Optim Lett (2013) 7:1503–1516
DOI 10.1007/s11590-012-0555-8

ORIGINAL PAPER

A multi-space sampling heuristic for the vehicle routing problem with stochastic demands

Jorge E. Mendoza · Juan G. Villegas

Received: 29 September 2011 / Accepted: 1 September 2012 / Published online: 19 September 2012

Abstract The vehicle routing problem with stochastic demands consists in designing transportation routes of minimal expected cost to satisfy a set of customers with random demands of known probability distributions. This paper proposes a simple yet effective heuristic approach that uses randomized heuristics for the traveling salesman problem, a tour partitioning procedure, and a set partitioning formulation to sample the solution space and find high-quality solutions for the problem. Computational experiments on benchmark instances from the literature show that the proposed approach is competitive with the state-of-the-art algorithm for the problem in terms of both accuracy and efficiency. In experiments conducted on a set of 40 instances, the proposed approach unveiled four new best-known solutions (BKSs) and matched another 24. For the remaining 12 instances, the heuristic reported average gaps with respect to the BKS ranging from 0.69 to 0.15 % depending on its configuration.

Keywords Vehicle routing · Stochastic demands · Heuristics · Sampling · Logistics

Electronic supplementary material The online version of this article (doi:10.1007/s11590-012-0555-8) contains supplementary material, which is available to authorized users.

J. E. Mendoza (✉)
LUNAM Université, Université Catholique de l'Ouest, LISA (EA CNRS 4094),
3 Place André Leroy, 49008 Angers, France
e-mail: jorge.mendoza@uco.fr

J. G. Villegas
Departamento de Ingeniería Industrial, Facultad de Ingeniería, Universidad de Antioquia,
Calle 70 No. 52 - 21, 050010 Medellin, Colombia
e-mail: jvillega@udea.edu.co

1 Introduction

The vehicle routing problem with stochastic demands (VRPSD) can be defined on a complete and undirected graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{0, \ldots, n\}$ is the vertex set and $\mathcal{E} = \{(v, u) : v, u \in \mathcal{V}, v \neq u\}$ is the edge set. Vertices $v = 1, \ldots, n$ represent the customers and vertex $v = 0$ represents the depot. A distance d_e is associated with edge $e = (v, u) = (u, v) \in \mathcal{E}$, and it represents the travel cost between vertices v and u. Each customer v has a random demand ξ_v for a given product. Customer demands are met using an unlimited fleet of homogeneous vehicles with capacity Q located at the depot. The exact quantity demanded by each customer is not known until the vehicle arrives at the customer location. It is assumed, however, that each customer's demand follows an independent and known probability distribution and that all demand realizations (actual quantities) are nonnegative and less than the capacity of the vehicle.

Different frameworks can be applied to model and solve the VRPSD. In general, these frameworks are classified into *dynamic* or *static* approaches [24]. Routing decisions in dynamic approaches are made in multiple stages and are based on demand realizations. Given the remaining vehicle capacity and the set of unvisited customers, the routes are *re-optimized* at each stage. On the other hand, routing decisions made in static approaches remain unchanged regardless of the demand realizations. While dynamic strategies produce more accurate solutions, static strategies are preferred from the computational point of view since the problem, known to be computationally intractable, is solved only once [15]. Static strategies are also particularly useful when a stable solution is sought [4] or when re-optimization during route execution is impossible because of the lack of information. Among static frameworks, the most widely used in the literature, and the one selected for this research, is two-stage stochastic programming.

As the name suggests, in two-stage stochastic programming the problem is solved in two stages. In the first stage, a set $\overline{\mathcal{R}}$ of *planned* routes is designed. Each route $r \in \overline{\mathcal{R}}$ is a sequence of vertices $r = (0, v_1, \ldots, v_i, \ldots, v_{n_r}, 0)$, where $v_i \in \mathcal{V}\backslash\{0\}$ and n_r is the number of customers serviced by the route. During the planning phase, each route is designed so that the total expected demand does not exceed the capacity of the vehicle (i.e., $\sum_{v \in r\backslash 0} E[\xi_v] \leq Q \ \forall \ r \in \overline{\mathcal{R}}$) and every customer is visited by exactly one route. In the second stage, each route is executed until a *route failure* occurs, that is, the capacity of the vehicle is exceeded. A *recourse action* is then applied to recover the feasibility of the failing route. The recourse action is classically defined as a return to the depot to reload the vehicle, followed by a trip back to the customer location to complete the service. The route is then resumed from that point as originally planned. It is worth noting that the literature includes more sophisticated recourse actions (see for instance [1,9,27]). We decided to retain the classical policy because it is simple, suitable for many practical applications, and allows a more direct comparison with previously published results. The second-stage solution is then the actual set of routes traveled by the vehicles. The problem is to determine in the first stage the set of planned routes $\overline{\mathcal{R}}$ that minimizes the expected cost $E[C]$ of the second-stage solution given by:

$$E[C] = \sum_{r \in \overline{\mathcal{R}}} E\left[l_r + G_r(\boldsymbol{\xi})\right] = \sum_{r \in \overline{\mathcal{R}}} l_r + \sum_{r \in \overline{\mathcal{R}}} E\left[G_r(\boldsymbol{\xi})\right] \quad (1)$$

where l_r is the planned length (planned cost) and $E\left[G_r(\boldsymbol{\xi})\right]$ is the expected length of the return trips to the depot, or the cost of recourse, caused by route failures for each route $r \in \overline{\mathcal{R}}$. The planned cost of a route is the sum of the lengths of the arcs traversed by the route. On the other hand, the estimation of the expected cost of recourse is slightly more complicated. Under the selected recourse action, the expected cost of the failures of a route is given by:

$$E\left[G_r(\boldsymbol{\xi})\right] = 2 \times \sum_{i=2}^{n_r} \sum_{f=1}^{i-1} Pr\left(\sum_{j=2}^{i-1} \xi_{v_j} \leq f \cdot Q < \sum_{j=2}^{i} \xi_{v_j}\right) \times d_{v_i,0} \quad (2)$$

where the probability term represents the probability of having the fth failure while servicing customer v_i. The expected cost of failures in (2) can be efficiently computed when the customer demands follow a probability distribution Ψ with the following property: the sum of two independent and Ψ-distributed random variables is also Ψ distributed (as is the case for the normal, Poisson, and Gamma distributions). For the details of the derivation of (2) the reader is referred to [6,15,25].

The two-stage stochastic programming formulation with the classic recourse action presented above has been the main building block of solution methods for VRPs with stochastic demands. Exact methods based on this formulation include that of Laporte et al. [15] who proposed an implementation of the L-Shaped algorithm to solve a formulation with an additional constraint on the maximum number of vehicles. Their approach solved to optimality instances of up to 50 and 100 customers with normally and Poisson distributed demands, respectively. Later, Christiansen and Lysgaard [6] introduced a branch-and-price algorithm that reported optimal solutions for instances of up to 60 customers with Poisson distributed demands. In the field of heuristic approaches, Secomandi [23] proposed a rollout strategy built on top of the cyclic algorithm introduced by Bertsimas [4] for the single-vehicle VRPSD (SVRPSD), a variant in which a single route servicing all customers is to be designed. In addition to a formulation with classical recourse, this author also considered recourse with *restocking*, or preventive trips to the depot, introduced by Yang et al. [27]. Gendreau et al. [11] proposed a tabu search (TS) algorithm, known as *tabustoch*, designed for a VRPSD in which customers are present, or not, with a given probability (i.e., the customers are also stochastic). In this case, the classical recourse is extended to avoid customers that are not present. Mendoza et al. [16] generalized the formulation with classical recourse to consider the case in which each customer demands several incompatible products that are transported in different vehicle compartments. By setting the number of products and compartments to one, they obtain a classical VRPSD. More recently, Goodson et al. [13] introduced a set of neighborhood schemes to perform local search in the cyclic-order representation of a VRP solution [21]. They used their neighborhood schemes to build an effective simulated-annealing heuristic for the VRPSD which, to the best of our knowledge, finds the best-known solutions for most of the instances proposed by Christiansen and Lysgaard [6].

This paper introduces a new heuristic for the VRPSD that follows a two-phase solution strategy. In the first phase it samples multiple solution representation spaces, while in the second phase, it uses the sampled elements to build a solution to the problem. To accomplish its mission, the proposed heuristic combines various components from the literature, namely, a set of randomized heuristics for the traveling salesman problem, a tour partitioning procedure, and a set partitioning model. The paper also discusses computational experiments conducted on the 40-instance testbed of Christiansen and Lysgaard [6]. The experiments showed that the proposed heuristic is competitive with the state-of-the-art algorithm for the problem in terms of both accuracy and efficiency: the approach unveiled four new best-known solutions and matched another 24 in shorter execution times. In addition to accuracy and efficiency, the approach also offers simplicity and flexibility. As observed by Cordeau et al. [7], the latter two attributes foster the transfer of optimization technology to industry but are often neglected in academic research. Our heuristic is simple to understand and implement and can be extended to other vehicle routing problems. Last but not least, the paper also presents a detailed analysis of the contribution of each component to the overall performance of the method. We believe that this analysis provides the reader with valuable insight into the proposed method and how its principles can be used to design or enhance other vehicle routing heuristics.

The remainder of the paper is organized as follows. Section 2.1 reviews different heuristic-search strategies commonly applied to vehicle routing problems and discusses how our heuristic relates to those strategies. Section 2.2 presents a detailed description of our heuristic and its components. Section 3 discusses extensive computational experiments conducted on standard instances from the literature and presents the analysis of the components. Finally, Sect. 4 concludes the paper and outlines future research perspectives.

2 Multi-space sampling heuristic

2.1 Motivation

The ultimate goal of any vehicle routing heuristic is to find the best possible element s in the set of all feasible solutions to the problem $\mathcal{S}$. To accomplish this goal, routing heuristics may search $\mathcal{S}$ directly or may search alternative spaces and then map the result of the search to a solution in $\mathcal{S}$. The alternative search spaces that are commonly used in vehicle routing include: the set $\mathcal{T}$ of TSP-like tours (i.e., giant tours visiting all customers that start and end at the depot); the set $\mathcal{R}$ of all feasible routes (i.e., routes that verify all the side constraints of the problem); the set $\mathcal{C}$ of all clusters of customers from which feasible routes may be built; and the set $\mathcal{O}$ of cyclic orders (i.e., circular orderings of the set of customers).

Figure 1 depicts some examples of multi-space search strategies for vehicle routing. For instance, cluster-first route-second approaches, such as the sweep algorithm [12], select a subset of elements from $\mathcal{C}$ and then solve a series of TSP problems to map the selected elements to a solution in $\mathcal{S}$. In contrast, route-first cluster-second approaches

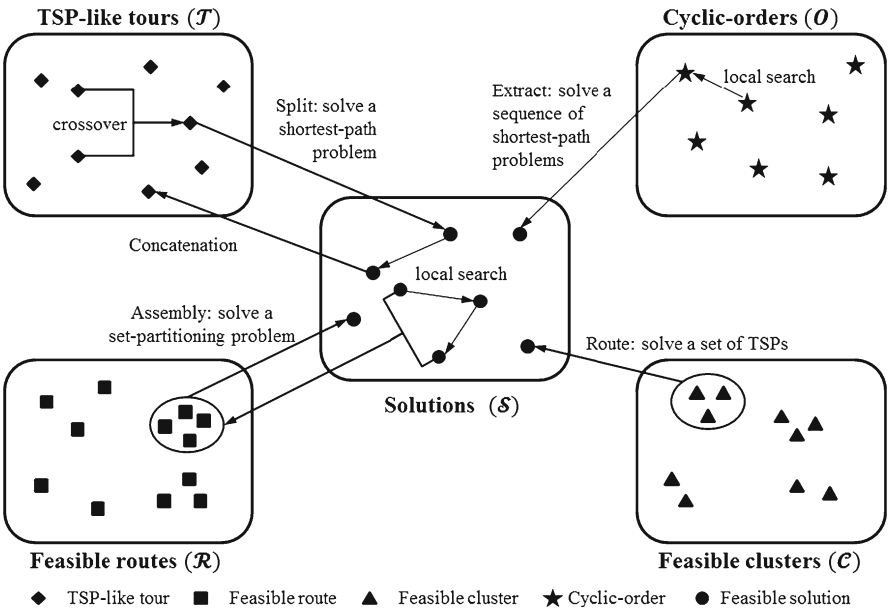

Fig. 1 Multi-space search strategies for vehicle routing problems

[2] use a TSP heuristic to select a tour in $\mathcal{T}$ and then split it into feasible routes to map it to a solution in $\mathcal{S}$. The closely related cyclic-order heuristics, such as that of Ryan et al. [21], build a cyclic order in $\mathcal{O}$ and then extract from it a solution by solving a sequence of shortest path problems. Petal heuristics, such as that of Foster and Ryan [10], build (one by one) feasible routes that are added to a set $\Omega \subset \mathcal{R}$ and then solve a set partitioning problem over Ω to map to a solution in $\mathcal{S}$.

More elaborate approaches map elements between different spaces in an iterative fashion; Fig. 1 also illustrates some of these approaches. One widely used strategy is to explore $\mathcal{S}$ using a (meta)heuristic and to map elite solutions found during the search (e.g., local optima) to a set $\Omega \subset \mathcal{R}$. The set Ω is later used in a post-optimization phase to map to a solution in $\mathcal{S}$ following the principle of petal heuristics. Some successful approaches that are based on this strategy include the TS for the capacitated VRP (CVRP) by Rochat and Taillard [20] and that by Kelly and Xu [14], and the GRASP for the truck-and-trailer routing problem by Villegas et al. [26]. A different strategy, recently introduced by Goodson et al. [13], consists in performing the search in the space of cyclic orders (i.e., $\mathcal{O}$) while iteratively mapping cyclic orders to solutions in order to obtain information that is used to guide the search. Other approaches not only map elements between spaces iteratively but also simultaneously explore multiple spaces. One good example is Prins' well-known evolutionary algorithm for the CVRP [18]. At each iteration, the algorithm selects two tours from $\mathcal{T}$ and applies a crossover operator to obtain a new tour. The new tour is split to map to a solution in $\mathcal{S}$, and then local search is applied to that solution. When the local search is completed, the new solution is mapped back to $\mathcal{T}$ using a concatenation procedure and the whole routine starts over.

The heuristic proposed in this paper takes advantage of existing procedures that are able to effectively map elements between different sets in the context of the VRPSD; it uses them to implement a new multi-space search strategy. The proposed approach works in two phases. In the first phase (*sampling*), the algorithm uses a set of randomized TSP heuristics (henceforth called *sampling heuristics*) to draw a biased sample from the set of TSP-like tours (i.e., $\mathcal{T}$). From each sampled tour, the approach extracts every feasible route that can be obtained without altering the order of the customers, and it uses these routes to build a set $\Omega \subset \mathcal{R}$. In the second phase (*assembly*), the approach follows the principle of petal heuristics and maps set Ω to a solution $s \in \mathcal{S}$ by solving a set partitioning formulation of the problem.

2.2 Detailed algorithm description

Algorithm 1 outlines the general structure of the proposed heuristic. The procedure starts by entering the sampling phase (lines 4–17). At each iteration $t \leq T$, the algorithm selects a sampling heuristic from a set $\mathcal{H}$ (line 6) and uses it to build a TSP tour p^t (line 7). Then, the algorithm makes a call to a procedure known as *s-split* (line 8) to retrieve a tuple $\langle \Omega^t, s^t \rangle$, where Ω^t is the set of all feasible routes that can be obtained from p^t without altering the order of the customers, and $s^t \in \mathcal{S}$ is the best VRPSD solution that can be built using routes from Ω^t. The routes in Ω^t join Ω (line 9), while s^t is used to update an upper bound $f(s^*)$ on the objective function of the final solution (lines 10–14). In the assembly phase (line 18), the heuristic invokes a procedure called *SetPartitioning* to solve a set partitioning formulation over Ω using $f(s^*)$ as an upper bound. The resulting solution $\overline{\mathcal{R}}$ is that reported by the heuristic (line 19).

Algorithm 1 Multi-space sampling heuristic: General structure

```
1: function MultiSpaceSampling(G,H,T)
2:     Ω ← ∅
3:     t ← 1
4:     while t ≤ T do
5:         for j = 1 to j = |H| do
6:             h ← H_j
7:             p^t ← h(G)
8:             ⟨Ω^t, s^t⟩ ← s-split(G, p^t)
9:             Ω ← Ω ∪ Ω^t
10:            if t = 1 then
11:                s* ← s^t
12:            else if f(s^t) < f(s*) then
13:                s* ← s^t
14:            end if
15:            t ← t + 1
16:        end for
17:    end while
18:    R̄ ← setPartitioning(G, Ω, s*)
19:    return R̄
20: end function
```

Figure 2 illustrates the operation of the proposed heuristic with $\mathcal{H} = \{h_1, h_2\}$ and $T = 2$. In the first iteration of the sampling phase (top of the leftmost frame) sampling heuristic h_1 builds $p^1 = (0, 1, 2, 3, 4, 5, 0)$. From p^1, the s-split procedure extracts a set Ω^1 made up of nine routes and a VRPSD solution

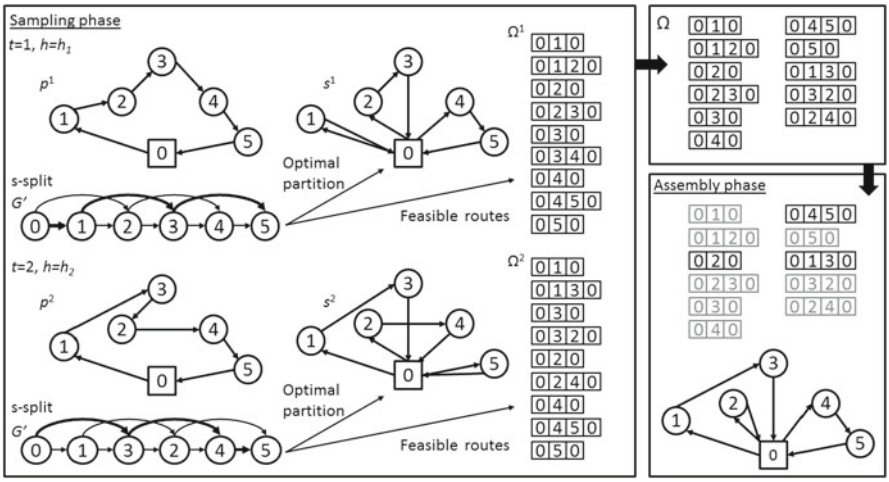

Fig. 2 Multi-space sampling heuristic: example of an execution with $\mathcal{H} = \{h_1, h_2\}$ and $T = 2$

$s^1 = \{(0, 1, 0), (0, 2, 3, 0), (0, 4, 5, 0)\}$. Similarly, in the second iteration, heuristic h^2 builds $p^2 = (0, 1, 3, 2, 4, 5, 0)$ from which s-split extracts a set Ω^2 made up of nine routes and $s^2 = \{(0, 1, 3, 0), (0, 2, 4, 0), (0, 5, 0)\}$. In the assembly phase (rightmost frames), the set partitioning selects from $\Omega = \Omega^1 \cup \Omega^2$ the subset of routes $\overline{\mathcal{R}} = \{(0, 2, 0), (0, 4, 5, 0), (0, 1, 3, 0)\}$ to assemble the final solution. The remainder of this section presents the components employed by the heuristic to accomplish each task.

2.2.1 Sampling heuristics

To sample $\mathcal{T}$ (line 7 in Algorithm 1), our heuristic uses randomized versions of four TSP constructive heuristics: randomized nearest neighbor (RNN), randomized nearest insertion (RNI), randomized best insertion (RBI), and randomized farthest insertion (RFI). We chose these heuristics based on their trade-off between implementation simplicity, accuracy, and computational performance [3]. Our implementation is based on the description of the original TSP heuristics presented in [19]. Although the strategies we used to generate the randomized versions of the four heuristics are rather intuitive, for the sake of completeness we briefly describe them here.

Let p^t be an ordered set representing the TSP tour being built by a given sampling heuristic at iteration t, $\mathcal{W}$ the set of vertices visited by p^t, and $\mathcal{N} = \mathcal{V} \backslash \mathcal{W}$ an ordered set of non-routed vertices. For the sake of simplicity, we assume that sets $\mathcal{W}$ and $\mathcal{N}$ are updated every time a customer is added to p^t. Let us also define three metrics for every customer $v \in \mathcal{N}$, namely, $d_{min}(v) = \min\{d_{(v,u)} | u \in \mathcal{W}\}$, $d_{max}(v) = \max\{d_{(v,u)} | u \in \mathcal{W}\}$, and $\Delta_{min}(v) = \min\{d_{(u,v)} + d_{(v,w)} - d_{(u,w)} | (u, w) \in p\}$. Finally, let k be a random integer in $\{1, \ldots, \min\{K, |\mathcal{N}|\}\}$, where parameter K denotes the *randomization factor* of each heuristic. The four sampling heuristics operate as follows:

- **RNN:** set $p^t = (0)$ and $u = 0$. At each iteration: identify the vertex v that is the kth nearest vertex to u, append v to p^t, and set $u = v$. Stop when $|\mathcal{N}| = 0$ and append 0 to p^t to complete a tour.
- **RNI:** initialize p^t as a tour starting at the depot and performing a round trip to a randomly selected customer (henceforth we will refer to this procedure simply as initialize p^t). At each iteration: sort $\mathcal{N}$ in non-decreasing order of $d_{min}(v)$. Insert $v = \mathcal{N}_k$ (i.e., the kth element in the ordered set $\mathcal{N}$) in the best possible position in the tour p^t (i.e., the position generating the smallest increment in the cost of the tour). Stop when $|\mathcal{N}| = 0$.
- **RFI:** initialize p^t. At each iteration: sort $\mathcal{N}$ in non-decreasing order of $d_{max}(v)$ and insert $v = \mathcal{N}_k$ in the best possible position in the tour p^t. Stop when $|\mathcal{N}| = 0$.
- **RBI:** Initialize p^t. At each iteration: sort $\mathcal{N}$ in non-decreasing order of $\Delta_{min}(v)$ and insert $v = \mathcal{N}_k$ in the best possible position in the tour p^t. Stop when $|\mathcal{N}| = 0$.

2.2.2 S-split

To extract $\langle \Omega^t, s^t \rangle$ from p^t (line 8 in Algorithm 1), our heuristic uses an adaptation of the s-split procedure for the VRPSD proposed in [17]. S-split builds a directed and acyclic graph $G' = (\mathcal{V}', \mathcal{A})$ composed of the ordered vertex set $\mathcal{V}' = (v_0, v_1, \ldots, v_i, \ldots, v_n)$ and the arc set $\mathcal{A}$. Vertex $v_0 = 0$ is an auxiliary vertex, while vertices $v_1, \ldots, v_n \in p^t \backslash \{0\}$. The vertices in $\mathcal{V}'$ are arranged in the same order in which they appear in p^t. Arc $(v_i, v_{i+n_r}) \in \mathcal{A}$ represents a feasible route r with expected cost $E[C_r]$ starting and ending at the depot and traversing the sequence of customers from v_{i+1} to v_{i+n_r}. The leftmost frame in Fig. 2 depicts two examples of auxiliary graphs G' built by s-split. To build the output, s-split adds to Ω^t one route for each arc in $\mathcal{A}$. On the other hand, to retrieve s^t, the procedure finds the set of arcs (i.e., routes) along the shortest path connecting 0 and v_n in G'. It is worth noting that since G' is directed and acyclic, building the graph and finding the shortest path can be done simultaneously using Bellman's algorithm [16]. It is also interesting to note that by transforming p^t into a cyclic order (i.e., eliminating the depot at the start and end of the tour) one could use the mapping procedure of Goodson et al. [13] to obtain a tuple $\langle \Omega^{t'}, s^{t'} \rangle$, where $\Omega^{t'} \supset \Omega^t$ and $f(s^{t'}) \leq f(s^t)$. After some preliminary experimentation, we discarded this variant because the gain in accuracy did not seem to offset the loss of simplicity. Although s-split is not an original contribution of this research, for the sake of completeness we have included a detailed pseudocode in Supplementary material 1.

2.2.3 Set partitioning

In the assembly phase, our approach maps the set $\Omega = \Omega^1 \cup \ldots \cup \Omega^T$ to a solution in $\mathcal{S}$ by solving a set partitioning formulation of the VRPSD proposed in [6] $\left(\min_{\overline{\mathcal{R}} \subseteq \Omega} \left\{ \sum_{r \in \overline{\mathcal{R}}} E[C_r] : \bigcup_{r \in \overline{\mathcal{R}}} = \mathcal{V}; r_i \bigcap r_j = \{0\} \, \forall r_i, r_j \in \overline{\mathcal{R}} \right\}\right)$. The objective is then to select the best subset of routes from Ω to build the set of planned routes $\overline{\mathcal{R}}$ (i.e., the final solution) guaranteeing that each customer will be visited by exactly one route. Since Ω does not have the column-circular structure [5], the resulting set

Table 1 Summary of results for the 40-instance testbed of Christiansen and Lysgaard

Performance metric	T					Goodson et al.
	1,000	2,000	5,000	10,000	20,000	
Avg. gap best (%)	0.36	0.26	0.14	0.09	0.05	0.01
Avg. gap (%)	0.67	0.49	0.31	0.23	0.15	0.33
Max. gap (%)	2.52	2.24	1.62	1.34	1.16	1.89
Avg. CPU (s)	12.90	22.75	50.19	93.43	180.78	268.66
Max. CPU (s)	45.00	82.05	179.61	314.09	782.77	603.80
Min. CPU (s)	0.69	1.00	1.90	3.29	5.91	9.00
Matched BKSs	12	15	18	22	24	37*
Improved BKSs	1	2	3	4	4	–

* 33 if the four new BKSs are considered

partitioning problem cannot be solved using a sequence of shortest path problems as in [21] or the linear relaxation of the formulation as in [10]. We therefore delegate this task to a commercial optimizer.

3 Computational experiments

We implemented the proposed heuristic in Java (jre V.1.6.0_22-b04) and used the Gurobi Optimizer (version 4.5.1) to solve the set partitioning model. To test our approach, we ran it on the 40-instance testbed proposed by Christiansen and Lysgaard [6]. These instances range from 16 to 60 customers and assume Poisson distributed demands. To assess the effectiveness of our heuristic, we compared our results to the best-known solutions (BKSs) for the testbed which, to the best of our knowledge, were by [6,13], or [17], or by all three. It is worth mentioning that [6] provided optimality certificates for 19 of the 40 instances. For each instance, we executed ten runs with five different values for parameter T: 1,000, 2,000, 5,000, 10,000, and 20,000. For the 2,000 runs ($=40 \times 10 \times 5$) we set the value of K to six for RNI, RBI, and RFI and to three for RNN. All the experiments were run on a PC with an Intel Xeon processor running at 2.4 GHz under Windows Server 2008 (64 bits) with 12 GB of RAM.

Table 1 summarizes the results of the experiments: the first column describes the performance metric, the following five columns report the results for the five different algorithm configurations, and column six presents the results for the simulated annealing approach of Goodson et al. [13] which to our knowledge is currently the best-performing heuristic approach for the VRPSD. For columns 2–7, the table reports: (1) the average gap between the best solution found for each instance and the BKS; (2) the average and maximum gaps with respect to the BKS over the 400 ($=40\times10$) runs conducted for each configuration; (3) the average, maximum, and minimum running times over the 400 runs; (4) the number of BKSs matched over the 40 instances; and (5) the number of improved BKSs over the 40 instances. The gaps are computed as $(f(s) - f(ref))/f(ref)$, where $f(s)$ is the objective function of a given solution s

and $f(ref)$ is the objective function of the reference solution (e.g., the BKS). Detailed results for each instance are reported in Supplementary material 2–6.

The results show that our approach is competitive in terms of both accuracy and efficiency. In its most effective configuration (i.e., $T = 20{,}000$), our heuristic matched 24 of the 40 BKSs and improved another four (i.e., 10 % of the instances) while delivering solutions with an average gap of 0.15 %. Moreover, the latter figure reduces to just 0.05 % if we consider only the best solution found over the 10 runs for each instance. The results show that the SA of Goodson et al. [13] finds a larger number of BKSs (33 vs. 28) and reports a slightly better average gap when only best solutions are considered (0.01 vs. 0.05 %). Nonetheless, the results for the average and worst-case behavior over multiple runs (i.e., Avg. Gap and Max. Gap, respectively) suggest that our heuristic is more stable (i.e., finds close-to-BKS solutions more often). The data on the coefficient of variation of the objective function reported for the two approaches in columns 5 and 11 of the detailed results (Supplementary material 2–6) support this observation.

In terms of computational efficiency, the results suggest that our approach outperforms the state-of-the-art SA. In its most effective, but most expensive, configuration, our heuristic reports CPU times that are comparable to those reported by Goodson et al. [13]. In a faster configuration, i.e., $T = 10{,}000$, our algorithm is able to deliver solutions of a similar quality while investing (on average) less than half of the computational effort. Note that we did not scale the CPU times reported by Goodson et al. [13] to account for differences in the testing environment (programming language, operating system, processing power, etc.). However, their testing environment is in theory at least as powerful as ours, so we believe that our conclusion is valid.

In practice, the two-stage stochastic programming formulation for the VRPSD models medium-term decisions (e.g., a set of routes that will be used repeatedly); thus, rapid solution is desirable but not critical. On the other hand, in dynamic environments (see for instance [22]) speed becomes an important factor. The computational performance and stability exhibited by our heuristic, especially in fast configurations (i.e., $T = 1{,}000$ or $T = 2{,}000$), suggest that it could be embedded in methods that make dynamic routing decisions.

3.1 Component analysis

To gain insight into our approach, we collected several statistics during our experiments. These data allowed us to analyze the impact of the different components on the performance of our heuristic. The first analysis focuses on the impact of each phase on the effectiveness and efficiency of the approach. Figure 3a shows the average and maximal improvement of the solution reported at the end of the assembly phase with respect to the best solution found in the sampling phase (over the 400 runs conducted for each value of T). The results show that the ability of the assembly phase to improve the best solution from the sampling phase (i.e., s^*) decreases as T increases. This result is explained by the fact that drawing more samples during the sampling phase increases the chances of finding a good solution. Nonetheless, as shown in Fig. 3a, even at the largest value of T, the assembly phase is able to improve by about 3 % the best solu-

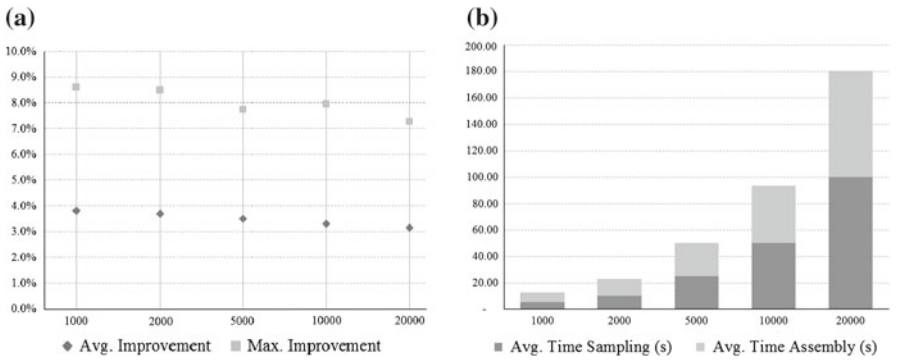

Fig. 3 Component analysis: impact of the sampling and assembly phases on the performance of the heuristic. Average values over the 400 runs conducted for each value of T

tion found during the sampling phase. The latter observation along with the results in Table 1 shed some light on the synergy between the two phases of our heuristic. On the one hand, an approach based only on drawing a large number of solutions using the sampling heuristics would not be competitive enough. On the other hand, in order to find better solutions, the assembly phase needs an extensive sampling phase that provides a rich pool of routes (i.e., Ω) as input.

Figure 3b shows the total average CPU time for each value of T (over the 400 runs conducted for each value of T) and how the execution time is split between the two phases of the heuristic. The results show that at any given value of T, the CPU time is equally distributed between the sampling and assembly phases.

The second analysis focuses on the ability of the sampling heuristics to contribute routes to Ω and to the final solution $\overline{\mathcal{R}}$ reported by our approach. To measure this contribution, we saved for each route in Ω a counter of the number of times that the route was generated by each sampling heuristic (i.e., was extracted from a TSP tour built by a given heuristic). Using these data, we calculated five metrics for each sampling heuristic. The first metric, *diversity ratio*, measures the ability of a sampling heuristic h to generate different routes; it is calculated as $\left(|\overline{\Omega}_h|/|\Omega_h|\right) \times 100$, where $|\overline{\Omega}_h|$ is the number of routes generated by h after we eliminate all duplicates, and $|\Omega_h|$ is the total number of routes generated by h. The second metric, *total contribution to* Ω = $\left(|\overline{\Omega}_h|/|\Omega|\right) \times 100$, measures the contribution of the heuristic to the pool of routes. Note that $\Omega = \bigcup_{h\in\mathcal{H}} \overline{\Omega}_h$ but $\overline{\Omega}_h \cap \overline{\Omega}_{h'} \neq \emptyset \; \forall \, h, h' \in \mathcal{H}, h \neq h'$. The third metric, *exclusive contribution to* Ω, measures the percentage of routes exclusively generated by h. The metric is calculated as $\left(|\overline{\Omega}'_h|/|\Omega|\right) \times 100$, where $\overline{\Omega}'_h = \overline{\Omega}_h \setminus \bigcup_{h'\in\mathcal{H}\setminus\{h\}} \overline{\Omega}_{h'}$. The fourth metric, *routes in* $\overline{\mathcal{R}}$, measures the ability of sampling heuristic h to contribute routes to the final solution. It is computed as $\left(|\overline{\mathcal{R}}_h|/|\overline{\mathcal{R}}|\right) \times 100$, where $|\overline{\mathcal{R}}_h|$ is the number of routes generated by h that appear in the final solution. Note that $\overline{\mathcal{R}}_h \cap \overline{\mathcal{R}}_{h'}$ for any $h, h' \in \mathcal{H}, h \neq h'$ is not necessarily empty. Finally, the fifth metric, *exclusive routes in* $\overline{\mathcal{R}}$, measures the percentage of routes in the final solution that were exclusively generated by h. The metric is calculated as $\left(|\overline{\mathcal{R}}'_h|/|\overline{\mathcal{R}}|\right) \times 100$, where

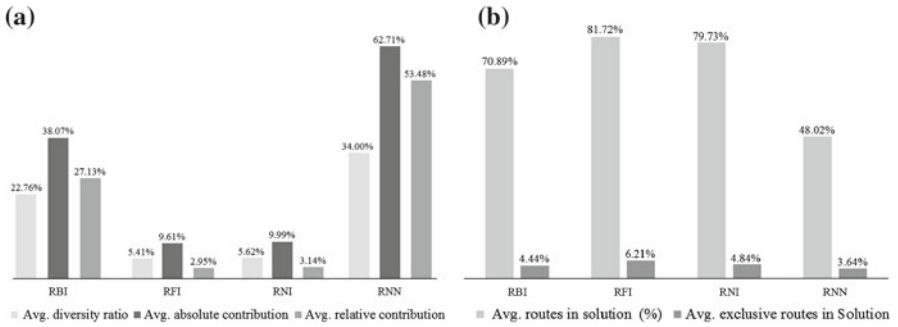

Fig. 4 Component analysis: ability of the sampling heuristics to contribute routes to Ω and to the final solution. Average values over ten runs for $T = 10{,}000$

$\overline{\mathcal{R}}'_h = \overline{\mathcal{R}}_h \setminus \bigcup_{h' \in \mathcal{H} \setminus \{h\}} \overline{\mathcal{R}}_{h'}$. Figure 4a and b present, for each metric, the average value over the forty test instances for $T = 10{,}000$.

The results in Fig. 4a show that RNN is the sampling heuristic with the highest diversity ratio (i.e., 34.00 %). In other words, RNN is the heuristic that is most sensitive to randomization. A plausible explanation for this behavior is that RBI (22.76 %), RFI (5.41 %), and RNI (5.62 %) always insert the selected node into its best possible position in the TSP tour being built, which is often the same position regardless of the state of the tour. The internal ability of the sampling heuristics to generate diverse routes is reflected in the pool Ω. The data show that 62.71 % of the routes in the pool were generated at least once by RNN while this figure is 38.07, 9.61, and 9.99 % for RBI, RFI, and RNI, respectively. A close look at the data on the percentage of routes exclusively generated by each heuristic reveals that the sampling heuristics play a complementary role in the construction of a diverse pool of routes. According to our observations, in a typical (average) pool 53.48 % of the routes are exclusively generated by RNN, and 27.13, 3.14, and 2.95 % of the routes are exclusively generated by RBI, RFI, and RNI, respectively. On the other hand, in a typical pool, only 13.30 % of the routes are generated by more than one sampling heuristic. In summary, this analysis shows that including different randomized heuristics in the sampling phase leads to a pool of routes that is more diverse than the pools built independently by each heuristic.

The results in Fig. 4b reveal that although RFI, RBI, and RNI generate fewer routes than RNN does, the former have a greater impact on the final solutions. For instance, 6.21 % of the routes in the final solutions are exclusive to RFI. In other words, eliminating RFI from $\mathcal{H}$ would have caused our heuristic to miss 6 % of the routes that appeared in its high-quality final solutions. A similar observation can be made about RBI and RNI. Based on the contribution of RFI and RNI to the final solutions (81.72 and 79.73 %, respectively) we decided to perform a simple experiment (for $T = 10{,}000$) with $\mathcal{H} = \{RFI\}$ and $\mathcal{H} = \{RNI\}$. The version of the heuristic that uses only RFI gave solutions with an average gap of 0.32 % with respect to the BKSs, and that using only RNI produced results with an average gap of 0.37 %. In both cases the results are dominated by those obtained by the original multi-space sampling heuristic. In conclusion, using different sampling heuristics in the sampling phase fosters diversity

and contributes to the effectiveness of the approach. This is an important observation since VRP heuristics that are based on drawing samples from the solution space (e.g., GRASP) traditionally incorporate only one randomized heuristic.

4 Concluding remarks and future research

This paper introduces an effective multi-space sampling heuristic for the VRPSD. The approach uses three components—a set of four randomized heuristics for the TSP, a tour partitioning procedure, and a set partitioning formulation—to sample the solution space and find solutions. Experiments conducted on a set of forty instances from the literature showed that our heuristic is competitive with the state-of-the-art metaheuristic in terms of both solution quality and computational efficiency. Our approach found four new BKSs for the testbed and matched another 24. For the remaining 12 instances, the heuristic reported average gaps with respect to the BKSs ranging from 0.15 to 0.69 % depending on its configuration.

Our heuristic was also designed to be simple and flexible. It is simple to understand and implement and can be extended to other VRPs. The feasibility and cost of the solutions are controlled at two points in the heuristic: (1) when we extract routes from the TSP tour in the sampling phase and (2) when we solve the set partitioning model in the assembly phase. The literature discusses a number of splitting procedures designed to map TSP-like tours to solutions in the context of different VRP variants (for a comprehensive review, the reader is referred to [8]). It is, in general, fairly easy to adapt these procedures (as we adapted s-split in this research) so that they return not only the optimal partition of the tour but also the set of routes evaluated during the splitting. One could use these adapted procedures (or develop new ones) to build pools of feasible routes for different VRP variants during the sampling phase. As for the assembly phase, the set partitioning model described in Section 2.2.3 fits several VRP variants and thus may remain untouched. However, if there are additional constraints on the solution (e.g., a maximum number of routes) or an objective function that is not given by the sum of the cost of the individual routes, the model can be extended (at the risk of making the problem intractable for the optimizer). Research currently underway builds on this flexibility; it includes the extension of our approach to the VRPSD with a heterogeneous fleet and the VRP with stochastic and correlated demands.

Acknowledgments The authors would like to thank two anonymous referees for insightful comments and valuable advice that helped to improve the paper. This research was partially funded by *Region Pays de la Loire (France)* through project LigéRO and by the *Universidad de Antioquia (Colombia)* through project: *Desarrollo de metaheurísticos híbridos y métodos cooperativos para problemas de rutas de vehículos con flota heterogénea y restricciones adicionales*.

References

1. Ak, A., Erera, A.: A paired-vehicle recourse strategy for the vehicle-routing problem with stochastic demands. Transp. Sci. **41**(2), 222–237 (2007)
2. Beasley, J.E.: Route-first cluster-second methods for vehicle routing. Omega **11**, 403–408 (1983)
3. Bentley, J.B.: Fast algorithms for geometric traveling salesman problems. INFORMS J. Comput. **4**, 387–411 (1992)

4. Bertsimas, D.: A vehicle routing problem with stochastic demand. Oper. Res. **40**(3), 574–585 (1992)
5. Boctor, F.F., Renaud, J.: The column-circular, subsets-selection problem: Complexity and solutions. Comput. Oper. Res. **27**(4), 383–398 (2000)
6. Christiansen, C., Lysgaard, J.: A branch-and-price algorithm for the capacitated vehicle routing problem with stochastic demands. Oper. Res. Lett. **35**(6), 773–781 (2007)
7. Cordeau, J.F., Gendreau, M., Laporte, G., Potvin, J.Y., Semet, F.: A guide to vehicle routing heuristics. J. Oper. Res. Soc. **53**(5), 512–522 (2002)
8. Duhamel, C., Lacomme, P., Prodhon, C.: Efficient frameworks for greedy split and new depth first search split procedures for routing problems. Comput. Oper. Res. **38**(4), 723–739 (2011)
9. Erera, A., Savelsbergh, M., Uyar, E.: Fixed routes with backup vehicles for stochastic vehicle routing problems with time constraints. Networks **54**(4), 270–283 (2009)
10. Foster, B.A., Ryan, D.M.: An integer programming approach to the vehicle scheduling problem. Oper. Res. Q. **27**(2), 367–384 (1976)
11. Gendreau, M., Laporte, G., Séguin, R.: A tabu search heuristic for the vehicle routing problem with stochastic demands and customers. Oper. Res. **44**(3), 469–477 (1996)
12. Gillett, B.E., Miller, L.R.: A heuristic algorithm for the vehicle dispatch problem. Oper. Res. **22**(2), 340–349 (1974)
13. Goodson, J.C., Ohlmann, J.W., Thomas, B.W.: Cyclic-order neighborhoods with application to the vehicle routing problem with stochastic demand. Euro. J. Oper. Res. **217**(2), 312–323 (2012)
14. Kelly, J.P., Xu, J.: A set-partitioning-based heuristic for the vehicle routing problem. INFORMS J. Comput. **11**(2), 161–172 (1999)
15. Laporte, G., Louveaux, F., Van Hamme, L.: An integer L-shaped algorithm for the capacitated vehicle routing problem with stochastic demands. Oper. Res. **50**(3), 415–423 (2002)
16. Mendoza, J.E., Castanier, B., Guéret, C., Medaglia, A.L., Velasco, N.: A memetic algorithm for the multi-compartment vehicle routing problem with stochastic demands. Comput. Oper. Res. **37**(11), 1886–1898 (2010)
17. Mendoza, J.E., Castanier, B., Guéret, C., Medaglia, A.L., Velasco, N.: Constructive heuristics for the multicompartment vehicle routing problem with stochastic demands. Transp. Sci. **45**(3), 335–345 (2011)
18. Prins, C.: A simple and effective evolutionary algorithm for the vehicle routing problem. Comput. Oper. Res. **31**(12), 1985–2002 (2004)
19. Reinelt, G.: Construction heuristics. In: The traveling salesman, lecture notes in computer science, vol. 840, pp. 73–99. Springer, Berlin (1994)
20. Rochat, Y., Taillard, E.D.: Probabilistic diversification and intensification in local search for vehicle routing. J. Heuristics **1**, 147–167 (1995)
21. Ryan, D.M., Hjorring, C., Glover, F.: Extensions of the petal method for vehicle routeing. J. Oper. Res. Soc. **44**(3), 289–296 (1993)
22. Secomandi, N.: A rollout policy for the vehicle routing problem with stochastic demands. Oper. Res. **49**(5), 796–804 (2001)
23. Secomandi, N.: Analysis of a rollout approach to sequencing problems with stochastic routing applications. J. Heuristics **9**(4), 321–352 (2003)
24. Secomandi, N., Margot, F.: Reoptimization approaches for the vehicle-routing problem with stochastic demands. Oper. Res. **57**(1), 214–230 (2009)
25. Teodorović, D., Pavković, G.: A simulated annealing technique approach to the vehicle routing problem in the case of stochastic demands. Transp. Plan. Technol. **16**(4), 261–273 (1992)
26. Villegas, J.G., Prins, C., Prodhon, C., Medaglia, A.L., Velasco, N.: A matheuristic for the truck and trailer routing problem. Department of Industrial Engineering, Universidad de Antioquia, Technical report (2011)
27. Yang, W.H., Mathur, K., Ballou, R.: Stochastic vehicle routing with restocking. Transp. Sci. **34**(1), 99–112 (2000)