an example that illustrates the difference between restocking fixed route policies and the fixed route policies of §5.3.1. In §6.3, we review literature related to restocking policies for fixed routes. In §6.4, we describe our restocking-based pre-decision rollout policy. In §6.5, we develop a computationally tractable method to estimate the value of a restocking fixed route policy. We summarize our computational experience in §6.6 and provide concluding remarks in §6.7. Unless otherwise noted, we adopt the notation of §5 throughout this chapter.

6.2 Illustrative Example

In this section, we present an example that illustrates the difference between restocking fixed route policies and the fixed route policies of §5.3.1, which refer to as *a priori* fixed route policies. As in §5.3.1, the fixed route policy we consider for a vehicle $c \in \mathcal{M}$ is characterized by a fixed route $v^c = (v_1^c, v_2^c, \ldots, v_b^c)$. After fully serving demand at a customer v_i^c, instead of requiring the vehicle to travel directly to customer v_{i+1}^c, the vehicle may first replenish capacity at the depot.

The first five columns of Table 6.1 display data for a single fixed route $v = (23, 22, 13, 15, 2)$ for problem R101(25) with vehicle capacity $Q = 50$ and route duration limit $L = 171.681$. The column labeled "x_{v_k}" displays the deterministic demand at customer v_k. The horizontal and vertical coordinates of each customer are given in the columns labeled "x-coord" and "y-coord," respectively. The depot is located at $(35, 35)$ and one time unit is equal to one unit of distance, as measured by the Euclidian metric. The remainder of Table 6.1 compares an a priori policy for fixed route v with a restocking policy for fixed route v. The a priori policy requires the vehicle to visit customers in the order given on the fixed route, only returning to the depot in the event of a route failure or if proceeding to the next customer will result in a violation of the route duration limit.

The restocking policy we consider requires the vehicle to replenish capacity after fully serving demand at customer $v_1 = 23$ and before beginning service at customer $v_2 = 22$. Beginning at customer $v_2 = 22$, the restocking policy requires the vehicle to travel directly to each subsequent customer on the fixed route. As in the a priori policy, the vehicle must return to the depot in the event of a route failure or if servicing additional demand will result in a violation of the route duration limit. We note that this represents only one of 16 possible restocking strategies for fixed route v. In §6.5, we develop a procedure to obtain the optimal restocking policy for

Table 6.1: Example Data and Results for A Priori and Restocking Policies

Data					**A Priori Policy**			**Restocking Policy**		
k	v_k	x_{v_k}	x-coord	y-coord	Q_{v_k}	A_{v_k}	R_k	Q_{v_k}	A_{v_k}	R_k
1	23	39	55	5	50	36.0555	39	50	36.0555	39
2	22	18	45	10	11	47.2359	18	50	99.0368	18
3	13	23	30	25	43	122.301	23	32	120.25	23
4	15	8	30	5	20	142.301	0	9	140.25	8
5	2	7	35	17	12	155.301	0	1	153.25	1

a given fixed route. For each policy, Table 6.1 displays the capacity upon arrival to each customer (denoted Q_{v_k}), the arrival time at each customer (denoted A_{v_k}), and the demand served at each customer (denoted R_k).

The a priori policy begins service at customer $v_1 = 23$ at time $A_{23} = 36.0555$ (the time required to travel from the depot to customer 23) and with initial vehicle capacity $Q_{23} = 50$. After fully serving demand at customer $v_1 = 23$, vehicle capacity is positive, thus the a priori policy requires the vehicle to travel directly to customer $v_2 = 22$. Because vehicle capacity upon arrival $Q_{22} = 11$ is insufficient to fully serve demand $x_{22} = 18$, a route failure occurs. As required by the a priori policy, the vehicle serves 11 units of demand and then makes a return trip to the depot to replenish capacity. After serving the remaining seven units of demand, the vehicle travels directly to customer $v_3 = 13$ and fully serves demand $x_{13} = 23$. At this point, the a priori policy requires the vehicle to conclude its route and return to the depot because visiting customer $v_4 = 15$ results in a violation of the route duration limit. Thus, zero demand is served at customers $v_4 = 15$ and $v_5 = 2$ and the total demand

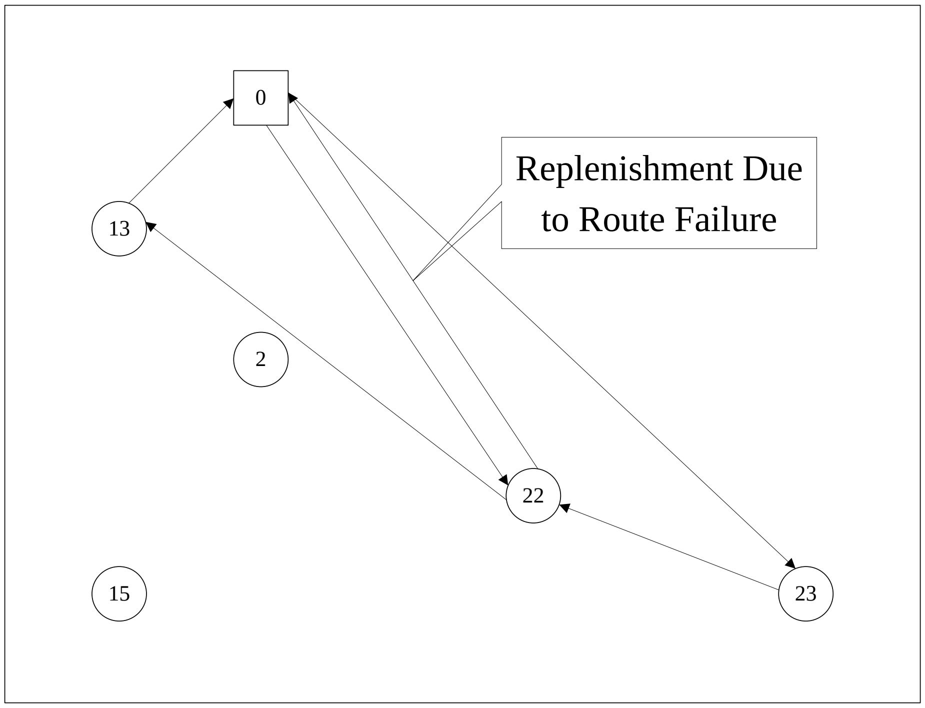

Figure 6.1: Example A Priori Route

served by the a priori fixed route policy is 80 units. The route traversed by the vehicle under the a priori policy is displayed in Figure 6.1.

The restocking policy also begins service at customer $v_1 = 23$ at time $A_{23} = 36.0555$ and with initial vehicle capacity $Q_{23} = 50$. Although vehicle capacity is positive after fully serving demand at customer $v_1 = 23$, the restocking policy we employ requires the vehicle to replenish capacity at the depot before visiting customer $v_2 = 22$. This proactive capacity replenishment, not permitted in the a priori policy, enables the vehicle to directly visit each subsequent customer on the fixed route without violating the route duration limit. As a result, demand is fully served at all customers except customer $v_5 = 2$, where only one of $x_2 = 7$ demand units is served.

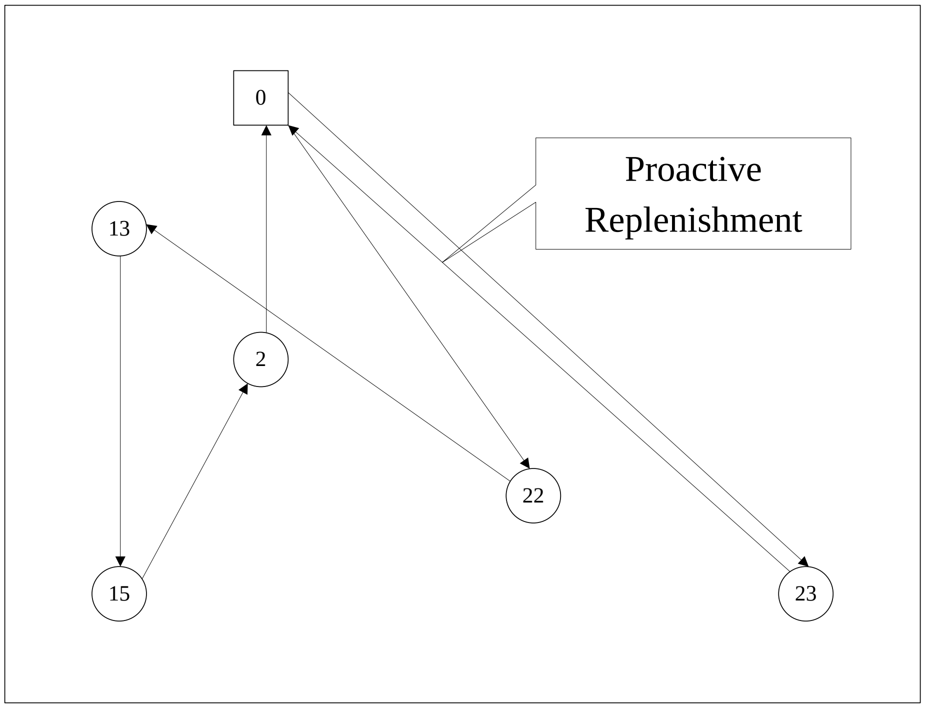

Figure 6.2: Example Restocking Route

The total demand served by the restocking policy is 89 units, nine units higher than the demand served by the a priori policy. The route traversed by the vehicle under the restocking policy is displayed in Figure 6.2.

In this example, the restocking policy is the optimal restocking policy for fixed route v (we discuss how to obtain the optimal restocking fixed route policy in a subsequent section). In general, for a given fixed route, the optimal restocking policy always serves at least as much expected demand as the a priori policy. This is because optimal restocking fixed route policies are a relaxation of a priori fixed route policies.

6.3 Restocking-Related Literature

In this section, we review literature related to restocking policies for a given fixed route. The literature we review focuses on obtaining restocking policies with an objective of minimizing some measure of travel cost. Although the methods we develop focus on maximizing demand served, there are methodological similarities between our work and the literature.

Much of the literature on restocking policies is founded on the work of Bertsimas et al. (1995), which considers restocking policies for a single-vehicle routing problem with stochastic demand. For a given fixed route, Bertsimas et al. (1995) formulate a dynamic program to determine the expected length of the route that results from following the optimal restocking policy. Because Bertsimas et al. (1995) assume that customer demands are discrete, the dynamic program can be solved in polynomial time. We also formulate a dynamic program to determine the demand served that results from following the optimal restocking policy for a given fixed route. However, because we model arrival times to customers as continuous (rather than discrete), we are unable to obtain a similar complexity result. In §6.5, we derive structural results and a forward dynamic programming procedure that overcome this difficulty. Our results apply to both discrete- and continuous-time models.

Yang et al. (2000) establish a key structural property for the dynamic program proposed by Bertsimas et al. (1995): an optimal policy is a threshold policy on available vehicle capacity. Yang et al. (2000) also show that routing a single vehicle along a single fixed route is equivalent to using multiple vehicles unless additional constraints are imposed, such as the route duration constraints we consider. Although the structural result reduces the computation required to evaluate a fixed route as a restocking policy, Yang et al. (2000) find that searching for an optimal fixed route is still computationally prohibitive. To further reduce computation, Yang et al. (2000) develop a method to approximate the change in the expected length of a fixed route when it is modified by a local search procedure. Bianchi et al. (2006) embed this approximation strategy in metaheuristic procedures and demonstrate that the computational results of Yang et al. (2000) can be improved by considering more complex search methods.

Secomandi (2003) develops a rollout procedure to search for restocking policies in a single-vehicle routing problem with stochastic demand. Given an initial fixed route, the method is guaranteed to return a restocking fixed route policy at least as good as the initial policy.

Tsirimpas et al. (2008) and Tatarakis and Minis (2009) consider three variations of the basic model proposed by Bertsimas et al. (1995) to evaluate the expected length of a given fixed route: the case of multi-product deliveries when each product is stored in its own compartment in the vehicle, the case of multi-product deliveries when all products are stored together in the vehicle's single compartment, and the case in which the vehicle picks up from and delivers a single product to each customer. The authors identify structural properties that aid in solving the dynamic programs.

6.4 A Pre-Decision Rollout Policy

Our pre-decision rollout policy employs a local search based fixed route heuristic identical to that of §5.3.1, except that fixed routes are evaluated as restocking policies, not as a priori policies. To differentiate this heuristic from that of §5.3.1, we refer to it as $\tilde{\mathcal{H}}(\cdot)$ and to the resulting pre-decision rollout policy as $\pi_{\mathcal{R}\tilde{\mathcal{H}}''}$.

Denote a restocking fixed route policy for vehicle $c \in \mathcal{M}$ by $\tilde{\pi}(v^c)$. Similar to the discussion in §5.3.1, M demand samples are employed to estimate the value of $\tilde{\pi}(v^c)$. Let $V^{\tilde{\pi}(v^c)}(\hat{x}^j)$ be the value of policy $\tilde{\pi}(v^c)$ when customer demands are given by the j^{th} demand sample $\hat{x}^j$ (we discuss calculation of $V^{\tilde{\pi}(v^c)}(\hat{x}^j)$ in §6.5). Then, we estimate the expected value of $\tilde{\pi}(v^c)$ as

$$\widehat{V}^{\tilde{\pi}(v^c)} = \frac{1}{M}\sum_{j=1}^{M} V^{\tilde{\pi}(v^c)}(\hat{x}^j). \tag{6.1}$$

Denoting a collection of fixed routes by $v = (v^c)_{c\in\mathcal{M}}$ and the corresponding collection of restocking policies as $\tilde{\pi}(v)$, we estimate the value of policy $\tilde{\pi}(v)$ as

$$\widehat{V}^{\tilde{\pi}(v)} = \sum_{c\in\mathcal{M}} \widehat{V}^{\tilde{\pi}(v^c)}. \tag{6.2}$$

Algorithm 6.1 summarizes the steps required to follow a pre-decision rollout policy $\pi_{\mathcal{R}\tilde{\mathcal{H}}''}$ for a given VRPSDL instance.

6.5 Evaluating Restocking Policies

In this section, we develop a method to calculate $V^{\tilde{\pi}(v^c)}(\hat{x}^j)$, the value of the restocking policy for fixed route $v^c = (v_1^c, v_2^c, \ldots, v_b^c)$ when observing demand

Algorithm 6.1 Pre-Decision Rollout Policy for the VRPSDL

```
1: $k \leftarrow 0$
2: while $s_k \notin \mathcal{S}_K$ do
3:     From pre-decision state $s_k$, execute $\tilde{\mathcal{H}}(s_k)$ to obtain $\tilde{\pi}_{\tilde{\mathcal{H}}(s_k)}$
4:     $\bar{a} \leftarrow \delta^{\tilde{\pi}_{\tilde{\mathcal{H}}(s_k)}}(s_k)$
5:     Make state transitions: $s_k \rightarrow s_k^{\bar{a}} \rightarrow s_{k+1}$.
6:     $k \leftarrow k+1$
```

$\hat{x}^j$. For notational brevity, we refer to fixed route v^c as v and to demand sample $\hat{x}^j$ as x, except when doing so leads to ambiguity. To obtain a restocking policy, we must determine, after fully serving demand at each customer on fixed route v, whether to proceed directly to the next customer or to first replenish capacity at the depot. Given fixed route v and customer demands x, an optimal restocking policy can be obtained by solving a deterministic dynamic program, which we formulate in §6.5.1. In §6.5.2, we derive structural properties that, when coupled with our forward solution approach of §6.5.3, significantly reduce the computational burden of solving the dynamic program. Our empirical experiences reveals that the structural results and forward solution approach significantly decrease computation time, particularly for fixed routes with more than 25 customers.

6.5.1 Dynamic Programming Formulation

Let Q_{v_k} be the vehicle capacity upon arrival to customer v_k, A_{v_k} the time of arrival at customer v_k, and x_{v_k} the demand at customer v_k. As in §2, $t(i,j)$ denotes the time required to travel from location i to location j. The state of the system is captured by $s_k = (Q_{v_k}, A_{v_k})$. We denote by a_{k-1}^d the action to proceed directly to customer v_k and by a^r_{k-1} the action to replenish capacity before proceeding to customer v_k. A transition from state s_{k-1} to s_k is a function of state s_{k-1} and the action selected when in state s_{k-1}. We denote the capacity and arrival time transition functions as $Q(s_{k-1}, a_{k-1})$ and $A(s_{k-1}, a_{k-1})$, respectively, for $a_{k-1} \in \{a^d_{k-1}, a^r_{k-1}\}$. If action a^d_{k-1} is selected, we separate the calculation of $Q(s_{k-1}, a^d_{k-1})$ and $A(s_{k-1}, a^d_{k-1})$ into three cases:

$$Q(s_{k-1}, a^d_{k-1}) = \begin{cases} Q_{v_{k-1}} - x_{v_{k-1}}, & x_{v_{k-1}} < Q_{v_{k-1}} \\ Q, & \frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} = \left\lfloor \frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} \right\rfloor \\ \left\lceil \frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} \right\rceil Q - x_{v_{k-1}} + Q_{v_{k-1}}, & \frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} > \left\lfloor \frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} \right\rfloor \end{cases} \tag{6.3}$$

and

$$A(s_{k-1}, a^d_{k-1}) = \begin{cases} A_{v_{k-1}} + t(v_{k-1}, v_k), & x_{v_{k-1}} < Q_{v_{k-1}} \\ A_{v_{k-1}} + \left(\frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} + 1\right) t(v_{k-1}, 0) & \frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} \\ \quad + \left(\frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q}\right) t(0, v_{k-1}) + t(0, v_k), & \quad = \left\lfloor \frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} \right\rfloor, \\ A_{v_{k-1}} + t(v_{k-1}, v_k) + \left\lceil \frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} \right\rceil & \frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} \\ \quad \times \left(t(v_{k-1}, 0) + t(0, v_{k-1})\right), & \quad > \left\lfloor \frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} \right\rfloor \end{cases} \tag{6.4}$$

where $v_1 = l_c$, $Q_{v_1} = q_c$, and $A_{v_1} = t_c$ are given by the current state of vehicle c, as described in §2.2. In the first case, demand at customer v_{k-1} is less than vehicle capacity upon arrival to v_{k-1}, thus vehicle capacity is simply decremented by the amount of demand and the vehicle travels directly from v_{k-1} to v_k. The second and third cases account for situations where demand at customer v_{k-1} is greater than or equal to vehicle capacity upon arrival to v_{k-1}, thereby requiring return trips to the depot to replenish capacity. The number of return trips required is $\lfloor (x_{v_{k-1}} - Q_{v_{k-1}})/Q \rfloor$. In the second case, satisfying demand at v_{k-1} exactly depletes vehicle capacity, thus requiring the vehicle to replenish at the depot one additional time and travel directly to customer v_k with full capacity. In the third case, there is some capacity remaining after serving demand at customer v_{k-1}. After making the necessary return trips to the depot, the vehicle travels directly from v_{k-1} to v_k with the remaining capacity.

If action a^r_{k-1} is selected, then vehicle capacity upon arrival to customer v_k is $Q(s_{k-1}, a^r_{k-1}) = Q$ and arrival time at customer v_k is $A(s_{k-1}, a^r_{k-1})$. We consider three cases when calculating $A(s_{k-1}, a^r_{k-1})$:

$$A(s_{k-1}, a^r_{k-1}) = \begin{cases} A_{v_{k-1}} + t(v_{k-1}, 0) + t(0, v_k), & x_{v_{k-1}} < Q_{v_{k-1}} \\ A_{v_{k-1}} + \left(\frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} + 1\right) t(v_{k-1}, 0) & \frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} \\ \quad + \left(\frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q}\right) t(0, v_{k-1}) + t(0, v_k), & \quad = \left\lfloor \frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} \right\rfloor \\ A_{v_{k-1}} + t(v_{k-1}, 0) + t(0, v_k) + \left\lceil \frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} \right\rceil & \frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} \\ \quad \times \left(t(v_{k-1}, 0) + t(0, v_{k-1})\right), & \quad > \left\lfloor \frac{x_{v_{k-1}} - Q_{v_{k-1}}}{Q} \right\rfloor \end{cases} . \tag{6.5}$$

In the first case, demand at customer v_{k-1} is less than vehicle capacity upon arrival to v_{k-1}, thus demand $x_{v_{k-1}}$ is fully served without replenishing capacity at the depot and the arrival time at customer v_k is simply the arrival time at customer v_{k-1} plus the travel time required to visit the depot and then travel to customer v_k. The second and third cases account for situations where demand at customer v_{k-1} is greater than or equal to vehicle capacity upon arrival to v_{k-1}. These cases account for the $\lfloor (x_{v_{k-1}} - Q_{v_{k-1}})/Q \rfloor$ return trips to the depot and the additional travel time required to replenish capacity before traveling to customer v_k.

To compute $R_k(s_k)$, the demand served at customer v_k, we consider the route duration limit, noting that violations of the route duration limit result in zero demand served. Three cases are considered in the calculation:

$$R_k(s_k) = \begin{cases} 0, & A_{v_k} > L - t(v_k, 0) \\ \left\lfloor \frac{L - t(v_k,0) - A_{v_k}}{t(v_k,0) + t(0,v_k)} \right\rfloor Q + Q_{v_k}, & \left\lfloor \frac{L - t(v_k,0) - A_{v_k}}{t(v_k,0) + t(0,v_k)} \right\rfloor \leq \left\lfloor \frac{x_{v_k} - Q_{v_k}}{Q} \right\rfloor \\ x_{v_k}, & x_{v_k} \leq Q_{v_k} \text{ and } A_{v_k} \leq L - t(v_k, 0), \\ & \text{or } \left\lfloor \frac{L - t(v_k,0) - A_{v_k}}{t(v_k,0) + t(0,v_k)} \right\rfloor > \left\lfloor \frac{x_{v_k} - Q_{v_k}}{Q} \right\rfloor \end{cases} . \tag{6.6}$$

In the first case, zero demand is served because the route duration limit is violated. In the second case, only a portion of demand is served because the vehicle does not have enough time to make the return trips to the depot necessary to serve demand in full. In the third case, demand is served in full either because the vehicle arrives prior to the route duration limit and with sufficient capacity, or because the vehicle has enough time to make any necessary replenishments.

For a fixed route v, we seek a restocking policy that maximizes the demand served at each customer $v_1, v_2, \ldots, v_b$. Let $\tilde{\Pi}(v)$ be the set of all restocking policies for fixed route v. We seek a restocking policy $\tilde{\pi}^\star(v)$ such that $V_1^{\tilde{\pi}^\star(v)} \geq V_1^{\tilde{\pi}(v)}$ for all $\tilde{\pi}(v) \in \tilde{\Pi}(v)$, where $V_1^{\tilde{\pi}(v)} = \sum_{k=1}^{b} R_k(s_k)$ is the total demand served at customers $v_1, v_2, \ldots, v_b$. The optimal policy $\tilde{\pi}^\star(v)$ can be found by solving the optimality equations (Puterman, 1994):

$$V_k(s_k) = R_k(s_k) + \max_{a \in \{a_k^d, a_k^r\}} \left\{ V_{k+1}\Big(s_{k+1} = \big(Q(s_k, a), A(s_k, a) \big) \Big) \right\}, \tag{6.7}$$

for $k = 1, \ldots, b-1$ and

$$V_b(s_b) = R_b(s_b). \tag{6.8}$$

6.5.2 Structural Properties

A sequence of states from stage 1 to stage b in the dynamic program represents one possible sequence of arrival time and capacity upon arrival to each customer $v_1, \ldots, v_b$ on the fixed route. Consider a sequence of states $s_1, s_2, \ldots, s_{k'}, \ldots, s_b$, where k' is the first stage such that the demand at customer $v_{k'}$ is not fully served, i.e., $R(s_{k'}) < x_{v_{k'}}$. By (6.6), it must be that demand is not fully served because doing so violates the route duration limit. Assuming the triangle inequality holds for travel times, then because we require demand to be served in full at customer $v_{k'}$ before proceeding to customer $v_{k'+1}$, it is not possible to serve any demand at customers $v_{k'+1}, v_{k'+2}, \ldots, v_b$ because doing so violates the route duration limit. We formalize this observation in Proposition 6.1. In §6.5.3, we use this result to prune the state space in a forward dynamic programming solution approach.

Proposition 6.1. *Assume travel times $t(\cdot,\cdot)$ satisfy the triangle inequality. Consider a sequence of states $s_1, s_2, \ldots, s_b$. Let k' be the smallest k such that $R_k(s_k) < x_{v_k}$. If k' exists, then:*

(i). $R_k(s_k) = 0$ for $k = k'+1, k'+2, \ldots, b$; and

(ii). $R_k(s_k) = x_{v_k}$ for $k = 1, 2, \ldots, k'-1$.

Proof. We first prove property (i). Since $R_{k'}(s_{k'}) < x_{v_{k'}}$, we have one of two cases. In the first case, $R_{k'}(s_{k'}) = 0$, implying by (6.6) that $A_{v_{k'}} + t(v_{k'}, 0) > L$. By the triangle inequality, this implies that for any $A_{v_{k'+1}}$ that may result from (6.4) or (6.5), $A_{v_{k'+1}} + t(v_{k'+1}, 0) > L$. Thus, $R_{k'+1}(s_{k'+1}) = 0$. The same argument applies for $k = k'+2, \ldots, b$. In the second case, $0 < R_{k'}(s_{k'}) < x_{v_{k'}}$, implying by (6.6) that $\lfloor (L - t(v_{k'}, 0) - A_{v'_k})/(t(v_{k'}, 0) + t(0, v_{k'})) \rfloor \leq \lfloor (x_{v_{k'}} - Q_{v_{k'}})/Q \rfloor$, meaning the number of replenishments required to satisfy demand in full is greater than the number of replenishments possible before violating the route duration limit L. Let t' be the time at which demand at customer $v_{k'}$ is served in full. It must be that $t' + t(v_{k'}, 0) > L$. Then, by the triangle inequality, $t' + t(v_{k'}, v_{k'+1}) + t(v_{k'+1}, 0) > L$ and $t' + t(v_{k'}, 0) + t(0, v_{k'+1}) + t(v_{k'+1}, 0) > L$, meaning that $A_{v_{k'+1}} + t(v_{k'+1}, 0) > L$ regardless of the action selected at decision epoch k'. Thus, $R_{k'+1}(s_{k'+1}) = 0$. Then, using the arguments presented for the first case, $R_k(s_k) = 0$ for $k = k'+2, \ldots, b$.

We prove property (ii) by contradiction. Suppose there exists some $k \in$

$\{1, 2, \ldots, k' - 1\}$ such that $R_k(s_k) < x_{v_k}$. Then, by property (i), $R_j(s_j) = 0$ for $j = k+1, k+2, \ldots, b$. Yet, by assumption, $R_j(s_j) = x_{v_j}$ for $j = 1, 2, \ldots, k' - 1$. $\square$

Proposition 6.2 shows that the demand served from a given state s_k through s_b is non-decreasing as the capacity upon arrival Q_{v_k} increases and as the arrival time A_{v_k} decreases. We use this result, in conjunction with Proposition 6.1, to further prune the state space in the forward dynamic programming approach we develop in §6.5.3.

Proposition 6.2. *For* $s_k = (Q_{v_k}, A_{v_k})$ *and* $s'_k = (Q'_{v_k}, A'_{v_k})$ *such that* $0 \leq Q_{v_k} \leq Q'_{v_k}$ *and* $0 \leq A'_{v_k} \leq A_{v_k}$, $V_k(s_k) \leq V_k(s'_k)$.

Proof. The proof is by induction. First, note that if $V_b(s_b) < x_{v_b}$, then $V_b(s_b)$ increases as Q_{v_b} increases and as A_{v_b} decreases. If $V_b(s_b) = x_{v_b}$, then $V_b(s_b)$ is constant as it is bounded above by x_{v_b}. Thus, the result holds for stage b. Assume the result holds for stages $b-1, b-2, \ldots, k+1$. At stage k, it follows from (6.6) that $R_k(s_k)$ increases as Q_{v_k} increases and as A_{v_k} decreases. As in stage b, $R_k(s_k)$ is constant if $R_k(s_k) = x_{v_k}$. By the induction hypothesis, the reward-to-go, $V_{k+1}(s_{k+1})$, also increases as Q_{v_k} increases and as A_{v_k} decreases. Because the value function at stage k is the sum of two functions that increase as Q_{v_k} increases and as A_{v_k} decreases, the result holds at stage k. $\square$

6.5.3 Solution Approach

Because we do not require travel times to be discrete, the state space of the dynamic program we formulate in §6.5.1 is infinite, thereby rendering the standard backward dynamic programming procedure computationally intractable. To overcome this obstacle, we develop a forward dynamic programming approach that only considers states that may actually occur as a result of a given initial state. Our approach mirrors the well-known reaching algorithm for dynamic programming (Denardo, 2003) and leverages the structural results of §6.5.2 to prune. Our forward dynamic programming procedure is able to obtain the values of optimal restocking policies for large fixed routes. Without the pruning afforded by our structural results, our forward approach enumerates the entire solution space. We note that our structural results and forward solution approach are also applicable to the discrete-time case discussed in §6.3.

Our solution approach utilizes a graph structure. Each node in the graph is labeled by a state, the demand served in that state, and the total demand served so far. Thus, a node associated with a state s_k is represented by the tuple $(s_k = (Q_{v_k}, A_{v_k}), R_k(s_k), \lambda_k = \sum_{i=1}^{k} R_k(s_k))$. Nodes are connected by arcs representing actions. An arc from a node s_{k-1} to a node s_k represents an action $a_{k-1} \in \{a^d_{k-1}, a^r_{k-1}\}$ denoting the decision to travel directly from v_{k-1} to v_k or to first replenish capacity at the depot. The graph is constructed in stages, one for each customer on the fixed route. We refer to the set of nodes belonging to stage k as Λ_k. Stage 1 of the graph is constructed via s_1, the given initial state of the vehicle. Stage 2 is constructed by *extending* the initial node corresponding to state s_1, i.e., by considering the states that result by taking actions a^d_1 and a^r_1 from the initial state. Any stage $k+1$ is constructed in a similar manner by extending the nodes in stage Λ_k. The node in Λ_b

that achieves the largest λ_b indicates the value of the optimal restocking policy for the given fixed route. The optimal policy is represented by the sequence of actions leading to this node.

The total number of nodes in the graph is $2^b - 1$, where b is the number of customers on the fixed route. The total number of node sequences from stage 1 through stage b, each of which represents a policy, is 2^{b-1}. In our computational experience, storing the graph in memory becomes problematic as b approaches 25. Further, because heuristic $\tilde{\mathcal{H}}(\cdot)$ requires us to obtain optimal restocking policies for many fixed routes, evaluating fixed routes can be computationally prohibitive even when b is much smaller.

These computational issues can be mitigated by exploiting Propositions 6.1 and 6.2. Consider a partial path through the graph, which we denote by the sequence of states $s_1, s_2, \ldots, s_{k'}$, where k' is the first stage along this path such that the demand at customer $v_{k'}$ is not fully served. By Proposition 6.1, any extension of this path will result in zero demand served at customers $v_{k'+1}, v_{k'+2}, \ldots, v_b$, i.e., $R_k(s_k) = 0$ for $k = k'+1, k'+2, \ldots, b$. Thus, to obtain the value of the optimal restocking policy, it is only necessary to extend nodes that fully serve demand. Thus, at stage k, we extend only nodes $\Lambda'_k = \{(s_k, R_k(s_k), \lambda_k) : R_k(s_k) = x_{v_k}\}$.

Additional pruning is possible by using the result of Proposition 6.2 to further refine Λ'_k. For any two nodes $(s_k = (Q_{v_k}, A_{v_k}), R_k(s_k), \lambda_k)$ and $(s'_k = (Q'_{v_k}, A'_{v_k}), R_k(s'_k), \lambda_k)$ in Λ'_k, if $Q_{v_k} \leq Q'_{v_k}$ and $A'_{v_k} \leq A_{v_k}$, then it is not necessary to extend s_k because the total demand served by extending s'_k will be at least as large. More formally, let

$\Lambda''_k = \{(s_k, \cdot, \cdot) \in \Lambda'_k : \nexists\ (s'_k, \cdot, \cdot) \in \Lambda'_k$ such that $Q_{v_k} \leq Q'_{v_k}$ and $A'_{v_k} \leq A_{v_k}\}$ be the set of *non-dominated* nodes in Λ'_k. Proposition 6.2 guarantees that an optimal policy will be obtained by extending only the non-dominated $\Lambda''_k \subseteq \Lambda'_k$.

Algorithm 6.2 details our forward dynamic programming procedure. The EVALUATE(v, x, s_1) procedure in Algorithm 6.2 takes as input a fixed route v, customer demands x, and initial state s_1 at customer v_1. It returns $V^{\pi(v)}(x)$, the value of the optimal restocking policy for fixed route v. The procedure begins in line 2 by initializing Λ_1 with the given initial state s_1, $R_1(s_1)$, and $\lambda_1 = R_1(s_1)$, the demand served in state s_1. For $k = 2, \ldots, b$, Λ_k is initialized to the empty set. Line 4 begins the process of identifying nodes in Λ_k to extend. Per Proposition 6.1, states not fully serving demand at customer v_k need not be extended. Thus, $\Lambda'_k \subseteq \Lambda_k$ restricts attention to nodes such that $R_k(s_k) = x_{v_k}$. If Λ'_k is empty, then, by Proposition 6.1, it is not necessary to extend any nodes because doing so will not increase the total demand served. Line 6 accomplishes this by exiting the for loop. In line 7, the set of nodes to be extended is further refined by applying Proposition 6.2. When identifying nodes for inclusion in Λ''_k, it may be that several nodes are identical, and therefore each node dominates the other. In such cases, we extend one of these nodes, provided that it is not dominated by another node. Lines 9 and 10 construct Λ_{k+1} by extending the nodes in Λ''_k. Line 11 identifies the non-empty node set with the largest index, $\bar{k}$. Finally, the procedure returns the maximum demand served by the nodes in $\Lambda_{\bar{k}}$.

Figure 6.3 depicts the full graph for the example fixed route we consider in

Algorithm 6.2 Valuation of Optimal Restocking Policy

```
 1: procedure EVALUATE($v, x, s_1$)
 2:     $\Lambda_1 \leftarrow \{(s_1, R_1(s_1), \lambda_1 = R_1(s_1))\}$, $\Lambda_k \leftarrow \emptyset$ for $k = 2, 3, \dots, b$
 3:     for $k = 1$ to $b - 1$ do
 4:         $\Lambda'_k \leftarrow \{(s_k, R_k(s_k), \lambda_k) \in \Lambda_k : R_k(s_k) = x_{v_k}\}$
 5:         if $\Lambda'_k = \emptyset$ then
 6:             break
 7:         $\Lambda''_k \leftarrow \{(s_k, \cdot, \cdot) \in \Lambda'_k : \nexists\ (s'_k, \cdot, \cdot) \in \Lambda'_k$ such that $Q_{v_k} \leq Q'_{v_k}$ and $A'_{v_k} \leq$
    $A_{v_k}\}$
 8:         for $(s_k, \lambda_k) \in \Lambda''_k$ do
 9:             $\Lambda_{k+1} \leftarrow \Lambda_{k+1} \cup \{(s_{k+1} = (Q(s_k, a^d_k), A(s_k, a^d_k)), \lambda_k + R_{k+1}(s_{k+1})\}$
10:             $\Lambda_{k+1} \leftarrow \Lambda_{k+1} \cup \{(s_{k+1} = (Q(s_k, a^r_k), A(s_k, a^r_k)), \lambda_k + R_{k+1}(s_{k+1})\}$
11:     $\bar{k} \leftarrow$ largest $k \in \{1, 2, \dots, b\}$ such that $\Lambda_k \neq \emptyset$
12:     return $\max\{\lambda_{\bar{k}} : (s_{\bar{k}}, R_{\bar{k}}(s_{\bar{k}}), \lambda_{\bar{k}}) \in \Lambda_{\bar{k}}\}$
```

Table 6.1. For convenience, each node is numbered in the upper-left corner. We demonstrate Algorithm 6.2 by stepping through the construction of the graph in Figure 6.3. Given $v = (23, 22, 13, 15, 2)$, $x = (29, 18, 23, 8, 7)$, and $s_1 = (50, 36.0555)$, the procedure begins by calling EVALUATE(v, x, s_1). Node set Λ_1 is initialized via s_1, $R_1(s_1) = 39$, and $\lambda_1 = 39$. Because $\Lambda_1 = \Lambda'_1 = \Lambda''_1 = \{1\}$, node 1 is extended to create $\Lambda_2 = \{2, 3\}$. Because $\Lambda_2 = \Lambda'_2 = \Lambda''_2$, nodes 2 and 3 are extended to create $\Lambda_3 = \{4, 5, 6, 7\}$. All four nodes in Λ_3 fully serve demand at customer $v_3 = 13$, thus $\Lambda'_3 = \Lambda_3$. However, node 5 is dominated by node 7, thus we only extend the nodes in $\Lambda''_3 = \{4, 6, 7\}$ to create $\Lambda_4 = \{8, 9, 12, 13, 14, 15\}$. Only node 12 fully serves demand at customer $v_4 = 15$, thus $\Lambda'_4 = \Lambda''_4 = \{4\}$. Extending node 12 results in $\Lambda_5 = \{24, 25\}$. The procedure concludes by returning $\max\{89, 88\} = 89$, which is the value of the optimal restocking policy for fixed route v when demand is x and the

initial state is s_1. The optimal policy is represented by the sequence of nodes 1, 3, 6, 12, 24, which corresponds to an optimal sequence of actions $a_1^r, a_2^d, a_3^d, a_4^d$.

In this example, Algorithm 6.2 decreases the number of nodes in the graph from 31 to 15. In our experience, for large fixed routes, Algorithm 6.2 can decrease the number of nodes by several orders of magnitude, thereby making it computationally feasible to solve the dynamic program required to obtain an optimal restocking policy for a given fixed route. For smaller problem instances where the number of customers in a fixed route is typically smaller, we observe runtime reductions up to 75 percent when embedding Algorithm 6.2 in heuristic $\tilde{\mathcal{H}}(\cdot)$.

6.6 Computational Experience

In this section, we examine the effectiveness of our pre-decision rollout policy for the VRPSDL. Ideally, our rollout policy would be compared to an optimal policy. However, we experience the same difficulties we discuss in §5.4 and are unable to obtain optimal policies or bounds on the value of optimal policies. Given these limitations, we develop two benchmarks for our results. First, we compare the demand collected by our pre-decision rollout policy to the demand collected by a stand-alone restocking policy. This benchmark restocking policy is obtained by applying $\tilde{\mathcal{H}}(\cdot)$ to the benchmark a priori policy described in §5.4.1. This comparison allows us to compare dynamic updating to the static routing often implemented in practice (Erera et al., 2010). Second, we compare the demand collected by our pre-decision rollout policy to the demand collected by the hybrid rollout policy developed in §5.3. This

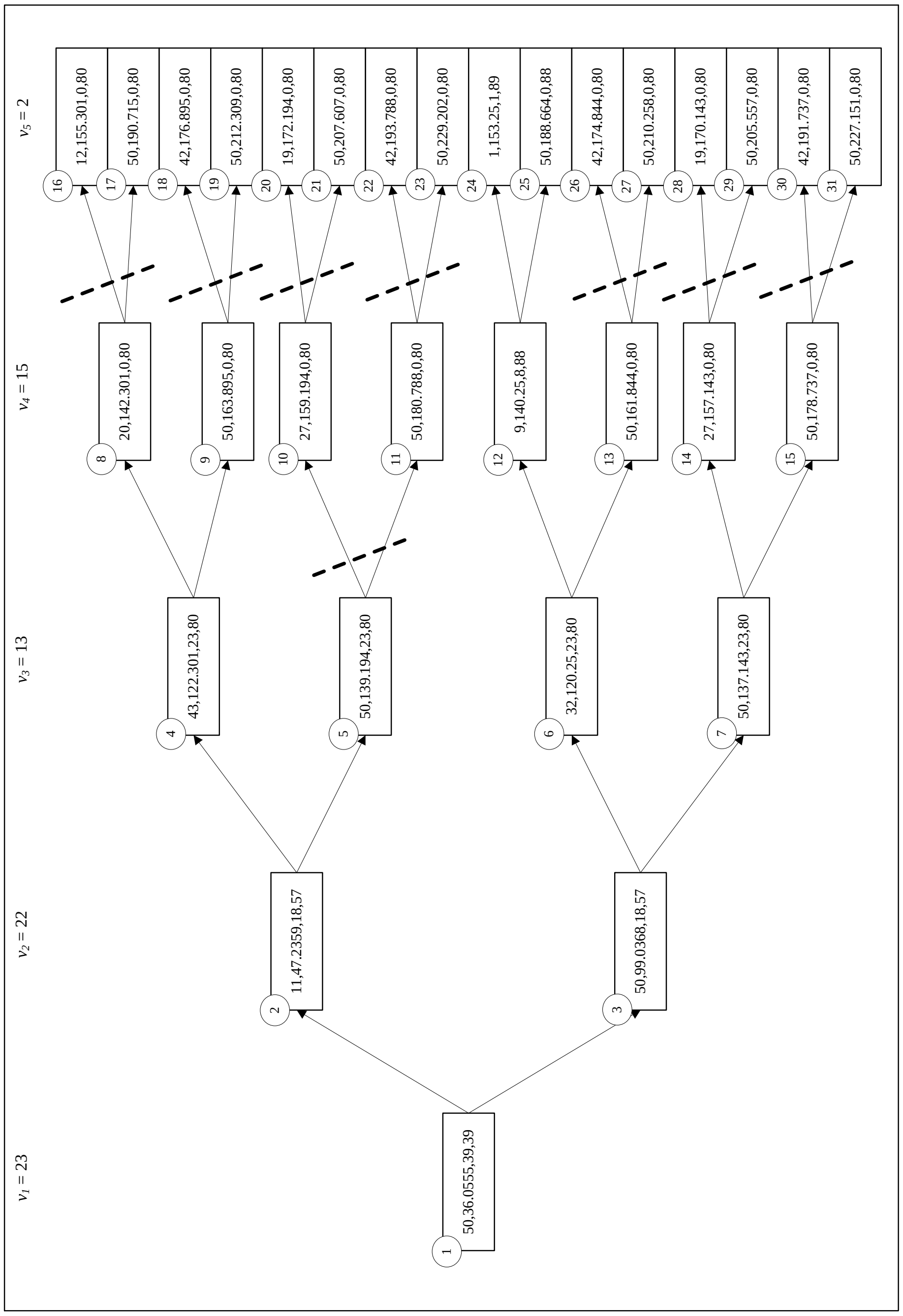

Figure 6.3: Example Dynamic Program

comparison allows us to gauge the benefit of using a restocking policy (instead of an a priori policy) to approximate future rewards.

To facilitate our experiments, we use the same set of 216 problem instance described in §5.4.2. We also utilize the same 500 realizations for each problem instance. As in §5.4.2, we present a confidence interval for the demand served by one policy over the demand served by another policy. When estimating the expected demand served by a restocking policy in $\tilde{\mathcal{H}}(\cdot)$, we use $M = 30$ demand samples. We use the same set of M demand samples throughout the procedure.

We implement our experiments, which required more than 180 CPU days, in C++ and execute them on a computing cluster provided by the University of Iowa Information Technology Services. The cluster contains eight nodes with the CentOS 5.3 operating system. Each node consists of dual quad core AMD Opteron Processors 2350 at 2GHz and 16GB of ECC DDR2 667MHz SDRAM.

Table 6.2 displays confidence intervals for the percent increase in demand served by our pre-decision rollout policies over the corresponding benchmark restocking policies. Confidence intervals to the right of zero are in bold and indicate a statistically significant increase in demand served by our pre-decision rollout policies over the benchmark restocking policies. Confidence intervals containing zero are in normal typeface and indicate no statistically significant difference in demand collected by the two policies.

Across all 216 problem instances, the upper end of the confidence intervals in Table 6.2 is always greater than zero, thereby suggesting that following our pre-decision rollout policy weakly improves upon the benchmark restocking policy. In some cases the improvement is dramatic, while in others, there is little to no benefit. We observe the same trends in Table 6.2 as we observe in Table 5.2, which compares demand collected by the hybrid rollout policy of §5.3 to benchmark a priori policies. First, for a given duration limit, as vehicle capacity decreases, the the percent demand served over benchmark restocking policies tends to increase. Second, when variability in customer demand is high, pre-decision rollout policies tend to perform better on clustered instances than on randomized instances. Third, the benefit of our pre-decision rollout policies over the benchmark restocking policies generally decreases as variability in customer demand decreases. The explanations for the same observations, provided in §5.4.4, apply.

Table 6.3 displays confidence intervals for the percent increase in demand served by our pre-decision rollout policies over the corresponding hybrid rollout policies. Confidence intervals to the right of zero are in bold and indicate a statistically significant increase in demand served by our pre-decision rollout policies over the hybrid rollout policies. Confidence intervals containing zero are in normal typeface and indicate no statistically significant difference in demand collected by the two policies. Confidence intervals to the left of zero are in italics and indicate a statistically significant decrease in demand served by our pre-decision rollout policies over the hybrid rollout policies.

The confidence intervals in Table 6.3 show both statistically significant increases and decreases in demand served by the pre-decision rollout policies over the

Table 6.2: Confidence Intervals for Percent Increase in Demand Served by Pre-Decision Rollout Policy $\pi_{\mathcal{R}\tilde{\mathcal{H}}''}$ Over Benchmark Restocking Policies

Duration	short			medium			long		
Capacity	small	medium	large	small	medium	large	small	medium	large
				Low Variability					
R101 (25)	**[0.30, 0.50]**	[−0.02, 0.20]	[−0.20, 0.54]	**[0.17, 0.80]**	[−0.01, 0.76]	[−0.23, 0.57]	**[0.96, 1.66]**	[−0.30, 0.51]	[−0.24, 0.71]
C101 (25)	**[0.53, 1.32]**	[−0.19, 0.69]	[−0.21, 0.75]	**[0.54, 1.29]**	**[0.41, 1.24]**	[−0.28, 0.58]	**[0.23, 0.79]**	[−0.24, 0.59]	[−0.26, 0.63]
R101 (50)	**[0.17, 0.85]**	**[0.12, 0.62]**	**[0.24, 0.61]**	**[0.70, 1.26]**	**[0.24, 0.81]**	[−0.14, 0.49]	**[0.51, 1.00]**	[−0.25, 0.38]	[−0.27, 0.92]
C101 (50)	**[0.79, 1.45]**	**[0.04, 0.64]**	**[0.42, 0.91]**	**[0.63, 1.11]**	**[1.78, 2.48]**	[−0.06, 0.60]	**[0.69, 1.15]**	[−0.24, 0.38]	[−0.21, 0.53]
R101 (75)	**[0.69, 1.21]**	**[0.73, 1.19]**	**[0.03, 0.42]**	**[0.52, 0.96]**	[0.00, 0.51]	[−0.20, 0.33]	**[0.79, 1.21]**	[−0.22, 0.31]	[−0.18, 0.46]
C101 (75)	**[1.01, 1.50]**	**[0.61, 1.07]**	**[0.00, 0.50]**	**[0.55, 0.97]**	[−0.12, 0.41]	[−0.21, 0.34]	**[0.46, 0.89]**	[−0.22, 0.33]	[−0.20, 0.59]
R101 (100)	**[1.03, 1.46]**	**[0.74, 1.20]**	**[0.73, 1.12]**	**[0.54, 0.93]**	**[0.07, 0.52]**	[−0.16, 0.31]	**[1.01, 1.36]**	[−0.17, 0.29]	[−0.17, 0.54]
C101 (100)	**[0.81, 1.25]**	**[0.56, 0.91]**	**[0.22, 0.68]**	**[0.94, 1.34]**	**[0.21, 0.63]**	[−0.14, 0.34]	**[1.03, 1.37]**	[−0.19, 0.29]	[−1.35, 4.31]
				Moderate Variability					
R101 (25)	**[2.36, 3.41]**	**[0.95, 1.40]**	**[0.68, 1.20]**	**[2.75, 3.39]**	**[2.17, 2.98]**	**[0.56, 1.07]**	**[2.02, 2.50]**	**[0.05, 0.18]**	[0.00, 0.00]
C101 (25)	**[4.47, 5.52]**	**[2.49, 3.21]**	**[0.92, 1.26]**	**[2.72, 3.38]**	**[4.19, 5.14]**	**[0.27, 0.48]**	**[2.40, 2.94]**	**[0.22, 0.42]**	**[0.02, 0.06]**
R101 (50)	**[1.97, 2.48]**	**[0.23, 0.48]**	**[0.36, 0.59]**	**[3.23, 3.71]**	**[1.73, 2.10]**	**[0.18, 0.38]**	**[3.33, 3.70]**	**[0.09, 0.26]**	**[0.00, 0.03]**
C101 (50)	**[2.90, 3.38]**	**[1.87, 2.46]**	**[0.54, 0.75]**	**[2.56, 3.01]**	**[1.53, 2.70]**	**[1.11, 2.65]**	**[2.70, 3.68]**	**[0.26, 1.85]**	[−0.03, 1.74]
R101 (75)	**[2.62, 3.85]**	**[0.33, 1.57]**	**[1.11, 2.31]**	**[2.39, 3.38]**	**[1.00, 2.36]**	[−0.15, 1.35]	**[2.58, 3.61]**	[−0.34, 1.15]	[−0.28, 1.29]
C101 (75)	**[4.27, 5.44]**	**[1.66, 2.65]**	**[2.04, 3.30]**	**[3.05, 4.02]**	**[0.85, 2.23]**	[−0.31, 1.19]	**[2.95, 4.07]**	[−0.17, 1.32]	[−0.26, 1.38]
R101 (100)	**[3.07, 4.13]**	**[0.98, 1.94]**	**[0.98, 1.95]**	**[3.03, 3.89]**	**[0.82, 1.95]**	**[0.05, 1.33]**	**[3.08, 4.03]**	[−0.26, 1.01]	[−0.26, 1.17]
C101 (100)	**[3.46, 4.44]**	**[1.78, 2.57]**	**[2.35, 3.44]**	**[3.54, 4.41]**	**[1.61, 2.67]**	[−0.04, 1.22]	**[3.28, 4.14]**	[−0.14, 1.11]	[−0.29, 1.49]
				High Variability					
R101 (25)	**[4.05, 5.78]**	**[1.77, 2.93]**	[−0.11, 0.04]	**[4.26, 5.51]**	**[3.73, 4.81]**	**[1.64, 2.15]**	**[5.13, 6.17]**	**[0.29, 0.49]**	[0.00, 0.01]
C101 (25)	**[7.40, 8.84]**	**[3.84, 4.77]**	**[2.79, 3.53]**	**[7.85, 8.96]**	**[5.75, 6.73]**	**[3.54, 4.32]**	**[6.23, 7.24]**	**[1.32, 1.83]**	**[0.15, 0.22]**
R101 (50)	**[4.19, 4.93]**	**[1.05, 2.18]**	**[1.22, 3.58]**	**[4.04, 6.31]**	**[2.66, 5.51]**	**[0.66, 3.75]**	**[5.13, 7.53]**	**[0.64, 3.79]**	**[0.23, 3.65]**
C101 (50)	**[3.07, 5.60]**	**[3.77, 6.50]**	**[1.38, 4.65]**	**[6.19, 8.58]**	**[3.57, 5.88]**	**[2.92, 5.62]**	**[6.47, 8.57]**	**[2.43, 5.29]**	**[0.05, 3.18]**
R101 (75)	**[5.66, 7.97]**	**[2.52, 4.59]**	**[2.23, 4.28]**	**[4.36, 6.25]**	**[2.53, 4.85]**	**[0.10, 2.72]**	**[5.22, 7.22]**	**[0.00, 2.62]**	[−0.07, 2.63]
C101 (75)	**[7.53, 9.64]**	**[3.77, 5.68]**	**[4.17, 6.32]**	**[6.73, 8.57]**	**[3.47, 5.81]**	[−0.02, 2.60]	**[7.06, 9.06]**	**[0.30, 2.93]**	[−0.08, 2.67]
R101 (100)	**[6.16, 8.06]**	**[3.54, 5.28]**	**[2.22, 3.95]**	**[5.19, 6.74]**	**[2.35, 4.36]**	**[0.57, 2.78]**	**[5.64, 7.38]**	**[0.10, 2.28]**	[−0.15, 2.18]
C101 (100)	**[6.98, 8.82]**	**[4.25, 5.81]**	**[2.59, 4.38]**	**[6.62, 8.21]**	**[4.10, 5.96]**	**[0.62, 2.79]**	**[6.67, 8.26]**	**[0.70, 2.86]**	[−0.27, 3.38]

hybrid rollout policies. In the latter case, differences in demand collected are small, almost never exceeding one percent. On the other hand, the pre-decision rollout policy sometimes collects as much as five percent more demand than the hybrid rollout policy. In general, while there does not appear to be an overarching pattern in these results, there is one notable trend: pre-decision rollout policies tend to perform better than hybrid rollout policies when vehicle capacities are small. As vehicle capacities increase to medium and large, the confidence intervals tend to shift to the left, indicating a decrease in this benefit. Intuition suggests the following explanation. Vehicles with smaller capacities experience more route failures/replenishments than vehicles with larger capacities. The hybrid rollout policy projects the impact of route failures via an a priori fixed route policy, while the pre-decision rollout policy utilizes a restocking fixed route policy. Because restocking fixed route policies relax a priori fixed route policies by permitting capacity replenishment prior to route failures, restocking fixed route policies are able to better plan for capacity replenishments. This is corroborated by the fact that we observe no statistically significant differences in the distances traveled by the hybrid and pre-decision rollout policies. Thus, the difference in demand collected is most likely due to more intelligent capacity replenishment actions. As vehicle capacities increase to medium and large, the number of route failures tends to decrease, thus providing fewer opportunities to leverage the ability of restocking fixed route policies to better respond to route failures.

Although solution quality is our primary consideration, potential for real-time decision making is also important. Table 6.4 displays per epoch computing times

Table 6.3: Confidence Intervals for Percent Increase in Demand Served by Pre-Decision Rollout Policy $\pi_{\mathcal{R}\tilde{\mathcal{H}}''}$ Over Hybrid Rollout Policy $\pi_{\mathcal{HY}}$

Duration	short			medium			long		
Capacity	small	medium	large	small	medium	large	small	medium	large
				Low Variability					
R101 (25)	**[2.84, 3.30]**	**[0.71, 1.12]**	**[0.01, 0.03]**	**[0.16, 0.38]**	**[0.08, 0.20]**	**[0.01, 0.05]**	**[0.30, 0.53]**	[0.00, 0.00]	[0.00, 0.00]
C101 (25)	**[2.10, 2.40]**	**[0.05, 0.12]**	[–0.01, 0.01]	**[0.16, 0.43]**	[–0.09, 0.14]	[0.00, 0.00]	[–0.03, 0.07]	[–0.03, 0.01]	[0.00, 0.00]
R101 (50)	*[–0.78, –0.56]*	**[0.55, 0.72]**	**[0.02, 0.08]**	**[1.15, 1.34]**	**[0.13, 0.22]**	[–0.01, 0.00]	**[0.55, 0.68]**	[0.00, 0.00]	[0.00, 0.00]
C101 (50)	**[0.10, 0.32]**	[–0.17, 0.20]	*[–0.68, –0.51]*	**[0.07, 0.19]**	**[0.16, 0.39]**	[0.00, 0.00]	**[0.04, 0.16]**	[–0.04, 0.00]	[0.00, 0.00]
R101 (75)	**[0.51, 0.65]**	**[0.09, 0.23]**	**[0.65, 0.74]**	**[1.11, 1.27]**	*[–0.04, –0.01]*	[0.00, 0.00]	[–0.04, 0.08]	[0.00, 0.00]	[0.00, 0.00]
C101 (75)	*[–0.15, –0.02]*	*[–0.37, –0.23]*	[–0.01, 0.05]	**[0.02, 0.13]**	[–0.03, 0.00]	[0.00, 0.00]	*[–0.18, –0.07]*	[0.00, 0.00]	[0.00, 0.00]
R101 (100)	**[0.43, 0.58]**	**[0.44, 0.59]**	[–0.07, 0.01]	**[0.86, 1.00]**	[0.00, 0.04]	[0.00, 0.00]	[–0.06, 0.06]	[0.00, 0.00]	[0.00, 0.00]
C101 (100)	**[0.03, 0.21]**	**[0.09, 0.20]**	**[0.12, 0.20]**	**[0.12, 0.22]**	[–0.05, 0.02]	*[–0.03, –0.01]*	[–0.03, 0.09]	[0.00, 0.00]	[0.00, 0.00]
				Moderate Variability					
R101 (25)	**[5.01, 6.50]**	**[1.97, 2.80]**	**[0.52, 0.93]**	**[0.95, 1.48]**	[–0.06, 0.38]	[–0.11, 0.05]	**[1.17, 1.78]**	*[–0.05, –0.01]*	[0.00, 0.00]
C101 (25)	**[3.39, 4.30]**	**[0.26, 0.60]**	**[0.05, 0.19]**	**[0.18, 0.75]**	**[1.24, 1.81]**	[–0.06, 0.14]	*[–0.47, –0.06]*	*[–0.57, –0.32]*	[0.00, 0.00]
R101 (50)	**[3.59, 4.24]**	**[0.93, 1.19]**	*[–1.58, –1.16]*	**[0.01, 0.41]**	[–0.02, 0.24]	[–0.01, 0.05]	**[0.24, 0.56]**	[–0.01, 0.02]	[0.00, 0.00]
C101 (50)	**[2.24, 2.74]**	*[–0.31, –0.02]*	*[–0.16, –0.05]*	**[0.71, 1.12]**	[–0.14, 0.31]	[–0.06, 0.14]	[–0.05, 0.23]	[–0.12, 0.05]	[–0.01, 0.02]
R101 (75)	**[0.81, 1.22]**	**[1.74, 2.14]**	*[–0.60, –0.36]*	**[0.39, 0.69]**	*[–0.29, –0.16]*	[–0.03, 0.00]	**[0.14, 0.35]**	[0.00, 0.00]	[0.00, 0.00]
C101 (75)	[–0.10, 0.25]	*[–0.56, –0.31]*	*[–0.29, –0.08]*	[–0.09, 0.14]	[–0.04, 0.10]	[–0.01, 0.00]	[–0.13, 0.10]	[–0.01, 0.01]	[0.00, 0.00]
R101 (100)	**[0.27, 0.59]**	**[1.14, 1.49]**	**[0.01, 0.16]**	**[0.51, 0.79]**	*[–0.20, –0.08]*	[0.00, 0.01]	**[0.09, 0.29]**	[0.00, 0.00]	[0.00, 0.00]
C101 (100)	*[–0.33, –0.03]*	**[0.04, 0.26]**	**[0.24, 0.39]**	*[–0.33, –0.12]*	**[0.18, 0.37]**	*[–0.06, –0.02]*	*[–0.28, –0.09]*	*[–0.04, –0.02]*	[0.00, 0.00]
				High Variability					
R101 (25)	**[0.86, 2.39]**	[–0.84, 0.66]	**[0.48, 1.08]**	**[0.11, 1.16]**	[–0.37, 0.37]	*[–0.85, –0.48]*	[–0.79, 0.00]	[–0.02, 0.09]	[0.00, 0.00]
C101 (25)	**[0.50, 1.60]**	**[0.62, 1.43]**	**[0.09, 0.50]**	**[0.34, 1.20]**	**[0.87, 1.74]**	[–0.05, 0.32]	[–0.45, 0.14]	[–0.02, 0.27]	[0.00, 0.00]
R101 (50)	**[2.16, 3.07]**	*[–0.65, –0.08]*	[–0.12, 0.54]	**[0.85, 1.55]**	[–0.34, 0.10]	*[–0.26, –0.11]*	*[–0.79, –0.30]*	[–0.04, 0.03]	[0.00, 0.00]
C101 (50)	**[0.35, 1.19]**	**[0.31, 0.87]**	*[–0.41, –0.12]*	**[0.88, 1.50]**	[–0.14, 0.59]	*[–0.49, –0.07]*	*[–0.50, –0.03]*	**[0.35, 0.64]**	*[–0.10, –0.06]*
R101 (75)	[–0.56, 0.05]	[–0.52, 0.07]	[–0.14, 0.18]	[–0.12, 0.34]	[–0.15, 0.16]	[–0.01, 0.00]	*[–0.56, –0.23]*	[0.00, 0.01]	[0.00, 0.00]
C101 (75)	**[0.17, 0.69]**	**[0.35, 0.78]**	[–0.02, 0.29]	*[–0.60, –0.26]*	[–0.26, 0.04]	[–0.01, 0.01]	*[–0.62, –0.29]*	*[–0.16, –0.11]*	[0.00, 0.00]
R101 (100)	[–0.13, 0.41]	*[–1.15, –0.68]*	[–0.01, 0.30]	**[0.13, 0.51]**	*[–0.61, –0.38]*	*[–0.04, –0.02]*	*[–0.52, –0.21]*	[0.00, 0.00]	[0.00, 0.00]
C101 (100)	*[–1.10, –0.58]*	**[0.73, 1.10]**	[–0.28, 0.03]	*[–0.39, –0.02]*	**[0.16, 0.50]**	*[–0.24, –0.12]*	*[–0.55, –0.25]*	[–0.04, 0.02]	[0.00, 0.00]

for the pre-decision rollout policy. The first number of each pair is the average number of CPU seconds required to select an action across all decision epochs in all 500 realizations. The second number of each pair is the average of the maximum number of seconds required to select an action across all 500 realizations. For problem C101(100) with high variability, a long duration limit, and medium capacity, the average of the maximum computing time required to select an action is nearly 10 minutes. This problem instance appears to be an exception, however, as the average of the maximum computing time required to select an action rarely exceeds one minute across the remaining 215 problem instances.

Compared to the per epoch computing times of hybrid rollout policies displayed in Table 5.3, pre-decision rollout policies are generally more computationally efficient. Typically, the a priori-based fixed routes heuristic employed by hybrid rollout policies requires less time to execute than the restocking-based fixed routes heuristic employed by pre-decision rollout policies. However, at a given a decision epoch, a hybrid rollout policy must execute the fixed routes heuristic at least twice, whereas a pre-decision rollout policy executes a fixed routes heuristic only once.

6.7 Conclusion and Future Research

We propose a pre-decision rollout policy to obtain dynamic solutions for the multi-vehicle routing problem with stochastic demand and duration limits (VRPSDL). In contrast to the a priori-based fixed routes heuristic underlying the hybrid rollout policy we develop in §5.3, the restocking-based fixed routes heuristic underlying our

Table 6.4: Average and Maximum Per Epoch CPU Seconds for Pre-Decision Rollout Policies

Duration	short			medium			long		
Capacity	small	medium	large	small	medium	large	small	medium	large
				Low Variability					
R101 (25)	(0.01, 0.29)	(0.01, 0.33)	(0.02, 0.43)	(0.02, 0.71)	(0.02, 0.66)	(0.03, 0.82)	(0.04, 1.00)	(0.03, 1.00)	(0.03, 0.92)
C101 (25)	(0.02, 0.65)	(0.01, 0.36)	(0.02, 0.58)	(0.03, 0.98)	(0.02, 0.77)	(0.02, 0.75)	(0.05, 1.00)	(0.04, 1.00)	(0.02, 0.90)
R101 (50)	(0.11, 1.00)	(0.10, 1.00)	(0.09, 1.01)	(0.15, 1.02)	(0.13, 1.02)	(0.11, 1.03)	(0.25, 2.28)	(0.17, 1.21)	(0.19, 1.65)
C101 (50)	(0.11, 1.01)	(0.09, 1.00)	(0.09, 1.00)	(0.14, 1.01)	(0.13, 1.02)	(0.14, 1.58)	(0.20, 1.18)	(0.28, 3.08)	(0.16, 1.14)
R101 (75)	(0.41, 1.43)	(0.38, 2.07)	(0.47, 3.37)	(0.76, 5.35)	(0.84, 7.45)	(0.71, 6.56)	(1.13, 9.87)	(0.92, 6.95)	(0.99, 7.82)
C101 (75)	(0.46, 2.28)	(0.47, 2.77)	(0.60, 6.32)	(0.79, 4.61)	(0.83, 8.72)	(0.56, 5.18)	(1.29, 10.71)	(0.65, 5.60)	(0.61, 5.79)
R101 (100)	(1.23, 9.09)	(1.64, 18.59)	(1.81, 23.98)	(2.55, 20.10)	(2.86, 24.94)	(2.48, 21.06)	(5.33, 48.73)	(3.42, 24.44)	(3.20, 25.15)
C101 (100)	(1.17, 7.99)	(1.41, 9.66)	(1.48, 13.76)	(2.74, 19.76)	(2.75, 30.32)	(1.81, 16.15)	(3.76, 32.56)	(2.43, 19.04)	(2.27, 20.80)
				Moderate Variability					
R101 (25)	(0.02, 0.48)	(0.02, 0.44)	(0.02, 0.39)	(0.03, 0.88)	(0.02, 0.76)	(0.02, 0.77)	(0.04, 1.00)	(0.05, 1.00)	(0.03, 0.92)
C101 (25)	(0.02, 0.64)	(0.02, 0.55)	(0.02, 0.55)	(0.03, 0.96)	(0.03, 0.97)	(0.03, 0.99)	(0.04, 1.00)	(0.08, 1.01)	(0.10, 1.08)
R101 (50)	(0.13, 1.01)	(0.10, 1.00)	(0.10, 1.00)	(0.21, 1.83)	(0.18, 1.50)	(0.14, 1.06)	(0.30, 2.26)	(0.20, 1.50)	(0.14, 1.06)
C101 (50)	(0.13, 1.02)	(0.10, 1.00)	(0.10, 1.00)	(0.17, 1.07)	(0.17, 1.26)	(0.21, 2.03)	(0.28, 2.09)	(0.28, 2.28)	(0.21, 1.51)
R101 (75)	(0.50, 2.43)	(0.41, 2.34)	(0.42, 3.60)	(0.97, 7.87)	(1.15, 11.79)	(0.88, 10.14)	(1.60, 15.27)	(1.05, 9.20)	(0.88, 7.31)
C101 (75)	(0.52, 3.35)	(0.48, 3.56)	(0.48, 4.89)	(1.19, 12.63)	(0.93, 8.71)	(0.79, 9.08)	(1.42, 16.56)	(1.05, 7.16)	(0.77, 6.64)
R101 (100)	(1.44, 10.34)	(1.92, 15.19)	(1.67, 16.57)	(3.88, 36.33)	(4.18, 42.86)	(2.76, 24.04)	(5.94, 61.33)	(3.19, 23.56)	(3.08, 24.65)
C101 (100)	(1.53, 12.40)	(1.96, 19.29)	(1.83, 32.29)	(3.78, 37.17)	(2.92, 28.96)	(2.73, 22.07)	(4.31, 52.01)	(2.92, 19.85)	(2.43, 20.77)
				High Variability					
R101 (25)	(0.03, 0.58)	(0.02, 0.47)	(0.02, 0.51)	(0.04, 0.98)	(0.03, 0.97)	(0.03, 0.94)	(0.04, 1.00)	(0.04, 1.00)	(0.03, 0.98)
C101 (25)	(0.03, 0.79)	(0.02, 0.68)	(0.02, 0.54)	(0.04, 1.00)	(0.03, 0.97)	(0.04, 1.00)	(0.04, 1.00)	(0.04, 1.00)	(0.10, 1.02)
R101 (50)	(0.16, 1.02)	(0.15, 1.02)	(0.14, 1.21)	(0.23, 1.58)	(0.18, 1.26)	(0.22, 2.09)	(0.30, 2.09)	(0.24, 2.63)	(0.16, 1.06)
C101 (50)	(0.16, 1.01)	(0.13, 1.00)	(0.11, 1.00)	(0.21, 1.33)	(0.18, 1.45)	(0.22, 2.06)	(0.28, 2.04)	(0.22, 2.21)	(0.36, 6.21)
R101 (75)	(0.63, 5.99)	(0.66, 4.51)	(0.47, 3.22)	(1.04, 8.41)	(1.02, 9.32)	(0.89, 12.51)	(1.95, 22.31)	(1.23, 9.43)	(0.91, 6.83)
C101 (75)	(0.65, 4.00)	(0.69, 7.94)	(0.58, 5.67)	(1.13, 12.28)	(1.15, 13.68)	(0.90, 6.51)	(1.68, 20.13)	(1.34, 11.59)	(0.72, 5.46)
R101 (100)	(2.33, 18.15)	(2.98, 39.71)	(2.56, 38.91)	(5.30, 51.22)	(4.25, 42.77)	(3.33, 46.88)	(6.92, 76.45)	(3.74, 35.00)	(3.07, 21.80)
C101 (100)	(2.55, 26.37)	(2.73, 28.51)	(1.92, 17.06)	(3.61, 35.57)	(3.74, 43.15)	(2.29, 19.89)	(4.53, 57.45)	(16.44, 574.99)	(2.35, 19.40)

pre-decision rollout policy explicitly considers capacity replenishment actions prior to route failures. As a result, our pre-decision rollout policy is able to consider the full set of feasible actions from a pre-decision state via only one application of the fixed route heuristic, thereby yielding shorter per epoch computing times than the hybrid rollout policy of §5. Evaluation of restocking policies poses a computational challenge, which we address by exploiting structural properties of a dynamic program. Our computational experiments indicate that our pre-decision rollout policy can significantly improve upon benchmark restocking policies, a method frequently implemented in practice. Compared to the hybrid rollout policy of §5.3, our pre-decision rollout policy tends to perform better when vehicle capacity is small, sometimes collecting as much as five percent more demand than the hybrid rollout policy. However, we observe some problem instances where the hybrid rollout policy performs better than the pre-decision rollout policy. In these instances, the difference in demand collected is small, almost never exceeding one percent. Overall, the performance of the pre-decision and hybrid rollout policies is similar. Because pre-decision rollout policies typically require less computing time to select an action than hybrid policies, they may offer an advantage in circumstances where real-time decision making is important.

One avenue for future research is to develop a restocking-based hybrid rollout policy for a VRPSDL that includes *waiting* actions. At a given decision epoch, instead of requiring vehicles to travel to the depot or to customers with pending or unknown demand, vehicles may also wait at their current location until the next decision epoch.

This modification leads to an expansion of the feasible action set developed in §2.2.2. When waiting actions are considered, the set of actions available in state s_k becomes

$$\begin{aligned}
\mathcal{A}(s_k) = \{\, a \in \mathcal{N}^m : \\
& a_i = l_i \,\forall\, i \in \{\mathcal{M} \setminus \mathcal{M}'\}, && (6.9) \\
& a_i = 0 \,\forall\, \{i \in \mathcal{M}' : q_i \le d_{l_i}, l_i \ne 0\}, && (6.10) \\
& a_i \ne a_j \,\forall\, \{i, j \in \mathcal{M} : i \ne j, a_i \ne 0, a_j \ne 0\}, && (6.11) \\
& a_i \notin \{j \in \{\cup_{h \in \mathcal{M}'} l_h \setminus \{0 \cup l_i\}\} : d_j \le q_{\text{veh}(j)}\}, && (6.12) \\
& a_i \notin \{j \in \{\mathcal{N} \setminus \{0 \cup l_i\}\} : d_j = 0\}, && (6.13) \\
& a_i \notin \{j \in \mathcal{N} : T_k + t(l_i, j) + t(j, 0) > L\}, && (6.14) \\
& a_i \ne l_i \,\forall\, \{i \in \mathcal{M}' : T_{k+1} + t(l_i, 0) > L\}\}. && (6.15)
\end{aligned}$$

Condition (6.9) requires that vehicles en route continue to their current destination. Condition (6.10) requires vehicles in $\mathcal{M}'$ to return to the depot if capacity will be depleted by serving customer demands at current locations. Condition (6.11) prevents assignment of multiple vehicles to customer locations, except in the case of assignments to the depot. Except for a vehicle's current location, (6.12) and (6.13) disallow assignment of vehicles to locations with zero demand (veh(j) denotes the vehicle at location j). Condition (6.14) prohibits the assignment of vehicles to locations (other than the current location) that will result in violations of the route duration limit. Condition (6.15) does not allow vehicles in $\mathcal{M}'$ to wait at their current location if doing so will result in a violation of the route duration limit; note that the feasibility of a vehicle waiting at a particular location is dependent on T_{k+1}, the time of decision epoch $k+1$, which is determined by the actions taken by the other vehicles.

When waiting actions are considered, the transition to the post-decision state, originally stated in §2.2.3, must be modified. Arrival times are set to

$$t_i^a = \begin{cases} t_i + t(l_i, l_i^a), & \forall\, i \in \{j \in \mathcal{M}' : l_j \neq l_j^a\} \\ t_i, & \forall\, i \in \{\mathcal{M} \setminus \mathcal{M}'\} \end{cases}, \tag{6.16}$$

and then for all $i \in \{g \in \mathcal{M}' : l_g = l_g^a\}$, we set

$$t_i^a = \min_{\substack{j \in \{\{\mathcal{M} \setminus \mathcal{M}'\} \cup \\ \{h \in \mathcal{M}' : l_h \neq l_h^a\}\}}} t_j^a, \tag{6.17}$$

in order to correctly calculate arrival times for vehicles waiting at their current locations.

Using the restocking-based fixed routes heuristic $\tilde{\mathcal{H}}(\cdot)$ in a hybrid rollout policy for a VRPSDL with waiting actions is appealing because applying $\tilde{\mathcal{H}}(\cdot)$ in the pre-decision state implicitly evaluates all actions in $\mathcal{A}(\cdot)$ except actions that allow vehicles to wait at their current locations. Heuristic $\tilde{\mathcal{H}}(\cdot)$ can then be applied, from the post-decision state, to each action allowing vehicles to wait. Because inclusion of waiting actions relaxes the Markov decision process formulation of §2.2, we anticipate that policies can be devised to collect more demand than when waiting actions are ignored.

CHAPTER 7
PARAMETRIC APPROXIMATE DYNAMIC PROGRAMMING FOR THE VEHICLE ROUTING PROBLEM WITH STOCHASTIC DEMAND AND DURATION LIMITS

7.1 Introduction

In this chapter, we present the results of our first steps toward a more traditional approximate dynamic programming (ADP) procedure to obtain dynamic solutions for the vehicle routing problem with stochastic demand and duration limits (VRPSDL). Unlike the rollout procedures of §5 and §6, which approximate the value functions (Bellman, 1957) via fixed route policies, we consider a parameterized linear function to approximate the value functions. Our linear value function approximation exploits key information in the state variable, such as the expected demand in a particular geographic region or the vehicle capacity allocated to a given set of customers. We estimate the parameters in our linear value function approximation via a simulation-based algorithm advocated by Powell (2007). The simulation procedure is analogous to estimating the parameters in a linear regression.

Three observations motivate consideration of a parametric functional form to approximate the value functions. First, when evaluating actions, a parametric functional form has the potential to capture future interactions among vehicle routes. The rollout procedures of §5 and §6 evaluate actions via fixed route policies, which, by definition, do not consider future interactions among vehicle routes. Second, the simulation procedure we employ to estimate parameters can be executed offline. After determining an appropriate set of parameters, it is not time consuming to select actions in real time, even for large-scale problems. Third, the body of literature on traditional ADP procedures for vehicle routing problems is small. This work will further the comparison between parametric ADP procedures and rollout procedures for vehicle routing problems.

This chapter is organized as follows. In §7.2, we review related literature. We describe our value function approximation in §7.3. In §7.4, we provide the results of computational experiments. We conclude with remarks and directions for future research in §7.5.

7.2 Related Literature

In this section, we review literature that applies ADP techniques to dynamic and stochastic vehicle routing and related problems. Often, ADP methods restrict the value function to lie within a specified class of functions and then seek to find the optimal value function in this class. Challenges associated with these methods include determining the best class of functions to use for a given problem and determining the optimal approximation within a chosen class. As of this writing, the choice of functional class remains as much an art as a science. However, determining the optimal approximation within a chosen class is typically done analytically or via simulation.

Analytical approaches build on the work of Schweitzer and Seidman (1985), with more recent work by Adelman (2004, 2005) and de Farias and Van Roy (2003, 2006, 2004). A typical analytical solution method proceeds as follows. First, transform the Markov decision process (MDP) into its equivalent linear program (Puterman, 1994). Second, approximate the value functions by assuming a specific parameterized form. This step typically involves the identification of *basis functions* that simplify the state representation. For example, in the VRPSDL, a basis function might represent the state as the number of vehicles in given geographic regions. Third, use the chosen approximation in the linear program to create the approximate linear program. In the approximate linear program, the decision variables are the parameters, or weights, associated with each basis function. Fourth, solve the approximate linear program to obtain the optimal set of parameters. Finally, use this optimal linear value function approximation to select actions for any visited states. The difficulty with analytical approaches is solving the approximate linear program, which may be quite large despite the reduction in the state space afforded by basis functions. An example of this methodology applied to a stochastic inventory routing problem is Adelman (2004), who approximates the value function as the sum of single-customer value functions.

Simulation-based approaches (Powell, 2007) generate sample paths of the problem and seek to update the parameters that determine the chosen class of functions in an iterative fashion. Similar to analytical methods, these methods typically employ basis functions to simplify the state space. Simulation-based methods suffer from the fact that not only is the true value function approximated, but a further source of approximation is introduced through sampling error. Despite these limitations, simulation-based methods can be very effective, especially when the general shape of the value functions is known. For example, Topaloglu and Powell (2006) consider time-staged integer multicommodity-flow problems and approximate the value function by a piecewise-linear, concave function. The approach approximates the true shape of the value functions and enables the successful management of large-scale business-jet operations. Maxwell et al. (2010) develop linear value function approximations for ambulance redeployment. The basis functions they consider are based on insights from static deployment policies. Parameters are tuned via simulation and the authors demonstrate practically significant improvements in performance relative to benchmark static policies. Meisel et al. (2009) construct linear value function approximations for a dynamic and stochastic routing problem where a single, uncapacitated vehicle serves customer requests that arrive randomly over a given time horizon. The authors tune their approximations via simulation and demonstrate improvement over state-of-the-art heuristics.

Although analytical and simulation-based methods are the foundation of mainstream ADP techniques, other ADP procedures have also proven successful for routing problems. Kleywegt et al. (2004) decompose a stochastic inventory routing problem into tractable MDP subproblems by partitioning customers to be served into small groups. Parameters for each MDP subproblem, determined via simulation, are set to mimic the behavior of the original MDP. Such an approach alleviates the curses of dimensionality via decomposition, but transfers these difficulties to determining appropriate parameters for each subproblem.

Hvattum and Løkketangen (2009) and Hvattum et al. (2009) simplify the state space of a stochastic inventory routing problem by considering only a subset of random events, which they refer to as a *scenario tree.* At each decision epoch, actions are selected by solving the associated *scenario tree problem*, which is a math programming representation of the state-reduced MDP from the current state forward. The potential advantage of this approach is that it does not impose a functional form on the cost-to-go, thereby making it possible to obtain better approximations.

Similar to scenario trees, Hvattum et al. (2006, 2007) apply the concept of *scenario-based planning* to a multi-vehicle routing problem with stochastic demand. Scenario-based planning heuristically selects actions based on a subset of random events, called scenarios. Each scenario is a random sample of stochastic parameters in the problem. For each scenario, the sample-based planning method typically solves a deterministic problem induced by the scenario. From the corresponding deterministic solutions, an action is extracted from a *distinguished* solution selected via a *consensus* function designed to select the solution that bears the most similarity to the other solutions. The underlying idea is that the distinguished solution will lead to the most "robust" action.

Finally, as we discuss in §4, rollout procedures offer an additional ADP framework. The reader is referred to §5.2 for a review of literature that applies rollout procedures to routing problems.

7.3 Value Function Approximation

To apply value function approximation to the VRPSDL, recall the MDP formulation in §2.2. Also, recall the state transition functions introduced in §4:

$$s_{k+1} = S^{M}(s_k, a, W_{k+1}) = S^{M,W}(S^{M,a}(s_k, a), W_{k+1}).$$

The transition from a pre-decision state s_k to a post-decision state s_k^a is accomplished via function $S^{M,a}(s_k, a)$. The transition from a post-decision state s_k^a to a pre-decision state s_{k+1} is accomplished via function $S^{M,W}(s_k^a, W_{k+1})$, where W_{k+1} represents the random customer demands first observed at decision epoch $k+1$.

The value functions formulated around the post-decision state are

$$V_{k-1}^a(s_{k-1}^a) = \mathbb{E}\left\{ \max_{a \in \mathcal{A}(s_k)} \left\{ R_k(s_k, a) + V_k^a(s_k^a) \right\} \middle| s_{k-1}^a \right\}, \tag{7.1}$$

where $V_{k-1}^a(s_{k-1}^a)$ is the optimal reward-to-go from the $(k-1)^{\text{st}}$ decision epoch when the process is in post-decision state s_{k-1}^a. The expectation is taken over the random variable $s_k = S^{M,W}(s_{k-1}^a, W_k)$. A post-decision state $s_k^a = S^{M,a}(s_k, a)$ is deterministic conditional on pre-decision state s_k and action a.

To approximate (7.1), we consider linear functions of the form

$$\widehat{V}_k(s) = \sum_{j=1}^{J} \theta_j \phi_j(s), \tag{7.2}$$

where each $\phi_j(s)$ is a basis function that maps a state $s \in \mathcal{S}$ to a real number and each θ_j is a real-valued weight, or parameter, associated with a basis function. We discuss our choice of basis functions $\phi_1(\cdot), \ldots, \phi_J(\cdot)$ in §7.3.1. In §7.3.3, we describe a simulation-based procedure to determine parameters $\theta_1, \ldots, \theta_J$. Substituting (7.2) into (7.1), the approximate value functions are

$$\widehat{V}^a_{k-1}(s^a_{k-1}) = \mathbb{E}\left\{ \max_{a \in \mathcal{A}(s_k)} \left\{ R_k(s_k, a) + \sum_{j=1}^{J} \theta_j \phi_j(s^a_k) \right\} \middle| s^a_{k-1} \right\}. \tag{7.3}$$

As we will show, the advantage of working with the approximate value functions formulated around the post-decision state is that, for a given sample realization of customer demands, (7.3) is a deterministic problem. As Powell (2007) points out, the more traditional choice of formulating the value functions around the pre-decision state requires the calculation of an expected reward-to-go, which can be computationally prohibitive. Solving deterministic problems reduces the computational burden of the simulation-based parameter estimation procedure we describe in §7.3.3.

7.3.1 Basis Functions

We now define the basis functions $\phi_1(\cdot), \ldots, \phi_J(\cdot)$, each of which maps a state $s = ((l, t, q), (d, x))$ to a real number. Our choice of basis functions is based on intuition and reflects what we believe are important elements of the state variable. Our basis functions are also similar to those of Secomandi (2000). For any state $s \in \mathcal{S}$, $\phi_1(s) = 1$, thereby allowing θ_1 to serve as a location parameter, much like the intercept in a regression model. For each vehicle in $\mathcal{M} = \{1, \ldots, m\}$, we define a basis function that returns the time until the route duration limit for that vehicle:

$$\phi_{1+j}(s) = L - t_j, \forall\, j \in \{1, \ldots, m\}, \tag{7.4}$$

where t_j is the time at which vehicle $j \in \mathcal{M}$ arrives at its current destination.

The remainder of our basis functions require the customers in $\mathcal{N}$, not including the depot, to be partitioned into disjoint sets. We denote the collection of these sets as $\mathcal{P}$. For example, if $\mathcal{N} = \{0, 1, 2, 3, 4, 5\}$, then $\mathcal{P}$ might be $\{\{1, 4\}, \{3\}, \{2, 5\}\}$. We refer to the j^{th} customer set in $\mathcal{P}$ as p_j. In §7.3.2, we describe a method to obtain $\mathcal{P}$.

For each customer set $p \in \mathcal{P}$, we define a basis function that returns the number of vehicles assigned to visit a customer in p:

$$\phi_{1+m+j}(s) = |\{l_i : i \in \mathcal{M}, l_i \in p_j\}|, \forall\, j \in \{1, \ldots, |\mathcal{P}|\}, \tag{7.5}$$

where l_i is the current destination of vehicle $i \in \mathcal{M}$.

For each customer set $p \in \mathcal{P}$, we define a basis function that returns the sum of the expected and pending demand in set p:

$$\phi_{1+m+|\mathcal{P}|+j}(s) = \sum_{i \in p_j : x_i = ?} \mathbb{E}\{x_i\} + \sum_{i \in p_j : x_i \neq ?} d_i, \forall\, j \in \{1, \ldots, |\mathcal{P}|\}, \tag{7.6}$$

where x_i is the random amount of demand at customer $i \in \mathcal{N}$ and d_i is the pending demand at customer $i \in \mathcal{N}$.

For each customer set $p \in \mathcal{P}$, we define a basis function that returns the sum of the standard deviation of demand in set p:

$$\phi_{1+m+2|\mathcal{P}|+j}(s) = \sum_{i \in p_j : x_i = ?} \sigma_{x_i}, \forall\, j \in \{1, \ldots, |\mathcal{P}|\}, \tag{7.7}$$

where σ_{x_i} is the standard deviation of demand at customer $i \in \mathcal{N}$.

For each customer set $p \in \mathcal{P}$, we define a basis function that returns the total vehicle capacity allocated to set p:

$$\phi_{1+m+3|\mathcal{P}|+j}(s) = \sum_{i:i\in\mathcal{M},l_i\in p_j} q_i, \forall\, j \in \{1,\ldots,|\mathcal{P}|\}, \tag{7.8}$$

where q_i is the available capacity in vehicle $i \in \mathcal{M}$. The total number of basis functions is $J = 1 + m + 4|\mathcal{P}|$.

7.3.2 Partitioning Method

We partition customers into sets by heuristically solving the k-means clustering problem (Lloyd, 1982), a standard problem in the machine learning community. The goal is to partition the customers in $\mathcal{N}$ into k sets $\mathcal{P} = (p_1, p_2, \ldots, p_k)$ so as to minimize the within cluster sum of squares:

$$\min_{\mathcal{P}} \sum_{i=1}^{k} \sum_{j\in p_i} g(j,\mu_i)^2, \tag{7.9}$$

where μ_i is the demand-weighted centroid of cluster p_i and we define the function $g(j,\mu_i)$ to be the Euclidian distance from customer j to μ_i multiplied by the expected/pending demand at customer j.

To solve the k-means clustering problem, we employ the iterative heuristic of Lloyd (1982), displayed in Algorithm 7.1. We execute Algorithm 7.1 1000 times and choose the clustering that yields the lowest objective value.

Algorithm 7.1 k-Means Clustering Algorithm

```
1: Randomly initialize cluster means.
2: Assign each customer to the cluster with the closest mean.
3: Calculate the new cluster means.
4: Repeat steps 2 and 3 until the objective (7.9) does not improve and return the
   clustering with the lowest objective value.
```

7.3.3 Parameter Estimation

Our simulation-based procedure for estimating the parameter vector $\theta = (\theta_1, \theta_2, \ldots \theta_J)$ is equivalent to the classical problem of estimating parameters in a linear regression, except that we perform the estimation in an iterative fashion. Specifically, each iteration of our procedure returns a set of state-value pairs. The value associated with each pair is analogous to the dependent variable in a regression. Applying the basis functions of §7.3.1 to the state associated with each pair yields the independent variables. At each iteration of our procedure, these "dependent" and "independent" variables are used to update, and hopefully improve, our current estimate of θ.

Algorithm 7.2 gives an overview of our procedure, which is termed by Powell (2007) as a *double pass* procedure because each iteration consists of a *forward* and *backward pass*. We denote the parameter vector at the i^{th} iteration of our procedure by θ^i. For a given realization of customer demands, the forward pass begins in initial state s_0 and steps forward through time using the current parameter vector to select actions and using the MDP model to make state transitions. Lines 4-8 constitute the forward pass, which continues until an absorbing state $s_K \in \mathcal{S}_K$ is reached. Then, the backward pass traverses the visited post-decision states $s_0^a, s_1^a, \ldots, s_{K-1}^a$ in reverse order to calculate the reward-to-go achieved at each post-decision state by using the current parameter vector to select actions. At the i^{th} iteration, we denote the reward-to-go from state s_k^a on as $\hat{v}_k^i$ and we define $\hat{v}_K^i = 0$ because s_K is an absorbing state. Lines 9-11 constitute the backward pass. A single iteration of the procedure concludes in line 12 by updating the parameter vector using information from the forward and backward passes. We denote the updating process by a function $U(\cdot)$, which takes as arguments the current parameter vector, the post-decision states visited during the current iteration, and the calculated rewards-to-go. We perform a total of I iterations, where each iteration consists of a forward and backward pass on a randomly generated set of customer demands.

Algorithm 7.2 Double Pass Parameter Estimation Procedure

```
1: Initialize $\theta^0$
2: for $i = 1$ to $I$ do
3:     Randomly generate demands $x^i$
4:     $k \leftarrow 0$
5:     while $s_k \notin \mathcal{S}_K$ do
6:         $\bar{a}_k \leftarrow \arg\max_{a \in \mathcal{A}(s_k)} \left\{ R_k(s_k, a) + \sum_{j \in \mathcal{J}} \theta_j^{i-1} \phi_j(s_k^a) \right\}$
7:         $s_{k+1} \leftarrow S^M(s_k, \bar{a}_k, x^i)$
8:         $k \leftarrow k + 1$
9:     while $s_k \neq s_0$ do
10:        $\hat{v}_{k-1}^i \leftarrow R_k(s_k, \bar{a}_k) + \hat{v}_k^i$
11:        $k \leftarrow k - 1$
12:    $\theta^i \leftarrow U\left(\theta^{i-1}, (s_k^a)_{k=0}^{K-1}, (\hat{v}_k^i)_{k=0}^{K-1}\right)$
```

The updating procedure $U(\cdot)$ results in a parameter vector that minimizes the squared errors of the entire history of state-value pairs observed through the current iteration. Let

$$\left(\left(\phi_j \left(s_k^a \right) \right)_{j=1}^{J}, \hat{v}_k^i \right)_{k=0}^{K-1}$$

be the set of state-value pairs that result from the i^{th} iteration of Algorithm 7.2 and let

$$\left(\left(\left(\phi_j \left(s_k^a \right) \right)_{j=1}^{J}, \hat{v}_k^i \right)_{k=0}^{K-1} \right)_{i=1}^{h}$$

be the set of state-value pairs observed during iterations 1 through $h < I$. At iteration h, the updating function $U(\cdot)$ returns the parameter vector θ^h that solves

$$\min_{\theta^h} \sum_{i=1}^{h} \sum_{k=0}^{K_i - 1} \left(\hat{v}_k - \sum_{j=1}^{J} \theta_j^h \phi_j(s_k) \right)^2, \tag{7.10}$$

where K_i denotes the final decision epoch in the i^{th} iteration. Powell (2007) describe procedures to obtain θ^h in a recursive fashion by updating θ^{h-1}. We employ these procedures because they eliminate the need to maintain the entire history of state-value pairs.

In our computational experiments, we consider two methods for initializing θ. The first method simply initializes each element of θ^0 to zero, as is common practice in the literature. The second method collects state-value pairs from the hybrid rollout policy of §5 and uses these in (7.10) to obtain θ^0.

7.4 Computational Experience

In this section, we present the results of preliminary computational experiments. The aim of our experiments is to gauge the potential effectiveness of the value function approximation strategy proposed in §7.3. To accomplish this, we compare the results of our value function approximation to the benchmark fixed route policies and the hybrid rollout policies of §5. Our experiments are conducted on problem instance A-n32-k5 (Ralphs, 2008), which consists of 31 customers and five vehicles. Vehicle capacity is small, the route duration limit is long, and variability in customer demand is high, as defined in §5.4.2.

Our first experiment examines the influence of the number of customer sets, $|\mathcal{P}|$, on the effectiveness of the value function approximation. Figures 7.1, 7.2, and 7.3 display the results of each of $I = 900$ iterations of Algorithm 7.2 when the number of customers sets, or regions, is two, five, and 11, respectively. As described in §7.3.3, we obtain the initial parameter vector by executing the hybrid rollout policy of §5 on 100 demand realization and using the resulting state-value pairs to obtain θ^0. A single data point represents the percent increase in demand served by our linear value function approximation over the benchmark fixed route policy. A negative number indicates a decrease in demand served compared to the benchmark fixed route policy, while a positive number indicates an increase. The confidence band for the demand served by the hybrid rollout policy is also displayed.

In Figures 7.1, 7.2, and 7.3, it would be ideal to observe an upward trend over the course of 900 iterations. Such a trend would indicate learning, or improvement,

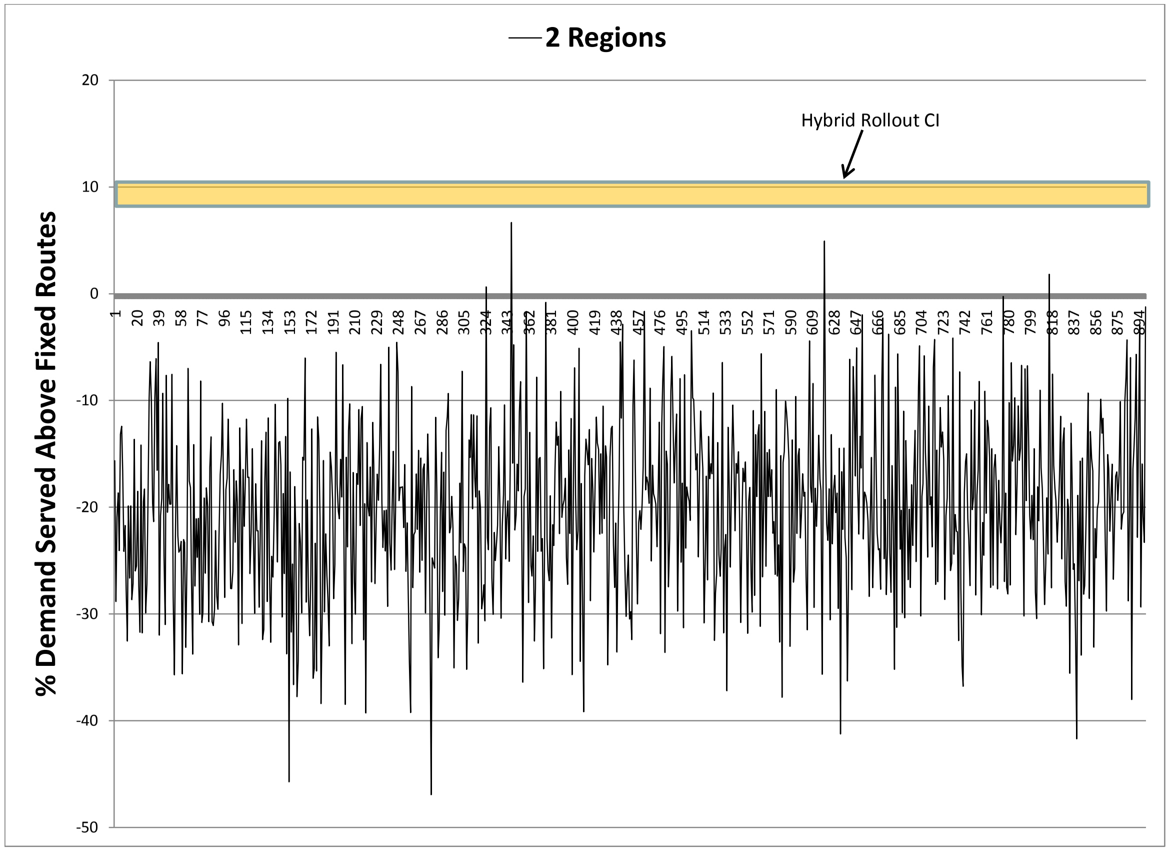

Figure 7.1: ADP Results with $|\mathcal{P}| = 2$ Regions

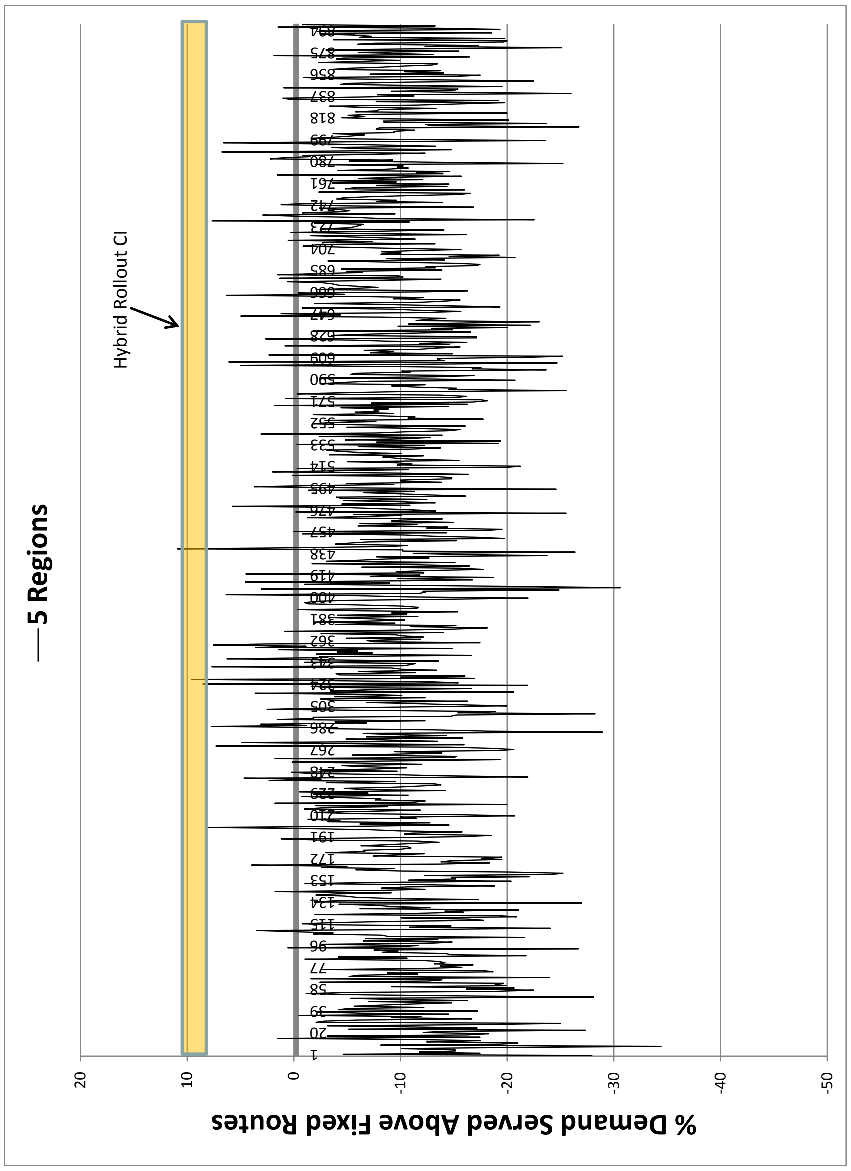

Figure 7.2: ADP Results with $|\mathcal{P}| = 5$ Regions

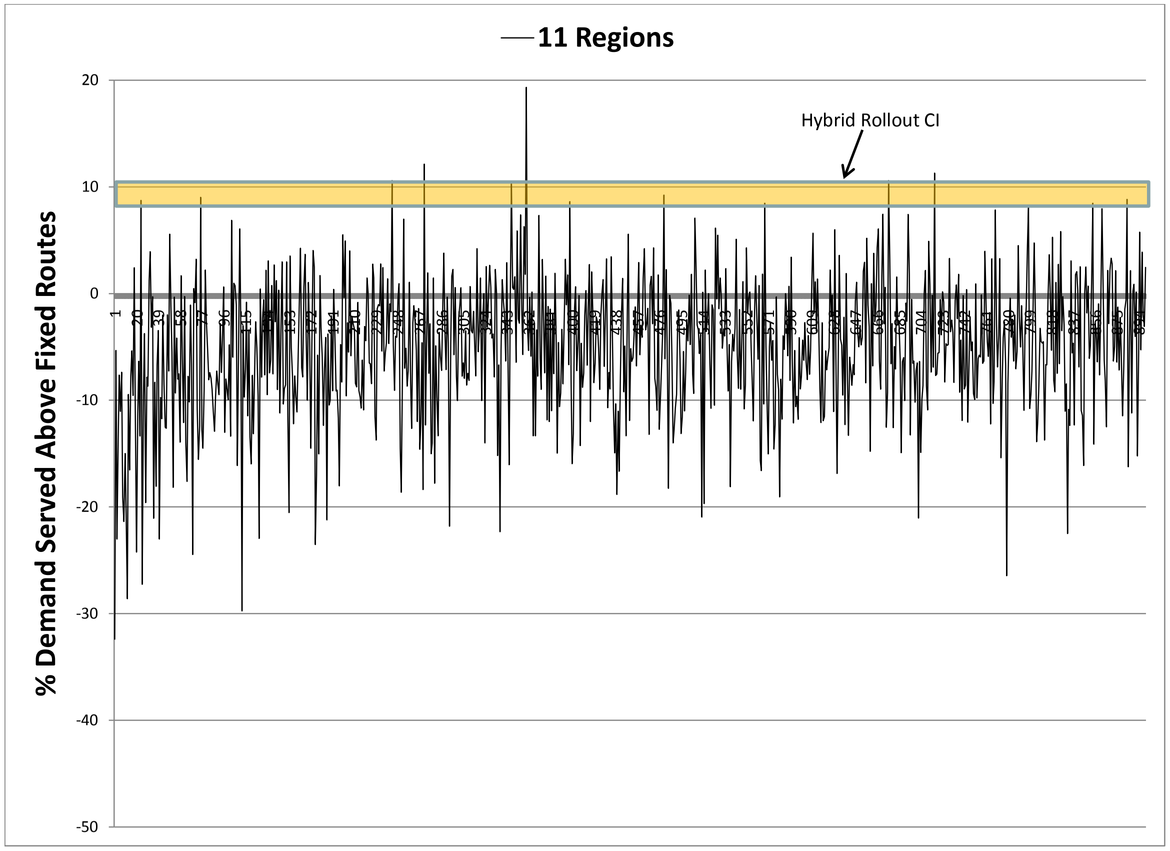

Figure 7.3: ADP Results with $|\mathcal{P}| = 11$ Regions

in the parameter vector θ. While there does appear to be somewhat of an upward trend in the first iterations of Figures 7.2 and 7.3, performance appears to plateau after iteration 100. At least two explanations exist for this behavior. First, more iterations may be required to observe a more pronounced improving trend. Second, our linear value function approximation may adequately capture the behavior of the true value functions.

Despite the lack of an improving trend, the data series corresponding to 11 regions generally performs better than when there are two or five regions. For this problem, increasing the number of regions beyond 11 results in little improvement. Moreover, our linear value function approximation is unable to consistently outperform either the benchmark fixed route policy or the hybrid rollout policy. Thus, the primary conclusion of this experiment is that additional regions are beneficial up to a point, but additional work is required to compete with the benchmark fixed route and hybrid rollout policies.

Our second experiment examines the influence of the two methods of parameter initialization we discuss in §7.3.3. A visual comparison of the two methods is displayed in Figures 7.4 and 7.5, which, for each method, show the results of $I = 900$ iterations of Algorithm 7.2. For each method, the number of customer sets is 11. As in Figures 7.1, 7.2, and 7.3, a single data point represents the percent increase in demand served by our linear value function approximation over the benchmark fixed route policy. A negative number indicates a decrease in demand served compared to the benchmark fixed route policy, while a positive number indicates an increase. The confidence band for the demand served by the hybrid rollout policy is also displayed.

While neither data series in Figures 7.4 and 7.5 contains an improving trend beyond iteration 100, it is clear that initializing parameters via the hybrid rollout policy is beneficial. However, over the course of 900 iterations, neither method of initialization leads to an approximation that consistently serves more demand than the benchmark fixed route or hybrid rollout policies.

7.5 Conclusion and Future Research

We propose a linear value function approximation in a simulation-based approximate dynamic programming algorithm to obtain dynamic solutions to the multi-vehicle routing problem with stochastic demands and duration limits. Our computational experiments make first steps in this direction, but indicate that further research is necessary. Below, we discuss two directions for future research.

The linear value function approximation we consider in this chapter is preliminary, and we believe improved results can be obtained by adjusting the approximation. As we note in §7.2, value function approximations are typically better when they leverage structural properties of the associated MDP. Although we are unable to show general structural properties for the MDP model of §2.2, intuition suggests that a non-linear structure may be advantageous. For example, it seems reasonable that the value of assigning vehicles to visit a particular region should initially increase as the number of vehicles assigned is small. However, as the number of vehicles assigned becomes larger, the value of assigning additional vehicles should eventually decrease.

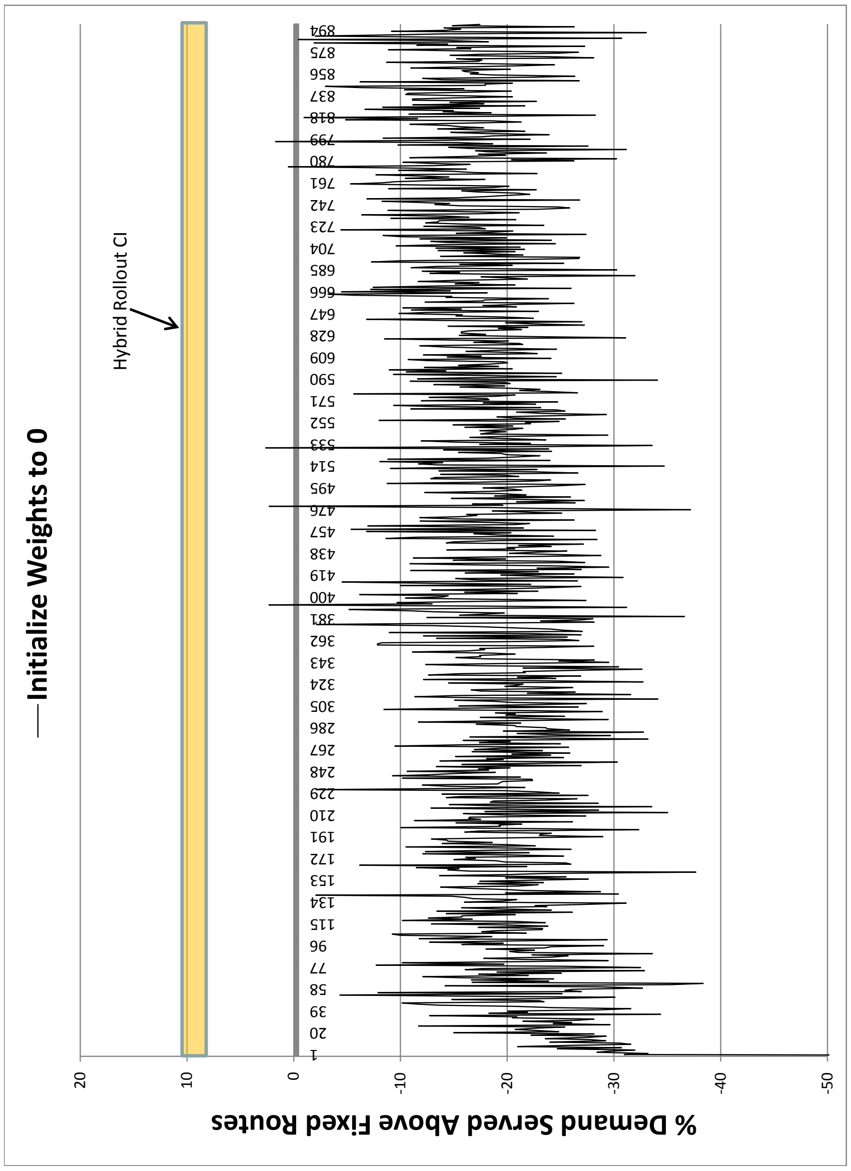

Figure 7.4: ADP Results when Parameters are Initialized to Zero

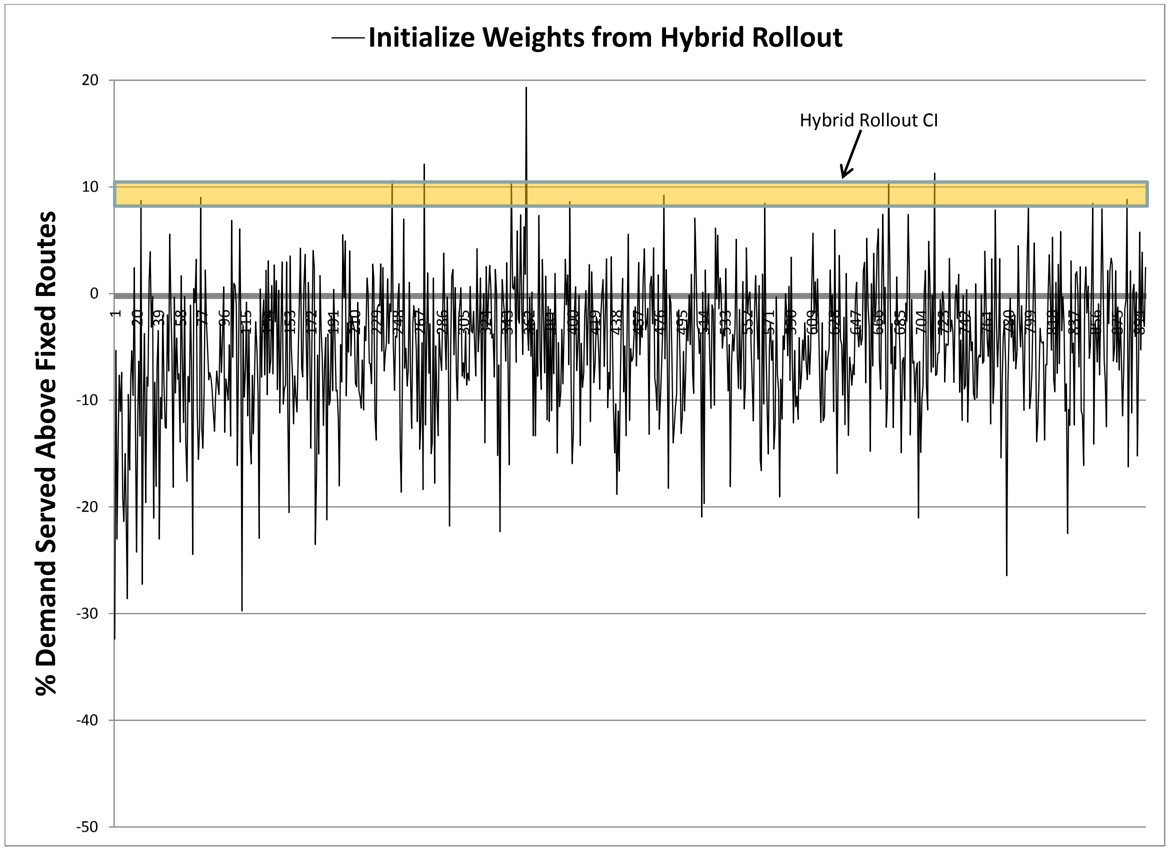

Figure 7.5: ADP Results when Parameters are Initialized via the Hybrid Rollout

This intuitive behavior can be modeled by a concave function and can be estimated via piecewise-linear functions, similar to Topaloglu and Powell (2006).

Another possibility for improving the quality of the value function approximation is to include basis functions that correspond to different levels of aggregation. For example, we might include basis functions for two, five, *and* eleven regions. The *Bias Adjusted Kalman Filter*, proposed by Powell (2007), provides a mechanism to dynamically adjust weights to account for these different levels of aggregation. In the early iterations of the procedure, higher weights are placed on higher levels of aggregation (e.g., two regions). As the algorithm progresses and more information is obtained, these weights decrease and the weights associated with lower levels of aggregation increase. Powell (2007) reports success with this method in dynamic resource allocation problems, particularly when it is not possible to leverage problem structure.

CHAPTER 8
CONCLUSION

In this thesis, we present solution methodologies to address vehicle routing problems (VRPs) with stochastic demand, an important class of problems in supply chain management. We consider two extensions of the base problem description provided in §1: the *vehicle routing problem with stochastic demand* (VRPSD) and the *vehicle routing problem with stochastic demand and duration limits* (VRPSDL). The VRPSD and the VRPSDL are fundamental problems underlying many operational challenges in the field of logistics. In this conclusion, we highlight the major contributions of each chapter and summarize directions for future research.

In §3, we develop a general methodology for solving a broad class of VRPs and demonstrate its effectiveness by obtaining high-quality fixed route policies for the VRPSD. The methodology employs a *cyclic order* solution representation for VRPs. The primary contribution of §3 is the design and implementation of neighborhoods to search cyclic orders. We demonstrate the potential of cyclic-order neighborhoods to facilitate the discovery of high quality solutions by embedding them within a simulated annealing framework to solve the classical VRP and the VRPSD. We propose an updating procedure and demonstrate its ability to reduce the computational expense of our approach. Our results underscore the potential for the use of cyclic-order neighborhoods as a general solution method for a variety of routing problems.

Due to the computational effort required to search a neighborhood of cyclic orders, one direction for future research is to investigate techniques to reduce the size of the cyclic-order search space. This could perhaps be accomplished by exploiting information pertaining to customer location. Specifically, cyclic orders may be removed from a neighborhood that would generate candidate routes unlikely to be observed in optimal solutions. For example, customers on opposite sides of the depot are unlikely to be serviced on the same route in an optimal solution. Thus, including such routes in the set of candidate routes is unnecessary. Such reductions in neighborhood size may be especially advantageous when it is necessary to evaluate all cyclic orders in a given neighborhood, such as in best-improving search, variable neighborhood search, or tabu search. How to use memory in the search procedure is another direction for future research. This might be accomplished by exploiting information contained in a history of candidate route sets, or by identifying common sequences of customers in a history of cyclic orders.

In §4, we lay the foundation for the dynamic routing policies of §5 and §6 by developing new *rollout policies* for *approximate dynamic programming* (ADP). ADP is a general solution methodology for dynamic and stochastic sequential decision problems and seeks to overcome the well-known curses of dimensionality associated with dynamic programs. Traditional rollout policies employ heuristic optimization to approximate value functions in dynamic programs, but can be difficult to obtain for problems with large action spaces. By partitioning the state transition of an Markov decision process (i.e., a stochastic dynamic program) into two parts, we introduce two extensions which we refer to as *pre-decision* and *post-decision* rollout policies. Furthermore, we discuss how these two new types of rollout policy may be used in combination, resulting in a *hybrid* rollout policy that, in our experience, achieves high quality solutions and results in a computational reduction sufficient for real-time implementation in large-scale settings. We also provide analytical results that position our extensions within the rollout literature and identify conditions under which our proposed rollout methods are equivalent to the traditional form.

In §5 and §6, we use the Markov decision process (MDP) formulation of §2 and the rollout framework of §4 to develop hybrid and pre-decision rollout policies for the VRPSDL. Similar to the literature on dynamic solution methodologies for single-vehicle VRPs with stochastic demand, a fixed route heuristic forms the basis of our rollout policy. We demonstrate the effectiveness of our methods by solving large-scale problems. By testing our procedure across a broad range of problem parameters, we empirically establish conditions under which the demand served by our rollout policies is significantly higher than the demand served by a method frequently implemented in practice. We also identify circumstances in which our rollout policies appear to offer little or no benefit compared to this benchmark. These observations can guide managerial decision making regarding when the use of our procedures is justifiable. Finally, we demonstrate that our methodology lends itself to real-time implementation, thereby providing a mechanism to make high-quality, dynamic routing decisions for large-scale operations.

One avenue for future research related to §6 is to develop a restocking-based hybrid rollout policy for a VRPSDL that includes *waiting* actions. At a given decision epoch, instead of requiring vehicles to travel to the depot or to customers with pending or unknown demand, vehicles may also wait at their current location until the next decision epoch. Using a restocking-based fixed routes heuristic in a hybrid rollout policy for a VRPSDL with waiting actions is appealing because applying executing the heuristic from the pre-decision state implicitly evaluates all feasible actions except those that allow vehicles to wait at their current locations. The fixed route heuristic can then be applied, from the post-decision state, to each action allowing vehicles to wait. Because inclusion of waiting actions relaxes the MDP formulation of §2.2, we anticipate that policies can be devised to collect more demand than when waiting actions are ignored.

A potential drawback of using the fixed route-based rollout policies of §5 and §6 to approximate future demand served is that the approximation does not account for possible interactions among vehicle routes. This is because fixed route policies restrict vehicles to disjoint sets of customers. Accounting for these interactions may lead to improved policies. A method that may account for this shortcoming is to use more sophisticated fixed route policies, such as those suggested by Ak and Erera (2007), which coordinate vehicle routes in pairs. However, incorporating these more complex recourse actions may lead to increased computing times, thereby making it difficult to implement such methods in real time.

Another method that has the potential to account for interactions among vehicle routes when approximating future demand served is the parametric functional approximation we consider in §7. The method we employ in this chapter is more typical of traditional ADP. We develop a parameterized linear function to approximate the value functions and estimate the parameters via a simulation-based algorithm. The computational results we present indicate that initializing parameter values via the rollout methods of §5 and §6 leads to significant improvements. However, we conclude that additional research is required to develop a parametric ADP methodology comparable or superior to the rollout policies of §5 and §6.

Value function approximations are typically better when they leverage structural properties of the associated MDP. Although we are unable to show general structural properties for the MDP model of §2.2, intuition suggests that a non-linear structure may be advantageous. For example, it seems reasonable that the value of assigning vehicles to visit a particular region should initially increase as the number of vehicles assigned is small. However, as the number of vehicles assigned becomes larger, the value of assigning additional vehicles should eventually decrease. This intuitive behavior can be modeled by a concave function and can be estimated via piecewise-linear functions, similar to Topaloglu and Powell (2006).

Another possibility for improving the quality of the value function approximation is to include basis functions that correspond to different levels of aggregation. For example, we might include basis functions for two, five, *and* eleven regions. The *Bias Adjusted Kalman Filter*, proposed by Powell (2007), provides a mechanism to dynamically adjust weights to account for these different levels of aggregation. In the early iterations of the procedure, higher weights are placed on higher levels of aggregation (e.g., two regions). As the algorithm progresses and more information is obtained, these weights decrease and the weights associated with lower levels of aggregation increase. Powell (2007) reports success with this method in dynamic resource allocation problems, particularly when it is not possible to leverage problem structure.

In this thesis, we consider objectives that optimize expected values, e.g., minimize expected travel costs or maximize expected demand served. Another avenue for future research is to incorporate some measure of risk into the objective, much like the ideas underlying "robust" optimization. Such objectives have many practical applications and may lead to different route structures.

REFERENCES

Adelman, D. (2004). A price-directed approach to stochastic inventory/routing. *Operations Research 52*(4), 499–514.

Adelman, D. (2005). Dynamic bid-prices in revenue management. *Operations Research 55*, 647–661.

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

ATA Economics & Statistical Analysis Department (1999). Standard trucking and transportation statistics. Technical report, American Trucking Association, Alexandria, VA.

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

Balaprakash, P., M. Birattari, T. Stützle, and M. Dorigo (2010). Estimation-based metaheuristics for the probabilistic traveling salesman problem. *Computers and Operations Research 37*(11), 1939–1951.

Bellman, R. (1957). *Dynamic Programming.* Princeton, NJ: Princeton University Press.

Bent, R. and P. Van Hentenryck (2004). A two-stage hybrid local search for the vehicle routing problem with time windows. *Transportation Science 38*(4), 515–530.

Bertsekas, D. (2000). *Dynamic programming and optimal control* (2nd ed.), Volume I. Belmont, MA: Athena Scientific.

Bertsekas, D., J. Tsitsiklis, and C. Wu (1997). Rollout algorithms for combinatorial optimization. *Journal of Heuristics 3*, 245–262.

Bertsimas, D., P. Chervi, and M. Peterson (1995). Computational approaches to stochastic vehicle routing problems. *Transportation Science 29*(4), 342–352.

Bertsimas, D. and I. Popescu (2003). Revenue management in a dynamic network environment. *Transportation Science 37*(3), 257–277.

Bianchi, L., M. Birattari, M. Chiarandini, M. Manfrin, M. Mastrolilli, L. Paquete, O. Rossi-doria, and T. Schiavinotto (2006). Hybrid metaheuristics for the vehicle routing problem with stochastic demands. *Journal of Mathematical Modelling and Algorithms 5*, 91–110.

Birattari, M., P. Balaprakash, T. Stützle, and M. Dorigo (2008). Estimation-based local search for stochastic combinatorial optimization using delta evaluations: a case study on the probabilistic traveling salesman problem. *INFORMS Journal on Computing 20*(4), 644–658.

Boctor, F. and J. Renaud (2000). The column-circular, subsets-selection problem: complexity and solutions. *Computers and Operations Research 27*, 383–398.

Brusco, M., L. Jacobs, and G. Thompson (1999). A morphing procedure to supplement a simulated annealing heuristic for cost- and coverage-correlated weighted set covering problems. *Annals of Operations Research 86*, 611–627.

Burke, E., G. Kendall, and G. Whitwell (2009). A simulated annealing enhancement of the best-fit heuristic for the orthogonal stock-cutting problem. *INFORMS Journal on Computing 21*(3), 505–516.

Campbell, A. and B. Thomas (2008). Challenges and advances in a priori routing. In B. Golden, S. Raghavan, and E. Wasil (Eds.), *The vehicle routing problem: latest advances and new challenges*. New York, NY: Springer.

Chepuri, K. and T. Homem-de Mello (2005). Solving the vehicle routing problem with stochastic demands using the cross-entropy method. *Annals of Operations Research 134*, 153–181.

Christiansen, C. and J. Lysgaard (2007). A branch-and-price algorithm for the capacitated vehicle routing problem with stochastic demands. *Operations Research Letters 35*, 773–781.

Christofides, N., A. Mingozzi, and P. Toth (1979). The vehicle routing problem. In N. Christofides, A. Mingozzi, P. Toth, and C. Sandi (Eds.), *Combinatorial Optimization*, pp. 315–338. Chichester: Wiley.

de Farias, D. and B. Van Roy (2003). The linear programming approach to approximate dynamic programming. *Operations Research 51*(6), 850–865.

de Farias, D. and B. Van Roy (2004). On constraint sampling in the linear programming approach to approximate dynamic programming. *Mathetmatics of Operations Research 29*(3), 462–478.

de Farias, D. and B. Van Roy (2006). A cost-shaping linear program for average-cost approximate dynamic programming with performance guarantees. *Mathematics of Operations Research 31*(3), 597–620.

Denardo, E. (2003). *Dynamic programming: models and applications*. Mineola, NY: Dover Publications.

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

Erera, A., M. Savelsbergh, and E. Uyar (2010). Fixed routes with backup vehicles for stochastic vehicle routing problems with time constraints. Forthcoming in *Networks*. doi: 10.1002/net.20338.

Fan, J., X. Wang, and H. Ning (2006). A multiple vehicles routing problem algorithm with stochastic demand. In *Proceedings of the 6th World Congress on Intelligent Control and Automation*, Volume 1, pp. 1688–1692.

Foster, B. and D. Ryan (1976). An integer programming approach to the vehicle scheduling problem. *Operational Research Quarterly 27*, 367–384.

Gendreau, M., G. Laporte, and R. Séguin (1995). An exact algorithm for the vehicle routing problem with stochastic demands and customers. *Transportation Science 29*(2), 143–155.

Gendreau, M., G. Laporte, and R. Séguin (1996). A tabu search heuristic for the vehicle routing problem with stochastic demands and customers. *Operations Research 44*(3), 469–477.

Gendreau, M., J. Potvin, O. Bräsy, G. Hasle, and A. Løkketangen (2008). Metaheuristics for the vehicle routing problem and its extensions: a categorized bibliography. In B. Golden, S. Raghavan, and E. Wasil (Eds.), *The vehicle routing problem: latest advances and challenges*, Volume 43 of *Operations Research/Computer Science Interfaces*. New York, NY: Springer.

Gillett, B. and L. Miller (1974). A heuristic algorithm for the vehicle-dispatch problem. *Operations Research 22*(2), 340–349.

Guerriero, F., M. Mancini, and R. Musmanno (2002). New rollout algorithms for combinatorial optimization problems. *Optimization methods and software 17*(4), 627–654.

Hjorring, C. and J. Holt (1999). New optimality cuts for a single-vehicle stochastic routing problem. *Annals of Operations Research 86*, 569–584.

Hvattum, L. and A. Løkketangen (2009). Using scenario trees and progressive hedging for stochastic inventory routing problems. *Journal of Heuristics 15*(6), 527–557.

Hvattum, L., A. Løkketangen, and G. Laporte (2006). Solving a dynamic and stochastic vehicle routing problem with a sample scenario hedging heuristic. *Transportation Science 40*(4), 421–438.

Hvattum, L., A. Løkketangen, and G. Laporte (2007). A branch-and-regret heuristic for stochastic and dynamic vehicle routing problems. *Networks 10*, 330–340.

Hvattum, L., A. Løkketangen, and G. Laporte (2009). Scenario tree-based heuristics for stochastic inventory-routing problems. *INFORMS Journal on Computing 21*(2), 268–285.

Johnson, D., C. Aragon, L. McGeoch, and C. Schevon (1989). Optimization by simulated annealing: an experimental evaluation; part I, graph partitioning. *Operations Research 37*(6), 865–892.

Johnson, D., C. Aragon, L. McGeoch, and C. Schevon (1991). Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning. *Operations Research 39*(3), 378–406.

Kindervater, G. and M. Savelsbergh (2003). Vehicle routing: handling edge exchanges. In E. Aarts and L. J. (Eds.), *Local search in combinatorial optimization*. Princeton, NJ: Princeton University Press.

Kirkpatrick, S., J. Gelatt, C., and M. Vecchi (1983). Optimization by simulated annealing. *Science 220*, 671–680.

Kleywegt, A., V. Nori, and M. Savelsbergh (2004). Dynamic programming approximations for a stochastic inventory routing problem. *Transportation Science 38*(1), 42–70.

Laporte, G., F. Louveaux, and L. Van Hamme (2002). An integer L-shaped algorithm for the capacitated vehicle routing problem with stochastic demands. *Operations Research 50*(3), 415–423.

Lloyd, S. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory 28*(2), 129–137.

Maxwell, M., M. Restrepo, S. Henderson, and H. Topaloglu (2010). Approximate dynamic programming for ambulance redeployment. *INFORMS Journal on Computing 22*(2), 266–281.

Meisel, S., U. Suppa, and D. Mattfeld (2009). GRASP based approximate dynamic programming for dynamic routing of a vehicle. In *MIC 2009: The VII Metaheuristics International Conference*.

Nagata, Y. and O. Bräysy (2008). Efficient local search limitation strategies for vehicle routing problems. In J. van Hemert and C. Cotta (Eds.), *Evolutionary Computation in Combinatorial Optimization*, pp. 48–60. Heidelberg: Springer Berlin.

Novoa, C., R. Berger, J. Linderoth, and R. Storer (2006). A set-partitioning-based model for the stochastic vehicle routing problem. Technical Report 06T-008, Department of Engineering and Technology Texas State University.

Novoa, C. and R. Storer (2008). An approximate dynamic programming approach for the vehicle routing problem with stochastic demands. *European Journal of Operational Research 196*(2), 509–515.

Patrick, J., M. Puterman, and M. Queyranne (2008). Dynamic multipriority patient scheduling. *Operations Research 56*(6), 1507–1525.

Powell, W. (2007). *Approximate Dynamic Programming.* New York, NY: John Wiley and Sons.

Powell, W. (2010a). The languages of stochastic optimization. *INFORMS Journal on Computing 22*(1), 23–25.

Powell, W. (2010b). Merging AI and OR to solve high-dimensional stochastic optimization problems using approximate dynamic programming. *INFORMS Journal on Computing 22*(1), 2–17.

Proper, S. and P. Tadepalli (2006). Scaling model-based average-reward reinforcement learning for product delivery. In J. Fürnkranz, T. Scheffer, and M. Spiliopoulou (Eds.), *Machine Learning: ECML 2006*, Volume 4212/2006 of *Lecture Notes in Computer Science*, pp. 735–742. Berlin: Springer.

Puterman, M. (1994). *Markov decision processes: discrete stochastic dynamic programming.* New York, NY: Wiley.

Ralphs, T. (2008). Vehicle routing data sets. http://www.branchandcut.org/. Accessed on October 6, 2008.

Renaud, J. and F. Boctor (2002). A sweep-based algorithm for the fleet size and mix vehicle routing problem. *European Journal of Operational Research 140*, 618–628.

Renaud, J., F. Boctor, and G. Laporte (1996). An improved petal heuristic for the vehicle routeing problem. *Journal of the Operational Research Society 47*, 329–336.

Rochat, Y. and E. Taillard (1995). Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics 1*, 147–167.

Ruszczyński, A. (2010). Post-decision states and separable approximations are powerful tools of approximate dynamic programming. *INFORMS Journal on Computing 22*(1), 20–22.

Ryan, D., C. Hjorring, and F. Glover (1993). Extensions of the petal method for vehicle routeing. *Journal of the Operational Research Society 44*(3), 289–296.

Savelsbergh, M. and M. Goetschalchx (1995). A comparison of the efficiency of fixed versus variable vehicle routes. *Journal of Business Logistics 16*(1), 163–187.

Schweitzer, P. and A. Seidman (1985). Generalized polynomial approximation in markovian decision processes. *Journal of Mathematical Analysis and Applications 110*, 568–582.

Secomandi, N. (2000). Comparing neuro-dynamic programming algorithms for the vehicle routing problem with stochastic demands. *Computers and Operations Research 27*, 1201–1225.

Secomandi, N. (2001). A rollout policy for the vehicle routing problem with stochastic demands. *Operations Research 49*(5), 796–802.

Secomandi, N. (2003). Analysis of a rollout approach to sequencing problems with stochastic routing applications. *Journal of Heuristics 9*, 321–352.

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

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

Tatarakis, A. and I. Minis (2009). Stochastic vehicle routing with a predefined customer sequence and multiple depot returns. *European Journal of Operational Research 197*, 557–571.

Teodorović, D. and G. Pavković (1992). A simulated annealing technique approach to the vehicle routing problem in the case of stochastic demand. *Transportation Planning and Technology 16*, 261–273.

Thompson, P. and H. Psaraftis (1993). Cyclic transfer algorithms for multi-vehicle routing and scheduling problems. *Operations Research 41*, 935–946.

Topaloglu, H. and W. Powell (2006). Dynamic-programming approximations for stochastic time-staged integer multicommodity-flow problems. *INFORMS Journal on Computing 18*(1), 31–42.

Tsirimpas, P., A. Tatarakis, I. Minis, and E. Kyriakidis (2008). Single vehicle routing with a predefined customer sequence and multiple depot returns. *European Journal of Operational Research 187*, 483–495.

Tsitsiklis, J. (2010). Perspectives on stochastic optimization over time. *INFORMS Journal on Computing 22*(1), 18–19.

van de Klundert, J. and L. Wormer (2010). ASAP: the after-salesman problem. Forthcoming in *Manufacturing and Service Operations Management*. doi: 10.1287/msom.1100.0292.

Yang, W., K. Mathur, and R. Ballou (2000). Stochastic vehicle routing problem with restocking. *Transportation Science 34*(1), 99–112.