PERGAMON

Computers & Operations Research 27 (2000) 1201–1225

computers & operations research

www.elsevier.com/locate/dsw

Comparing neuro-dynamic programming algorithms for the vehicle routing problem with stochastic demands

Nicola Secomandi*

Department of Decision and Information Sciences, College of Business Administration, University of Houston, Houston, TX 77204-6282, USA

Abstract

The paper considers a version of the vehicle routing problem where customers' demands are uncertain. The focus is on dynamically routing a single vehicle to serve the demands of a known set of geographically dispersed customers during real-time operations. The goal consists of minimizing the expected distance traveled in order to serve all customers' demands. Since actual demand is revealed upon arrival of the vehicle at the location of each customer, fully exploiting this feature requires a dynamic approach.

This work studies the suitability of the emerging field of neuro-dynamic programming (NDP) in providing approximate solutions to this difficult stochastic combinatorial optimization problem. The paper compares the performance of two NDP algorithms: optimistic approximate policy iteration and a rollout policy. While the former improves the performance of a nearest-neighbor policy by 2.3%, the computational results indicate that the rollout policy generates higher quality solutions. The implication for the practitioner is that the rollout policy is a promising candidate for vehicle routing applications where a dynamic approach is required.

Scope and purpose

Recent years have seen a growing interest in the development of vehicle routing algorithms to cope with the uncertain and dynamic situations found in real-world applications (see the recent survey paper by Powell et al. [1]). As noted by Psaraftis [2], dramatic advances in information and communication technologies provide new possibilities and opportunities for vehicle routing research and applications. The enhanced capability of capturing the information that becomes available during real-time operations opens up new research directions. This informational availability provides the possibility of developing dynamic routing algorithms that take advantage of the information that is dynamically revealed during operations. Exploiting such information presents a significant challenge to the operations research/management science community.

* Current affiliation and address: PROS Strategic Solutions, 3223 Smith Street, Suite 100, Houston, TX, 77006, USA. Tel.: + 1-713-335-5840; fax: + 1-713-523-8144.

E-mail address: nsecomandi@prosweb.com (N. Secomandi)

The single vehicle routing problem with stochastic demands [3] provides an example of a simple, yet very difficult to solve exactly, dynamic vehicle routing problem [2, p. 157]. The problem can be formulated as a stochastic shortest path problem [4] characterized by an enormous number of states.

Neuro-dynamic programming [5,6] is a recent methodology that can be used to approximately solve very large and complex stochastic decision and control problems. In this spirit, this paper is meant to study the applicability of neuro-dynamic programming algorithms to the single-vehicle routing problem with stochastic demands. © 2000 Elsevier Science Ltd. All rights reserved.

Keywords: Stochastic vehicle routing; Neuro-dynamic programming; Rollout policies; Heuristics

1. Introduction

The deterministic vehicle routing problem (VRP) is well studied in the operations research literature (see [3,7–10] for reviews). Given a set of geographically dispersed customers, each showing a positive demand for a given commodity, VRP consists of finding a set of tours of minimum length for a fleet of vehicles located at a central depot, such that customers' demands are satisfied and vehicles' capacities are not exceeded.

This work deals with a variation where customers' demands are uncertain and are assumed to follow given discrete probability distributions. This situation arises in practice whenever a distribution company faces the problem of deliveries to a set of customers, whose demands are uncertain. This work focuses on deliveries, but all the discussion carries through in case of collections. The problem of finding a tour through the customers that minimizes expected distance traveled is known as the vehicle routing problem with stochastic demands (SVRP for short) [11–14]. Other authors have considered the additional case where the presence of customers is also uncertain (i.e., customers are present with a known probability). This variation is known as the vehicle routing problem with stochastic customers and demands [12,14–16]. This paper only considers the case of stochastic demands.

Stochastic vehicle routing problems may be employed to model a number of business situations that arise in the area of distribution. Some examples follow (note that some of the references do not necessarily indicate a stochastic or dynamic formulation). In a strategic planning scenario, after having identified some major customers a distribution company might be interested in estimating the expected amount of travel to serve these customers on a typical day [12]. Central banks face the problem of delivering money to branches or automatic teller machines whose requirements vary on a daily basis [17]. In less-than-truckload operations the amount of goods to be collected on any given day from a set of customers is generally uncertain [15]. Other examples of applications reported in the literature include the delivery of home heating oil [18], sludge disposal [19], the design of a "hot meals" delivery system [20], and routing of forklifts in a cargo terminal or in a warehouse [12]. Psaraftis [2] provides examples of more "dynamic" situations, where the time dimension is an essential feature of the problem: courier services, intermodal services, tram ship operations, combined pickup and delivery services, and management of container terminals.

Most works in the literature [14] dealing with versions of the stochastic VRP focus on computing a fixed order of customers' visitation, eventually supplemented by decision rules on when to return to the depot to replenish, in anticipation of route failures. A route failure occurs when, due to the variability associated with customers' demands, the vehicle capacity is exceeded. Recourse actions are taken when a route failure occurs. A popular recourse action employed in the literature consists of having the vehicle return to the depot, replenish and resume its tour from the location where the failure occurred.

More specifically, properties and formulations of the problem are investigated by Dror et al. [11], Dror [21], and Bastian and Rinnooy Kan [22]. Bertsimas [12] proposes an asymptotically optimal heuristic, and an expression to efficiently compute the expected length of a tour. Bertsimas et al. [13] investigate the empirical behavior of this heuristic, and enhance it by using dynamic programming (DP) to supplement the a priori tour with rules for selecting return trips to the depot. Yang et al. [23] extend this approach to the multi-vehicle case when the length of each tour cannot exceed a given threshold. Secomandi [24] develops rollout algorithms to improve on any initial tour. A simulated annealing heuristic is proposed by Teodorović and Pavković [25]. Other heuristics are developed by Dror and Trudeau [26], Dror et al. [27], and Bouzaïene-Ayari et al. [28]. Gendreau et al. [15] propose an exact algorithm based on the integer L-shaped method of Laporte and Louveaux [29] for the case of both stochastic customers and demands. They also derive a DP-based algorithm to compute the expected length of a tour. The same authors [16] develop a tabu search heuristic to be used in case of large size instances. The chance constrained version of the problem is considered by Golden and Yee [30], Stewart and Golden [31], and Laporte et al. [32].

These approaches, with the partial exception of Bertsimas et al. [13], Yang et al. [23], and Secomandi [24] treat the problem as static, whereby the order of customers' visitation is not changed during its real-time execution when the actual demands' realizations become available.

The recent survey by Psaraftis [2] emphasizes the effect on the field of vehicle routing of dramatic advances in computer-based technology, such as electronic data interchange, geographic information systems, global positioning systems, and intelligent vehicle-highway systems. He stresses the concept that the enhanced availability of real-time data allowed by these technologies opens new challenges and research directions for the operations research/management science community. In particular, such technologies provide the necessary informational support for developing dynamic vehicle routing algorithms that can exploit the wealth of information that becomes available during real-time operations. The work of Powell et al. [1] provides an excellent review of dynamic and stochastic routing problems from an operations research/management science perspective.

In this context, Psaraftis presents the work of Dror et al. [11] as the first dynamic formulation of SVRP. He considers this model as a simple, yet difficult to solve exactly, dynamic routing problem. These authors develop a large-scale Markov decision process (MDP) model for SVRP, but no computational experience is provided. A slightly modified version of this model is discussed by Dror [21]. No computational experience is performed, and the author considers instances with more than three customers as computationally intractable (a comment also made by Dror et al. [27]). Secomandi [24] reformulates the problem as a stochastic shortest path problem (SSPP) and by exploiting a state-space decomposition develops an exact DP algorithm that can solve small instances (about 10 customers) to optimality. He also derives properties of the optimal policy.

Neuro-dynamic programming, also known as reinforcement learning (NDP for short), is a recent methodology that deals with the approximate solution of large and complex DP problems. (See the recent books by Bertsekas and Tsitsiklis [5] and by Sutton and Barto [6] for excellent presentations of NDP.) NDP combines simulation, learning, approximation architectures (e.g., neural networks) and the central ideas of DP to break the curse of dimensionality typical of DP.

In this spirit, the goal of this work is to study the applicability of the NDP methodology to the SSPP formulation of the single-vehicle SVRP in order to compute a suboptimal dynamic routing policy. The large-scale SSPP formulation analyzed in Secomandi [24] is adopted. The paper compares two NDP algorithms based on the approximate policy iteration (API) scheme: optimistic API and a rollout policy (RP). (The reader is referred to Secomandi [24] for a discussion of the theoretical properties of RP when applied to SVRP.)

Although a simplified model of a potential real-world distribution application, the single-vehicle SVRP does present some features that characterize it as an interesting dynamic routing problem. In a real-world application a distribution manager is likely to operate a fleet of vehicles. In this work the following assumption holds throughout: a set of customers has been assigned to be served by a given vehicle. Therefore, the multi-vehicle problem decomposes into independent single-vehicle subproblems. A scheme of this type has been proposed in the context of a stochastic and dynamic routing problem discussed by Bertsimas and van Ryzin [33–35]. In their work, requests for service arriving over time are assigned to the log of individual vehicles, effectively reducing the routing part of the problem to a situation somewhat similar to that studied here.

The paper is organized as follows. Section 2 introduces the problem. Section 3 presents a SSPP formulation for SVRP. Section 4 discusses the NDP approach. The NDP methodology is applied to SVRP in Section 5. Experimental analysis of the algorithms is conducted in Section 6. Section 7 concludes the paper.

2. Problem description

2.1. Notation and assumptions

Let the set of nodes of a given complete network be $\{0, 1, \dots, n\}$. Node 0 denotes the depot and $V = \{1, 2, \dots, n\}$ represents the set of customers' locations. Distances $d(i, j)$ between nodes are assumed to be known and symmetric, and they satisfy the triangle inequality: $d(i, j) \leqslant d(i, k) + d(k, j)$. Q denotes vehicle capacity. A single vehicle is assumed. This is equivalent to assuming that a set of customers has been assigned to receive service by a given vehicle. Let D_i, $i = 1, 2, \dots, n$, be the random variable that describes the demand of customer i. The probability distribution of D_i is discrete and known, and is denoted by $p_i(k) \equiv \Pr\{D_i = k\}$, $k = 0, 1, \dots, K \leqslant Q$. Customers' demands are assumed to be stochastically independent, and their realizations become known upon the first arrival of the vehicle at each customer location. The vehicle is initially located at the depot. During service, when capacity is reached or exceeded, a return trip to the depot is performed in order to restore capacity up to Q. This implies that the depot capacity is assumed to be at least equal to nK. When all demands have been satisfied, the vehicle returns to the depot. The objective is to determine a service/routing policy such that demand at each node is met, and expected distance traveled is minimized.

Fig. 1 provides a pictorial illustration of the problem. Customers are arranged in a network (which in the figure is not complete) and are represented by the solid nodes. The "piles" associated

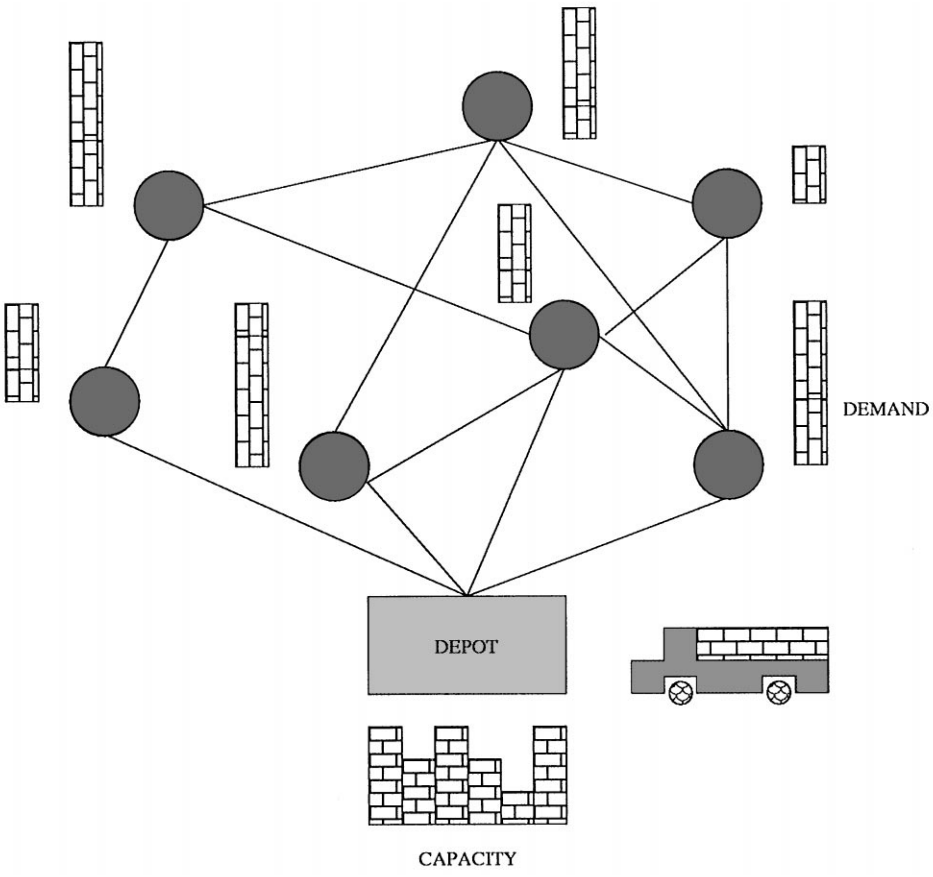


Fig. 1. A pictorial representation of the problem.

with each node in the network represent the uncertain amount of commodity to be delivered there. The depot is also shown together with the amount of available commodity and the vehicle. The limited capacity of the vehicle is emphasized.

2.2. Types of policies

One can envision the following types of routing policies: static, dynamic, and mixed. Static policies prescribe a sequence τ of customers (a tour through the nodes that starts and ends at the depot) to be visited in that order by the vehicle. In case of a route failure, a recourse action of the type described earlier is taken [12,13,15,16]. Dynamic policies provide a policy π that given the current state of the system prescribes which location should be visited next [11,21,24]. Mixed policies combine elements of both static and dynamic policies: e.g., the vehicle follows a tour, but is also enabled with state-dependent rules that allow for early replenishments [13,23,24].

Figs. 2–4 illustrate the different types of policies. Figs. 2 and 3 only differ in the "timing" of the replenishments: these are reactive in case of a static policy and proactive in case of a mixed policy.

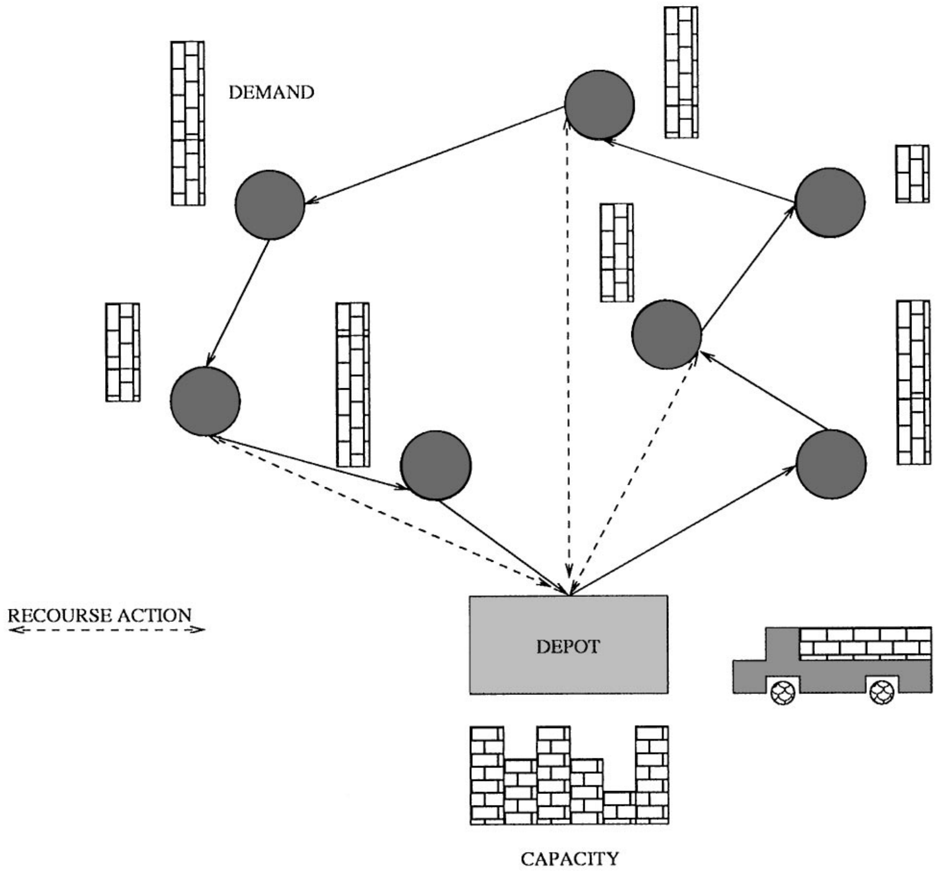


Fig. 2. An illustration of static policies.

Fig. 4 emphasizes the dynamism of the decision-making activity typical of a dynamic policy, whereby the location to be visited next is a function of the current state of the system.

It is easy to show that optimal dynamic policies provide a smaller expected distance, followed by optimal mixed and static policies, in that order [24]. The focus of the present work is on computing approximate dynamic policies.

3. A stochastic shortest path formulation

This section presents a stochastic shortest path problem (SSPP) formulation of SVRP. (Properties of this formulation are derived in Secomandi [24].) SSPPs are a subclass of MDP models [4,5] describing a discrete-time dynamic system whose state transition depends on a control. At state x a control u must be chosen from a given set $U(x)$. The choice of u specifies the transition probability $p_{xx'}(u)$ to the next state x'. Transition costs are denoted by $g(x, u, x')$. SSPPs are characterized by the existence of a cost-free termination state. Once the system reaches that state, it remains there at no further cost. The number of transitions from the initial to the termination state

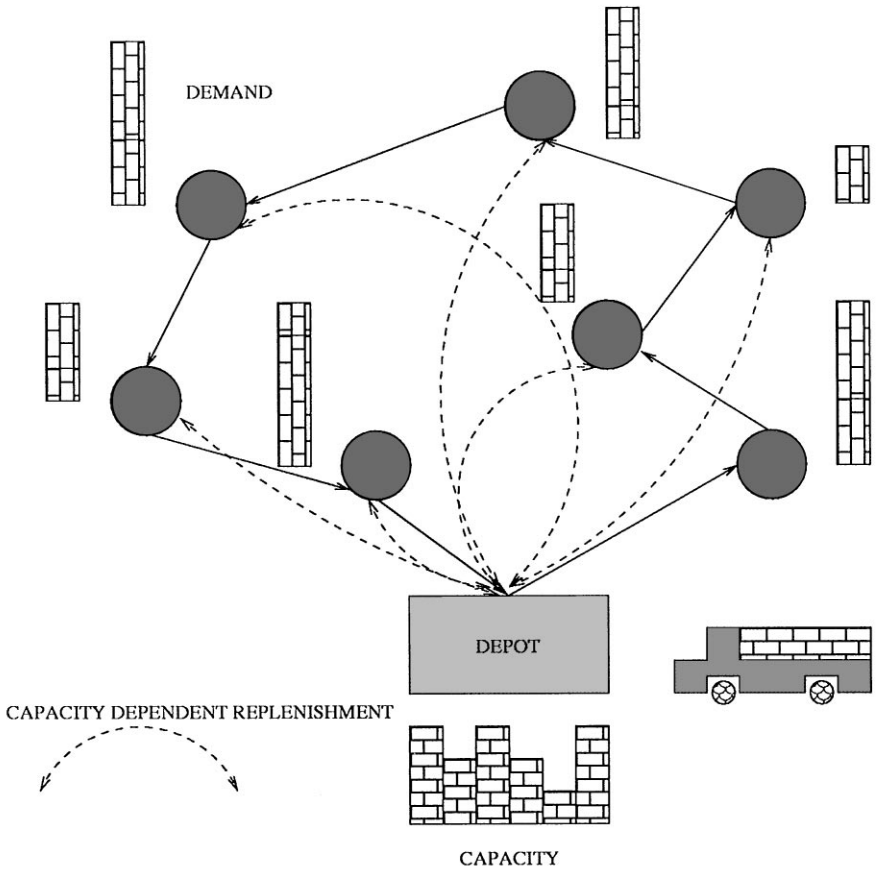


Fig. 3. An illustration of mixed policies.

is a random variable N, governed by the random variables defining the problem and the controls being used. Let x_k denote the state the system is in at stage k; x_N represents the termination state. A policy is a sequence $\pi = \{\mu_0, \mu_1, \dots, \mu_{N-1}\}$; μ_k is a function that maps states into controls with $\mu_k(x_k) \in U_k(x_k)$ for all states x_k. The objective is to find a policy that minimizes

$$J_N^{\pi}(x) = E\left[\sum_{k=0}^{N-1} g(x_k, \mu(x_k), x_{k+1}) \,|\, x_0 = x\right] \tag{1}$$

for all states $x \in S$, where S is the state space. Here the expectation is taken with respect to the probability distribution of the Markov chain $\{x_0, x_1, \dots, x_N\}$. This depends on the initial state and the policy π. The optimal N-stage cost-to-go from state x is

$$J_N^*(x) = \min_{\pi} J_N^{\pi}(x). \tag{2}$$

It is well known [4] that $J_N^*(x)$ satisfies Bellman's equation

$$J_N^*(x) = \min_{u_k \in U_k(x_k)} \sum_{x_{k+1} \in S} p_{x_k x_{k+1}}(u_k)\{g(x_k, u_k, x_{k+1}) + J_N^*(x_{k+1}) \,|\, x_k = x\} \tag{3}$$

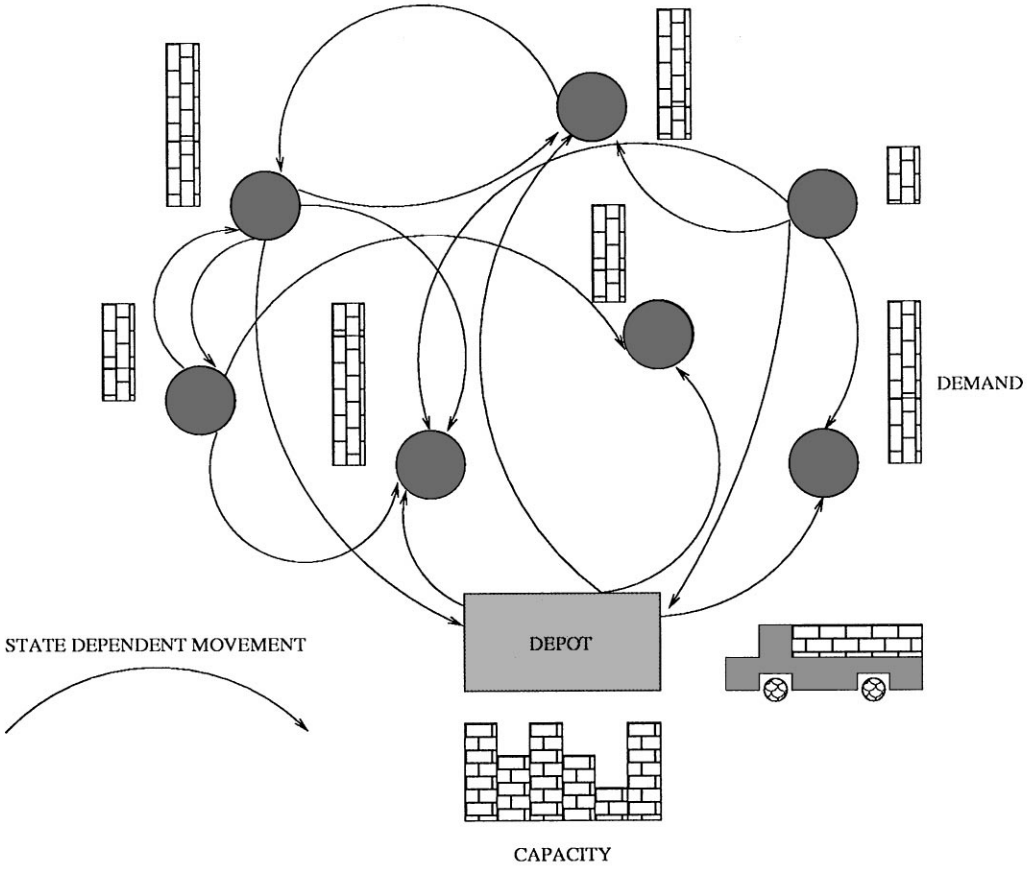


Fig. 4. An illustration of dynamic policies.

for all states $x \in S$. If one knew $J_N^*(x)$ for all the states, then an optimal control at each state x could be computed by performing the following minimization:

$$\mu_k^*(x) = \arg \min_{u_k \in U_k(x_k)} \sum_{x_{k+1} \in S} p_{x_k x_{k+1}}(u_k)\{g(x_k, u_k, x_{k+1}) + J_N^*(x_{k+1}) \mid x_k = x\}. \tag{4}$$

Computing the optimal cost-to-go is a task whose computational complexity grows with the cardinality of S [4].

In order to model SVRP as a SSPP, let the state be the following array $x = (\ell, q_\ell, j_1, \ldots, j_n)$; $\ell \in \{0, 1, \ldots, n\}$ denotes the current location of the vehicle, $q_\ell \in \{0, 1, \ldots, Q\}$ describes its current capacity before delivery to customer ℓ, and j_i is the amount of demand yet to be delivered to customer i. The exact amount of demand of each customer becomes known upon the first arrival of the vehicle at each location, before delivery begins. Prior to that, only the probability distribution of D_i is known. Therefore, letting "?" denote an unknown demand value, each j_i can take values in the set $\{?, 0, 1, \ldots, K\}$. The cardinality of the state space is extremely large: $O(nQK^n)$. The starting state is $(0, Q, ?, ?, \dots, ?)$, the final $(0, Q, 0, 0, \dots, 0)$. At a given state x, the control set is $U(x) = \{\{m \in \{1, 2, \dots, n\} \mid j_m \neq 0\} \cup \{0\}\} \times \{a\colon a \in \{0, 1\}\}$; i.e., any not yet visited customer m, or one with a pending demand, directly from ℓ ($a = 0$), or by first visiting the depot ($a = 1$); the depot is also included when all demands are satisfied, so that the vehicle moves there and the system enters the termination state. Alternative $a = 1$ allows some flexibility, whereby preventive trips to the depot are executed in order to avoid future expensive route failures. The control set of a given state is constrained such that at a customer location the demand is satisfied to the maximum extent, given the current capacity. Therefore, a control u assumes the form of a pair (m, a). (When all demands are satisfied $u \equiv (0, 0)$.) From state $x = (\ell, q_\ell, j_1, \dots, j_m, \dots, j_n)$ the system moves to state $x' = (m, q_m, j_1, \dots, j'_m, \dots, j_n)$. The transition cost from state x to state x' under control u is

$$g(x, u, x') = \begin{cases} d(\ell, m) & \text{if } u = (m, 0), \\ d(\ell, 0) + d(0, m) & \text{if } u = (m, 1). \end{cases}$$

The capacity at x' is updated to be

$$q_m = \begin{cases} \max(0, q_\ell - j_\ell) & \text{if } u = (m, 0), \\ q_\ell + Q - j_\ell & \text{if } u = (m, 1). \end{cases}$$

If $j_m = ?$ at x, j'_m is defined to be a realization of the random variable D_m, otherwise $j'_m = j_m$. Transition probabilities are simply

$$p_{xx'} = \begin{cases} 1 & \text{if } j_m \text{ is known,} \\ \Pr\{D_m = j'_m\} & \text{otherwise.} \end{cases}$$

This formulation is exploited in Section 5 to develop NDP algorithms for computing a suboptimal policy for SVRP. The key point here is that $J^*_N(x)$ is not known and its computation is an intractable task. Therefore, computationally tractable ways to approximate such function are sought. The next section discusses this issue from the NDP standpoint.

4. Neuro-dynamic programming

NDP is a methodology designed to deal with large or complex DP problems (see the work of Bertsekas and Tsitsiklis [5] and that of Sutton and Barto [6] for excellent introductions to the NDP field). With such problems, the large state space or the lack of an explicit model of the system prevent the applicability of the exact methods of DP [4]. In the present setting, a model of the system is available, but the size of its state space is large. Therefore computing the optimal cost-to-go is computationally prohibitive. To motivate NDP, observe that an approximate policy can still be implemented if the optimal cost-to-go $J^*(\cdot)$ is replaced by an approximation $\tilde{J}(\cdot)$. (In the remainder of the paper the subscript N on the cost-to-go is dropped for notational convenience.) This is accomplished by computing approximate controls $\tilde{\mu}_k(\cdot)$ as follows:

$$\tilde{\mu}_k(x) = \arg \min_{u_k \in U_k(x_k)} \sum_{x_{k+1} \in S} p_{x_k x_{k+1}}(u_k) \{ g(x_k, u_k, x_{k+1}) + \tilde{J}(x_{k+1}) \mid x_k = x \}. \tag{5}$$

Note that (5) is essentially an approximation of minimization (4) with $J^*(\cdot)$ replaced by $\tilde{J}(\cdot)$. This is the idea at the heart of NDP. The methodology consists of a rich set of algorithms to compute the approximation $\tilde{J}(\cdot)$. In the following a few possibilities are considered.

The model-free case where an explicit model of the system is not available (e.g., the transition probabilities are not available in closed form) can also be dealt with by NDP, but is not discussed here. The reader is referred to the recent monographs by Bertsekas and Tsitsiklis [5] and by Sutton and Barto [6].

4.1. Approximate policy iteration

The main idea of NDP is to break the barriers posed by a large state space or the lack of an explicit model of the system by employing simulation and parametric function approximations. Approximate policy iteration (API) is a NDP algorithm that focuses on the approximate execution of the policy iteration algorithm (see [4,5, pp. 269–284]). The exact policy iteration algorithm is described below. Assume that an initial policy μ^1 is available.

(1) Perform a *policy evaluation* step to compute $J^{\mu^1}(x)$, for all states $x \in S$, by solving the system of linear equations

$$J^{\mu^1}(x) = \sum_{x_{k+1} \in S} p_{x_k x_{k+1}}(\mu_k^1(x_k))\{g(x_k, \mu_k^1(x_k), x_{k+1}) + J^{\mu^1}(x_{k+1}) \mid x_k = x\}, \quad \forall x \in S. \tag{6}$$

(2) Perform a *policy improvement* step which computes a new policy μ^2 as follows:

$$\mu_k^2(x) = \arg \min_{u_k \in U_k(x_k)} \sum_{x_{k+1} \in S} p_{x_k x_{k+1}}(u_k)\{g(x_k, u_k, x_{k+1}) + J^{\mu^1}(x_{k+1}) \mid x_k = x\}, \quad \forall x \in S. \tag{7}$$

These steps are repeated until $J^{\mu^m} = J^{\mu^{m+1}}$, $\forall x \in S$, in which case the algorithm can be shown to terminate with the optimal policy μ^m [4].

It is clear that when dealing with large-scale systems this algorithm is not a viable avenue. But its steps can be approximated. NDP replaces the cost-to-go $J^\mu(x)$ of a policy μ by an approximation $\tilde{J}^\mu(x, r)$ that depends on a vector of parameters r. This approximation could be linear or nonlinear (e.g., a neural network). This process is now outlined [5].

(1) *Approximate policy evaluation.*
   - Assume that under the current policy μ pairs of states and sample cost-to-go values belonging to μ have been collected by simulation. In the neural network terminology, this consists in the generation of a "training set" for the policy μ. If an initial policy is not available, one can always be obtained by arbitrarily fixing the vector of parameters r, and taking approximately greedy controls with respect to the cost-to-go defined by r using (5).
   - Perform a least-squares fitting (linear or nonlinear, depending on the type of function approximation being used) on this training set in order to obtain $\tilde{J}^\mu(x, r)$, i.e., a function that approximates the cost-to-go of μ. This step provides a vector of parameters r that defines the cost-to-go of policy μ for all states x.

(2) *Approximate policy improvement.*
   - Simulate the system by performing a policy improvement step for the states visited during the simulation by employing $\tilde{J}^\mu(x, r)$ in Eq. (5). This amounts to generating an approximately

greedy policy with respect to μ, and provides a new training set of states and sample cost-to-go values corresponding to a new policy μ'. One is now ready to perform step 1 with the newly obtained training set.

The above steps are repeated until convergence to a given policy occurs or a prespecified number of iterations is exceeded. However, convergence is not guaranteed.

By comparing the approximate algorithm with exact policy iteration, it can be noticed that simulation is used in place of the policy improvement phase, while the policy evaluation phase is replaced by the solution of a least-squares problem. This point is crucial, since by means of an approximation, an evaluation of the cost-to-go from any given state under a given policy is made available.

The success of such an approach depends on the "quality" of the training sets collected during the simulation and the accuracy of the resulting cost-to-go approximations. If an accurate approximation of the cost-to-go of all visited policies for all possible states is available, then a policy that is close to being optimal can be generated (see [5, Chapter 6], for a formal proof of this statement). In general this does not happen, and the approximations are of varying quality. In particular, the sequence of policies generated by the algorithm is typically non-monotonically improving (a marked difference from exact policy iteration).

A more detailed discussion of the approximate policy iteration (API) algorithm is now provided. A functional form for the cost-to-go approximation must be assumed. As already mentioned, it could be a linear or a nonlinear architecture, e.g., a neural network. A linear architecture has the significant advantage that the least-squares fitting can be performed very efficiently. If a given policy μ^1 is available, the system is simulated under this policy to generate trajectories from the initial to the terminal state. This provides a set S^1 of states x and associated sample cost-to-go values $c(x)$. This also provides a sample evaluation $\bar{J}^{\mu_1}(x_s)$ of the cost-to-go of policy μ^1 from the starting state x_s, i.e.,

$$\bar{J}^{\mu^1}(x_s) = \sum_{(x_s, c(x_s)) \in S^1} \frac{c(x_s)}{m}, \tag{8}$$

where m is the number of trajectories used to generate S^1. If such a policy is not available, the algorithm starts with a given set of parameters r^0. A training set S^1 of states x and associated sample cost-to-go values $c(x)$ belonging to a policy μ^1 (greedy with respect to $\tilde{J}^{\mu^0}(\cdot, r^0)$) is obtained by simulating trajectories from the initial to the terminal state according to the following minimization:

$$\mu^1(x) = \arg\min_{u \in U(x)} \sum_{x' \in S} p_{xx'}(u)\{g(x,u,x') + \tilde{J}^{\mu^0}(x', r^0)\}. \tag{9}$$

The second step consists of performing the following least-squares fitting:

$$r^1 = \operatorname*{argmin}_{r} \sum_{(x, c(x)) \in S^1} (\tilde{J}^{\mu^1}(x,r) - c(x))^2. \tag{10}$$

At this point a new set S^2 of states and sample cost-to-go values is generated in a similar manner, and a new least-squares problem is solved. The algorithm alternates between simulation and fitting phases. As mentioned above, the termination is quite arbitrary, since it is not guaranteed that the algorithm converges to a unique vector of parameters $\hat{r}$. Upon termination one needs to pick a vector of parameters r that can be used to control the system by implementing a greedy policy with respect to $\tilde{J}(\cdot, r)$. Assuming that m vectors r^t, $t = 0, 1, \ldots, m-1$, have been generated, a possibility is to choose r such that

$$r = \arg \min_{r^t, t=0,1,\ldots,m-1} \bar{J}^{\mu^{t+1}}(x_s). \tag{11}$$

(If $r = r^0$ and an initial policy was available so that r^0 is not defined, it is assumed that the initial policy is chosen as best policy.)

There are two main differences between NDP and more traditional neural networks applications. First, a training set is not available before the training starts. Instead, its creation brings in an additional complication. Indeed one is not faced with a single training set, but with multiple ones, each corresponding to a given control policy. Second, the training of the approximation architecture must be performed repeatedly. Clearly, this calls for fast algorithms or linear architectures, that can be efficiently trained.

4.2. Optimistic approximate policy iteration

Following Bertsekas and Tsitsiklis [5, p. 315] a version of API can be envisioned, that is particularly interesting in case of linear architectures. This variant is called optimistic API (OAPI). The main idea is to generate a smaller number of trajectories (under a given policy) than one would otherwise do with API. Therefore, a least-squares problem is solved more frequently than in API. In particular, let r^t be the current vector of parameters under which a greedy policy is being simulated, and S^{t+1} the newly obtained training set. Let r' be the vector of parameters obtained by applying (10) to S^{t+1}. Then r^{t+1} is computed by interpolation between r^t and r':

$$r^{t+1} = r^t + \gamma^t(r' - r^t), \tag{12}$$

where γ^t is a step size that satisfies $0 < \gamma^t \leqslant 1$ and diminishes as t increases. This corresponds to performing a policy update before "fully" evaluating the current policy. The practical advantage of this algorithm is a smaller memory requirement, since the cardinalities of the training sets are typically smaller than they are in API. On the other side, the number of least-squares problems that need be solved increases, and this is why OAPI is mainly of interest with linear architectures.

4.3. Rollout policies

A rollout policy (RP) is a simplified, yet powerful, NDP algorithm (see [5, pp. 266–267, 455]; see also Bertsekas et al. [36] for the application of the rollout ideas to combinatorial optimization problems, and Bertsekas and Castanon [37] for their application to stochastic scheduling). RP represents another variant of the approximate policy iteration algorithm. Assume that a given heuristic base policy μ for the problem at hand is known. Also assume that the cost-to-go of this base policy from any given state x can be easily computed. Denote it by $H(x)$. Then improved decisions over those available under policy μ can be taken when controlling the system on-line by performing the following minimization:

$$\bar{\mu}(x) = \arg \min_{u \in U(x)} \sum_{x' \in S} p_{xx'}(u)\{g(x, u, x') + H(x')\}. \tag{13}$$

RP corresponds to performing a single policy improvement step over policy μ on the states actually encountered during on-line control of the system. It is particularly interesting when $H(x)$ is available in closed form or is easily computable. Otherwise $H(x)$ has to be estimated via simulation, and the method becomes less efficient (because of the corresponding computational overhead) and less reliable (because of simulation error).

Note that with API and OAPI learning occurs during the simulation and least-squares phases, i.e., during the generation of the training set and the training itself of the architecture that approximates the cost-to-go function. With RP, the training is immediate. In fact, here the approximation architecture is of the form

$$\tilde{J}^{\mu}(x,r) = r_0 + r_1 H(x), \tag{14}$$

and the parameters r_0 and r_1 are trivially "trained" to be 0 and 1, respectively. For RP learning occurs in a different form. It consists of sequentially adjusting the decisions belonging to the initial policy (by using this same policy as a guide) to take advantage of the information that is released during real-time control of the system. While this is certainly a less sophisticated type of learning, it also constitutes a less ambitious endeavor.

5. Applying NDP to SVRP

This section illustrates how the previously described NDP algorithms are applied to the problem at hand.

5.1. API and OAPI

A linear architecture is employed as cost-to-go approximation. For this purpose, let any state $x = (\ell, q_\ell, j_1, \dots, j_i, \dots, j_n)$ (see Section 3) be represented as

$$y = (1, f_1(\ell), f_2(\ell), f_3(q_\ell), \{(f_4(i), f_5(i)),\ i = 1, 2, \dots, n\}). \tag{15}$$

The first component of y is a constant term; the other entries express features of state x:

- $f_1(\ell)$ and $f_2(\ell)$ are the coordinates of the current location ℓ of the vehicle in state x (as explained in the next section the depot and the customers are assumed to be located on an Euclidean plane; if this were not the case an indicator variable should be used to represent the current location of the vehicle);
- $f_3(q_\ell)$ is the current available capacity in state x, i.e., $f_3(q_\ell) = q_\ell$;
- $f_4(i)$ is a feature expressing the demand yet to be delivered to customer i:

$$f_4(i) = \begin{cases} E[D_i] & \text{if } j_i = \text{"?"}, \\ j_i & \text{otherwise;} \end{cases} \tag{16}$$

- $f_5(i)$ is a feature expressing the uncertainty associated with the demand yet to be delivered to customer i:

$$f_5(i) = \begin{cases} SD(D_i) & \text{if } j_i = \text{"?"}, \\ 0 & \text{otherwise,} \end{cases} \tag{17}$$

where $SD(D_i)$ is the standard deviation of D_i. Essentially $f_5(i)$ is an indicator that is on or off whether customer i has not yet or has already been visited, respectively, when the system is in state x.

The linear approximation is then defined as

$$\tilde{J}(x,r) = r_0 + r_1 f_1(\ell) + r_2 f_2(\ell) + r_3 f_3(q_\ell) + \sum_{i=1}^{n} [r_{4+2(i-1)} f_4(i) + r_{5+2(i-1)} f_5(i)]. \tag{18}$$

An initial policy is obtained by setting the initial vector of parameters r equal to the zero vector. This implies that the initial policy is assumed to be a dynamic nearest-neighbor heuristic that always visits the closest customer. As a hard-coded rule, whenever the vehicle is empty, it drives back to the depot in order to replenish. In this way all policies are proper (i.e., termination is reached from any state with probability 1 [4,5]). This rule is needed since when approximating the cost-to-go of a given policy one may face a situation where the vehicle cycles empty indefinitely between two states, and algorithms API or OAPI never terminate.

With this cost-to-go approximation in place, the API and OAPI algorithms presented in Section 4 are implemented using the formulation given in Section 3. An example is provided after discussing the implementation of RP.

5.2. *A rollout policy*

A RP policy for SVRP and its properties are provided in Secomandi [24]. In particular, it can be shown that RP cannot perform worse than the initial policy it is based on. This work provides additional computational experience with this algorithm and a comparison against OAPI. For completeness, the RP algorithm is now briefly summarized.

An initial static policy is obtained by computing a deterministic nearest neighbor traveling salesman tour improved by 2-Int moves [38] ignoring customers' demands. Let $\tau = (0,1,\dots,n,0)$ (where nodes have been ordered) be this tour; τ provides a static policy whose expected length can be computed by the algorithms presented in Bertsimas [12] and Gendreau et al. [15]. For completeness, a slightly modified version of this latter algorithm is given in Appendix A. Tour τ is used as a guiding solution to build a dynamic routing policy. At any state x the next customer m to be visited is selected by modifying τ to yield a new static policy τ_m. Its expected length $E[L_{\tau_m}|q_m]$ (see Appendix A) provides an estimate of the optimal cost-to-go to termination from the new state x'. The modified tour τ_m is obtained by employing the cyclic heuristic (CH) proposed in Bertsimas [12] as follows:

$$\tau_m \equiv (m, m+1, \dots, n, 1, \dots, m-1, 0), \tag{19}$$

where customers already visited are dropped. For example, let $n = 5$, $Q = 3$, $x = (2,3,?,1,0,?,?)$, and $\tau = (0,1,2,3,4,5,0)$. Then, for $m = 1$, $\tau_1 = (1,4,5,0)$, for $m = 4$, $\tau_4 = (4,5,1,0)$, and for $m = 5$, $\tau_5 = (5,1,4,0)$. Under the control $\mu(x) = (m,\cdot)$ at state x, the cost-to-go from the new state x' assumes the form

$$H(x') = E[L_{\tau_m}|q_m]. \tag{20}$$

This expression for $H(\cdot)$ is used in (13). Then a rollout policy is implemented exploiting the formulation of Section 3.

5.3. An example

This subsection provides an example to illustrate the OAPI and RP algorithms. The compact notations

$$Q(x,u,r) \equiv \min_{u\in U(x)} \sum_{x'\in S} p_{xx'}(u)\{g(x,u,x') + \tilde{J}(x',r)\} \tag{21}$$

and

$$Q(x,u) \equiv \min_{u\in U(x)} \sum_{x'\in S} p_{xx'}(u)\{g(x,u,x') + H(x')\}. \tag{22}$$

are adopted.

Example 1. Consider a network with $n=5$ customers. Vehicle capacity is set to $Q=15$. Table 1 provides the distance matrix and the demands' distributions. These are discrete uniform random variables with different supports.

Fig. 5 illustrates the behavior of OAPI, using a constant $\gamma_t = 0.01$: part (a) is a plot of the average cost-to-go from the initial state in each of 50 iterations, while part (b) illustrates how the values of the vector of parameters evolve over time.

The first part of Table 2 summarizes the computation of the control at the initial state $x=(0,15,?,?,?,?,?)$ for the best set of parameters r generated by OAPI according to (11). Even though $\tilde{J}(x',r)$ underestimates the true optimal cost-to-go from each next state, $Q(x,u,r)$ maintains the relative order among controls and a "good" decision is taken (good in the sense that it agrees with that taken by RP).

Fig. 6 shows the nearest neighbor and the improved 2-Int static policies. The second part of Table 2 provides the computation of the first RP control. The values of $Q(x,u)$ appear to be a better estimate of the optimal expected costs to termination. Nevertheless, in this case the controls under RP and OAPI coincide.

Table 3 reports the trajectories generated by both algorithms on the set of demands' realizations $\{1,\ 5,\ 3,\ 9,\ 6\}$, expressed as a sequence of location and available capacity pairs (ℓ, q). Fig. 7 graphically illustrates these trajectories. Notice how both policies perform replenishment trips to the

Table 1
Example 1: Distances and probability distributions ($Q=15$)

Node	Node						D_i
	0	1	2	3	4	5	
0	0.0000	0.8753	0.4749	0.3590	0.3813	0.4350	
1		0.0000	0.5956	0.6517	0.5387	0.6802	U(1, 5)
2			0.0000	0.1164	0.4446	0.0866	U(3, 9)
3				0.0000	0.3873	0.0944	U(1, 5)
4					0.0000	0.4768	U(6, 12)
5						0.0000	U(3, 9)

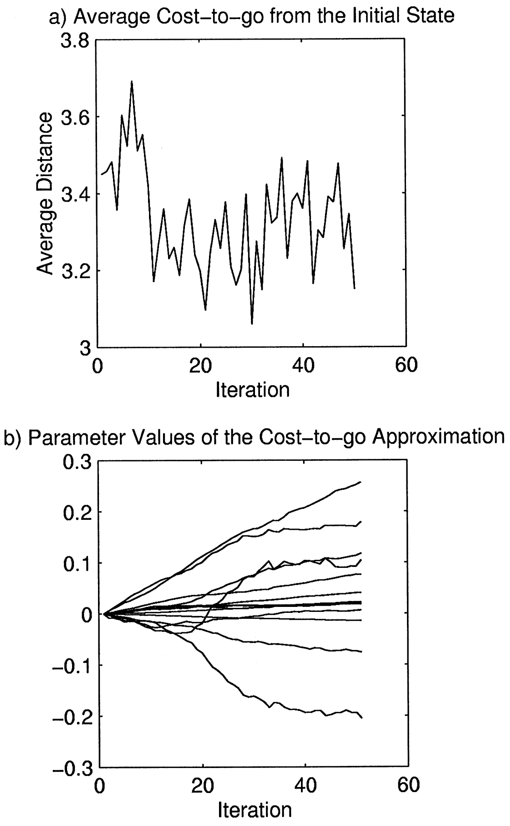


Fig. 5. Example 1: OAPI: (a) cost-to-go values, (b) parameter values.

depot in order to anticipate route failures. In this instance and for these demands' realizations, RP outperforms OAPI by about 2.43%.

6. Computational experience

This section reports the results of a computational experience with algorithms OAPI and RP. The results for API are not given, since OAPI generally performed better than API. The goal of the experiment is to compare the performance of OAPI and RP. A set of random instances are generated according to a scheme similar to that proposed in Gendreau et al. [15]. The instances differ for the number of customers and the relative amount of expected customers' demands with respect to vehicle capacity.

The characteristics of the instances are now illustrated. All instances are single-vehicle ones. Three levels for factor n (number of customers, depot excluded) are analyzed: $n \in \{5, 10, 15\}$. These may seem small sizes, but one should note that these are single-vehicle instances. In a typical multi-vehicle instance reported in the literature (say with at most 200 customers, as discussed in [39] in the context of deterministic VRP), each vehicle would be assigned a number of customers

Table 2
Example 1: control computation from the initial state

OAPI		
$x = (0,\ 15,\ ?,\ ?,\ ?,\ ?,\ ?)$		
$y = (1, 0, 0, 15, 3, 1.4142, 7, 2, 3, 1.4142, 10, 2, 7, 2)$		
$r = (0.17,\ 0.07,\ -0.16,\ -0.01,\ 0.05,\ 0.15,\ 0.02,\ -0.01,\ 0.02,\ 0.07,\ 0.01,\ 0.01,\ 0.02,\ -0.06)$		
u	$(x', \tilde{J}(x', r))$	$Q(x,u,r)$
1	((1, 15, {1..5}, ?, ?, ?, ?), {0.6429, 0.5972, 0.5515, 0.5058, 0.4601})	1.4268
2	((2, 15, ?, {3..9}, ?, ?, ?), {0.8669, 0.8501, 0.8333, 0.8165, 0.7997})	1.3250
3[a]	((3, 15, ?, ?, {1..5}, ?, ?), {0.7796, 0.7645, 0.7494, 0.7343, 0.7192})	1.1083
4	((4, 15, ?, ?, ?, {6..12}, ?), {0.7670, 0.7551, 0.7432, 0.7313, 0.7194})	1.1364
5	((5, 15, ?, ?, ?, ?, {3..9}), {0.9799, 0.9551, 0.9303, 0.9055, 0.8807})	1.3901
RP		
$\tau = (0,\ 3,\ 5,\ 2,\ 1,\ 4,\ 0)$		
u	τ'	$Q(x,u)$
1	(0, 1, 4, 3, 5, 2, 0)	3.3978
2	(0, 2, 1, 4, 3, 5, 0)	3.4222
3[a]	(0, 3, 5, 2, 1, 4, 0)	3.2408
4	(0, 4, 3, 5, 2, 1, 0)	3.5500
5	(0, 5, 2, 1, 4, 3, 0)	3.5439

[a]Best.

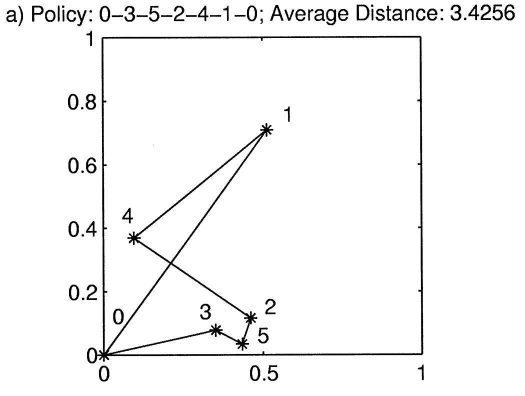
approximately in the range [5,20]. Previous experiments in the context of SVRP [15] consider instances with only two vehicles and up to 70 customers, so that in such case more customers are assigned to each vehicle. Other experiments [25,26] present instances that are closer to the deterministic case and the instances discussed here.

The filling rate is an index of the tightness of a given VRP instance. In a stochastic environment, the expected filling rate $\bar{f}$ can be defined as

$$\bar{f} = \sum_{i=1}^{n} \frac{E[D_i]}{mQ}, \tag{23}$$

where $E[D_i]$ is the expected demand of customer i, m is the number of available vehicles, and Q denotes vehicle capacity. When $m = 1, \bar{f}$ can be interpreted as the expected number of replenishments needed to serve all customers' demands, including the first load. Therefore $\bar{f}' = \bar{f} - 1$ is the expected number of route failures in a given instance. Three levels are considered for this factor: $\bar{f}' \in \{1.0, 1.5, 2.0\}$.

Customers' demands are divided into three categories: low, medium, and high demand, corresponding to the following three discrete uniform probability distributions, $U(1,5)$, $U(3,9)$, $U(6,12)$,



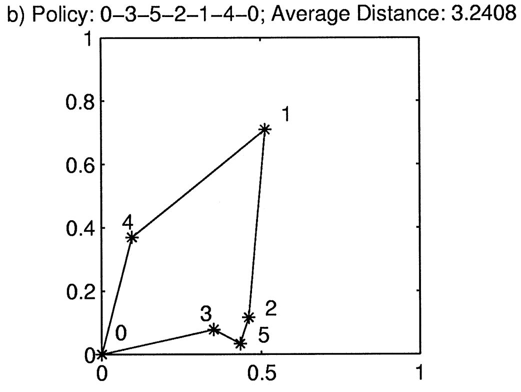


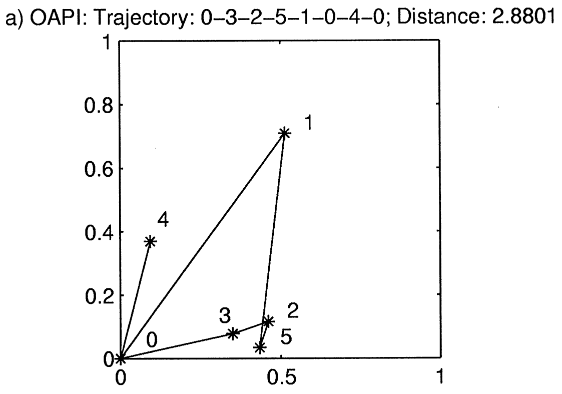
Fig. 6. Example 1: static policies: (a) nearest neighbor, (b) 2-Int.

Table 3
Example 1: trajectories and distances on demands realizations $\{1, 5, 3, 9, 6\}$

Algorithm	$\{(\ell, q)\}$	Distance
OAPI	{(0, 15), (3, 15), (2, 12), (5, 7), (1, 1), (0, 15), (4, 15), (0, 15)}	2.8801
RP	{(0, 15), (3, 15), (5, 12), (2, 6), (0, 15), (1, 15), (4, 14), (0, 15)}	2.8101

respectively. In each instance, each customer is assigned to any of the three groups with equal probability. It follows that, a priori, in all instances $E[D_i] = (3 + 6 + 9)/3 = 6$, for any customer i, and $\bar{f} = 6n/Q$. Customers locations are random points in $[0,1]^2$, with the depot fixed at $(0,0)$. Randomization is adopted within each factor combination (a cell). Five instances, each corresponding to an initial seed value for the random number generator, are generated according to the rules just explained.

Table 4 summarizes the instances characteristics. The first column reports levels of factor $\bar{f}'$, the first row those of factor n. All other entries give the value of Q, for a given combination of $\bar{f}'$ and n,



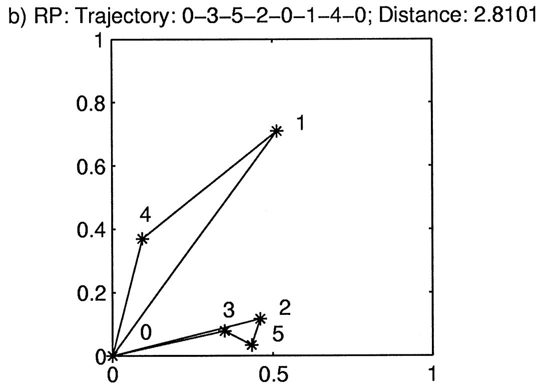


Fig. 7. Example 1: Trajectories for Demands' Realizations $\{1, 5, 3, 9, 6\}$: (a) OAPI (b) RP.

computed as

$$Q = \frac{6n}{\bar{f}} = \frac{6n}{1+\bar{f}'} \tag{24}$$

rounded to the nearest integer.

The following subsections analyze the performance of the two algorithms. These are implemented in MATLAB 5.1; the experiment run on a 233 MHz Pentium PC, under the Linux operating system. The expected distance achievable by each algorithm is evaluated through simulation of the corresponding policy by generating 100 trajectories from the initial to the termination state. Each algorithm is evaluated using the same seed for the random number generator, so that the comparisons are meaningful. The sample mean of the distance traveled in each trajectory is used as performance indicator. In OAPI, the least squares is efficiently solved by the linear algebra routines available in MATLAB. Generating a single trajectory is faster than it is with RP. In fact there is no need to evaluate $H(x)$, since $\tilde{J}^{\mu}(x,\cdot)$ is available. But obtaining the vector of approximating parameters requires simulation, whereas this is not needed with RP.

6.1. Analysis of OAPI

The first half of Table 5 reports results of the experiment with OAPI. In all instances, OAPI is started by simulating a nearest neighbor policy (NNP); then 50 approximate policy improvements, each consisting of 30 trajectories, and associated approximate policy evaluations (i.e., least-squares) are executed. Table 6 presents the average CPU seconds needed to complete a single approximate policy iteration. The stepsize γ_t is set to 0.01 and kept constant over all the execution of the algorithm. Finally the best policy according to Eq. (11) is evaluated by generating 100 trajectories from the initial to the final state. The quality of NNP is also evaluated in this manner (i.e., by using the same seeds used for OAPI in generating the evaluation trajectories).

Table 4
Vehicle capacity for each factor level combination

$\bar{f}'$	n		
	5	10	15
1.0	15	30	45
1.5	12	24	36
2.0	10	20	30

Table 5
Average distance traveled for each factor level combination

$\bar{f}'$	Seed	OAPI			RP		
		n			n		
		5	10	15	5	10	15
	1	3.3288	4.4347	5.7589	3.0303	4.1133	5.7524
	2	4.5211	5.8824	7.0522	4.0943	5.7893	6.5419
1.0	3	2.6153	3.9949	6.6047	2.4666	3.8278	5.4517
	4	6.1131	5.7692	6.7177	5.4874	6.1631	6.1865
	5	5.3540	6.2328	5.4147	4.7546	5.9630	5.3689
	1	3.8638	5.0445	6.4868	3.4909	4.7292	6.2884
	2	5.4616	7.1922	7.7820	5.0742	6.4282	7.1279
1.5	3	3.1785	4.9790	7.0964	2.9243	4.4900	6.3790
	4	7.1372	6.5739	7.5677	6.3089	6.4649	6.9754
	5	6.1933	6.7057	7.2314	5.6471	6.3492	6.1126
	1	4.2878	5.5286	7.8073	3.8598	5.1217	6.9270
	2	6.3452	8.0910	9.1746	5.6394	7.7434	8.2239
2.0	3	3.4949	5.2650	8.1509	3.2060	4.8733	7.2838
	4	8.0640	7.8583	8.7490	7.3697	7.2566	8.1559
	5	6.8770	8.1366	7.7732	6.5284	7.3958	6.7186

A comparison of the average performance of OAPI against that of NNP is presented in Table 7 (each entry is the average over five instances). OAPI generally improves the quality of NNP, but this improvement is somehow small amounting on average to 2.3133%. Nevertheless, it must be recognized that, given the combinatorial structure of the problem, the task OAPI is trying to accomplish is extremely challenging.

6.2. Analysis of RP

As shown in Table 8, RP significantly improves the quality of the base heuristic policy (HP) (both RP and HP are evaluated in the same way of OAPI and NNP); the average reduction in expected distance is 7.0332%. As mentioned, HP is a static heuristic traveling salesman tour that ignores the inherent stochasticity of the problem, supported by recourse actions (a trip to the depot to replenish) in case of route failures. The second half of Table 5 reports the quality of the policy generated by RP in each instance. Clearly RP outperforms OAPI across all factor combinations. The average improvements of RP over OAPI are presented in Table 9. On average, RP outperforms OAPI by 7.7189%. The reason of such a result is to be attributed to the fact that RP works with an exact evaluation of the cost-to-go $H(\cdot)$ of the policy that is being improved, while OAPI must rely on an approximation $\tilde{J}(\cdot, r)$. In the experiments, it was observed that even a single "wrong" evaluation of the available controls, due to the imprecision of $\tilde{J}(\cdot, r)$, can lead to a 10% deterioration in the solution quality. On the contrary, even though RP only executes a single

Table 6
OAPI: average CPU seconds per approximate policy iteration

$\bar{f}'$	n		
	5	10	15
1.0	3.9	14.5	33.5
1.5	4.0	14.6	33.8
2.0	4.2	15.1	34.8

Table 7
OAPI vs. NNP: percentage improvements in average distance traveled (2.3133)[a]

$\bar{f}'$	n		
	5	10	15
1.0	0.9507	5.2158	5.4812
1.5	0.1585	4.7135	2.5820
2.0	0.6901	0.3016	0.7269

[a] Average.

Table 8
RP vs. HP: percentage improvements in average distance traveled (7.0332)[a]

$\bar{f}'$	n		
	5	10	15
1.0	8.0401	8.0968	8.8898
1.5	5.9113	6.2811	8.1915
2.0	5.4249	5.2688	7.1950

[a] Average.

Table 9
RP vs. OAPI: percentage improvements in average distance traveled (7.7189)[a]

$\bar{f}'$	n		
	5	10	15
1.0	9.5705	1.7386	7.1209
1.5	9.2473	6.6698	9.0724
2.0	8.4815	7.1360	10.4333

[a] Average.

Table 10
RP: average CPU seconds per trajectory

$\bar{f}'$	n		
	5	10	15
1.0	0.3	3.5	17.5
1.5	0.3	3.1	15.0
2.0	0.3	2.8	13.6

on-line policy improvement step, it benefits from accurate evaluation of the controls at each decision point. Average CPU seconds needed to complete a single trajectory are given in Table 10. As it can be noticed, RP can be efficiently implemented in a real-time environment. As a result, RP poses itself as a robust algorithm for SVRP. Overall, given its simplicity, its performance is quite impressive.

7. Conclusion

The paper presents a comparison of neuro-dynamic programming algorithms for the single vehicle routing problem with stochastic demands. Two such algorithms are analyzed: optimistic approximate policy iteration and a rollout policy. The computational experiment shows that the rollout policy outperforms optimistic approximate policy iteration. The percentage improvements are substantial and consistent over all the instances used in the experiment. The performance of RP is quite impressive, if one considers the simplicity of the algorithm. RP is a robust algorithm that holds promise for a successful application to VRP business situations. (For instance, the same idea on which RP is based can be applied to deterministic VRPs that are heuristically solved.) This does not seem to be the case with other “more ambitious” and less robust neuro-dynamic programming algorithms, as optimistic approximate policy iteration. Further research (e.g., the use of a nonlinear architecture for approximating the cost-to-go) and experimentation should be performed to assess the full potential of such algorithms.

Acknowledgements

I thank Sean Curry at The MathWorks, Inc. for the excellent support provided in the use of MATLAB. The comments and suggestions of two anonymous referees and of the guest editor led to an improved version of the original paper.

Appendix A. The expected length of a static policy

Let $\tau = (0, 1, \dots, n, n+1 \equiv 0)$ be a tour through the customers that starts and ends at the depot. Let L_τ be the random variable denoting the length of tour τ. Consider the case where the vehicle is at location ℓ, $\ell = 0, 1, \dots, n$, with q_ℓ denoting its currently available capacity, $q_\ell = 1, \dots, Q$. When $\ell \neq 0$ it is assumed that customer ℓ has not yet been served, so that q_ℓ represents the available capacity before service. If $\ell = 0$, then $q_\ell \equiv Q$. Assume that some of the customers have already been served under an arbitrary policy. Then $\tau_\ell = (\ell, \ell + 1, \dots, 0)$ denotes a partial tour through the remaining customers. Moreover, some of the customers may already have received a visit, so that their demands are known, but not completely served, perhaps because a route failure occurred. In this case the random variable describing the demand of each such customer is redefined to be a degenerate random variable assuming a value equal to the amount of pending demand with probability 1. For notational convenience, if $\ell = 0$ then D_0 is assumed to be a degenerate random variable identically equal to 0 with probability 1. Then Proposition 2 in Gendreau et al. [15] can be modified to compute $E[L_\tau \mid q_\ell]$, the expected length of the partial tour τ_ℓ [24]:

$$E[L_{\tau_\ell} \mid q_\ell] = \sum_{i=\ell}^{n} d(i, i+1) + \alpha_\ell(q_\ell), \tag{A.1}$$

where $\alpha_n(q_n) = 2d(0,n)\sum_{k=q_n+1}^{K} p_n(k)$, for $q_n = 1, \dots, Q$, and, for $i = \ell, \dots, n-1$, $q_i = 1, \dots, Q$,

$$\begin{aligned} \alpha_i(q_i) = &\sum_{k=0}^{q_i - 1} p_i(k)\alpha_{i+1}(q_i - k) \\ &+ p_i(q_i)[d(i, 0 - d(i, i+1) + d(0, i+1) + \alpha_{i+1}(Q)] \\ &+ \sum_{k=q_i+1}^{K} p_i(k)[2d(i, 0) + \alpha_{i+1}(Q + q_i - k)]. \end{aligned} \tag{A.2}$$

The first term in (A.1) is the tour length if no route failures occur. The terms $\alpha_i(q_i)$ represent the expected extra distance traveled due to route failures before termination, given that the vehicle is at location i with q_i available capacity, and the remaining tour is τ_i. The terms $\alpha_n(q_n)$ are the boundary conditions. All other terms are computed by DP backward recursion by conditioning on the demand levels of the current customer in the tour.

In a DP context, $E[L_{\tau_\ell} \mid q_\ell]$ assumes the interpretation of a cost-to-go function, that is the expected distance to termination, given that the current location is ℓ, the current capacity is q_ℓ, and the remaining portion of the tour is τ_ℓ.

References

[1] Powell WB, Jaillet P, Odoni A. Stochastic and Dynamic Networks and Routing. In: Ball MO, Magnanti TL, Monma CL, Nemhauser GL, editors. Network routing, vol. 8, Handbooks in Operations Research and Management Science. Amsterdam: Elsevier, 1995 [chapter 3].
[2] Psaraftis HN. Dynamic vehicle routing. Annals of Operations Research 1995;61:143–64.
[3] Bertsimas DJ, Simchi-Levi D. A new generation of vehicle routing research: robust algorithms, addressing uncertainty. Operations Research 1996;44:286–304.
[4] Bertsekas DP. Dynamic programming and optimal control. Belmont, MA: Athena Scientific, 1995.
[5] Bertsekas DP, Tsitsiklis JN. Neuro-dynamic programming. Belmont, MA: Athena Scientific, 1996.
[6] Sutton RS, Barto AG. Reinforcement learning. Cambridge, MA: MIT Press, 1998.
[7] Golden BL, Assad AA, editors. Vehicle routing: methods and studies. Amsterdam: Elsevier, 1988.

[8] Fisher M. Vehicle routing. In: Ball MO, Magnanti TL, Monma CL, Nemhauser GL, editors. Network routing, Vol. 8, Handbooks in Operations Research and Management Science. Amsterdam: Elsevier, 1995 [Chapter 1].
[9] Crainic TG, Laporte G. Planning models for freight transportation. European Journal of Operational Research 1997;97:409–38.
[10] Bramel J, Simchi-Levi D. The logic of logistics. New York: Springer, 1997.
[11] Dror M, Laporte G, Trudeau P. Vehicle routing with stochastic demands: properties and solution frameworks. Transportation Science 1989;23:166–76.
[12] Bertsimas DJ. A vehicle routing problem with stochastic demands. Operations Research 1992;40:574–85.
[13] Bertsimas DJ, Chervi P, Peterson M. Computational approaches to stochastic vehicle routing problems. Transportation Science 1995;29:342–52.
[14] Gendreau M, Laporte G, Séguin R. Stochastic vehicle routing. European Journal of Operational Research 1996;88:3–12.
[15] Gendreau M, Laporte G, Séguin R. An exact algorithm for the vehicle routing problem with stochastic demands and customers. Transportation Science 1995;29:143–55.
[16] Gendreau M, Laporte G, Séguin R. A tabu search heuristic for the vehicle routing problem with stochastic demands and customers. Operations Research 1996;44:469–77.
[17] Lambert V, Laporte G, Louveaux FV. Designing collection routes through bank branches. Computers and Operations Research 1993;20:783–91.
[18] Dror M, Ball MO, Golden BL. Computational comparison of algorithms for inventory routing. Annals of Operations Research 1985;4:3–23.
[19] Larson RC. Transportation of sludge to the 106-mile site: an inventory routing algorithm for fleet sizing and logistic system design. Transportation Science 1988;22:186–98.
[20] Bartholdi JJ, Platzman LK, Collins RL, Warden WH. A minimal technology routing system for meals on wheels. Interfaces 1983;13:1–8.
[21] Dror M. Modeling vehicle routing with uncertain demands as a stochastic program: properties of the corresponding solution. European Journal of Operational Research 1993;64:432–41.
[22] Bastian C, Rinnooy Kan AHG. The stochastic vehicle routing problem revisited. European Journal of Operational Research 1992;56:407–12.
[23] Yang WH, Mathur K, Ballou RH. Stochastic vehicle routing with restocking. Transportation Science, to appear.
[24] Secomandi N. Exact and heuristic dynamic programming algorithms for the vehicle routing problem with stochastic demands. Dissertation, University of Houston, USA, 1998.
[25] Teodorović D, Pavković G. A simulated annealing technique approach to the vehicle routing problem in the case of stochastic demand. Transportation Planning and Technology 1992;16:261–73.
[26] Dror M, Trudeau P. Stochastic vehicle routing with modified savings algorithm. European Journal of Operational Research 1986;23:228–35.
[27] Dror M, Laporte G, Louveaux FV. Vehicle routing with stochastic demands and restricted failures. Zeitschrift für Operations Research 1993;37:273–83.
[28] Bouzaïene-Ayari B, Dror M, Laporte G. Vehicle routing with stochastic demands and split deliveries. Foundations of Computing and Decision Sciences 1993;18:63–9.
[29] Laporte G, Louveaux FV. The integer L-shaped method for stochastic integer programs with complete recourse. Operations Research Letters 1993;13:133–42.
[30] Golden BL, Yee JR. A framework for probabilistic vehicle routing. AIIE Transactions 1979;11:109–12.
[31] Stewart WR, Golden BL. Stochastic vehicle routing: a comprehensive approach. European Journal of Operational Research 1983;14:371–85.
[32] Laporte G, Louveaux FV, Mercure H. Models and exact solutions for a class of stochastic location-routing problems. European Journal of Operational Research 1989;39:71–8.
[33] Bertsimas DJ, van Ryzin G. A stochastic and dynamic vehicle routing problem in the Euclidean plane. Operations Research 1991;39:601–15.
[34] Bertsimas DJ, van Ryzin G. Stochastic and dynamic vehicle routing in the Euclidean plane with multiple capacitated vehicles. Operations Research 1993;41:60–76.

[35] Bertsimas DJ, van Ryzin G. Stochastic and dynamic vehicle routing with general demand and interarrival time distributions. Advances in Applied Probability 1993;25:947–78.
[36] Bertsekas DP, Tsitsiklis JN, Wu C. Rollout algorithms for combinatorial optimization. Journal of Heuristics 1997;3:245–62.
[37] Bertsekas DP, Castanon DA. Rollout algorithms for stochastic scheduling problems. Journal of Heuristics, to appear.
[38] Nemhauser GL, Wolsey LA. Integer and combinatorial optimization. New York: Wiley, 1988.
[39] Gendreau M, Hertz A, Laporte G. A tabu search heuristic for the vehicle routing problem. Management Science 1994;40:1276–90.

Nicola Secomandi received his Ph.D. degree in operations research from the University of Houston in 1998. He is now a research scientist in the R&D department of PROS Strategic Solutions, a Houston based company that specializes in the application of revenue management. His current research interests involve stochastic and dynamic transportation problems, rollout algorithms, metaheuristics, financial engineering, and the theory and applications of revenue management, in particular as it applies to the energy industry.