A Temporal Difference Learning Approach to Network Revenue Management

by

Taylor Egner

A thesis submitted to the faculty of the Graduate School of the University of Colorado Boulder Leeds School of Business, in partial fulfillment of the requirement for the degree of

Master of Science

Strategy, Entrepreneurship, and Operations Department

2022

Committee Members: Dan Zhang, Rui Zhang, Thomas Vossen
Leeds School of Business, University of Colorado Boulder
dan.zhang@colorado.edu, rui.zhang@colorado.edu, thomas.vossen@colorado.edu

This version: May 19, 2022

Egner, Taylor (M.S., Strategy, Entrepreneurship, and Operations)
A Temporal Difference Learning Approach to Network Revenue Management
Thesis directed by Professor Dan Zhang

Abstract The network revenue management (NRM) problem has been well-studied in the literature, with the large majority of research focusing on the approximate linear programming (ALP) approach. The ALP approach has been shown to work quite well for many problem instances. The performance of the ALP approach depends critically on the approximation architecture. However, even separable piecewise linear (SPL) approximation, widely considered the strongest approximation, cannot fully account for the strong network effects that can exist in the NRM problem. This limitation is mainly due to the nature of the ALP approach. In this research, we explore simulation-based reinforcement learning methods that are more flexible and can utilize a broader class of approximation architectures beyond those viable for the ALP approach. Our primary focus is the widely used temporal difference (TD) learning algorithm. We develop two distinct adaptations of the TD learning algorithm that are tailored to fit the structure of the NRM problem and test the algorithms using various approximation architectures and initial policies. We introduce a novel eligibility trace which we call the salience trace that significantly increases TD learning performance for approximations that use binary features. We demonstrate via experiments that the TD learning algorithms do not rely on commercial mathematical programming solvers and can lift the policy performance to near optimal even when the initial policy is a naive one, which accepts a request whenever feasible. Furthermore, we show that the TD learning algorithms can adopt a recently proposed novel approximation architecture (Zhang et al. 2021b), which can better capture the network effects and provide smaller approximation errors than the SPL approximation, but is intractable for the ALP approach. Over a set of 40 problem instances, heuristic policies based on the learned value function approximations generate expected revenues that are competitive with the bid-price policy derived from the ALP approach in 60% of instances.

Contents

List of Tables

List of Figures

1. Introduction

Network revenue management (NRM) involves the management of a network of resources with limited capacity which are consumed in various combinations by different products. Product requests are assumed to arrive over a finite time horizon according to a stochastic process and the decision maker must decide whether to accept or reject each request upon arrival with the objective of maximizing the expected total revenue over the selling horizon. This sequential decision making problem naturally lends itself well to a dynamic programming (DP) formulation, with the state being the remaining capacity of each resource and the action set in each period being the feasible accept/reject decisions given the state (Talluri and van Ryzin (2004)). In practice, the exact solution of the DP formulation is difficult to achieve using standard algorithms (such as value iteration and policy iteration) as it suffers from the well-known "curse of dimensionality". For this reason, researchers have developed a number of solution methods based on approximations and heuristics.

A widely studied approach to solve the NRM problem is approximate linear programming (ALP) (de Farias and Van Roy 2003). In the ALP approach, the search for the optimal value function is restricted to the space generated by the span of a set of basis functions. The problem is then solved by computing the optimal set of weights for each basis function. In this way, the problem is reformulated such that there is a relatively small number of variables which allows for better computational tractability. However, ALP formulations still contain an exponential number of constraints. Traditionally, these problems have then been solved using approaches such as constraint sampling or column generation. The solution to the ALP approach yields a set of weights which can be interpreted as marginal values that inform a heuristic "bid-price" policy whereby a request is accepted if the fare is greater than the sum of the marginal values of the units of each resource to be consumed (Talluri and van Ryzin 1998). This "bid-price" policy heuristic is widely recognized in the literature as one of the strongest policies for NRM problems.

One of the first applications of the ALP approach to NRM was given by Adelman (2007), who introduced an approximation architecture for the NRM problem that was based on an affine function of the state vector. Using this architecture, Adelman was able to produce a tighter upper bound on the total expected revenue than was achievable using the well-known deterministic linear programming (DLP) approach (Talluri and van Ryzin 1998). Critically, the coefficients of the affine approximation can be interpreted as time-dependent marginal values of each resource in each time period, thus allowing for the development of

a time-dependent bid-price policy which was shown to outperform the bid-price policies given by the DLP formulation (Williamson 1992).

There has been a series of follow-up work to Adelman (2007) in the literature that considers stronger and more granular functional approximations that lead to tighter bounds and improved bid-price policy performance. The most popular approximation is the seperable piecewise linear (SPL) approximation (Farias and Van Roy 2007, Meissner and Strauss 2012)), which considers both the time period and the resource level when constructing the value function approximation. Topaloglu (2009) offered a Lagrangian relaxation approach which was subsequently shown to be equivalent to the SPL approximation (Kunnumkal and Talluri 2016). In this paper, we make use of the SPL approximation architecture with weights trained by a temporal difference (TD) learning approach as an alternative to the ALP approach. While bid-price policies derived from ALPs often perform quite well, Talluri and van Ryzin (1998) use a simple example to demonstrate the non-optimality of such policies, given that they ignore the network effects when multiple products share the same resources. We revisit the example in Talluri and van Ryzin (1998) and show that the bid-price policy derived from the SPL approximation is suboptimal, but a TD learning approach using the same approximation architecture can achieve optimality in this example.

ALPs are also known to pose significant computational challenges due to their size, as they have relatively few variables but still carry an exponential number of constraints. This fact has led researchers to make use of specialized algorithms such as column generation (Adelman 2007, Meissner and Strauss 2012) and constraint sampling (de Farias and Van Roy 2004, Farias and Van Roy 2007) in order to solve these formulations. However, these specialized algorithms can be very computationally expensive even for small problems. Recent research has demonstrated the possibility of constructing compact reformulations for some ALPs to address the issue of tractability (Tong and Topaloglu 2014, Vossen and Zhang 2015). Specifically, the SPL approximation for NRM has been shown to have an equivalent, compact LP formulation that allows for solutions that are faster by several orders of magnitude than the original formulations. These compact reformulations, however, are heavily dependent on the specific structure of the approximation architecture in question and are not guaranteed to exist for an arbitrary approximation architecture. It is easy to see that this reliance on compact reformulations severely restricts the class of approximation architectures that may be solved in a reasonable time frame using the ALP approach.

A recent paper (Zhang et al. 2021b) has proposed an alternative to the SPL approximation that better addresses network effects in the NRM problem. The authors propose a *product-based* approximation with a novel bid-price policy that considers the cost of "last-unit" consumption of resources. While the product-based approximation is shown to have smaller approximation errors than the SPL approximation, its resulting ALP formulation does not admit a compact reformulation and therefore can be computationally challenging. TD learning is one possible way of avoiding these computational issue while still leveraging the improved value function approximation architecture of the product-based approximation. The critical reason for this is that the TD learning algorithm does not need to solve mathematical programs, but rather uses a simulation-based approach.

The existing literature tends to focus primarily on mathematical programming techniques. In contrast, we explore a simulation-based approach. The approach involves the Monte Carlo simulation of a large number of booking horizons for fixed policies and iterative fitting of value function approximations. Recent literature suggests that simulation-based methods allow for a broader class of approximation architectures. Koch (2017) developed an approach based on least-squares approximate policy iterate that uses the linear least squares regression to mimic heuristic policies such as bid-price policies by enforcing a series of constraints on the parameters. Of special significance to this approach is the non-reliance on fixed models of customer choice. In general, this is one of the main advantages of simulation methods — they are often model-free and as such are highly flexible. Furthermore, simulation approaches are generally relatively simple, easy to implement, and do not require the use of commercial solvers.

The simulations techniques we examine belong to the broad class of reinforcement learning (RL) algorithms. These learning algorithms are widely used in many fields including engineering and robotics (Mahmud et al. (2018), Kober et al. (2013)), and are powerful tools for solving problems which can be cast as sequential decision making problems under uncertainty (Littman (1994)). There are many such problems in the OR literature to which reinforcement learning algorithms might be applicable. Mazyavkina et al. (2021) present a survey of RL approaches to combinatorial optimization problems which indicate the potential that these algorithms have in OR problem settings. Broadly, reinforcement learning involves the training of an agent through interaction between the agent and an environment with the express purpose of maximizing some reward signal. Fitting neither the paradigms of supervised or unsupervised learning, reinforcement learning is considered

a separate machine learning paradigm (Sutton and Barto 2018). In this paper, we use a classical reinforcement technique known as TD learning first proposed by Sutton (1988a). Over a set of 40 problem instances, heuristic policies based on our TD-learning algorithms generate expected revenues that are competitive with the bid-price policy derived from the ALP approach in 60% of instances.

The remainder of the paper is organized as follows. Section 2 formulates the NRM problem, reviews the ALP approach, and illustrates the non-optimality of the ALP bid-price policy using an example taken from Talluri and van Ryzin (1998). Section 3 formulates the TD learning algorithm for the NRM problem. Section 4 details the results of a computational study. Finally, we conclude in Section 5.

2. Preliminaries: The ALP Approach for Network Revenue Management

We consider the NRM problem as a finite-horizon dynamic program with J products each consuming combinations of I resources, with requests arriving over T periods. The set of resources $\mathcal{I} = \{1, \ldots, I\}$ has capacity vector $\mathbf{c} = (c_1, \ldots, c_I)$. The set of products $\mathcal{J} = \{1, \ldots, J\}$ has reward (fare) vector $f = (f_1, \ldots, f_J)$. We denote the product/resource co-incidence matrix $A = [a_{ij}]$ with characteristic element

$$\mathbf{a}_{i,j} = \begin{cases} 1, & \text{if resource } i \text{ is used by product } j, \\ 0, & \text{otherwise.} \end{cases}$$

The set of resources used by product j is the j-th column of A, which is denoted by $\mathbf{a}^j$.

In each period t, a request arrives for product j with probability $\lambda_{t,j}$. Without loss of generality, we assume there always exists a product representing the "null request", having zero fare and consuming no resources. Thus, $\sum_j \lambda_{t,j} = 1$ for all t. The state space is given by $\mathcal{X} = \{\mathbf{x} \in \mathbb{Z}_+^I | x_i \leq c_i, \forall i\}$ and the feasible action set in state $\mathbf{x}$ is given by $\mathcal{U}(\mathbf{x}) = \{\mathbf{u} \in \{0,1\}^{\mathcal{J}} : \mathbf{a}^j u_j \leq \mathbf{x}\}$. The decision maker seeks to maximize the total expected revenue over the horizon by accepting or rejecting requests when they arrive.

The NRM problem has an intuitive optimal policy whereby the firm accepts only such requests for which the reward is greater than the opportunity cost of consuming the resources. i.e,

$$u_j = \begin{cases} 1, & \text{if } \mathbf{x} \geq \mathbf{a}^j \wedge f_j \geq v_{t+1}^*(\mathbf{x}) - v_{t+1}^*(\mathbf{x} - \mathbf{a}^j), \\ 0, & \text{otherwise.} \end{cases}$$

Here, $v_t^*(\mathbf{x})$ denotes the optimal value-to-go function. The NRM is a sequential decision making problem and consequently can be intuitively cast as a dynamic program (DP) with optimality conditions given by:

$$v_t(\mathbf{x}) = \max_{\mathbf{u} \in \mathcal{U}(\mathbf{x})} \sum_j \lambda_{t,j} u_j [f_j - (v_{t+1}(\mathbf{x}) - v_{t+1}(\mathbf{x} - \mathbf{a}^j))] + v_{t+1}(\mathbf{x}) \quad \forall\, t, \mathbf{x} \in \mathcal{X}. \tag{1}$$

The boundary condition is $v_{T+1}(\mathbf{x}) = 0$ for all $\mathbf{x} \in \mathcal{X}$.

The dynamic program (1) can be equivalently written as a linear program as follows (Adelman 2007):

$$\begin{aligned} (\text{LP}) \quad \min_{\{v_t(\cdot)\}_{\forall t}} \quad & v_1(\mathbf{c}) & (2) \\ \text{s.t.} \quad & v_t(\mathbf{x}) - v_{t+1}(\mathbf{x}) + \sum_j \lambda_{t,j} u_{t,j} \left[v_{t+1}(\mathbf{x}) - v_{t+1}(\mathbf{x} - \mathbf{a}^j) \right] \geq \sum_j \lambda_{t,j} u_{t,j} f_j, \\ & \qquad \forall t, \mathbf{x} \in \mathcal{X}_t, \mathbf{u}_t \in \mathcal{U}(\mathbf{x}). & (3) \end{aligned}$$

However, the number of variables and constraints in (LP) increases exponentially in the number of resources I and the number of products J. Thus, solving (LP) is as hard as solving the dynamic program (1). To achieve tractability, a key idea in the ALP approach is to represent the value function $v_t(\mathbf{x})$ by the weighted basis functions:

$$v_t(\mathbf{x}) \approx \theta_t + \sum_{b\in\mathcal{B}} V_{t,b}\phi_b(\mathbf{x}), \quad \forall t, \mathbf{x} \in \mathcal{X}_t, \tag{4}$$

where $\phi_b : \mathcal{X} \to \mathbb{R}$ for $b \in \mathcal{B}$ is a set of pre-specified basis functions and $\mathcal{B}$ is some index set. The parameter $V_{t,b}$ is the weight of the basis function $\phi_b(\cdot)$ in period t, and θ_t is a constant offset.

Two common approximation architectures for the NRM problem are affine and SPL approximations. In this paper, we focus on the SPL approximation, which is known to be stronger than the affine approximation. The SPL approximation is given by (Vossen and Zhang 2015):

$$v_t(\mathbf{x}) \approx \theta_t + \sum_i \sum_{k=1}^{x_i} W_{t,i,k}, \quad \forall t, \mathbf{x} \in \mathcal{X}_t. \tag{5}$$

In (5), $W_{t,i,k}$ can be interpreted as the value of the k-th unit of resource i in period t.

Substituting (5) into (LP) yields a problem with a polynomial number of variables. However, it still has exponentially many constraints. Specialized column generation (Adelman 2007, Meissner and Strauss 2012) and constraint sampling (de Farias and Van Roy 2004, Farias and Van Roy 2007) methods were developed to handle these constraints. More recently, Vossen and Zhang (2015) show that the ALP associated with the approximation in (5) can be written compactly as the following *reduced program*:

$$\text{(R)}\ z_R = \max_{\mathbf{p},\mathbf{q},\mathbf{z}} \sum_{t,j} \lambda_{t,j} f_j q_{t,j}$$

$$\text{s.t. } p_{t,i,k} = \begin{cases} 1, & \text{if } t = 1, \\ p_{t-1,i,k} - \sum\limits_{m\in\mathbf{a}_i} \lambda_{t-1,l}(z_{t-1,i,m,k} - z_{t-1,i,m,k+1}), & \text{if } t > 1 \end{cases}$$

$$\forall t, i, k : a_{i,j} = 1, \tag{6}$$

$$q_{t,j} = z_{t,i,j,1}, \qquad \forall t, i, j : a_{i,j} = 1, \tag{7}$$

$$z_{t,i,j,k+1} \le z_{t,i,j,k}, \qquad \forall t, i, j, k : a_{i,j} = 1, \tag{8}$$

$$z_{t,i,j,k} \le p_{t,i,k}, \qquad \forall t, i, j, k : a_{i,j} = 1. \tag{9}$$

The dual solution to (R) can be used to construct a bid-price control policy. We will make use of this bid-price control policy in our computational study both as a comparison benchmark and an initialization policy for our TD learning algorithms. According to the definition in Talluri and van Ryzin (1998), a bid-price policy specifies a set of bid-prices for each resource at each point in time such that the request for a product is accepted if and only if there is available capacity and the fare exceeds the sum of the bid-prices for all units of the resources used by the product. It is natural to use the optimal dual values of constraint (6), $\{W^*_{t,i,k}\}_{\forall t,i,k}$, to approximate the value function following (5) as

$$v_t(\mathbf{x}) \approx v_t^R(\mathbf{x}) = \theta_t + \sum_i \sum_{k=1}^{x_i} W^*_{t,i,k}, \quad \forall t, \mathbf{x} \in \mathcal{X}_t. \tag{10}$$

A bid-price policy can be constructed such that

$$u_{t,j} = \begin{cases} \prod_{i \in \mathbf{a}^j} \mathbb{1}(x_i \geq 1), & \text{if } f_j \geq \sum_{i \in \mathbf{a}^j} W^*_{t+1,i,x_i}, \\ 0, & \text{otherwise,} \end{cases} \qquad \forall t, \mathbf{x}, j. \tag{11}$$

We refer to (11) as the *SPL* bid-price policy, which is one of the strongest policies in the literature for the NRM problem.

Talluri and van Ryzin (1998) use an example to illustrate the non-optimality of a particular form of bid-price policies. Their example can also be used to show that the bid-price policy specified in (11) is not optimal. We replicate their example below but change the time index to match our notation.

Example 1 (Talluri and van Ryzin (1998), Section 3.1). Consider a network with two resources and three products. There are two local products (P1 and P2), each with a fare of \$250, and one through product (P3) with a fare of \$500. Each local product uses one resource, while the through product uses both resources. Each resource has one unit of capacity. The problem data are shown in Table 1.

Period (t)	Product: $\mathbf{a}^j$	Fare	Probability
1	P1: (1, 0)	\$250	0.3
	P2: (0, 1)	\$250	0.3
	P3: (1, 1)	\$500	0.4
2	No arrival		0.2
	P3: (1, 1)	\$500	0.8

Table 1 Problem Data for Example 1.

In the first period, the probability of arrival for each of the local products is 0.3, and 0.4 for the through product. In the second period, there is no demand for local products, and

the probability of arrival for the through product is 0.8. An optimal policy will reject both local products and only accept the through product in period 1. If the through product does not arrive in period 1, the policy will accept the through product in period 2 if it arrives. The optimal expected revenue is \$440.

To enforce the optimal policy in Example 1, the bid-prices need to satisfy $W^*_{2,1,1} > 250$, $W^*_{2,2,1} > 250$, and $W^*_{2,1,1} + W^*_{2,2,1} \leq 500$, which obviously is impossible. According to the definition in Talluri and van Ryzin (1998), the best a bid-price policy can do in this example is to reject all demand in period 1 and accept only the through itinerary (if it arrives) in period 2, yielding an expected revenue of \$400. In the next section, we show that a TD learning approach can overcome the pitfall in this example.

3. Temporal Difference (TD) Learning for Policy Evaluation

TD learning, first proposed by Sutton (1988b), is a reinforcement learning technique that uses an iterative approach to train a prediction model in an online fashion using in part the previous predictions of the model. This is also known as *bootstrapping*, which is a central feature of DP that is accomplished via the backwards induction loop. Unlike DP, however, TD methods do not require a formalized model of the environment, the rewards for specific actions, or state-transition probabilities in order to converge on a solution. Rather, they require only signals relating to rewards and state transitions and are completely agnostic with respect to the methods used to generate these signals. This flexibility allows TD methods to handle a broader class of approximation architectures, including the one in Zhang et al. (2021b).

In Section 3.1, we present two tabular TD algorithms, which adopt the canonical TD learning algorithms for the NRM problem. They are used to demonstrate the full power of the TD methods. However, due to the explosion in the state space, the tabular TD algorithms are only suitable for small-size problems. To work around this, we adopt TD learning algorithms with value function approximation, such as the well-known TD(λ) algorithm, in Section 3.2. In the literature, the TD learning algorithms are designed for infinite horizon problems (Sutton and Barto 2018), while we apply them to the NRM problem, which is a finite-horizon problem.

3.1 The Tabular TD Algorithms

Algorithm 1 details the canonical TD learning algorithm used for policy evaluation that makes use of a tabular representation of the value function. This algorithm, however, does not address a crucial consideration that arises in reinforcement learning, the trade-off between *exploration* and *exploitation*. That is, an agent much choose whether to *exploit* the information it already has by selecting the estimated optimal action or it must *explore* the state-action space by selecting random, feasible actions in order to gather more information. When using TD for policy evaluation, a given policy π may never visit certain states which may hold important information for the value function. Therefore, it is desirable to allow the agent to deviate from the policy, π, with some small probability in order to gain more valuable information. Fortunately, it is very easy to adapt the canonical TD algorithm to incorporate this ϵ-exploration. **Algorithm 2** details the tabular TD algorithm with ϵ-exploration.

Algorithm 1: Tabular TD learning for NRM policy evaluation

Input: a policy, π, to be evaluated

Parameter: a step size $\alpha \in (0,1]$

Initialize $V_t(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{X}$, $t \in \{1,\ldots,T\}$ arbitrarily, incorporating boundary conditions

Loop for each sample path, P:

  Set initial state $\mathbf{x} = \mathbf{c}$

  Loop for each period, t:

    Set acceptance decision $\mathbf{u} \in \{0,1\}^J$ according to $\pi(\mathbf{x})$

    Take action $\mathbf{u}$, observe reward R, and let $\mathbf{x}' = \mathbf{x} - u_j \mathbf{a}^j$

    $V_t(\mathbf{x}) \leftarrow V_t(\mathbf{x}) + \alpha[R + V_{t+1}(\mathbf{x}') - V_t(\mathbf{x})]$

    $\mathbf{x} \leftarrow \mathbf{x}'$

Algorithm 2: Tabular TD learning for NRM policy evaluation with ϵ-exploration

Input: a policy, π, to be evaluated

Parameter: a step size $\alpha \in (0,1]$

Initialize $V_t(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{X}$, $t \in \{1,\ldots,T\}$ arbitrarily, incorporating boundary conditions

Loop for each sample path, P:

  Set initial state $\mathbf{x} = \mathbf{c}$

  Loop for each period, t:

    Set acceptance decision $\mathbf{u} \in \{0,1\}^J$ according to $\pi(\mathbf{x})$.

    With probability ϵ, set acceptance decision u_j as 0 or 1 with equal chance for each product j. Set $u_j = 0$ if $u_j = 1$ is infeasible.

    Take action $\mathbf{u}$, observe reward R and let $\mathbf{x}' = \mathbf{x} - u_j \mathbf{a}^j$

    $V_t(\mathbf{x}) \leftarrow V_t(\mathbf{x}) + \alpha[R + V_{t+1}(\mathbf{x}') - V_t(\mathbf{x})]$

    $\mathbf{x} \leftarrow \mathbf{x}'$

Policy	Expected Revenue
SPL	350
TD, $\epsilon = 0$	350
TD, $\epsilon = 0.03$	440

Table 2 Performance of TD Learning on Example 1

In order to motivate the application of TD learning to the NRM problem, we revisit Example 1. Table 2 lists the total expected revenues for the SPL bid-price policy and the greedy policies based on the value function approximations generated by the tabular TD learning algorithms with and without ϵ-exploration. Notice that the TD algorithm is able to discover the value function approximation that generates the optimal policy, with expected revenue \$440, only when the agent is allowed to deviate from the SPL bid-price policy, thus demonstrating the importance of ϵ-exploration to this technique.

This reliance on ϵ-exploration and sampling leads to large variability in the outcome of the TD algorithms. This variability is potentially one of the reasons that the literature has not given as much attention to reinforcement learning methods as those based on mathematical programming. However, the inherent randomness of reinforcement learning methods has been widely tolerated in other fields due to the fact that these methods are both easy to implement and, in many cases, quite effective.

3.2 TD Learning for Approximate Policy Evaluation

The initial results with the tabular TD algorithms on Example 1 demonstrate the potential of TD learning to improve over ALP based bid-price control policies, but such algorithms are limited by the reliance on the tabular representation of the states. For larger instances, the well known *curse of dimensionality*, or state space explosion, is a practical barrier to this methodology. However, we may adapt the method to utilize instead an approximation of the value function based on the combination of a set of basis functions. In this way, we may avoid the issue of state space explosion, but at the cost of the accuracy of our value function representation. TD learning with functional approximation received significant attention in the machine learning literature through the 1990s with early work suggesting that applications of the algorithm to complex problems and games had the potential for success (Tesauro 1992), but there were few guarantees of statistical performance in this early work. The work of Tsitsiklis and Van Roy (1997) provided some of the first convergence guarantees for TD algorithms with function approximation under general conditions including the boundedness of expectations and linear independence of basis functions.

One class of TD algorithms making use of value function approximations known as TD(λ), proposed in Sutton (1988a), has been well studied in the literature with applications in a wide number of fields including computer science, neuroscience, and psychology (Tesauro et al. 1995, O'Doherty et al. 2003). TD(λ) is more precisely a continuum of algorithms that are parameterized by $\lambda \in [0, 1]$. The λ parameter is used to define the relative

significance of previous feature-set observations via a vector, $\mathbf{z}$, known as the *eligibility trace*. This eligibility trace is a means of determining and maintaining an update direction, making TD learning similar to gradient descent methods. Like gradient descent methods, there are several ways to construct the eligibility trace. Van Seijen et al. (2016) provides a detailed analysis of various eligibility trace methods. We also propose a novel eligibility trace, which we call the *salience trace* that makes use of the binary structure of the set of basis functions we examine. We call the variation of the TD algorithm that uses this salience trace $TD\text{-}\rho$.

Implementing the TD(λ) algorithm in the NRM problem setting poses some unique challenges. First, the canonical application of TD(λ) assumes an infinite horizon problem while the NRM problem is a finite-horizon problem. It has been shown in the literature that, in general, finite horizon TD(λ) will converge (Asis et al. 2019). However, the ALP based approximation architectures that we use assume that the feature set, or equivalently the set of basis functions, is independent across time periods. The independence of basis functions across time periods motivates the decision to train a family of value function approximations, each representing a single time period, in the implementation given in this paper. **Algorithm 3** details the TD(λ) algorithm as it is applied to the NRM problem which follows closely with that presented in Sutton (1988a), but is adapted to train a family of time-dependent value function approximations.

Not surprisingly, the reliance of the TD(λ) algorithm on a set of predefined parameters suggests that strong performance of the algorithm requires careful tuning of the parameters. In general, this is an important practical consideration. The parameters which may be selected by the researcher include the approximation architectures, the step size (α), the trace decay rate (λ), the initialization of the weight vector $\mathbf{w}$, and the form of the eligibility trace. What follows is a discussion of each of these parameters.

Approximation Architectures: The choice of approximation architecture is critically important to both the ALP and reinforcement learning based methods. One of the advantages of reinforcement learning based methods is the flexibility afforded the researcher when selecting an approximation architecture. While the ALP methods are generally restricted to linear approximations due to computational tractability, the TD-learning method we propose is capable of operating with an arbitrary functional approximation to the value function. This allows for a much broader range of functional approximations. For the sake of comparison, we focus on two approximation architectures including the SPL approximation (5), as well as the product-based approximation as given in (Zhang et al. 2021b).

Algorithm 3: TD(λ) for approximate policy evaluation

Input: an initial policy, π, to be evaluated

Input: a family of basis functions $\hat{v}_t : \mathcal{X} \times \mathbb{R}^d \to \mathbb{R}$ for $t = 1, \ldots, T$, with the boundary condition enforced.

Parameter: a step size $\alpha > 0$, trace decay rate λ

Initialize weights, $\mathbf{w}_t$, for all t arbitrarily

Initialize $\mathbf{z}_t \leftarrow \mathbf{0} \, \forall \, t$

Loop for each sample path:

  Initialize initial state $\mathbf{x}$

  Loop for each period:

    $\mathbf{u} \in \{0,1\}^J \leftarrow$ action given by $\pi(\mathbf{x})$

    Take action $\mathbf{u}$, observe reward R and next state $\mathbf{x}' = \mathbf{x} - u_j \mathbf{a}^j$

    $\mathbf{z}_t \leftarrow \lambda \mathbf{z}_t + \Phi_t(\mathbf{x})$, where $\Phi_t(\mathbf{x})$ is the gradient of the value function approximation.

    $\delta \leftarrow R + v_{t+1}(\mathbf{x}') - v_t(\mathbf{x})$

    $\mathbf{w}_t \leftarrow \mathbf{w}_t + \alpha \delta \mathbf{z}_t$

    $\mathbf{x} \leftarrow \mathbf{x}'$

Zhang et al. (2021b) consider the following value function approximation:

$$v_t(\mathbf{x}) \approx \theta_t + \sum_j \mathbb{1}\left(\mathbf{x} \geq \mathbf{a}^j\right) \sum_{i \in \mathbf{a}^j} \sum_{k=1}^{x_i} \underline{W}_{t,i,j,k}, \quad \forall t, \mathbf{x} \in \mathcal{X}_t. \tag{12}$$

This approximation architecture is referred to as the *product-based* approximation (PB). We observe that the product-based approximation can capture some network effects of the NRM problem. $\underline{W}_{t,i,j,k}$ in the product-based approximation can be interpreted as the contribution of product j to the value of the k-th unit of resource i in period t. In this way, the product-based approximation allows the marginal value of each resource-unit to be distributed heterogeneously across the set of products that use resource i. Furthermore, the indicator function $\mathbb{1}\left(\mathbf{x} \geq \mathbf{a}^j\right)$ tracks the resource availability for each product. Zhang et al. (2021b) show that the product-based approximation yields a smaller approximation error and a tighter upper bound on expected revenue than the SPL approximation, but at the cost of a higher computational complexity, which can make the solution to the resulting ALP intractable even with high performance commercial LP solvers. However, the TD-learning algorithm is able to handle it.

We can construct a policy by using (12) as follows. Consider two products $j, m \in \mathcal{J}$. Let

$\underline{x}^{m,j} = \min_{i' \in \mathbf{a}^m \cap \mathbf{a}^j} x_{i'}$. As a convention, we take $\underline{x}^{m,j} = 0$ if $\mathbf{a}^m \cap \mathbf{a}^j = \emptyset$. Thus, $\underline{x}^{m,j}$ gives the minimum capacity of the resources shared by products j and m; it takes the value of 0 if products j and m do not share any resource. A bid-price that can be used to decide whether to accept product j is then given by

$$\Delta v^P_{t+1}(\mathbf{x}) = \sum_{m:\mathbf{a}^m \cap \mathbf{a}^j \neq \emptyset} \left\{ \mathbb{1}\left(\underline{x}^{m,j} = 1\right) \sum_{i \in \mathbf{a}^m} \sum_{k=1}^{x_i} W^*_{t+1,i,m,k} + \mathbb{1}\left(\underline{x}^{m,j} > 1\right) \sum_{i \in \mathbf{a}^m \cap \mathbf{a}^j} W^*_{t+1,i,m,x_i} \right\}. \tag{13}$$

The corresponding bid-price policy is given by

$$u_{t,j} = \begin{cases} \prod_{i \in \mathbf{a}^j} \mathbb{1}(x_i \geq 1), & \text{if } f_j \geq \Delta v^P_{t+1}(\mathbf{x}), \\ 0, & \text{otherwise,} \end{cases} \quad \forall \mathbf{x}, j. \tag{14}$$

We refer to this policy as the product based (PB) bid-price policy.

Eligibility Trace: Another important decision to make is the manner in which to make updates to the *eligibility trace*, $\mathbf{z}$. The eligibility trace is a vector that tracks the observations of the feature set over the course of the simulation, and can be considered to be a vector describing the *direction* of updates to the weight vector similar to gradient descent approaches. The distance of each update is determined by the size of the temporal difference, δ. In the canonical form of the TD(λ) algorithm, this vector is updated after each state transition, and reset to 0 at the end of each *episode*. An episode is simply a sample path between the initial state and the terminating state. In the case of the NRM problem, each episode might be considered to be a sample path over the finite time horizon. As a result of this particular structure, each episode of TD(λ) learning will encounter the eligibility trace associated with a given time period exactly one time. Therefore, if this vector were to be reset to 0, there would never be any accumulation of information about previous state space observations. For this reason, our implementation of the algorithm does not reset the traces at the beginning of each sample path. In this way, the entire simulation can be considered to be a single episode with information being accumulated over all simulated arrival sequences.

While there exist in the literature several different approaches to the eligibility trace, the canonical TD(λ) algorithm defines the trace by the update rule,

$$\mathbf{z} \leftarrow \lambda \mathbf{z} + \nabla \hat{v}(\mathbf{x}, \mathbf{w}).$$

We observe that when $\lambda = 0$, the algorithm operates much like a single-step backup algorithm because the trace at any given step is equal to the current gradient of the value

function approximation. Conversely, when $\lambda = 1$, the algorithm accumulates all previous feature set observations similar to a Monte-Carlo algorithm — though updates to the model are performed at each iteration rather than at the end of the simulation. In this way, the λ parameter places the algorithm in a continuum between single step backup methods and pure Monte-Carlo methods. For a given problem instance, λ may be optimized via some heuristic method, though in practice the choice of λ is often made arbitrarily, with a value near 0.5 often working quite well.

Returning to the update rule, let $\Phi(\mathbf{x})$ denote the gradient of the value function approximation. For the SPL approximation, the gradient w.r.t. the weight vector is as follows:

$$\phi^{SPL}_{t,i,k}(\mathbf{x}) = \mathbb{1}\{\mathbf{x}_i \geq k\} \quad \forall\, t = 1, \ldots, T.$$

For the product-based approximation, the gradient w.r.t. the weight vector is as follows:

$$\phi^{PB}_{t,i,j,k}(\mathbf{x}) = \mathbb{1}\left(\mathbf{x} \geq \mathbf{a}^j\right) \times \mathbb{1}\{\mathbf{x}_i \geq k\} \quad \forall\, t = 1, \ldots, T.$$

The update rule for the canonical eligibility trace for the NRM problem is simply

$$\mathbf{z} \leftarrow \lambda \mathbf{z} + \Phi(\mathbf{x}).$$

This observation about the peculiarity of the eligibility trace updates lead us to propose a modification of the canonical TD algorithm, TD-ρ, that makes use of the problem-specific feature set architecture to maintain an eligibility trace with a probabilistic interpretation. We first observe that the basis functions we have selected for our approximation architecture are derived from the two ALP methods discussed above, which use as their approximation architecture a collection of binary-valued indicator functions. The binary-valued nature of the feature set informs the development of the following update step for the eligibility trace

$$\mathbf{z} \leftarrow \mathbf{z} + \frac{\Phi(x) - \mathbf{z}}{k}.$$

This update step is simply computing an element-wise running average of the value of each basis function in a given time period across all simulated arrival sequences. The binary nature of each basis function implies a probabilistic interpretation wherein each element of z_t takes a value in $[0, 1]$ which corresponds to the observed proportion of simulated paths in which the basis function associated with that element took a value of 1 in the given period. This modified eligibility trace, which we call the *salience trace* tracks the relative importance of each basis function to the value function approximation within a given period, which we argue should yield a strong direction for our weight updates. **Algorithm 4** details the TD-ρ algorithm as it is applied to the NRM problem which incorporates the *salience trace*.

Algorithm 4: TD-ρ for approximate policy evaluation

Input: a policy, π, to be evaluated

Input: a family of basis functions $\hat{v}_t : \mathcal{X} \times \mathbb{R}^d \to \mathbb{R}$ for $t = 1, \ldots, T$, with the boundary condition enforced.

Parameter: a step size $\alpha > 0$

Initialize weights, $\mathbf{w}_t$, for all t arbitrarily

Initialize $\mathbf{z}_t \leftarrow \mathbf{0} \, \forall \, t$

Loop for each sample path, indexed by k:

  Initialize initial state, $\mathbf{x}$

  Loop for each period:

    $\mathbf{u} \in \{0,1\}^J \leftarrow$ action given by $\pi(\mathbf{x})$

    Take action $\mathbf{u}$, observe reward R and next state $\mathbf{x}' = \mathbf{x} - u_j \mathbf{a}^j$

    $\mathbf{z}_t \leftarrow \mathbf{z}_t + \frac{\mathbf{\Phi}_t(\mathbf{x}) - \mathbf{z}_t}{k}$, where $\Phi_t(\mathbf{x})$ is the gradient of the value function approximation.

    $\delta \leftarrow R + \hat{v}_{t+1}(\mathbf{x}', \mathbf{w}_{t+1}) - \hat{v}_t(\mathbf{x}, \mathbf{w}_t)$

    $\mathbf{w}_t \leftarrow \mathbf{w}_t + \alpha \delta \mathbf{z}_t$

    $\mathbf{x} \leftarrow \mathbf{x}'$

Step Size: Being an iterative updating method, TD-learning requires the researcher to provide a means of determining the step size each time an update occurs. Let α_k be the step size at step k. Various methods exist to determine step size so as to guarantee convergence under the classic conditions

$$\sum_{k=1}^{\infty} \alpha_k = \infty, \quad \sum_{k=1}^{\infty} \alpha_k^2 < \infty.$$

This set of conditions is one of the critical assumptions in the convergence proof given in Tsitsiklis and Van Roy (1997). Naturally, this implies that a step size of $\frac{1}{k}$ would guarantee convergence. However, in practice the performance of this step size is rather weak. It can also be shown that the algorithm converges with a constant step size if the value function is Lipschitz continuous and the step size is sufficiently small. Further, there exist a number of *adaptive* step size strategies that seek to leverage online information to control the step size. One such adaptive algorithm based on computing upper bounds on the allowable step size is given in Dabney and Barto (2012). We implement a small, constant step size of $1e^{-3}$ in our experiments.

Initial Policy The TD-learning approach we implement is a policy evaluation algorithm for which the performance is usually highly sensitive to the initial policy. To examine this sensitivity, our numerical study includes two initial policies. The first one is the naive initialization, which accepts any feasible request. The second one is the SPL bid-price policy from the ALP approach.

3.3 Iterative Policy Improvement

We also consider an iterative policy improvement algorithm that makes use of TD-learning to generate greedy policies with respect to the value function approximations generated at each iteration. The procedure is as follows: starting with some initial policy, π^0, a series of value function approximations, $\{V_t^{\pi^0} : t = 1, \ldots, T\}$, is generated using TD learning. The algorithm then trains a new family of value function approximations by conducting the simulation again using a heuristic policy, π^1, which is simply a greedy policy with respect to the family of value functions trained in the previous iteration.

This procedure is quite similar to the well-known Rollout algorithm in that it trains each value function approximation at iteration k, $V_t^{\pi^k}$, using an approximation of the optimal value function $V_{t+1}^{\pi^{k-1}}$. It is a well known result that Rollout algorithms which make use of value function approximations can result in policy improvement in practice, though they often require careful tuning (Bertsekas and Tsitsiklis 1996).

4. Computational Study

In order to evaluate the performance of TD-learning methods on the NRM problem, we designed a computational study that examined the expected revenue generated by policies derived from the two TD learning algorithms described in the previous section under various approximation architectures and initializations. The computational study was performed on a machine with Intel Core i7-8700 processor, 16 GB RAM, and the Windows 10 operating system. The computer code is implemented in Python 3.0 with extensive use of the NumPY library. The ALP approach makes use of Gurobi 9.0 solver.

4.1 Test Instances

For the initial experiments using tabular TD learning, we used a small problem instance, given in Figure 3, which is representative of a two node airline network with two fare classes. For the computational study on the TD learning algorithms with value function approximation we consider a set of larger instances with a hotel structure. These instances were selected due to their strong network effects, and are a reconstruction of the 50-period hotel instances evaluated in Zhang et al. (2021b). In the hotel instances, each resource represents a room-night. Each instance contains I resources. Products can be interpreted as requests for room reservations for one or more consecutive nights. A customer request may span a set of consecutive nights from $i = 1, \ldots, I$ to $i' = i, i+1, \ldots, I$. Thus, there are $\frac{I(I+1)}{2}$ different itineraries which are included in each instance. Each product request has two fare classes, with the high fare class offering a reward that is a κ-multiple of the low-fare class. The revenue for the low-fare class for each night, i, denoted as f_i, is drawn from a discrete uniform distribution between 1 and 100. Furthermore, the total revenue generated by the low-fare class for a given request is given as the sum of the low-fare class for each night included in the itinerary. The arrival probabilities, $\lambda_{t,j}$, for each product in each period are generated according to the method described in Ma et al. (2020) such that high-fare requests tend to arrive with higher probability in later periods of the selling horizon. The initial capacity of each room-night, i, is determined according to an instance-specific load factor, α. We consider 40 instances in total with $I \in \{2, 3, 4, 5\}$, $\kappa \in \{4, 8\}$, and $\alpha \in \{1.0, 1.2, 1.6, 2.2, 3.0\}$. For a more detailed description of the problem instances, refer to Zhang et al. (2021b).

	x_1	x_2	Reward
p_1	0	1	150
p_2	1	0	250
p_3	1	1	375
h_1	0	1	600
h_2	1	0	1000
h_3	1	1	1500

(a) Products

t	p_1	p_2	p_3	h_1	h_2	h_3	None
1	0.1	0.1	0.6	0.0	0.0	0.0	0.2
2	0.1	0.2	0.4	0.1	0.1	0.0	0.1
3	0.3	0.3	0.1	0.05	0.15	0.0	0.1
4	0.3	0.0	0.4	0.1	0.0	0.2	0.1
5	0.0	0.0	0.65	0.0	0.0	0.3	0.05

(b) Arrival Probabilities

Table 3 Tabular TD Problem Instance, Capacity = 2

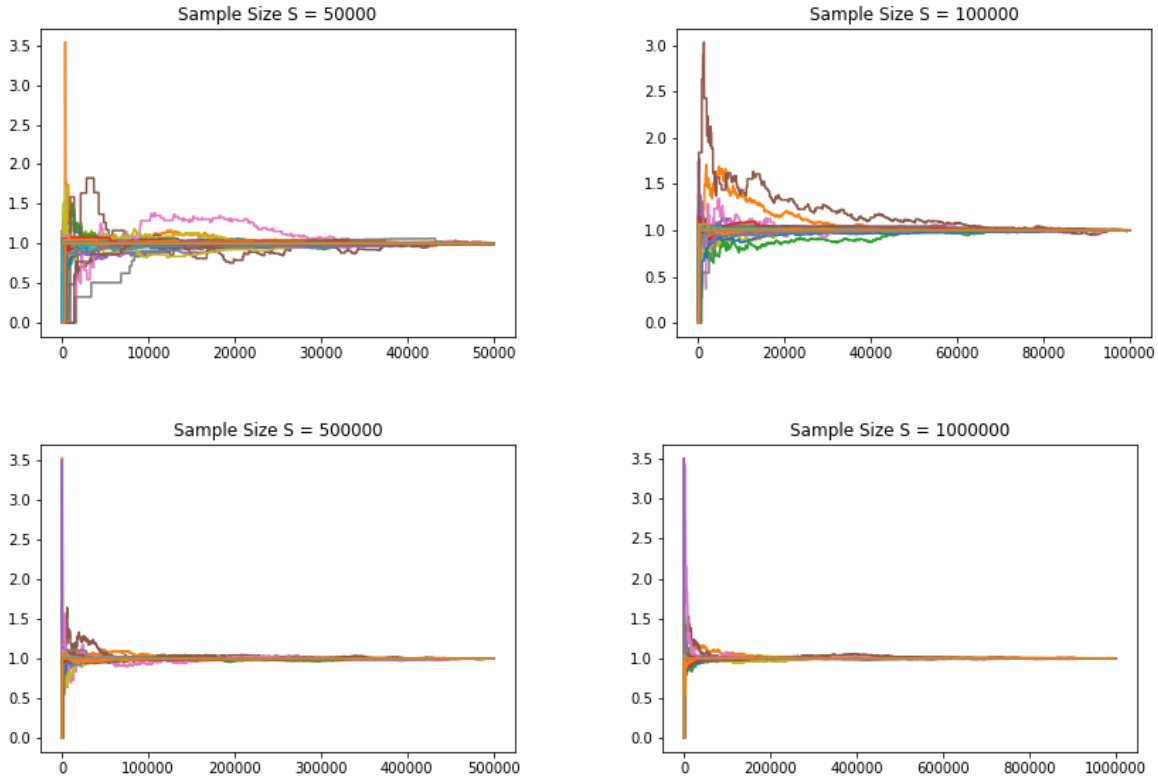

Figure 1 Convergence of Value Function Approximations under Tabular TD Learning

4.2 Performance of the Tabular TD Algorithms

The first question we chose to examine is whether, and how quickly, the tabular TD algorithm converges to a stable value approximation for each state. Figure 1 illustrates the observed convergence behavior of the value approximations for each state over various sample sizes. We observe that the value approximations for each state do converge to a stable value over time, and we used this information to set an appropriate sample size for other experiments.

Table 4 shows the results of our experiments with multiple policies. The table details the mean revenue over a sample of 100,000 paths for the initial policies, π^{SPL}, and for

Initial Policy	π^{SPL}	V^{π}
Naive	892	1327
SPL	1209	1334
Naive (ϵ)	894	1328
SPL (ϵ)	1209	1328

Table 4 Expected Revenue Results on Test Instance with $\epsilon = 0.03$

the policies given by the value function approximation trained on V^{π}. We use both naive policy and SPL bid-price policy to initialize. The rows marked with an ϵ in parenthesis were trained using an "exploration" technique, where with some small probability, ϵ, the agent is allowed to take a *random* action at a given time step instead of the action given by π. This is called *Epsilon Exploration*, which is discussed below.

Any reinforcement learning algorithm must concern itself with the precise nature of the agent's interaction with the environment, as it is the actions of the agent that determines the region of the state space examined by the algorithm. In the case of deterministic policies like the bid-price policies we have used, there is a possibility that the agent will never reach certain important states. For this reason, it is often considered a good practice in the reinforcement learning literature to include some element of *randomness* in the agent's behavior. This is the principle behind the ϵ-exploration we have used in our experiments. The algorithm is designed to allow the agent, with small probability ϵ, to take a random, feasible action at a given time step. Looking at Table 4, we observe that this exploration step may potentially yield some policy improvement for the Tabular TD algorithm, but the impact is unclear.

In addition to the policy evaluation via TD that we have studied, we are also interested in determining the potential for policy *improvement* using the TD algorithm. The value function associated with a given policy can be used to construct a new greedy policy. If we replace the original policy with this new value-function informed policy and use our TD algorithm to discover the value function associated with this new policy, we could potentially improve the policy over time.

	0	1	2	3	4	5	6	7	8	9
Naive	1336	1336	1336	1337	1337	1337	1337	1337	1337	1338
SPL	1339	1398	1339	1339	1339	1339	1339	1339	1339	1339
Naive (ϵ)	1328	1330	1332	1333	1334	1335	1335	1336	1336	1337
SPL (ϵ)	1339	1339	1339	1339	1339	1339	1339	1339	1339	1339

Table 5 Policy Improvement Results with Tabular TD

	0	1	2	3	4	5	6	7	8	9
TD(λ)	991	1048	1205	1200	1217	1258	1291	1215	1205	1273
TD-ρ	923	966	984	1050	1038	1073	1104	1108	1143	1239

Table 6 Policy Improvement Results with Naive initialization and SPL approximation architecture

Table 5 details the results of an experiment that performed 10 iterations of the procedure outlined in Section 3.3 using the tabular representation. Some slight policy improvement was observed for the Naive initialized trials. However it appears that in all cases the algorithm discovers a near-optimal policy with a single pass. Thus, it might have little room to improve. On the other hand, Table 6 details the results of the experiment performed using TD learning with the SPL approximations. The initial policy is the Naive policy. There are two noteworthy observations. First, there is a significant decrease in performance for TD learning with the SPL approximations relative to tabular TD. Second, there is evidence of policy improvement across successive iterations, though policy improvement is not guaranteed between iterations.

4.3 Policy Improvement with the TD Algorithms with Value Function Approximations

We incorporate a policy improvement method in our computational study. **Algorithm 5** details the policy improvement with TD-learning procedure. It is a relatively simple procedure that replaces the policy π with a greedy policy w.r.t. the family of value function approximations trained by the TD-learning algorithm in each iteration. The process continues until some stopping condition is met. A stopping condition can be a cap on the number of iterations, or something else.

Figure 2 shows the performance of the TD-ρ algorithm with a naive initialization policy and the SPL architecture across 15 policy improvement iterations. Each line represents the performance trajectory of one instance, normalized by the maximum value. The figure shows two general patterns in the policy improvement performance. The first pattern, occurring in 37.5% of instances, involves the algorithm achieving the best performance in the first iteration and then declining in each subsequent iteration. The instances that showed this pattern ended the trial with an average performance decay of 6.9% relative to the maximum performance. The second pattern, occurring in the remaining instances, shows the expected behavior involving the algorithm starting at a low level of performance and then improving over subsequent iterations. These instances ended the trial with an average performance decay of 0.9% relative to the maximum performance. 35% of the

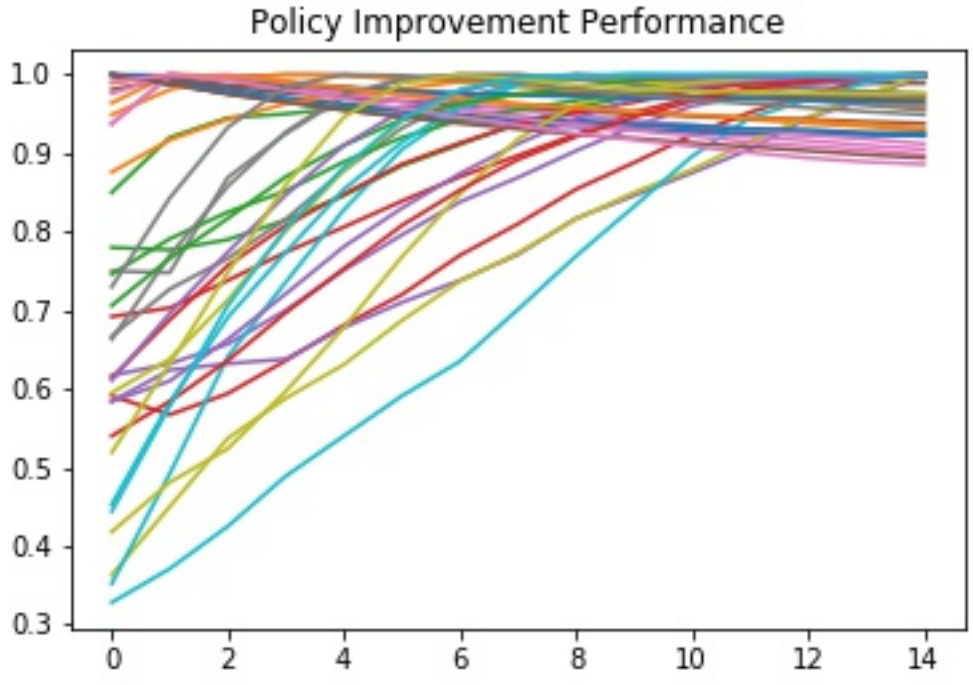

Figure 2 Policy Performance across Improvement iterations, TD-ρ with Naive initialization and SPL architecture

instances ended the trial at their maximum performance level, indicating that additional improvement might have been observed if the trial had been extended past 15 iterations. The average min-max gap in policy performance across the instances with the expected policy improvement pattern was 39.8%.

Our results indicate that there is a significant practical benefit to include the policy improvement procedure, though perhaps with some simple heuristics or early stopping conditions to prevent performance decay. In the computational study discussed in section 4.4, we set a stopping condition that enforced a limit of 20 iterations with an early stopping condition that would terminate the procedure if 3 consecutive iterations passed without any improvement in the performance of the policy. Additionally, we enforced a rule ensuring that the policy π would only be updated if the performance had improved in the current iteration.

4.4 Performance of the TD Algorithms with Value Function Approximations

We consider two initialization policies and two approximation architectures for TD(λ) and TD-ρ. Tables 7 and 8 detail the results of the $TD(\lambda)$ algorithm initialized with the bid-price policy derived from the SPL and with a naive initialization, respectively. Tables

Algorithm 5: TD-Learning with Policy Evaluation

Input: a policy, π, to be evaluated

Input: A TD-learning algorithm, $AlgTD$

Input: Stopping Condition for the policy improvement algorithm

Loop until Stopping Condition is met:

Let $\{v_t | t = 1, \ldots, T\} \leftarrow AlgTD(\pi)$

Let $\pi \leftarrow Greedy(\{v_t | t = 1, \ldots, T\})$

9 and 10 detail the results of the TD-ρ algorithms initialized again with the bid-price policy and the naive policy, respectively. The first column lists the instance parameters including the number of periods, τ, the number of unique resources, m, the fare scaling factor, κ, and the load parameter, α. The "Benchmarks" column contains the upper bound ("UB") computed from the ALP with SPL approximation and the expected revenue of the SPL bid-price policy ("π^{SPL}"). The "TD-ER" column lists the expected revenue obtained by TD-learning algorithm with the two approximation architectures, product-based (PB) approximation detailed in Zhang et al. (2021a) and the SPL approximation, respectively. Each test instance was trained on a sample of 200,000 arrival sequences and tested against an independent sample of 20,000 arrival sequences. The last two columns in each table report the percentage gain (loss) over the SPL bid-price policy.

Table 7 details the results of the $TD(\lambda)$ algorithm under the SPL bid-price initialization. The average loss with respect to the SPL bid-price policy is 9.06% for the PB approximation, and 6.75% for the SPL approximation, with a median loss of 4.44% and 5.87%, respectively. Across both approximation architectures, there was only one problem instance in which the $TD(\lambda)$ algorithm outperformed the SPL bid-price policy. The naive initialization of the $TD(\lambda)$ algorithm performed markedly worse in the computational study. With a mean loss of 16.5% and 38.4% for the PB and SPL approximation architectures, respectively.

The TD-ρ algorithm showed significant improvement over $TD(\lambda)$ on the hotel instances. Table 9 details the results of the study under the SPL bid-price initialization. In this study, the average loss with respect to the SPL bid-price policy was 2.42% for the PB approximation and only 0.14% for the SPL approximation. The median loss for PB approximation was 1.61%, and the SPL approximation showed a median gain over the bid-price policy of 0.29%. In total, the algorithm with the PB approximation outperformed the SPL bid-price policy on a single instance, while the SPL approximation outperformed the SPL bid-price

	Benchmarks		TD-ER		% Off π^{SPL}	
(τ, m, κ, α)	UB	π^{SPL}	PB	SPL	PB	SPL
(50, 2, 4, 1.0)	7251	7230	7090	7050	-1.936	-2.490
(50, 2, 4, 1.2)	5749	5698	5512	5645	-3.264	-0.930
(50, 2, 4, 1.6)	3638	3632	3394	3482	-6.553	-4.130
(50, 2, 4, 2.2)	6091	6014	5860	5966	-2.561	-0.798
(50, 2, 4, 3.0)	5159	5075	4993	4838	-1.616	-4.670
(50, 3, 4, 1.0)	6894	6768	6591	6605	-2.615	-2.408
(50, 3, 4, 1.2)	5175	5067	4822	4730	-4.835	-6.651
(50, 3, 4, 1.6)	3270	3208	3059	3039	-4.645	-5.268
(50, 3, 4, 2.2)	9484	9282	8841	8895	-4.751	-4.169
(50, 3, 4, 3.0)	2398	2324	2228	2143	-4.131	-7.788
(50, 4, 4, 1.0)	9747	9455	9080	9183	-3.966	-2.877
(50, 4, 4, 1.2)	8884	8599	8175	7681	-4.931	-10.676
(50, 4, 4, 1.6)	5524	5383	4850	4914	-9.902	-8.713
(50, 4, 4, 2.2)	7520	7211	6692	6589	-7.197	-8.626
(50, 4, 4, 3.0)	4694	4507	3948	3880	-12.403	-13.912
(50, 5, 4, 1.0)	15790	15136	11611	14246	-23.289	-5.880
(50, 5, 4, 1.2)	9406	8975	8319	8293	-7.309	-7.599
(50, 5, 4, 1.6)	8626	8274	7022	7405	-15.132	-10.503
(50, 5, 4, 2.2)	9090	8681	5461	7803	-37.093	-10.114
(50, 5, 4, 3.0)	3605	3433	2907	3050	-15.322	-11.156
(50, 2, 8, 1.0)	5505	5475	5496	5373	0.384	-1.863
(50, 2, 8, 1.2)	12224	12145	11974	12049	-1.408	-0.790
(50, 2, 8, 1.6)	3210	3194	3113	3165	-2.536	-0.908
(50, 2, 8, 2.2)	16838	16687	16587	15931	-0.599	-4.530
(50, 2, 8, 3.0)	8691	8590	8319	6977	-3.155	-18.778
(50, 3, 8, 1.0)	4288	4232	4118	4087	-2.694	-3.426
(50, 3, 8, 1.2)	8807	8673	8162	8307	-5.892	-4.220
(50, 3, 8, 1.6)	15815	15656	15248	15049	-2.606	-3.877
(50, 3, 8, 2.2)	12073	11883	11579	11792	-2.558	-0.766
(50, 3, 8, 3.0)	9376	9170	8234	7894	-10.207	-13.915
(50, 4, 8, 1.0)	16082	15783	15114	14887	-4.239	-5.677
(50, 4, 8, 1.2)	17284	16899	16597	15783	-1.787	-6.604
(50, 4, 8, 1.6)	18312	17935	17352	16781	-3.251	-6.434
(50, 4, 8, 2.2)	8771	8518	8184	8032	-3.921	-5.706
(50, 4, 8, 3.0)	16232	15628	14112	12850	-9.701	-17.776
(50, 5, 8, 1.0)	33103	32018	26323	30137	-17.787	-5.875
(50, 5, 8, 1.2)	7855	7605	7382	7104	-2.932	-6.588
(50, 5, 8, 1.6)	14099	13697	12414	12503	-9.367	-8.717
(50, 5, 8, 2.2)	9657	9399	4892	8762	-47.952	-6.777
(50, 5, 8, 3.0)	19878	19090	8196	15759	-57.067	-17.449

Table 7 Experimental Results for TD(λ) with the SPL Bid Price Initialization

policy on a total of 24 out of 40 test instances with the average increase in expected revenue across these instances being 0.67%. The naive initialization, as reported in Table 10, performed slightly worse than the bid-price initialization with an average loss with respect to the SPL bid-price policy of 7.47% for the PB and 1.85% for the SPL approximation.

	Benchmarks		TD-ER		% Off π^{SPL}	
(τ, m, κ, α)	UB	π^{SPL}	PB	SPL	PB	SPL
(50, 2, 4, 1.0)	7251	7230	6999	7096	-3.195	-1.853
(50, 2, 4, 1.2)	5749	5698	5600	5641	-1.720	-1.000
(50, 2, 4, 1.6)	3638	3632	3492	3327	-3.855	-8.398
(50, 2, 4, 2.2)	6091	6014	4643	4698	-22.797	-21.882
(50, 2, 4, 3.0)	5159	5075	3269	1771	-35.586	-65.103
(50, 3, 4, 1.0)	6894	6768	6689	6426	-1.167	-5.053
(50, 3, 4, 1.2)	5175	5067	4930	3971	-2.704	-21.630
(50, 3, 4, 1.6)	3270	3208	2681	2018	-16.428	-37.095
(50, 3, 4, 2.2)	9484	9282	7771	5217	-16.279	-43.794
(50, 3, 4, 3.0)	2398	2324	1699	972	-26.893	-58.176
(50, 4, 4, 1.0)	9747	9455	9059	9003	-4.188	-4.781
(50, 4, 4, 1.2)	8884	8599	8364	6465	-2.733	-24.817
(50, 4, 4, 1.6)	5524	5383	3925	3052	-27.085	-43.303
(50, 4, 4, 2.2)	7520	7211	5335	3162	-26.016	-56.150
(50, 4, 4, 3.0)	4694	4507	2730	1580	-39.428	-64.943
(50, 5, 4, 1.0)	15790	15136	11873	14040	-21.558	-7.241
(50, 5, 4, 1.2)	9406	8975	8368	7134	-6.763	-20.513
(50, 5, 4, 1.6)	8626	8274	7756	4600	-6.261	-44.404
(50, 5, 4, 2.2)	9090	8681	5471	3789	-36.977	-56.353
(50, 5, 4, 3.0)	3605	3433	2757	1359	-19.691	-60.414
(50, 2, 8, 1.0)	5505	5475	5379	5097	-1.753	-6.904
(50, 2, 8, 1.2)	12224	12145	12068	12001	-0.634	-1.186
(50, 2, 8, 1.6)	3210	3194	3067	3057	-3.976	-4.289
(50, 2, 8, 2.2)	16838	16687	9168	3639	-45.059	-78.193
(50, 2, 8, 3.0)	8691	8590	3990	1310	-53.551	-84.750
(50, 3, 8, 1.0)	4288	4232	4178	3830	-1.276	-9.499
(50, 3, 8, 1.2)	8807	8673	8519	7395	-1.776	-14.735
(50, 3, 8, 1.6)	15815	15656	14999	5309	-4.196	-66.090
(50, 3, 8, 2.2)	12073	11883	10376	10358	-12.682	-12.833
(50, 3, 8, 3.0)	9376	9170	5953	1851	-35.082	-79.815
(50, 4, 8, 1.0)	16082	15783	15010	14166	-4.898	-10.245
(50, 4, 8, 1.2)	17284	16899	16497	10607	-2.379	-37.233
(50, 4, 8, 1.6)	18312	17935	16823	6271	-6.200	-65.035
(50, 4, 8, 2.2)	8771	8518	6106	2136	-28.317	-74.924
(50, 4, 8, 3.0)	16232	15628	7968	2875	-49.015	-81.604
(50, 5, 8, 1.0)	33103	32018	27404	28455	-14.411	-11.128
(50, 5, 8, 1.2)	7855	7605	7416	4989	-2.485	-34.398
(50, 5, 8, 1.6)	14099	13697	11912	5192	-13.032	-62.094
(50, 5, 8, 2.2)	9657	9399	6555	2383	-30.259	-74.646
(50, 5, 8, 3.0)	19878	19090	13936	3664	-26.998	-80.807

Table 8 Experimental Results for TD(λ) with Naive Initialization

The PB approximation did not yield any instances which outperformed the SPL bid-price policy, but the SPL approximation outperformed the SPL bid-price policy on 13 of the 40 test instances with an average gain over those instances of 0.43%.

Table 11 shows the median performance of the two TD-learning algorithms with two

	Benchmarks		TD-ER		% Off π^{SPL}	
(τ, m, κ, α)	UB	π^{SPL}	PB	SPL	PB	SPL
(50, 2, 4, 1.0)	7251	7230	7089	7108	-1.950	-1.687
(50, 2, 4, 1.2)	5749	5698	5639	5690	-1.035	-0.140
(50, 2, 4, 1.6)	3638	3632	3551	3605	-2.230	-0.743
(50, 2, 4, 2.2)	6091	6014	5984	6022	-0.499	0.133
(50, 2, 4, 3.0)	5159	5075	5092	5104	0.335	0.571
(50, 3, 4, 1.0)	6894	6768	6695	6770	-1.079	0.030
(50, 3, 4, 1.2)	5175	5067	4935	5077	-2.605	0.197
(50, 3, 4, 1.6)	3270	3208	3169	3224	-1.216	0.499
(50, 3, 4, 2.2)	9484	9282	9216	9310	-0.711	0.302
(50, 3, 4, 3.0)	2398	2324	2287	2342	-1.592	0.775
(50, 4, 4, 1.0)	9747	9455	9276	9521	-1.893	0.698
(50, 4, 4, 1.2)	8884	8599	8421	8470	-2.070	-1.500
(50, 4, 4, 1.6)	5524	5383	5166	5329	-4.031	-1.003
(50, 4, 4, 2.2)	7520	7211	7077	7269	-1.858	0.804
(50, 4, 4, 3.0)	4694	4507	4499	4522	-0.178	0.333
(50, 5, 4, 1.0)	15790	15136	14391	15199	-4.922	0.416
(50, 5, 4, 1.2)	9406	8975	7275	9108	-18.942	1.482
(50, 5, 4, 1.6)	8626	8274	7273	8243	-12.098	-0.375
(50, 5, 4, 2.2)	9090	8681	8540	8754	-1.624	0.841
(50, 5, 4, 3.0)	3605	3433	3125	3453	-8.972	0.583
(50, 2, 8, 1.0)	5505	5475	5463	5451	-0.219	-0.438
(50, 2, 8, 1.2)	12224	12145	12098	12102	-0.387	-0.354
(50, 2, 8, 1.6)	3210	3194	3160	3181	-1.064	-0.407
(50, 2, 8, 2.2)	16838	16687	16641	16645	-0.276	-0.252
(50, 2, 8, 3.0)	8691	8590	8475	7437	-1.339	-13.423
(50, 3, 8, 1.0)	4288	4232	4161	4222	-1.678	-0.236
(50, 3, 8, 1.2)	8807	8673	8559	8730	-1.314	0.657
(50, 3, 8, 1.6)	15815	15656	15627	15630	-0.185	-0.166
(50, 3, 8, 2.2)	12073	11883	11858	11879	-0.210	-0.034
(50, 3, 8, 3.0)	9376	9170	8899	9133	-2.955	-0.403
(50, 4, 8, 1.0)	16082	15783	15527	15691	-1.622	-0.583
(50, 4, 8, 1.2)	17284	16899	16841	17062	-0.343	0.965
(50, 4, 8, 1.6)	18312	17935	17929	18075	-0.033	0.781
(50, 4, 8, 2.2)	8771	8518	8510	8542	-0.094	0.282
(50, 4, 8, 3.0)	16232	15628	15212	15709	-2.662	0.518
(50, 5, 8, 1.0)	33103	32018	31609	32444	-1.277	1.331
(50, 5, 8, 1.2)	7855	7605	7472	7707	-1.749	1.341
(50, 5, 8, 1.6)	14099	13697	13362	13842	-2.446	1.059
(50, 5, 8, 2.2)	9657	9399	9204	9471	-2.075	0.766
(50, 5, 8, 3.0)	19878	19090	17946	19241	-5.993	0.791

Table 9 Experimental Results for TD-ρ with the SPL Bid Price Initialization

initial policies and two approximation architecture. It is clear that *TD-ρ* with the SPL bid-price policy and the SPL approximation has the best performance and is able to out-performan the SPL bid-price policy for most of the 40 instances.

	Benchmarks		TD-ER		% Off π^{SPL}	
(τ, m, κ, α)	UB	π^{SPL}	PB	SPL	PB	SPL
(50, 2, 4, 1.0)	7251	7230	7075	7084	-2.144	-2.019
(50, 2, 4, 1.2)	5749	5698	5668	5690	-0.527	-0.140
(50, 2, 4, 1.6)	3638	3632	3044	3332	-16.189	-8.260
(50, 2, 4, 2.2)	6091	6014	5978	6011	-0.599	-0.050
(50, 2, 4, 3.0)	5159	5075	3105	5061	-38.818	-0.276
(50, 3, 4, 1.0)	6894	6768	6633	6760	-1.995	-0.118
(50, 3, 4, 1.2)	5175	5067	4950	5069	-2.309	0.039
(50, 3, 4, 1.6)	3270	3208	2781	2513	-13.310	-21.665
(50, 3, 4, 2.2)	9484	9282	7381	9212	-20.480	-0.754
(50, 3, 4, 3.0)	2398	2324	1708	2332	-26.506	0.344
(50, 4, 4, 1.0)	9747	9455	9262	9494	-2.041	0.412
(50, 4, 4, 1.2)	8884	8599	8400	8585	-2.314	-0.163
(50, 4, 4, 1.6)	5524	5383	4742	4531	-11.908	-15.828
(50, 4, 4, 2.2)	7520	7211	6251	7173	-13.313	-0.527
(50, 4, 4, 3.0)	4694	4507	3434	4521	-23.807	0.311
(50, 5, 4, 1.0)	15790	15136	14257	15100	-5.814	-0.238
(50, 5, 4, 1.2)	9406	8975	7872	8935	-12.287	-0.446
(50, 5, 4, 1.6)	8626	8274	7291	8254	-11.883	-0.242
(50, 5, 4, 2.2)	9090	8681	8482	8727	-2.291	0.530
(50, 5, 4, 3.0)	3605	3433	2805	3447	-18.293	0.408
(50, 2, 8, 1.0)	5505	5475	5463	5370	-0.219	-1.918
(50, 2, 8, 1.2)	12224	12145	12092	12068	-0.436	-0.634
(50, 2, 8, 1.6)	3210	3194	3166	3184	-0.877	-0.313
(50, 2, 8, 2.2)	16838	16687	16632	16525	-0.330	-0.971
(50, 2, 8, 3.0)	8691	8590	8517	7208	-0.850	-16.088
(50, 3, 8, 1.0)	4288	4232	4136	4225	-2.268	-0.165
(50, 3, 8, 1.2)	8807	8673	8544	8657	-1.487	-0.184
(50, 3, 8, 1.6)	15815	15656	15280	15200	-2.402	-2.913
(50, 3, 8, 2.2)	12073	11883	9913	11752	-16.578	-1.102
(50, 3, 8, 3.0)	9376	9170	8794	9063	-4.100	-1.167
(50, 4, 8, 1.0)	16082	15783	15473	15560	-1.964	-1.413
(50, 4, 8, 1.2)	17284	16899	16837	16958	-0.367	0.349
(50, 4, 8, 1.6)	18312	17935	17739	17920	-1.093	-0.084
(50, 4, 8, 2.2)	8771	8518	8334	8362	-2.157	-1.831
(50, 4, 8, 3.0)	16232	15628	15176	15704	-2.891	0.486
(50, 5, 8, 1.0)	33103	32018	31011	32348	-3.152	1.031
(50, 5, 8, 1.2)	7855	7605	7303	7645	-3.968	0.526
(50, 5, 8, 1.6)	14099	13697	12988	13701	-5.184	0.029
(50, 5, 8, 2.2)	9657	9399	9221	9430	-1.890	0.330
(50, 5, 8, 3.0)	19878	19090	14801	19244	-22.473	0.807

Table 10 Experimental Results for TD-ρ with Naive Initialization

Approx.	Initial	TD-ρ to π^{SPL}	TD(λ) to π^{SPL}
PB	SPL Bid Price	-1.61	-4.44
SPL	SPL Bid Price	0.29	-5.88
PB	Naive	-2.36	-12.86
SPL	Naive	-0.21	-37.16

Table 11 Median Performance Gap of TD-ρ and $TD(\lambda)$ Algorithm

5. Conclusion

Reinforcement learning offers an alternative solution approach to sequential decision making problems that do not require the rigid assumptions and commercial solvers that are necessary for linear programming based approximate dynamic programming. For small problem instances, there is evidence that the TD learning method can improve over policies generated by ALP techniques which are demonstrably non-optimal in certain scenarios due to their inability to fully capture the impact of strong network effects which may exist in NRM problem instances. Our results show that while the traditional $TD(\lambda)$ algorithm may struggle to yield high performing policies on the hotel test instances in our study, the TD-ρ algorithm that leverages the novel *salience trace* shows the potential to improve over ALP-based policies when initialized with a strong bid-price policy. The TD-ρ algorithm was able to outperform the bid-price policy derived from the SPL approximation on 24 out of the 40 test instances considered, demonstrating the potential for the application of reinforcement learning techniques for policy improvement.

Furthermore, the flexibility of reinforcement methods allows for the consideration of a broader class of approximation architectures as well as applications in problems which may violate the critical assumptions of ALPs. With a recent emphasis in the literature on the use of machine learning techniques for demand forecasting (Lee et al. (2020), Wang and Duggasani (2020)) and choice modeling (Chen et al. (2021)), we believe there is an opportunity for further research into model-free solution methodologies which are agnostic to the manner in which signals about rewards, state transitions, and customer arrivals are generated.

References

Adelman, D. (2007). Dynamic bid-prices in revenue management. *Operations Research*, 55(4):647–661.

Asis, K. D., Chan, A., Pitis, S., Sutton, R. S., and Graves, D. (2019). Fixed-horizon temporal difference methods for stable reinforcement learning. *CoRR*, abs/1909.03906.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA.

Chen, S.-S., Choubey, B., and Singh, V. (2021). A neural network based price sensitive recommender model to predict customer choices based on price effect. *Journal of Retailing and Consumer Services*, 61:102573.

Dabney, W. and Barto, A. (2012). Adaptive step-size for online temporal difference learning.

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

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

Farias, V. F. and Van Roy, B. (2007). An approximate dynamic programming approach to network revenue mangement. Working paper, MIT Sloan School of Management.

Kober, J., Bagnell, J. A., and Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274.

Koch, S. (2017). Least squares approximate policy iteration for learning bid prices in choice-based revenue management. *Computers & Operations Research*, 77:240–253.

Kunnumkal, S. and Talluri, K. (2016). On a piecewise-linear approximation for network revenue management. *Mathematics of Operations Research*, 41(1):72–91.

Lee, M., Mu, X., and Zhang, Y. (2020). A machine learning approach to improving forecasting accuracy of hotel demand: A comparative analysis of neural networks and traditional models. *Issues in Information Systems*, 21(1):12–21.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier.

Ma, Y., Rusmevichientong, P., Sumida, M., and Topaloglu, H. (2020). An approximation algorithm for network revenue management under nonstationary arrivals. *Operations Research*, 68(3):834–855.

Mahmud, M., Kaiser, M. S., Hussain, A., and Vassanelli, S. (2018). Applications of deep learning and reinforcement learning to biological data. *IEEE transactions on neural networks and learning systems*, 29(6):2063–2079.

Mazyavkina, N., Sviridov, S., Ivanov, S., and Burnaev, E. (2021). Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, page 105400.

Meissner, J. and Strauss, A. K. (2012). Network revenue management with inventory-sensitive bid prices and customer choice. *European Journal of Operational Research*, 216(2):459–468.

O'Doherty, J. P., Dayan, P., Friston, K., Critchley, H., and Dolan, R. J. (2003). Temporal difference models and reward-related learning in the human brain. *Neuron*, 38(2):329–337.

Sutton, R. (1988a). Learning to predict by the methods of temporal differences. *Machine Learning*, (3.1):9–44. TD Learning.

Sutton, R. S. (1988b). Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press Cambridge, MA.

Talluri, K. and van Ryzin, G. J. (1998). An analysis of bid-price controls for network revenue management. *Management Science*, 44(11):1577–1593.

Talluri, K. and van Ryzin, G. J. (2004). Revenue management under a general discrete choice model of consumer behavior. *Management Science*, 50(1):15–33.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, 8:257–277.

Tesauro, G. et al. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68.

Tong, C. and Topaloglu, H. (2014). On the approximate linear programming approach for network revenue management problems. *INFORMS Journal on Computing*, 26(1):121–134.

Tsitsiklis, J. and Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690.

Van Seijen, H., Mahmood, A. R., Pilarski, P. M., Machado, M. C., and Sutton, R. S. (2016). True online temporal-difference learning. *The Journal of Machine Learning Research*, 17(1):5057–5096.

Vossen, T. and Zhang, D. (2015). Reductions of approximate linear programs for network revenue management. *Operations Research*, 63(6):1352–1371.

Wang, J. and Duggasani, A. (2020). Forecasting hotel reservations with long short-term memory-based recurrent neural networks. *International Journal of Data Science and Analytics*, 9.

Williamson, E. (1992). *Airline Network Seat Control*. PhD thesis, Massachusetts Institute of Technology.

Zhang, D., Samiedaluie, S., and Zhang, R. (2021a). Product-based approximate linear programs for network revenue management. Working paper, University of Colorado Boulder.

Zhang, R., Samiedaluie, S., and Zhang, D. (2021b). Product-based approximate linear programs for network revenue management. Working paper, Leeds of Business, University of Colorado Boulder.