Cyclic-Order Neighborhoods with Application to the Vehicle Routing Problem with Stochastic Demand

Justin C. Goodson Jeffrey W. Ohlmann Barrett W. Thomas

Abstract

We examine neighborhood structures for heuristic search applicable to a general class of vehicle routing problems (VRPs). Our methodology utilizes a cyclic-order solution encoding, which maps a permutation of the customer set to a collection of many possible VRP solutions. We identify the best VRP solution in this collection via a polynomial-time algorithm from the literature. We design neighborhoods to search the space of cyclic orders. Utilizing a simulated annealing framework, we demonstrate the potential of cyclic-order neighborhoods to facilitate the discovery of high quality a priori solutions for the vehicle routing problem with stochastic demand (VRPSD). Without tailoring our solution procedure to this specific routing problem, we are able to match 16 of 19 known optimal VRPSD solutions. We also propose an updating procedure to evaluate the neighbors of a current solution and demonstrate its ability to reduce the computational expense of our approach.

Keywords: heuristics, logistics, cyclic-order neighborhoods, stochastic vehicle routing, simulated annealing

1 Introduction

We examine neighborhood structures for heuristic search applicable to a general class of vehicle routing problems (VRPs). The class of problems includes VRPs in which a homogenous fleet of 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 a cost function separable in the routes composing x; and (iii) any constraints pertain to restrictions on individual routes rather than collections of routes. Within this class, we focus on obtaining a priori solutions to the VRP with stochastic demand (VRPSD), an important and challenging problem in the field of logistics (see Campbell and Thomas 2008 for a review of recent advances in a priori routing).

The neighborhood structures we develop utilize the *cyclic-order* solution encoding of Ryan et al. (1993). A cyclic order is a permutation, or ordering, of the set of customers. Orders are called cyclic because our algorithmic framework treats them as circular, e.g., when advancing through a cyclic order, iterating beyond the last customer in the order advances to the first customer. As Ryan et al. (1993) show, by iteratively sweeping across the permutation of customers to generate a set of candidate routes, a cyclic-order encoding corresponds to a very large number of classical VRP solutions from which the best solution can be identified via a polynomial-time algorithm. Further, Ryan et al. (1993) show that there exists a family of cyclic orders that corresponds to an optimal solution for the classical VRP. Our work is motivated by these results, recognizing that effective means to obtain a cyclic order within this family has not been explored in the literature. Further, our work recognizes that cyclic orders apply to a broader class of VRPs.

We make two contributions. First, we design local search neighborhoods for the cyclic-order representation of a general class of VRPs, leveraging Ryan et al. (1993) to map cyclic orders to VRP solutions. Utilizing a simulated annealing framework, we demonstrate the potential of cyclic-order neighborhoods to facilitate the discovery of high quality a priori VRPSD solutions, matching 16 of 19 known optimal solutions. As our search procedure is designed for the general class of VRPs described above, and not tailored to the VRPSD, we find these results to be promising. Second, for each neighborhood, we propose an updating procedure to evaluate the neighbors of a current solution. Our experiments with VRPSD instances show substantial computational savings result from employing the updating procedure. To the best of the authors' knowledge, this is the first work to examine local search for VRPs using the cyclic-order representation.

Throughout the paper, we provide illustrative examples of our methodology applied to the classical VRP. We utilize the classical VRP for simplicity and to emphasize the general nature of our methodology. To transfer the logic in our examples to the VRPSD or to any other problem within the stated class, two adjustments may be required. First, the function $f(\cdot)$ must reflect the cost of a routing plan for the chosen VRP. In §7.1, we demonstrate how to calculate the expected cost of an a priori VRPSD route. For some stochastic VRPs, where the expected cost of a routing plan may be difficult to calculate, $f(\cdot)$ may be estimated via Monte Carlo simulation. Second, constraints specific to the chosen VRP might have to be incorporated when generating candidate routes. In the VRPSD formulation we adopt, the constraints for a priori routes mirror those of the classical VRP, therefore our examples transfer directly.

In §2, we review literature related to the VRPSD. In §3, we formally state the VRPSD. In §4, we provide a detailed explanation of the cyclic-order solution representation. We define neighborhoods to search cyclic orders in §5. In §6, we describe methods to update the set of candidate routes in local search schemes. In §7, we show how cyclic-order neighborhoods can be embedded in a simulated annealing procedure to solve the VRPSD. We summarize the paper and suggest directions for future research in §8.

2 Related Literature

Compared to the classical VRP, the VRPSD has received relatively little attention in the literature despite its applicability to many distribution systems, e.g., vendor-managed inventory networks. Much of the VRPSD literature focuses on *a priori*, or *fixed route* solutions. The a priori approach, suggested initially by Tillman (1969) and formalized by Dror et al. (1989) and Bertsimas (1992), requires a vehicle to visit a predetermined sequence of customers, returning to the depot as necessary to replenish capacity. Compared to more dynamic routing schemes, a priori solutions are easier to compute and offer the managerial benefit of regular associations between customers and operators.

The a priori VRPSD formulation we adopt is that of Laporte et al. (2002) and Christiansen and Lysgaard (2007). Their formulation requires a set of a priori vehicle routes with minimum expected cost, beginning and ending at the depot, such that each customer is assigned to exactly one route and the expected demand assigned to each route does not exceed vehicle capacity. Vehicles are required to continue on their assigned routes until capacity is depleted (i.e., a *route failure* occurs), in which case capacity is replenished at the depot and service continues at the location of the route failure (this is in contrast to the more dynamic policy of Yang et al. (2000), which allows preemptive capacity replenishment). Under the assumption that customer demands are independent, calculation of the expected cost of an a priori routing plan is separable in the routes (Teodorović and Pavković 1992). Thus, the formulation we adopt for the VRPSD belongs to the class of problems described in §1.

Exact algorithms for obtaining a priori VRPSD solutions 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 obtaining a priori VRPSD solutions include the methods of Gendreau et al. (1996), Yang et al. (2000), Chepuri and Homem-de Mello (2005), Novoa et al. (2006), Ak and Erera (2007), and Rei et al. (2010). Some of these methods focus on single-vehicle problems, while others consider the multiple vehicle case and 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.

A limited amount of research has focused on more dynamic routing schemes for the VRPSD. Unlike the a priori approach, where a routing plan is devised ahead of time and a simple recourse action handles route failures, routing decisions in dynamic schemes are dependent on the realizations of customer demand. Dror et al. (1989) formulate a fully dynamic, single-vehicle VRPSD as a Markov decision process. Notable efforts to solve this formulation include Secomandi (2000, 2001, 2003), which focus on obtaining rollout policies through iterative application of the restocking-based cyclic heuristic of Bertsimas et al. (1995). Secomandi and Margot (2009) obtain high quality solutions for the single-vehicle VRPSD by applying exact solution methods to a restricted state space.

With regards to the use of cyclic orders, Renaud et al. (1996) and Renaud and Boctor (2002) employ a similar idea as part of construction heuristics for VRPs. Their methods construct a number of potential vehicle routes and then identify the best possible VRP solution that can be assembled from the routes via the polynomial-time procedure of Boctor and Renaud (2000), which is identical to the polynomial-time procedure of Ryan et al. (1993). There are two main differences between these construction heuristics and our local search procedures. First, when generating candidate routes corresponding to a cyclic order, we directly implement Ryan et al. (1993), which generates routes via the sweep procedure of Gillett and Miller (1974). In Renaud et al. (1996) and Renaud and Boctor (2002), a larger set of routes is generated via a more involved sweep procedure. Second, we iteratively generate candidate routes and extract VRP solutions, whereas Renaud et al. (1996) and Renaud and Boctor (2002) execute their procedures only once. Although the one-time candidate route sets of Renaud et al. (1996) and Renaud and Boctor (2002) are larger than the candidate route sets we generate for a single cyclic order, our iterative generation of cyclic orders (and the corresponding candidate route sets) leads to consideration of a very diverse set of routes for inclusion in VRP solutions. To the best of the authors' knowledge, this is the first work to examine local search for VRPs using the cyclic-order representation.

Finally, we note that *cyclic* methods form the basis of other search procedures for VRPs. While these methods contain algorithmic features that can be described as cyclic, the only connection to our work is a similarity in name – they are fundamentally different from the cyclic orders of Ryan et al. (1993), which we employ in this paper. Bertsimas (1992) and Bertsimas et al. (1995) propose cyclic heuristics for the single-vehicle VRPSD. Given a sequence of n customers, the heuristic seeks to identify a sequence with lower expected cost by generating n alternative sequences, each one constructed by beginning the sequence at one of the n customers in the initial sequence (hence the heuristic *cycles* through the initial sequence). Thompson and Psaraftis (1993) apply *cyclic* *transfers* to several vehicle routing and scheduling problems. Based on the work of Thompson and Orlin (1989), cyclic transfer algorithms seek to improve the total cost of a set of routes by transferring small numbers of demands among routes, in a cyclical manner.

3 Problem Statement

The classical VRP is defined by the completely connected digraph $G = (V, E)$, where $V = \{0, 1, \ldots, n\}$ is a set of $n + 1$ nodes and $E = \{(i, j) : i, j \in V\}$ is a set of directed edges. Node 0 is a depot at which m identical vehicles each of capacity D are based, while the remaining nodes are customers. A deterministic symmetric travel cost matrix $C = (c_{i,j})$ is defined on E. With each customer is associated a nonnegative deterministic demand to be collected or delivered, but not both. The objective of the classical VRP is to obtain a set of vehicle routes, each route beginning and ending at the depot, with minimum travel cost such that each customer is visited exactly once by exactly one vehicle and the total demand served by a vehicle route does not exceed vehicle capacity.

The VRPSD is an extension of the classical VRP that defines customer demands as nonnegative random variables ξ_i, $i \in \{1, 2, \ldots, n\}$. The probability distribution of customer demands is known ahead of time, but actual demands are not realized until initial arrival at customer locations. Consequently, vehicle capacity may not be sufficient to fully serve customer demand on an initial visit. When such route failures occur, vehicle capacity is replenished at the depot before the vehicle can continue serving customer demand.

4 Cyclic-Order Solution Representation

In this section, we describe a cyclic-order solution encoding for VRPs identical to that of Ryan et al. (1993), the only difference being that we have changed the notation to accommodate the algorithmic framework developed in subsequent sections. We begin in §4.1 by describing our notation. For a given cyclic order, identifying the corresponding VRP solution is accomplished in two steps: generation of a set of candidate routes and extraction of the best VRP solution that can be assembled from the candidate routes. We address these topics in §4.2 and §4.3, respectively.

4.1 Cyclic-Order Notation

We denote a cyclic order, or permutation of the customer set, by π. For some index $i \in \{1, 2, \ldots, n\}$, we denote by $\pi(i)$ the customer at the i^{th} position of π. We define three functions to work with indices, each of which accounts for the cyclic, or circular, nature of π. For any $i, j \in \{1, 2, \ldots, n\}$,

$$\texttt{forw}(i, j) = \begin{cases} i + j, & i + j \leq n \\ i + j - n, & \text{otherwise,} \end{cases} \tag{1}$$

$$\texttt{back}(i, j) = \begin{cases} i - j, & i - j \geq 1 \\ n - j + i, & \text{otherwise,} \end{cases} \tag{2}$$

and

$$\texttt{dist}(i, j) = \begin{cases} j - i, & i \leq j \\ n - i + j, & \text{otherwise.} \end{cases} \tag{3}$$

A call to $\texttt{forw}(i, j)$ in (1) returns the index obtained by advancing index i "forward" j positions. If $i + j \leq n$, then this index is obtained by summing i and j. If $i + j > n$, then the index is obtained by advancing forward to the end of the cyclic order and then advancing forward from the beginning of the cyclic order, resulting in index $i + j - n$.

Similarly, a call to $\texttt{back}(i, j)$ in (2) returns the index obtained by moving index i "backward" j positions. If $i - j \geq 1$, then this index is obtained by subtracting j from i. If $i - j < 1$, then the index is obtained by advancing backward to the beginning of the cyclic order and then advancing backward from the end of the cyclic order, resulting in index $n - j + i$.

A call to $\texttt{dist}(i, j)$ in (3) calculates the number of times index i must be advanced forward such that it equals index j. If i precedes j in the cyclic order, then this number is j less i. If j precedes i, then this number is obtained by counting the number of forward advances required to reach the end of the cyclic order and then to reach position j from the beginning of the cyclic order, resulting in $n - i + j$ forward advances.

To illustrate these functions, consider $n = 5$ and $\pi = (5, 3, 4, 2, 1)$. A call to $\texttt{forw}(4, 3)$ returns index 2, which is obtained by advancing index 4 forward one time, wrapping around to the beginning of the cyclic order, and then advancing forward two more times. Thus, $\pi(\texttt{forw}(4, 3)) = \pi(2) = 3$. A call to $\texttt{back}(1, 2)$ returns index 4, which is obtained by first wrapping around to the end of the cyclic order, and then advancing backward two times. Thus, $\pi(\texttt{back}(1, 2)) = \pi(4) = 2$. A call to $\texttt{dist}(4, 2)$ returns 3, which is the number of forward advances required to go from $\pi(4) = 2$ to $\pi(2) = 3$. We also use equations (1) and (3) in combination to specify a contiguous subset of customers in π. For example, the subset of customers corresponding to positions i through j of π is denoted $\{\pi(i), \pi(\texttt{forw}(i, 1)), \pi(\texttt{forw}(i, 2)), \ldots, \pi(\texttt{forw}(i, \texttt{dist}(i, j)))\}$. In our example, if $i = 4$ and $j = 2$, then the subset is $\{\pi(4) = 2, \pi(5) = 1, \pi(1) = 5, \pi(2) = 3\}$, where the last element, $\pi(2)$, is $\texttt{dist}(4, 2) = 3$ positions forward of $\pi(4)$.

4.2 Generation of Candidate Routes

We denote by $\mathcal{R}(\pi)$ the feasible set of candidate routes consisting of contiguous elements of π. A route $r = \{\pi(i), \pi(\texttt{forw}(i,1)), \pi(\texttt{forw}(i,2)), \ldots, \pi(\texttt{forw}(i, \texttt{dist}(i,j)))\}$ 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: $\langle \texttt{first}(r), \texttt{last}(r) \rangle = \langle \pi(i), \pi(j) \rangle$. As an example, if $\pi = \{5,3,4,2,1\}$, $i = 4$, and $j = 1$, then $r = \langle \pi(4), \pi(1) \rangle$ is a route that visits the subset of customers $\{\pi(4) = 2, \pi(5) = 1, \pi(1) = 5\}$.

To generate $\mathcal{R}(\pi)$, we use a sweep procedure similar to that of Gillett and Miller (1974). Algorithm 1 describes our sweep procedure, which we employ throughout the paper. The $\texttt{Sweep}(\cdot)$ function in Algorithm 1 takes as input a cyclic order π, a route set $\mathcal{S}$, and indices $i, j \in \{1, \ldots, n\}$. A route set is a set of already defined routes, a set which may be empty. An arbitrary call to $\texttt{Sweep}(\pi, \mathcal{S}, i, j)$ begins on line 2 by initializing a generic index p to index j. At each iteration of the loop beginning on line 3, a route r is created that visits the subset of customers $\{\pi(i), \pi(\texttt{forw}(i,1)), \pi(\texttt{forw}(i,2)), \ldots, \pi(\texttt{forw}(i, \texttt{dist}(i,p)))\}$. As before, we denote such a route as $\langle \pi(i), \pi(p) \rangle$. If r is feasible, we insert it into $\mathcal{S}$, advance index p forward by one position, and repeat. Otherwise, we break from the loop, the procedure terminates, and the updated $\mathcal{S}$ is returned. In the case that all created routes are feasible and the loop is never broken, the loop eventually terminates after creating route $\langle \pi(i), \pi(\texttt{back}(i,1)) \rangle$, which visits all of the customers. In this case p is equal to $\texttt{back}(i,1)$, thus advancing p forward one position meets the termination criteria of $p = i$, which is checked at the end of each loop iteration.

Algorithm 1 Sweep Procedure for Customer $\pi(i)$ Beginning at Customer $\pi(j)$

```
1:  procedure SWEEP($\pi, \mathcal{S}, i, j$)
2:      $p \leftarrow j$
3:      repeat
4:          $r \leftarrow \langle \pi(i), \pi(p) \rangle$
5:          if $r$ is feasible then
6:              $\mathcal{S} \leftarrow \mathcal{S} \cup r$
7:          else
8:              break
9:          $p \leftarrow \texttt{forw}(p, 1)$
10:     until $p = i$
11:     return $\mathcal{S}$
```

To explain the use of Algorithm 1 to construct $\mathcal{R}(\pi)$, let $\mathcal{R}(\pi(i))$ denote the set of routes consisting of contiguous elements of π such that $\texttt{first}(r) = \pi(i)$. We generate $\mathcal{R}(\pi) =$

Customer	**x-coord**	**y-coord**	**Demand**
0	8.0	-6.0	0
1	10.0	-5.0	3
2	11.0	-2.0	3
3	4.0	-2.0	2
4	6.0	-4.5	3
5	3.0	-4.0	6
6	6.5	-5.5	4
7	3.0	-6.5	1
8	7.5	-10.5	4
9	8.4	-8.0	4
10	9.5	-10.0	4
11	14.0	-6.5	3
12	10.4	-5.5	5
13	14.5	-4.0	2

Table 1: Example Problem Data

$\cup_{i=1}^{n}\mathcal{R}(\pi(i))$ by iteratively calling $\texttt{Sweep}(\pi, \emptyset, i, i)$ for $i = 1, \ldots, n$ and storing the respective output in $\mathcal{R}(\pi(i))$. For example, consider the cyclic order π = (7, 8, 9, 10, 11, 12, 13, 1, 2, 3, 4, 5, 6) for the 13-customer classical VRP instance given in Ryan et al. (1993). Table 1 displays customer locations and demands for this problem. Vehicle capacity is 10 and distances are Euclidean. We construct $\mathcal{R}(\pi(1))$ by executing $\texttt{Sweep}(\pi, \emptyset, 1, 1)$ 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 $\{\langle 7, 7\rangle, \langle 7, 8\rangle, \langle 7, 9\rangle\}$ which we store in $\mathcal{R}(\pi(1))$. Repeating the call to $\texttt{Sweep}(\pi, \emptyset, i, i)$ 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 2 illustrates (the right half of Table 2 will be used in a subsequent illustration). We note that the VRPSD version of this 13-customer instance in which each demand figure is assumed to be the average demand of the stochastic customer demand results in the same set $\mathcal{R}(\pi)$ since routes are constructed so that expected demand assigned to each route does not exceed vehicle capacity.

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

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	$\langle 13,13\rangle$	$\langle 13,8\rangle^*$	$\langle 13,9\rangle^*$

Table 2: Example of updating $\mathcal{R}(\pi)$ to obtain $\mathcal{R}(\pi')$ for a 3-shift neighbor with $i = 2$ and $j = 8$

4.3 Solution Extraction

After generating the set of candidate routes, optimization is applied to identify the best VRP solution that can be assembled from the routes in $\mathcal{R}(\pi)$. Denote by x_π a feasible VRP solution that can be assembled from $\mathcal{R}(\pi)$. Ryan et al. (1993) solve a set partitioning problem (SPP) to find the feasible solution $x^\star_\pi$ such that $f(x^\star_\pi) \leq f(x_\pi)$ for every feasible solution $x_\pi \subseteq \mathcal{R}(\pi)$. The SPP consists of columns corresponding to each route in $\mathcal{R}(\pi)$ and rows representing each of the n customers. The cost of each column is the cost of the corresponding route (which can include a large fixed cost to reflect the cost of using a vehicle).

Ryan et al. (1993) show that the special structure of the routes produced by the sweep procedure allows such SPPs to be solved in polynomial time by finding the shortest compact cycle in a digraph. The digraph consists of n nodes, one for each customer, and one arc for each route in $\mathcal{R}(\pi)$. Ryan et al. (1993) demonstrate that the problem of finding the shortest compact cycle is equivalent to solving one shortest path problem for each of the n customers, hence the polynomial-time guarantee. The graph for the shortest path problem corresponding to a customer i is obtained by removing from the digraph all arcs that skip the node representing customer i. The least-cost path among all n shortest path problems represents the best VRP solution that can be assembled from $\mathcal{R}(\pi)$. Applying this two-phase procedure of route generation and optimization to the cyclic order example given in §4.2 results in the route set displayed in Figure 1, which consists of routes $\{\langle 12,1\rangle, \langle 2,4\rangle, \langle 5,6\rangle, \langle 7,9\rangle, \langle 10,11\rangle\}$. For a more detailed description of the optimization procedure, the reader is referred to Ryan et al. (1993) or to Boctor and Renaud (2000), which provides

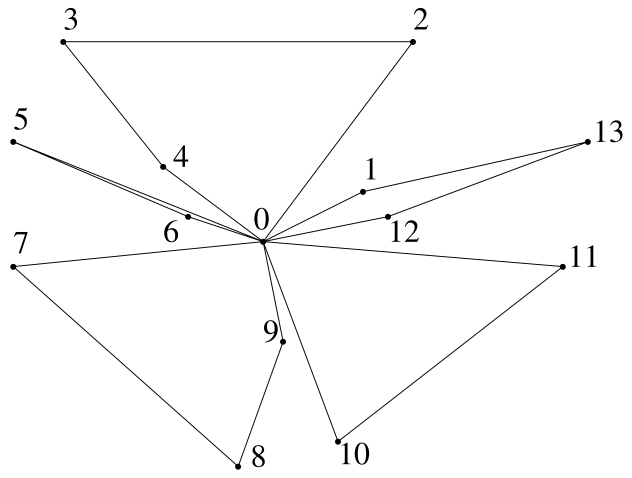

Figure 1: Best VRP solution corresponding to $\pi = (7, 8, 9, 10, 11, 12, 13, 1, 2, 3, 4, 5, 6)$. We note that this solution corrects a minor oversight in Ryan et al. (1993).

an equivalent result.

5 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:

- ***k*-shift**. For a given cyclic order π and a positive integer $k < n - 1$, select indices $i, j \in \{1, \ldots, n\}$ and shift the k contiguous elements $\pi(i), \pi(\texttt{forw}(i, 1)), \pi(\texttt{forw}(i, 2)), \ldots,$ $\pi(\texttt{forw}(i, k-1))$, as a group, to the positions immediately prior to $\pi(j)$. A k-shift neighborhood of π consists of the cyclic orders that can be constructed from the set of indices $\{i, j : \texttt{dist}(i, j) > k\}$.

- **Reverse**. For a given cyclic order π, select indices $i, j \in \{1, \ldots, n\}$ and reverse the order of $\pi(i), \pi(\texttt{forw}(i, 1)), \pi(\texttt{forw}(i, 2)), \ldots \pi(\texttt{forw}(i, \texttt{dist}(i, j)))$, which are the contiguous elements of π beginning at $\pi(i)$ and advancing forward to $\pi(j)$. A reverse neighborhood of π consists of the cyclic orders that can be constructed from all i, j pairs of indices. If $i = j$, then the entire cyclic order is reversed.

- **Exchange**. For a given cyclic order π, select indices $i, j \in \{1, \ldots, n\}$ and exchange the

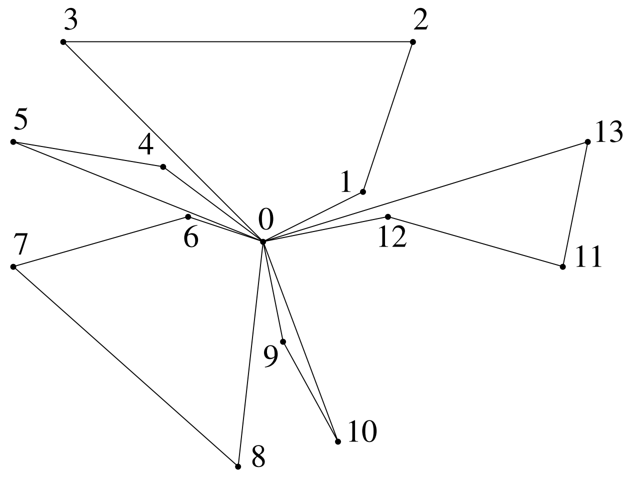

Figure 2: Best VRP solution corresponding to $\pi' = (7, 8, 9, 10, 12, 11, 13, 1, 2, 3, 4, 5, 6)$.

positions of $\pi(i)$ and $\pi(j)$. An exchange neighborhood of π consists of the cyclic orders that can be constructed from the set of indices $\{i, j : i < j\}$.

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 best VRP solution that can be assembled from the candidate routes. To illustrate the diversity of solutions that a cyclic-order neighborhood can generate, we compare Figures 1 and 2, which display the best classical 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 1 to those in Figure 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 2 from Figure 1 requires 10 customer insertions/removals if applying local search directly on the VRP solution rather than the cyclic order.

6 Updating Procedure for Neighboring Cyclic Orders

In §6.1, we motivate the need for an updating procedure. In §6.2, we identify the structural property of our algorithmic framework that enables our updating procedure. In §6.3, we provide a detailed example of how to apply our updating procedure.

6.1 Motivation

As discussed in §4, 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 $\mathcal{R}(\pi)$, where 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 calling $\texttt{Sweep}(\pi', \emptyset, i, i)$ for $i = 1, \ldots, n$. However, many of the candidate routes in $\mathcal{R}(\pi')$ may also be in $\mathcal{R}(\pi)$, therefore recreating these routes is redundant. In this section, we show how to obtain $\mathcal{R}(\pi')$ by updating $\mathcal{R}(\pi)$. We demonstrate in §7.4 that this updating procedure can significantly reduce computation time.

To motivate our updating procedures, we again consider the example presented in Table 2. 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 classical VRP instance of Ryan et al. (1993). The fourth column of Table 2 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 $*$, with all the other routes remaining unchanged. In comparison, totally reconstructing $\mathcal{R}(\pi')$ requires the creation of 33 routes, 26 of which already exist in $\mathcal{R}(\pi)$.

6.2 Structural Result

By considering the construction of $\mathcal{R}(\pi)$ (see §4), it is possible to systematically identify routes to retain, remove, and add to obtain $\mathcal{R}(\pi')$. Recall from §4 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 route subset $\mathcal{R}(\pi(1)) = \{\langle 7, 7\rangle, \langle 7, 8\rangle, \langle 7, 9\rangle\}$ in the left-hand portion of Table 2. Recall that the sweep procedure creates these routes because customers 7, 8, and 9 are positioned 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 $\texttt{Sweep}(\pi', \mathcal{R}(\pi'(10)), 10, 11)$ via Algorithm 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 route subsets $\mathcal{R}(\pi(i)), i = 2, \ldots, n$, 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 $\texttt{pos}(z)$ the position of customer z in π. If a route subset $\mathcal{R}(\pi(i))$ contains any routes visiting customers 7 and 8, then $|\mathcal{R}(\pi(i))| > \texttt{dist}(i, \texttt{pos}(8))$, where $\texttt{pos}(8) = 2$. Because $|\mathcal{R}(\pi(13))| = 3 > \texttt{dist}(13, 2) = 2$, we know that $\mathcal{R}(\pi(13))$ contains $3 - 2 = 1$ route that visits customers 7 and 8 (route $\langle 6, 8\rangle$). Because $|\mathcal{R}(\pi(12))| = 2 \leq \texttt{dist}(12, 2) = 3$, $\mathcal{R}(\pi(12))$ 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. These relationships are stated formally in Proposition 1, where we define $\Lambda\big(\pi(i), \pi(j)\big) = \{r \in \mathcal{R}(\pi(i)) : \texttt{dist}(i, \texttt{pos}(\texttt{last}(r))) \geq \texttt{dist}(i, j)\}$ to be the routes in $\mathcal{R}(\pi(i))$ that visit customers $\pi(i)$ and $\pi(j)$.

Proposition 1. *For all $i, j \in \{1, \ldots, n\}$, if $|\mathcal{R}(\pi(i))| > \texttt{dist}(i, j)$, then $|\Lambda\big(\pi(i), \pi(j)\big)| = |\mathcal{R}(\pi(i))| - \texttt{dist}(i, j)$. Otherwise, $\{\mathcal{R}(\pi(p)) : p \in (i, \texttt{back}(i, 1), \texttt{back}(i, 2), \ldots, \texttt{back}(i, \texttt{dist}(j, i) - 1))\}$ do not contain any routes that visit customers $\pi(i)$ and $\pi(j)$.*

We note that the sweep procedure in Algorithm 1 creates the routes in $\mathcal{R}(\pi(i))$ for $i = 1, \ldots, n$, in order of increasing $\texttt{pos}(\texttt{last}(\cdot))$, meaning that the routes in a given subset $\mathcal{R}(\pi(i))$ are sorted by the position in π of the last customer in the subset of customers visited by each route (we present the route sets in Table 2 in this order), thus facilitating the identification of $\Lambda\big(\pi(i), \pi(j)\big)$. To update $\mathcal{R}(\pi')$, we simply remove the last $|\mathcal{R}(\pi(i))| - \texttt{dist}(i, j)$ routes from the subset $\mathcal{R}(\pi(i))$, 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 1.

For a cyclic order π and a neighbor cyclic order π' generated from one of the neighborhoods in §5, we employ Proposition 1 to obtain $\mathcal{R}(\pi')$ by updating $\mathcal{R}(\pi)$. Algorithm 2 details this procedure for the k-shift neighborhood. Algorithms 3 and 4 detail this procedure for the reverse and exchange neighborhoods, respectively.

Algorithm 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: $\pi_A \leftarrow \big(\pi(i), \pi(\texttt{forw}(i,1)), \ldots, \pi(\texttt{forw}(i,k-1))\big)$
4: $\pi_B \leftarrow \big(\pi(j), \pi(\texttt{forw}(j,1)), \ldots, \pi(\texttt{forw}(j,\texttt{dist}(j,\texttt{back}(i,1))))\big)$
5: $\pi_C \leftarrow \big(\pi(\texttt{forw}(i,k)), \pi(\texttt{forw}(i,k+1)), \ldots, \pi(\texttt{forw}(i,k+\texttt{dist}(\texttt{forw}(i,k),\texttt{back}(j,1))))\big)$
6: $\pi' \leftarrow \pi_A \oplus \pi_B \oplus \pi_C$
7: $\mathcal{R}(\pi') \leftarrow \mathcal{R}(\pi)$
8: $p \leftarrow k$
9: repeat
10:     if $|\mathcal{R}(\pi'(p))| > \texttt{dist}(p,\texttt{forw}(k,1))$ then
11:         $\mathcal{R}(\pi'(p)) \leftarrow \mathcal{R}(\pi'(p)) \setminus \Lambda(\pi'(p), \pi'(\texttt{forw}(k,1)))$
12:         $\mathcal{R}(\pi'(p)) \leftarrow \texttt{Sweep}(\pi', \mathcal{R}(\pi'(p)), p, \texttt{forw}(k,1))$
13:     else if $|\mathcal{R}(\pi'(p))| = \texttt{dist}(p,\texttt{forw}(k,1))$ then
14:         $\mathcal{R}(\pi'(p)) \leftarrow \texttt{Sweep}(\pi', \mathcal{R}(\pi'(p)), p, \texttt{forw}(k,1))$
15:     else
16:         break
17:     $p \leftarrow \texttt{back}(p,1)$
18: until $p = n$
19: $p \leftarrow \texttt{forw}(k,\texttt{dist}(j,i))$
20: repeat
21:     if $|\mathcal{R}(\pi'(p))| > \texttt{dist}(p,\texttt{forw}(k,\texttt{dist}(j,i)+1))$ then
22:         $\mathcal{R}(\pi'(p)) \leftarrow \mathcal{R}(\pi'(p)) \setminus \Lambda(\pi'(p), \pi'(\texttt{forw}(k,\texttt{dist}(j,i)+1)))$
23:         $\mathcal{R}(\pi'(p)) \leftarrow \texttt{Sweep}(\pi', \mathcal{R}(\pi'(p)), p, \texttt{forw}(k,\texttt{dist}(j,i)+1))$
24:     else if $|\mathcal{R}(\pi'(p))| = \texttt{dist}(p,\texttt{forw}(k,\texttt{dist}(j,i)+1))$ then
25:         $\mathcal{R}(\pi'(p)) \leftarrow \texttt{Sweep}(\pi', \mathcal{R}(\pi'(p)), p, \texttt{forw}(k,\texttt{dist}(j,i)+1))$
26:     else
27:         break
28:     $p \leftarrow \texttt{back}(p,1)$
29: until $p = k$
30: $p \leftarrow n$
31: repeat
32:     if $|\mathcal{R}(\pi'(p))| > \texttt{dist}(p,1)$ then
33:         $\mathcal{R}(\pi'(p)) \leftarrow \mathcal{R}(\pi'(p)) \setminus \Lambda(\pi'(p), \pi'(1))$
34:         $\mathcal{R}(\pi'(p)) \leftarrow \texttt{Sweep}(\pi', \mathcal{R}(\pi'(p)), p, 1)$
35:     else if $|\mathcal{R}(\pi'(p))| = \texttt{dist}(p,1)$ then
36:         $\mathcal{R}(\pi'(p)) \leftarrow \texttt{Sweep}(\pi', \mathcal{R}(\pi'(p)), p, 1)$
37:     else
38:         break
39:     $p \leftarrow \texttt{back}(p,1)$
40: until $p = \texttt{forw}(k,\texttt{dist}(j,i))$
```

6.3 Example

We illustrate Algorithm 2 by stepping through the example in Table 2, where π' is in the k-shift neighborhood of π with $k = 3$, $i = 2$, and $j = 8$. Lines 3-6 of Algorithm 2 construct neighbor π' by partitioning π into three sequences (π_A, π_B, and π_C), which are then concatenated (where $\oplus$ is the concatenation operator) in line 6 to form π'. The sequence $\pi_A = (8, 9, 10)$ in line 3 corresponds to the $k = 3$ elements of π to be shifted. The sequence $\pi_B = (1, 2, 3, 4, 5, 6, 7)$ in line 4 corresponds to the string of elements in π between position $j = 8$ (where π_A is to be inserted) forward through position $\texttt{back}(i = 2, 1) = 1$, the position immediately before π_A currently begins. Sequence $\pi_C = (11, 12, 13)$ in line 5 corresponds to the string of elements in π lying between the elements to be shifted and their insertion position, i.e., position 5 forward through position 7 of π. Concatenating the sequences in the order of $\pi_A \oplus \pi_B \oplus \pi_C$ in line 6 results in $\pi(i)$ occupying position 1 in π'. We do this for algorithmic convenience and note that the concatenation $\pi_B \oplus \pi_C \oplus \pi_A$ or $\pi_C \oplus \pi_A \oplus \pi_B$ results in an alternate assignment of position 1, but an equivalent cyclic order with an identical set of candidate routes and subsequent VRP solution.

The remainder of Algorithm 2 utilizes the observation that the k-shift neighborhood generates a neighbor solution by rearranging the sequences π_A, π_B, and π_C. That is, $\pi = \pi_A \oplus \pi_C \oplus \pi_B$ and $\pi' = \pi_A \oplus \pi_B \oplus \pi_C$. Therefore, $\mathcal{R}(\pi')$ can be conveniently updated from $\mathcal{R}(\pi)$ by leveraging Proposition 1 while inspecting how the reordering of these sub-sequences affects the contiguous elements.

To begin the updating process, line 7 initializes candidate route set $\mathcal{R}(\pi')$ to $\mathcal{R}(\pi)$. The three loops composing the remainder of Algorithm 2 update the route sets corresponding to sequences π_A, π_B, and π_C, respectively. Lines 8-18 treat sequence π_A beginning at $\pi'(k)$ and advancing backward to $\pi'(1)$. At each iteration of the loop beginning on line 9, routes visiting customers 10 and 11 are removed from route set $\mathcal{R}(\pi'(p))$, then routes visiting customers 10 and 1 are added. Line 10 employs Proposition 1 to identify routes for removal. In the first iteration of the loop, when $p = k = 3$, $\mathcal{R}(\pi'(3))$ is initially $\{\langle 10, 10\rangle, \langle 10, 11\rangle\}$, and the condition $|\mathcal{R}(\pi'(3))| = 2 > \texttt{dist}(3, 4) = 1$ alerts us that one route needs to be removed and new routes may have to be added. Line 11 removes route $\langle 10, 11\rangle$ and line 12 adds routes $\langle 10, 1\rangle$ and $\langle 10, 2\rangle$ by executing $\texttt{Sweep}(\pi', \mathcal{R}(\pi'(3)), 3, 4)$ via Algorithm 1. In the next iteration, when $p = 2$, $\mathcal{R}(\pi'(2))$ is initially $\{\langle 9, 9\rangle, \langle 9, 10\rangle\}$, and the condition $|\mathcal{R}(\pi'(2))| = 2 \le \texttt{dist}(2, 4) = 2$ informs us that no routes need to be removed. However, the equality in line 13 holds, so an attempt to add routes containing customers 9 and 1 is made by executing $\texttt{Sweep}(\pi', \mathcal{R}(\pi'(2)), 2, 4)$. 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 10 or 13 are met and the loop is terminated via the break statement on line 16.

If the criteria for breaking had not been satisfied, setting p to $\texttt{back}(1,1) = n$ on line 17 would have met the loop termination criteria on line 18, which is checked at the end of each iteration.

In a similar fashion, lines 19-29 iterate backward through sequence π_B. At each iteration of the loop beginning on line 20, routes visiting customers 7 and 8 are removed from route set $\mathcal{R}(\pi'(p))$, then routes visiting customers 7 and 11 are added. Line 21 employs Proposition 1 to detect routes to be removed. In the first iteration of the loop, when $p = \texttt{forw}(3,7) = 10$, $\mathcal{R}(\pi'(10))$ is initially $\{\langle 7,7\rangle, \langle 7,8\rangle, \langle 7,9\rangle\}$, and the condition $|\mathcal{R}(\pi'(10))| = 3 > \texttt{dist}(10,11) = 1$ indicates that two routes need to be removed and new routes may have to be added. Line 22 removes routes $\langle 7,8\rangle$ and $\langle 7,9\rangle$ and line 23 adds routes $\langle 7,11\rangle$ and $\langle 7,12\rangle$ by performing $\texttt{Sweep}(\pi', \mathcal{R}(\pi'(10)), 10, 11)$ via Algorithm 1. In the next iteration, when $p = 9$, $\mathcal{R}(\pi'(9))$ is initially $\{\langle 6,6\rangle, \langle 6,7\rangle, \langle 6,8\rangle\}$, and the condition $|\mathcal{R}(\pi'(9))| = 3 > \texttt{dist}(9,11) = 2$ notifies us that one route needs to be removed and new routes may have to be added. Line 22 removes route $\langle 6,8\rangle$ and line 23 adds route $\langle 6,11\rangle$. In the next iteration, when $p = 8$, $\mathcal{R}(\pi'(8))$ is initially $\{\langle 5,5\rangle, \langle 5,6\rangle\}$, and the condition $|\mathcal{R}(\pi'(8))| = 2 < \texttt{dist}(8,11) = 3$ tells us that no routes need to be removed or added. Thus, the loop is terminated via the break statement on line 27.

Finally, lines 30-40 iterate backward through sequence π_C. At each iteration of the loop beginning on line 31, routes visiting customers 13 and 1 are removed from route set $\mathcal{R}(\pi'(p))$, then routes visiting customers 13 and 8 are added. Line 32 employs Proposition 1 to identify routes to be eliminated. In the first iteration of the loop, when $p = 13$, $\mathcal{R}(\pi'(13))$ is initially $\{\langle 13,13\rangle, \langle 13,1\rangle, \langle 13,2\rangle, \langle 13,3\rangle\}$, and the condition $|\mathcal{R}(\pi'(13))| = 4 > \texttt{dist}(13,1) = 1$ signals us that three routes need to be removed and new routes may have to be added. Line 33 removes routes $\langle 13,1\rangle$, $\langle 13,2\rangle$, and $\langle 13,3\rangle$ and line 34 adds routes $\langle 13,8\rangle$ and $\langle 13,9\rangle$ by executing $\texttt{Sweep}(\pi', \mathcal{R}(\pi'(13)), 13, 1)$ via Algorithm 1. In the next iteration, when $p = 12$, $\mathcal{R}(\pi'(12))$ is initially $\{\langle 12,12\rangle, \langle 12,13\rangle, \langle 12,1\rangle\}$, and the condition $|\mathcal{R}(\pi'(12))| = 3 > \texttt{dist}(12,1) = 2$ advises us that one route needs to be removed and that new routes may have to be added. Line 33 removes route $\langle 12,1\rangle$, then line 34 attempts to add routes containing customers 12 and 8 by performing $\texttt{Sweep}(\pi', \mathcal{R}(\pi'(12)), 12, 1)$. No such routes are added to $\mathcal{R}(\pi'(12))$ because they are infeasible with respect to vehicle capacity. In the next iteration, when $p = 11$, $\mathcal{R}(\pi'(11))$ is initially $\{\langle 11,11\rangle, \langle 11,12\rangle, \langle 11,13\rangle\}$, and the condition $|\mathcal{R}(\pi'(11))| = 3 \leq \texttt{dist}(11,1) = 3$ indicates that no routes need to be removed. However, the equality in line 35 holds, so an attempt to add routes containing customers 11 and 8 is made by calling $\texttt{Sweep}(\pi', \mathcal{R}(\pi'(11)), 11, 1)$. No such routes are added to $\mathcal{R}(\pi'(11))$ because they are infeasible with respect to vehicle capacity. In the next iteration, $p = 10$, which meets the loop termination criteria.

Table 2 denotes the routes that we remove and insert during the procedure by superscripting them with $\ddagger$ and $*$, respectively. We note that if neither of the conditions for removing and adding routes (lines 10 and 13, lines 21 and 24, lines 32 and 35, 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 remaining indices of the loop.

In the worst case, where we examine each route set and create n routes, the complexity of Algorithm 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 routing constraints (e.g., vehicle capacity) allow all customers to be serviced on one route. As our computational results in §7.4 demonstrate, the updating procedure provides significant computational advantages. Algorithms 3 and 4 detail similar updating procedures for the reverse and exchange cyclic-order neighborhoods. These procedures also have a worst case complexity of $O(n^2)$.

Algorithm 3 Update $\mathcal{R}(\pi)$ to Obtain $\mathcal{R}(\pi')$ for a 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: $\pi' \leftarrow \pi$
 4: reverse the order of $\pi'(i), \pi'(\texttt{forw}(i,1)), \pi'(\texttt{forw}(i,2)), \dots \pi'(\texttt{forw}(i, \texttt{dist}(i,j)))$
 5: $\mathcal{R}(\pi') \leftarrow \mathcal{R}(\pi)$
 6: $p \leftarrow i$
 7: repeat
 8:     if $|\mathcal{R}(\pi'(p))| > \texttt{dist}(p, \texttt{forw}(p,1))$ then
 9:         $\mathcal{R}(\pi'(p)) \leftarrow \mathcal{R}(\pi'(p)) \setminus \Lambda(\pi'(p), \pi'(\texttt{forw}(p,1)))$
10:         $\mathcal{R}(\pi'(p)) \leftarrow \texttt{Sweep}(\pi', \mathcal{R}(\pi'(p)), p, \texttt{forw}(p,1))$
11:     else
12:         $\mathcal{R}(\pi'(p)) \leftarrow \texttt{Sweep}(\pi', \mathcal{R}(\pi'(p)), p, \texttt{forw}(p,1))$
13:     $p \leftarrow \texttt{forw}(p,1)$
14: until $p = \texttt{forw}(j,1)$
15: $p \leftarrow \texttt{back}(i,1)$
16: repeat
17:     if $|\mathcal{R}(\pi'(p))| > \texttt{dist}(p,i)$ then
18:         $\mathcal{R}(\pi'(p)) \leftarrow \mathcal{R}(\pi'(p)) \setminus \Lambda(\pi'(p), \pi'(i))$
19:         $\mathcal{R}(\pi'(p)) \leftarrow \texttt{Sweep}(\pi', \mathcal{R}(\pi'(p)), p, i)$
20:     else if $|\mathcal{R}(\pi'(p))| = \texttt{dist}(p,i)$ then
21:         $\mathcal{R}(\pi'(p)) \leftarrow \texttt{Sweep}(\pi', \mathcal{R}(\pi'(p)), p, i)$
22:     else
23:         break
24:     $p \leftarrow \texttt{back}(p,1)$
25: until $p = j$
```

Algorithm 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: $\pi' \leftarrow \pi$
 4: exchange the positions of $\pi'(i)$ and $\pi'(j)$
 5: $\mathcal{R}(\pi') \leftarrow \mathcal{R}(\pi)$
 6: **for** all $p \in \{i, j\}$ **do**
 7:     **if** $|\mathcal{R}(\pi'(p))| > \texttt{dist}(p, \texttt{forw}(p, 1))$ **then**
 8:         $\mathcal{R}(\pi'(p)) \leftarrow \mathcal{R}(\pi'(p)) \setminus \Lambda\big(\pi'(p), \pi'(\texttt{forw}(p, 1))\big)$
 9:         $\mathcal{R}(\pi'(p)) \leftarrow \texttt{Sweep}(\pi', \mathcal{R}(\pi'(p)), p, \texttt{forw}(p, 1))$
10:     **else**
11:         $\mathcal{R}(\pi'(p)) \leftarrow \texttt{Sweep}(\pi', \mathcal{R}(\pi'(p)), p, \texttt{forw}(p, 1))$
12: $p \leftarrow \texttt{back}(i, 1)$
13: **repeat**
14:     **if** $|\mathcal{R}(\pi'(p))| > \texttt{dist}(p, i)$ **then**
15:         $\mathcal{R}(\pi'(p)) \leftarrow \mathcal{R}(\pi'(p)) \setminus \Lambda\big(\pi'(p), \pi'(i)\big)$
16:         $\mathcal{R}(\pi'(p)) \leftarrow \texttt{Sweep}(\pi', \mathcal{R}(\pi'(p)), p, i)$
17:     **else if** $|\mathcal{R}(\pi'(p))| = \texttt{dist}(p, i)$ **then**
18:         $\mathcal{R}(\pi'(p)) \leftarrow \texttt{Sweep}(\pi', \mathcal{R}(\pi'(p)), p, i)$
19:     **else**
20:         break
21:     $p \leftarrow \texttt{back}(p, 1)$
22: **until** $p = j$
23: $p \leftarrow \texttt{back}(j, 1)$
24: **repeat**
25:     **if** $|\mathcal{R}(\pi'(p))| > \texttt{dist}(p, j)$ **then**
26:         $\mathcal{R}(\pi'(p)) \leftarrow \mathcal{R}(\pi'(p)) \setminus \Lambda\big(\pi'(p), \pi'(j)\big)$
27:         $\mathcal{R}(\pi'(p)) \leftarrow \texttt{Sweep}(\pi', \mathcal{R}(\pi'(p)), p, j)$
28:     **else if** $|\mathcal{R}(\pi'(p))| = \texttt{dist}(p, j)$ **then**
29:         $\mathcal{R}(\pi'(p)) \leftarrow \texttt{Sweep}(\pi', \mathcal{R}(\pi'(p)), p, j)$
30:     **else**
31:         break
32:     $p \leftarrow \texttt{back}(p, 1)$
33: **until** $p = i$
```

7 Application to the VRPSD

To demonstrate the effectiveness of our methodology, we embed cyclic-order neighborhoods in a simulated annealing procedure to solve the VRPSD. In §7.1, we describe a procedure to calculate the expected cost of a VRPSD route. In §7.2, we describe our simulated annealing procedure and discuss the details of our computer implementation. In §7.3, we report our experience with benchmark problem instances for the VRPSD. In §7.4, we examine the computational benefit of employing the updating procedure outlined in §6.

7.1 Route Cost Calculation

Teodorović and Pavković (1992) provide a formula to calculate the expected cost of an a priori VRPSD route. In this section, to ease the development of computer code to implement our method, we show how to apply the equation using our cyclic-order notation. The procedure we describe is implied in each call to $\texttt{Sweep}(\cdot)$ in Algorithm 1; whenever a route is added to the candidate route set, its cost must be calculated and given as input to the solution extraction procedure of §4.3.

As discussed in §2, for a VRPSD route we employ the a priori policy of Laporte et al. (2002), which requires a vehicle to visit customers in the sequence defined by a given vehicle route. When a route failure occurs, demand at that customer must be fully served via return trips to the depot before continuing on to the next customer on the route. As shown by Teodorović and Pavković (1992), under this policy the expected cost of a route is the sum of the planned or deterministic travel cost and the expected cost of route failures. For a route $\langle\pi(i),\pi(j)\rangle$, a call to $\texttt{Cost}(\langle\pi(i),\pi(j)\rangle)$ in Algorithm 5 returns the expected route cost.

Lines 2-7 of Algorithm 5 calculate the deterministic travel cost of traversing the route, where $c_{i,j}$, defined in §2, denotes the travel cost of traveling from customer i to customer j. Lines 8-16 calculate the expected cost of route failures and add this to the total cost, which is returned on line 17. We define $\xi_{\pi(i),\pi(t)} = \xi_{\pi(i)} + \xi_{\pi(\texttt{forw}(i,1))} + \xi_{\pi(\texttt{forw}(i,2))} + \cdots + \xi_{\pi(\texttt{forw}(i,\texttt{dist}(i,t)))}$ to be the convolution of the random customer demands beginning in position i and advancing forward through position t of a cyclic order π. Each iteration of the loop beginning on line 11 calculates the expected cost of the l^{th} route failure at customer $\pi(p)$, where

$$F^t(y) = \begin{cases} \mathbb{P}\left\{\xi_{\pi(i),\pi(t)} \le y\right\}, & t \in \{i, \texttt{forw}(i,1), \texttt{forw}(i,2), \ldots, \texttt{forw}(i,\texttt{dist}(i,j))\} \\ 1, & \text{otherwise.} \end{cases} \tag{4}$$

If the supports for customer demands are not bounded above, then an infinite number of route failures is possible, thus the loop beginning on line 11 should repeat for an infinite number of iterations. However, as the probability of a large number of route failures at a given customer is typically quite small, the loop is terminated based on some tolerance criteria, such as a small percentage increase in the total cost calculation.

Algorithm 5 Expected Cost Calculation of a VRPSD Route

```
 1: procedure Cost($\langle\pi(i),\pi(j)\rangle$)
 2:     cost $\leftarrow c_{0,\pi(i)}$
 3:     $p \leftarrow i$
 4:     repeat
 5:         cost $\leftarrow$ cost $+ c_{\pi(p),\pi(\texttt{forw}(p,1))}$
 6:     until $p = j$
 7:     cost $\leftarrow$ cost $+ c_{\pi(j),0}$
 8:     $p \leftarrow i$
 9:     repeat
10:         $l \leftarrow 1$
11:         repeat
12:             cost $\leftarrow$ cost $+ 2\left[F^{\texttt{back}(p,1)}(lD) - F^{p}(lD)\right] c_{0,\pi(p)}$
13:             $l \leftarrow l + 1$
14:         until termination criteria met
15:         $p \leftarrow \texttt{forw}(p,1)$
16:     until $p = \texttt{forw}(j,1)$
17:     return cost
```

We note that the $\texttt{Cost}(\cdot)$ procedure of Algorithm 5 iterates through the customers on a route $\langle\pi(i),\pi(j)\rangle$ in the order they appear in π. If the customer sequence is different, then Algorithm 5 is simply applied to the appropriate sequence. The sweep-based methods discussed in §4 utilize a traveling salesman heuristic to improve the sequence of customers in each sweep-generated route. In our code development, however, 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 $\langle\texttt{first}(r),\texttt{last}(r)\rangle$, 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 sub-route rather than by recomputing the entire cost via Algorithm 5. For example, for a VRPSD route $\langle\pi(i),\pi(\texttt{forw}(p,1))\rangle$, both portions of the expected route cost can be calculated from the expected cost of route $\langle\pi(i),\pi(p)\rangle$ as $\langle\pi(i),\pi(\texttt{forw}(p,1))\rangle$ simply inserts customer $\pi(\texttt{forw}(p,1))$ at the end of route $\langle\pi(i),\pi(p)\rangle$. This simple updating is not possible if the sequence of customers on a route is not defined by the cyclic order.

The drawback of this approach is that it may be possible to improve the route set corresponding to a cyclic order π by changing the sequence of the customers on one or more routes contained in $\mathcal{R}(\pi)$. Improving the quality of routes in $\mathcal{R}(\pi)$ in turn may make it possible to assemble a better VRP solution via the extraction procedure of §4.3. However, because the size of $\mathcal{R}(\pi)$ can be quite large, optimizing each individual route in the candidate route set 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.

Further, we consider only the forward orientation of each route. This is in contrast to some methods (e.g., Laporte et al. 2002) that, for each route, also calculate the expected cost of the backward orientation and reverse the visit sequence if the expected cost is less than that of the forward orientation. Our experience indicates that allowing the search procedure to systematically reverse route orientations is much faster than calculating the cost of two orientations for each route.

7.2 Simulated Annealing Procedure

We demonstrate the potential of cyclic-order neighborhoods to facilitate the discovery of high quality solutions for the VRPSD by embedding them within a simulated annealing framework. 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 VRP with time windows (Bent and Van Hentenryck 2004), and the orthogonal stock-cutting problem (Burke et al. 2009), to name a few. 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 on the quality of the final solution.

Our simulated annealing procedure is executed in two phases. In the first phase, we solve the classical VRP. Thus, for a given cyclic order π, when calculating the cost of a candidate route in $\mathcal{R}(\pi)$, we only consider the deterministic travel cost along each route segment. The second phase seeks to improve the solution by explicitly accounting for the cost of recourse actions resulting from potential route failures. In this phase, the cost of a candidate route includes the expected cost resulting from potential route failures (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. Although the second phase can serve as a stand-alone solution procedure (and this yields good VRPSD solutions), initializing the second phase with the first-phase solution accelerates the discovery of high quality VRPSD solutions. As the first phase can be executed quickly relative to the second phase (due to the extra computation required to calculate the expected cost of routes in the second phase), the two-phase procedure results in an overall reduction in computational effort.

Algorithm 6 outlines the steps required to execute a single phase of our simulated annealing procedure. Three cyclic orders are maintained – π_{best}, π_{curr}, and π_{neigh} – corresponding to the best-found, the current, and a neighbor cyclic order, respectively. An iteration of the inner loop begins on line 3 by randomly selecting one of five neighborhood structures: 1-shift, 2-shift, 3-shift, reverse, or exchange. Denoting the resulting neighborhood structure by $N(\cdot)$, line 4 randomly selects a solution π_{neigh} in $N(\pi_{\text{curr}})$, the neighborhood of the current solution. Line 5 probabilistically updates π_{curr} with π_{neigh}, where $(y)^+ = y$ if $y > 0$ and 0 otherwise, τ is the current temperature (a control parameter in simulated annealing), and $g(\cdot)$ is a function representing the evaluation of a cyclic order (see §4.2 and §4.3) and returns the cost $f(\cdot)$ of the associated route set. Lines 6 and 7 update the best-found cyclic order. The inner loop terminates after 7,000 iterations. The outer loop implements a geometric cooling schedule and terminates after at least 100 iterations and without having updated the best-found cyclic order for 75 successive iterations.

Algorithm 6 Simulated Annealing

```
1:  repeat
2:      repeat
3:          Randomly select neighborhood $N(\cdot) \in \{\text{1-shift, 2-shift, 3-shift, reverse, exchange}\}$
4:          Randomly select $\pi_{\text{neigh}} \in N(\pi_{\text{curr}})$
5:          $\pi_{\text{curr}} \leftarrow \pi_{\text{neigh}}$ with probability $\exp\{-((g(\pi_{\text{neigh}}) - g(\pi_{\text{curr}})))^+/\tau\}$
6:          if $g(\pi_{\text{curr}}) < g(\pi_{\text{best}})$ then
7:              $\pi_{\text{best}} \leftarrow \pi_{\text{curr}}$
8:      until 7,000 iterations
9:      $\tau \leftarrow \tau \times 0.97$
10: until At least 100 iterations and not update $\pi_{\text{best}}$ for 75 successive iterations
```

The first phase initializes π_{best} and π_{curr} with a radial ordering of customers about the depot (i.e., customers are ordered by their polar angle with respect to a horizontal axis through the depot). For some benchmark problems in the literature where customer locations are unknown (i.e., only a travel cost matrix is available), a randomly generated cyclic order can serve as the initial solution. The second phase initializes π_{best} and π_{curr} with the best cyclic order returned in the first phase. Each phase begins with an initial temperature of $\tau = 10$. Based on computational tests that explore a wide set of parameters, we observe these parameters to perform well for the VRPSD instances considered in this paper.

We implement our simulated annealing procedure for the VRPSD in C++. We execute all computational experiments on 2.8GHz Intel Xeon X5660 processors with 12GB of RAM and the CentOS 5.3 operating system (we do not utilize parallel processing).

7.3 VRPSD Results

In this subsection, we compare the performance of cyclic-order neighborhoods embedded within simulated annealing to benchmark results for the VRPSD. To facilitate the comparison, we utilize the same problem instances as Christiansen and Lysgaard (2007), who adapt VRP instances to the VRPSD by assuming that customer demands are independent Poisson random variables with the mean demand for each customer equal to the deterministic value of the demand given in the underlying VRP problem instance (publicly available via Ralphs 2008). The name of each problem instance indicates the number of customers (including the depot) and the minimum number of required vehicles. For example, instance A-n38-k5 requires 37 customers be serviced by at least 5 vehicles. All of the problem instances we consider use Euclidian distances.

Table 3 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 classical VRP, e.g., by taking demand uncertainty into account when evaluating the solutions, but not when computing them. The latter is included because for some instances Christiansen and Lysgaard (2007) do not obtain an optimal solution, and thus the expected cost of following high quality classical VRP routes serves as an additional benchmark. Further, a comparison to best known routing plans for the classical VRP allows one to measure the benefit of modeling uncertainty in the optimization. In the columns displaying the results of Christiansen and Lysgaard (2007), when a value is reported for both the number of vehicles and the cost, this reported cost 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 nor 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 (2007) 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.

The columns of Table 3 corresponding to our approach report, over 10 runs, the best solution values, the average solution values, the standard deviation of solution values, the average CPU seconds, and the "Gap," which is the percentage above the minimum of the solution values given in columns 3 and 6; if column 3 does not contain a value, then the value in column 6 is used to

	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	199.8	0.00
A-n33-k5	5	704.20	8	5	722.99	5	704.20	705.91	2.79	178.2	0.00
A-n33-k6	6	793.90	49	6	816.58	6	793.90	793.95	0.10	141.1	0.00
A-n34-k5	–	827.87	1200	5	839.95	6	826.87	827.26	1.23	236.4	-0.12
A-n36-k5	–	–	–	5	907.55	5	858.71	859.48	1.42	276.1	-5.38
A-n37-k5	–	708.34	1200	5	709.83	5	708.34	709.67	4.18	386.9	0.00
A-n37-k6	–	1030.75	1200	6	1069.32	7	1030.73	1031.74	3.20	205.5	0.00
A-n38-k5	–	778.09	1200	5	831.99	6	775.14	775.25	0.35	313.4	-0.38
A-n39-k5	6	869.18	3	5	903.26	6	869.18	869.58	0.46	257.6	0.00
A-n39-k6	6	876.60	279	6	960.81	6	876.60	883.44	7.29	239.9	0.00
A-n44-k6	–	1025.48	1200	6	1047.18	7	1025.48	1029.72	6.10	281.4	0.00
A-n45-k6	–	–	–	6	1096.19	7	1026.73	1027.92	1.36	301.4	-6.34
A-n45-k7	7	1264.83	882	7	1302.20	7	1264.99	1288.70	17.13	216.0	0.01
A-n46-k7	–	1002.41	1200	7	1069.66	7	1002.22	1003.19	1.07	314.1	-0.02
A-n48-k7	–	–	–	7	1248.27	7	1187.14	1188.06	1.96	292.0	-4.90
A-n53-k7	–	–	–	7	1180.10	8	1124.27	1129.36	4.06	468.8	-4.73
A-n54-k7	–	–	–	7	1342.87	8	1287.07	1292.29	8.87	409.4	-4.16
A-n55-k9	–	–	–	9	1264.18	10	1179.11	1184.77	5.70	265.5	-6.73
A-n60-k9	–	–	–	9	1608.40	10	1529.82	1535.35	7.38	393.7	-4.89
E-n22-k4	4	411.57	1	4	411.73	4	411.57	411.57	0.00	104.1	0.00
E-n33-k4	4	850.27	86	4	850.27	4	850.27	851.24	3.07	371.7	0.00
E-n51-k5	–	–	–	5	553.26	6	552.26	552.81	0.79	586.0	-0.18
P-n16-k8	8	512.82	0	8	512.82	8	512.82	512.82	0.00	9.0	0.00
P-n19-k2	3	224.06	153	2	229.68	3	224.06	224.06	0.00	234.8	0.00
P-n20-k2	2	233.05	352	2	233.05	2	233.05	233.05	0.00	269.6	0.00
P-n21-k2	2	218.96	5	2	218.96	2	218.96	218.96	0.00	332.7	0.00
P-n22-k2	2	231.26	219	2	231.26	2	231.26	231.26	0.00	352.3	0.00
P-n22-k8	9	681.06	0	8	707.80	9	681.06	681.06	0.00	28.6	0.00
P-n23-k8	9	619.52	1	8	662.31	9	619.53	619.57	0.12	21.1	0.00
P-n40-k5	5	472.50	6	5	475.45	5	472.50	472.50	0.00	367.2	0.00
P-n45-k5	–	–	–	5	546.05	5	533.52	537.13	4.19	603.8	-2.29
P-n50-k10	–	–	–	10	792.20	11	760.94	764.12	3.90	150.6	-3.95
P-n50-k7	–	–	–	7	606.41	7	582.37	584.95	2.33	343.0	-3.96
P-n50-k8	–	–	–	8	724.69	9	669.81	674.11	4.54	225.3	-7.57
P-n51-k10	11	809.70	430	10	859.24	11	812.74	816.95	4.27	159.9	0.37
P-n55-k10	–	–	–	10	797.21	10	745.70	752.36	3.94	208.9	-6.46
P-n55-k15	18	1068.05	792	15	1191.34	18	1068.05	1072.17	3.71	117.0	0.00
P-n55-k7	–	–	–	7	616.44	7	588.56	593.10	3.15	452.8	-4.52
P-n60-k10	–	–	–	10	831.24	11	804.24	810.22	2.55	296.1	-3.25
P-n60-k15	16	1085.49	1348	15	1133.30	16	1087.41	1098.11	6.91	134.8	0.18

Table 3: Computational Results for the VRPSD

compute the gap. When calculating the gap, for example, if the best known solution value is x and our method returns a solution value of y, then the gap is $(y - x) \times 100/x$ percent. A negative gap indicates that the solution value obtained by our two-phase procedure improves upon the solution value of Christiansen and Lysgaard (2007), 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 percent), P-n51-k10 (gap of 0.37 percent), and P-n60-k15 (gap of 0.18 percent). 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 the method of Christiansen and Lysgaard (2007) or the expected value of the best known deterministic solution. The average gap for these 21 instances is -3.33 percent. The difference in computer architectures makes it difficult to compare run times, but we find the average run time of 268.7 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 (Savelsbergh and Goetschalchx 1995). As our search procedure is designed for the general class of VRPs described in §1, and not tailored specifically to the VRPSD, we find these results to be promising.

7.4 Benefit of Updating Procedure

We demonstrate the computational savings of our proposed updating schemes on 10 of the problem instances considered by Christiansen and Lysgaard (2007). Table 4 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 4 by observing the CPU time required to evaluate each cyclic order in the neighborhood of π, where π is the radial ordering of customers.

Overall, the updating procedures provide substantial computational savings. 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 similar numbers of vehicles but different numbers of customers, we observe that as the number of customers increases, the computational savings also tend 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 reverse neighborhoods are, on average, 33 percent less than the savings afforded by the updating schemes for the k-shift and exchange neighborhoods. When updating $\mathcal{R}(\pi')$ from

Problem	**1-shift**	**5-shift**	**10-shift**	**Reverse**	**Exchange**
A-n38-k5	73.0	63.6	60.7	31.6	71.1
A-n44-k6	76.8	68.6	68.2	35.6	74.1
A-n45-k6	76.7	68.5	66.7	36.5	73.8
A-n53-k7	76.0	70.2	69.3	36.9	75.2
P-n20-k2	39.4	24.4	26.3	17.2	34.1
P-n21-k2	38.6	23.7	20.5	17.1	33.8
P-n22-k8	77.8	76.2	78.6	34.5	71.4
P-n23-k8	78.9	73.3	70.0	33.3	70.0
P-n50-k10	84.2	80.0	80.6	40.0	82.1
P-n50-k8	80.4	76.0	75.6	38.6	78.6
Average	**70.2**	**62.5**	**61.7**	**32.1**	**66.4**

Table 4: Percentage decreases in CPU time when $\mathcal{R}(\pi')$ is obtained by updating $\mathcal{R}(\pi)$

$\mathcal{R}(\pi)$, the reverse neighborhood requires 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. In general, 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 4 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 five 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 route subset $\mathcal{R}(\pi(i))$ in P-n22-k8 is generally smaller than the size of each route subset $\mathcal{R}(\pi(i))$ 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 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 relative to demand, the procedures outlined in Algorithms 2, 3, and 4 provide more of an advantage over totally reconstructing $\mathcal{R}(\pi')$.

8 Conclusion and Future Research Directions

This paper examines cyclic-order neighborhoods for a general class of 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 neighborhoods are capable of obtaining high quality a priori solutions for the VRPSD. Without tailoring our solution procedure to this specific routing problem, we are able to match 16 of 19 known optimal VRPSD solutions.

We suggest three possibilities for future research directions. First, it may be beneficial to investigate different methods to search the space of cyclic orders. One way to accomplish this is through search memory, which might exploit information contained in a history of candidate route sets or may identify common sequences of customers in a history of cyclic orders. Exploiting information pertaining to customer locations may also be helpful. For example, customers on opposite sides of the depot are unlikely to be serviced on the same route in an optimal solution. Thus, the effectiveness of the local search could possibly be improved by avoiding the consideration of neighbor cyclic orders that result in solutions with routes servicing distant customers. Such a criterion could serve as the basis for neighborhood reduction mechanisms such as "don't look bits" or candidate list strategies often employed in iterated local search (Bentley 1992) and tabu search (Glover and Laguna 1998). 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.

Second, it may be possible to expand the class of VRPs to which cyclic order neighborhoods apply (see §1) to include heterogeneous fleets of vehicles. This would likely require a modification to the graph structure of the shortest path problems used to extract the best solution from a set of candidate routes in Ryan et al. (1993) and Boctor and Renaud (2000).

Third, although this paper focuses on the VRPSD, we note that our methodology is applicable to a broader class of problems in which a minimum cost partition of a set of items must be obtained such that the cost of a given partition is separable in the individual parts and any constraints pertain to individual parts rather than collections of parts. Thus, cyclic-order neighborhoods may find application in multi-permutation and ordering problems beyond vehicle routing. Our intuition suggests that cyclic-order neighborhoods will be most effective for problems where the order of items within a part directly impacts the cost of the part. For example, this problem class includes some parallel machine scheduling problems (e.g., see Armentano and Filho 2007). If the objective is to minimize tardiness with respect to a set of due dates, then the order in which jobs are processed directly impacts the objective function. Some bin packing problems also belong to this problem class (see Lodi et al. 2002 for a review of recent advances in bin packing). However, because the value of a packing is typically independent of the packing order, cyclic orders obtained by simply rearranging the packing order may yield the same solution.

Acknowledgements

The authors wish to express appreciation to anonymous referees whose comments greatly improved the paper. Further, the authors also wish to express 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 the authors conducted preliminary computational experiments. The authors are also grateful for the use of a computer cluster provided by Saint Louis University. The computational experiments presented in the paper were carried out on this cluster.

References

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.

Armentano, V. and M. Filho (2007). Minimizing total tardiness in parallel machine scheduling with setup times: an adaptive memory-based GRASP approach. *European Journal of Operational Research 183*(1), 100–114.

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.

Bentley, J. (1992). Fast algorithms for geometric traveling salesman problems. *INFORMS Journal on Computing 4*(4), 387–411.

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

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

Boctor, F. and J. Renaud (2000). The column-circular, subsets-selection problem: complexity and solutions. *Computers and Operations Research 27*(4), 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*(0), 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*(1), 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*(6), 773–781.

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

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.

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

Glover, F. and M. Laguna (1998). *Tabu search*. Kluwer Academic Pub.

Hjorring, C. and J. Holt (1999). New optimality cuts for a single-vehicle stochastic routing problem. *Annals of Operations Research 86*(0), 569–584.

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.

Kirkpatrick, S., J. Gelatt, C., and M. Vecchi (1983). Optimization by simulated annealing. *Science 220*(4598), 671–680.

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.

Lodi, A., S. Martello, and D. Vigo (2002). Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics 123*(1-3), 379–396.

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, Lehigh University.

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

Rei, W., M. Gendreau, and P. Soriano (2010). A hybrid monte carlo local branching algorithm for the single vehicle routing problem with stochastic demands. *Transportation Science 44*(1), 136–146.

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*(3), 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*(2), 329–336.

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.

Secomandi, N. (2000). Comparing neuro-dynamic programming algorithms for the vehicle routing problem with stochastic demands. *Computers and Operations Research 27*(11-12), 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*(4), 321–352.

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

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*(4), 261–273.

Thompson, P. and J. Orlin (1989). Theory of cyclic transfers. Working paper, Operations Research Center, MIT.

Thompson, P. and H. Psaraftis (1993). Cyclic transfer algorithms for multi-vehicle routing and scheduling problems. *Operations Research 41*(5), 935–946.

Tillman, F. (1969). The multiple terminal delivery problem with probabilistic demands. *Transportation Science 3*(3), 192–204.

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