Improving Network Durability using Approximate Dynamic Programming[*]

Erik Hammel[†] John E. Mitchell[‡] Thomas C. Sharkey[§] William A. Wallace [¶]

April 4, 2016

Abstract

Our goal is to optimize the efficacy of reinforcing an existing flow network to prevent unmet demand from imminent disruptions. Estimates for the probabilities of failures for edges in the network are refined as the disaster draws nearer, and we are asked to find edges which will best provide durability to the network post-event. The problem is formulated as an approximate dynamic program (ADP). We derive several innovative adaptations from reinforcement learning concepts. We compare the performance of the policy resulting from the ADP against traditional two-stage stochastic programs with recourse utilizing a sample average approximation model. We provide empirical evidence which corroborates with basic theorems of convergence for more simplistic forms of the reinforcement learning process. The material presented here is developed in the context of preparing urban infrastructures against damages caused by disasters, however it is applicable to any flow network.

Keywords: disaster mitigation, infrastructure restoration, approximate dynamic programming

[*]The material in this paper is based upon work supported by the U.S. Department of Homeland Security under Grant Award Number, 2008-ST-061-ND0001-07. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Department of Homeland Security. The work of Thomas C. Sharkey was supported by the U.S. National Science Foundation under CMMI-1254258.

[†]RankMiner Inc., St. Petersburg, FL 33701.

[‡]Department of Mathematical Sciences, Rensselaer Polytechnic Institute, 110 8th St, Troy, NY 12180. mitchj@rpi.edu http://www.rpi.edu/~mitchj

[§]Department of Industrial and Systems Engineering, Rensselaer Polytechnic Institute, 110 8th St, Troy, NY 12180. sharkt@rpi.edu

[¶]Department of Industrial and Systems Engineering, Rensselaer Polytechnic Institute, 110 8th St, Troy, NY 12180. wallaw@rpi.edu

1 Introduction

Disasters, whether natural, biological, accidental, or terrorist, can cause major damage and cost great sums of money to restore an area to standard working conditions. Restoring service infrastructures is especially costly, and finding better, more economical ways of restoring these systems is of great interest to many. Likewise, finding the best ways to prevent such damage altogether has received much attention. A normally functioning system describes flow of goods and/or services from those entities which provide them to those that require them, connected through a network of intermediaries. During a disaster, these entities or connections will be subject to disruptions which prevent the system from functioning normally; that is, the flow of goods or services from supplier to consumer is halted or diminished. An example of where this might be employed, and an application to which this work is applied, is restoring various (notably power) infrastructures from disruptions caused by hurricanes [20].

Previous investigations into repairing known, post-event damage as quickly as possible [6, 8, 9, 18, 29, 30, 38] motivate work that might minimize unmet demand by preventing it altogether, and in this work we focus on mitigation. One hope is to avoid unnecessary costs incurred due to unmet demand by reinforcing and installing edges into the pre-event network. Now consider the case when we know a disaster is imminent, but there is enough time to weigh our options. We likely have more resources at hand at the early stages, and certainly more time to be able to make several reinforcements if we need to. However, installation costs may not be negligible, and we wish to avoid installing an edge that later on turns out to be a worthless investment due to changes in the nature of the disaster. For example, hurricane forecasts give a general direction of the path of the hurricane, but are limited in their accuracy by a cone of uncertainty. The hurricane has a not insignificant chance to move anywhere within this cone as it travels along the forecasted path, and it may either increase or decrease in intensity. In these cases of high variability in predictions, it behooves the user to wait for further updates to the forecast rather than install edges right away. The question we then seek to answer is: which edges can we install that will once again allow the supplier to provide its service to those who need it, and when do we install them?

At any given time, the network can be described by its set of edges and their capacities, including any previously installed edges. We refer to this set of information together with the current forecast of the event as the state of the network, and all possible configurations of the network are contained within the state space. For any given state, the action space is the set of edges we might install. Once an action is performed at a state, the state changes. Our goal in this paper is to generate a sequence of installation decisions to minimize unmet demand induced by changing outage forecasts. The purpose of constructing this model is to describe the system dynamics so that we can estimate losses and project state transitions for use in a process that learns to make better installation decisions.

Network analysis focused on addressing disruptions and uncertainties is a broad topic that has been approached in many different ways. The most basic treatment involves disaster modeling from logistical or socioeconomic matters [7], but can also be approached from a purely mathematical standpoint, analysis which has existed for quite a while [47], but which has become increasingly sophisticated over time [22, 23]. Perhaps the most popular trait addressed is network performance in the presence of uncertain supply and demand [1, 11, 14, 16, 34, 35, 40, 46, 48].

Another less popular system dynamic that has been addressed, and the area which we investigate, is actual damage to the network. However, whereas we look at edge capacity failures, it is node failures that have received more attention [3, 39]

The popular method for optimizing under uncertainty is the large field known as Stochastic Programming (SP) [19, 37]. Stochastic programming does have some disadvantages, however. First, the optimal solution depends on a probability distribution of outages which may not be known exactly. Second, the sample set for this distribution can be intractably large or infinite, making calculating the expectation impossible. These drawbacks can be addressed by taking sample outcomes, which inherently introduces error into the solution. The first stage objective is estimated using a weighted average over a number of samples, known as the *Sample Average Approximation* (SAA) method.

Our model will have two levels of uncertainty. We have a forecast timeline of outage probabilities, with this forecast evolving. Thus, the probability that a particular infrastructure component is damaged depends on the outage probabilities, and the outage probabilities are themselves evolving in a stochastic manner. The outage probabilities can be regarded as a representation of the forecast of the characteristics of the impending disaster; sample scenarios of the surviving system will be the original network without the amalgam of edges randomly selected to have failed, each according to their outage probability. Du and Peeta [12] also look at planning for disasters when the outage probabilities are evolving. In order to account for the evolving outage probabilities and the dynamic nature of our installation decisions, we will model our problem in an approximate dynamic programming framework.

1.1 Approximate Dynamic Programming

Reinforcement Learning (RL) [4, 41, 42] is a general term used to describe one particular mode of machine learning: to improve an artificial agent's choices in some context so as to minimize an expected loss. A key aspect of RL is that we are making decisions in an environment without perfect information. The agent is able to describe some aspects of the current state of its environment, but there are other aspects hidden to it. When it comes time for it to make a decision, it must weigh what it has learned from previous state transitions and the associated loss of the transition's action with the newly encountered state and its current valuation of loss for actions taken at that state. Once the agent has made its decision, it is only capable of evaluating whether it was a good choice or a bad one.

Our model will have multiple stages, so the solution approach we consider is approximate dynamic programming, or ADP [31], the method of choice for implementing a solution to RL problems in the operations research community. The linkages between stochastic programming, approximate dynamic programming, and reinforcement learning are explored in [32]. Combinations of installable edges increase exponentially with the number of edges available. The use of ADP is an attempt to overcome the three curses of dimensionality: evaluation of state space, action space, and uncertainty.

The value of being in a state is approximated using a dynamically updated value function approximation (VFA). One way to address the curses of dimensionality is to parametrize the VFA to be some linear combination of *basis functions* whose coordinates are the so-called VFA coefficients. A basis function is simply some quantity calculable from numeric data acquired from the state. The general approximation is

$$\tilde{V}(S) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(S), \tag{1}$$

where ϕ are the set of basis functions, θ are the set of coefficients to be determined, and $\mathcal{F}$ is a set of *features*. This VFA form effectively condenses down the need for storing every state's value individually and replaces them with storing only a unique combination of coefficients to describe all states. This formulation of the VFA is known as a *linear-in-the-parameters* model. This terminology does not preclude the basis functions from being nonlinear; in fact, the basis functions for our model are nonlinear.

Given a sampled scenario, an iteration of the basic ADP method occurs in three steps. First, the best policy is found for the current state by taking the action that will minimize long-term cost. Long-term cost is defined as the VFA evaluated for the expected state by taking a particular allowable action plus any cost incurred as a direct result of taking that action. Second, the next state is determined by taking the chosen action and applying any exogenous factors. Once this state is determined, its VFA evaluation is used in the third step to refine the VFA by modifying the coefficients of the basis functions to better capture the performance of the policy. These three parts of an iteration are known as the control step, transfer function evaluation, and the prediction step, respectively. Specifics of each step are presented in Section 2 and later.

The process of evaluating and improving policy on states in an arbitrary (yet complete) fashion is called *asynchronous policy iteration*. This method can also be shown to converge to an optimal value:

Theorem 1 (Theorem 4 in [4]). Asynchronous policy iteration convergence: *Consider the algorithm that starts with an arbitrary initial value function V^0 and at the n^{th} iteration, chooses a state s_n, replaces the s_nth component of the current value function with $\mathcal{T}(V^n(s_n))$, and leaves all other components of V^n unchanged; that is,*

$$V^{n+1}(s) = \begin{cases} \mathcal{T}\left(V^n(s)\right), & s = s_n \\ V^n(s), & \textit{otherwise} \end{cases}$$

with

$$\mathcal{T}(V) = \min_a \left[\sum_{s'} \mathbb{P}^a_{ss'} \mathbb{R}^a_{ss'} + \sum_{s'} \mathbb{P}^a_{ss'} V \right], \tag{2}$$

with a state transition probability

$$\mathbb{P}^a_{ss'} = Pr\left[S_{t+1} = s' \,|\, S_t = s,\, a_t = a\right] \tag{3}$$

and a corresponding one-step expected return

$$\mathbb{R}^a_{ss'} = \mathbb{E}\left[l_{t+1} \,|\, S_t = s,\, a_t = a,\, S_{t+1} = s'\right] \tag{4}$$

where l_{t+1} denotes the loss at time $t+1$. Assume that all states are chosen for iteration infinitely often. Then

$$\lim_{n\to\infty} V^n = V^*,$$

where V^* is the optimum satisfying Bellman's optimality equation.

Topaloglu and Powell [44] have worked on an ADP algorithm in a time-expanded multicommodity flow network. Like us, they employ a linear form (as well as a piecewise linear form) for the VFA, however other facets of their work differ. The simplest difference is that they consider the network under demand uncertainty. It is mainly this research and the research presented in [5, 10, 13, 15, 21, 27, 28, 33, 34, 36, 43] along with the seminal works [2, 4, 31, 37, 42] that provide the tools and motivation for our work.

1.2 Present material and contributions

The problem to which we ascribe all of the above methods is motivated by disaster mitigation.

The predictive model is time-expanded with a "first-tier" or scenario level stochastic process representing the forecast timeline of outage probabilities (not the sampled hypothetical outages), which is a constrained Dirichlet process. Thus there are two layers of uncertainty present in the model: uncertainty of what the outage forecast will look like, and uncertainty for which sample outcomes will be under consideration for that time period's forecast. We will refer to the latter as the "second-tier" or outcome level outages, and this is a Bernoulli process whose random variables are the sampled outcomes of the Dirichlet process. In the context of planning for an impending hurricane, the predicted path and associated probabilities of damage are updated as the event approaches and they constitute the Dirichlet process, while the actual outages are drawn from the Bernoulli process. Our state space includes both the sampled deterministic forecast data and the set of hypothetically damaged edges for a given time step along with any installed edges from previous time periods. State transitions are performed by adding edges one chose to install in the current time step, taking new samples from the first-tier random process to obtain forecast data, then sampling from second-tier random variables constructed with this data.

The model starts with the basic linear-in-the-parameters VFA described earlier. One of the innovative aspects of our work is that these basis functions use only the forecast data, and thus can be considered to map from the first-tier Dirichlet process. Having the basis functions depend only on the Dirichlet process opens up the potential for using basis functions that use a variety of statistical calculations. We introduce a basis function based on a shortest path heuristic for each installable edge. The shortest path heuristic provides the probability that at least one edge will fail in the path most likely to remain intact. This basis function is later refined using a multiplicative factor based on the amount of flow that can pass through this shortest path relative to the bandwidth of the network, and we conduct a study on the efficacy of various values of this factor.

The bulk of the contribution lies in the field of ADP. There has been little study with flow network resiliency using ADP methods or any other predictive methods, and this work expands upon it. Because of our two-tiered model, all state transitions can occur at the scenario level rather than the outcome level. This means the basis functions more accurately represent the fact that unique combinations of VFA coefficients correspond to a whole family of states. Further, we introduce the idea of using varying and unknown probability distributions over time, where previous work has only employed static, standard distributions [44]. ADP models have also only concerned themselves with unknown demand and/or diminished node storage capacity in facility location problems, whereas our model changes the very nature of the network by adding and removing connections, which in turn changes the size of the state space. We incorporate sample averages in the prediction step in possibly a novel manner.

The remainder of the document is laid out as follows: Section 2 describes the ADP algorithm, including the definition of the basis functions and the control step. The prediction step is the subject of Section 3. The ADP model is compared with two two-stage stochastic programming approximations to our multi-stage model, described in Section 4. Initial computational results appear in Section 5. A refinement to the basis functions is the subject of Section 6 and conclusions and suggested avenues of future study are contained in Section 7.

A word on notation: we use undecorated variables to represent the target or "true" depictions of quantities, while using hats for random quantities and tildes for estimated quantities or calculated constants.

2 The ADP algorithm

2.1 ADP formulation from Bellman's equation

Our model has two levels of complexity: sampling a forecast scenario from a Monte Carlo (MC) simulation and performing an ADP algorithm by sampling outcomes from that forecast. The first-tier random process is denoted $\hat{\Psi}$, with deterministic forecast data ψ sampled from this process. Thus, ψ_t is the MC simulated forecast, a **deterministic** realization, at time t from the random variable $\hat{\Psi}_t$. In this way, $\hat{\Psi}$ maps "forecasts" to probabilities of occurrence. The forecasts evolve over time, so Ψ_t depends on the previous realization ψ_{t-1}. This realization ψ_t consists of the probabilities that each original edge will be damaged. The second tier random variables are drawn from the Bernoulli random variables specified by ψ_t and they comprise a vector u_t of edge capacities, each of which is equal to its original capacity or 0.

The state S_t at time t consists of the deterministic forecast data ψ_{t-1} sampled at time $t-1$, the network's edge capacities as modified by ψ_{t-1}, and a list of the installation decisions performed at prior times a_{t-1}. The set of all states at time t is denoted $\mathcal{S}_t$. By taking a set $\Omega_t \subseteq \Psi_t(\psi_{t-1})$ of samples, we can construct a sample average approximation that we can use to obtain a form of Bellman's equation given by

$$\tilde{V}_t(S_t) = \min_{a_t} C(S_t, a_t) + \sum_{\psi_t \in \Omega_t \subseteq \Psi_t(\psi_{t-1})} \mathbb{P}_{\Omega_t}[\psi_t]\, \tilde{V}_{t+1}(S_{t+1}(a_t, \psi_t)) \quad \forall S_t \in \mathcal{S}_t, \tag{5}$$

where $C(S_t, a_t)$ is the one-step cost. The set of feasible states $\mathcal{S}_{t+1}$ available at time $t+1$ depends on the actions a_t we take at time t and on the forecast scenario ψ_t. We use the tilde notation on V to emphasize that since we are only looking at a subset of samples from the distribution, we are now only approximating the actual value of being in a state. It is still a valid estimation because if

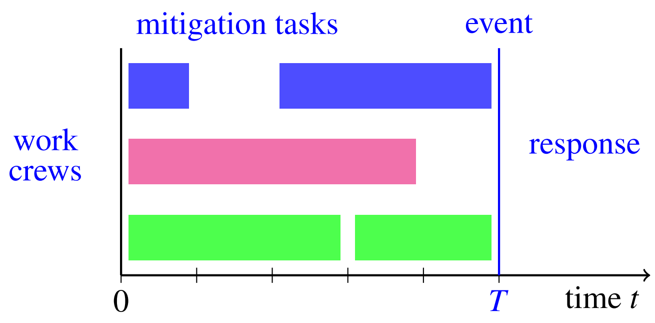


Figure 1: Work crews are assigned to mitigation tasks before the event at time T.

we sample many times, the sample average approaches the actual expectation over $\hat{\Psi}_t$ by the Law of Large Numbers. Currently, although we do not take more than one sample per iteration, we perform this procedure over many iterations, achieving the same effect.

We are looking to perform installation decisions prior to an event at time T. These installation decisions may require the allocation of work crews, as illustrated in Figure 1. Installation decisions are made at times $t = 0, \ldots, T-1$ based on the forecast scenario ψ_{t-1} available at that time. It may be optimal to employ a wait-and-see strategy at certain times, as illustrated in the figure. It is possible that an installed edge is parallel to an existing edge, which would correspond to increasing the capacity of the existing edge. In our computational testing, we assume that an installed edge will survive the event, but the model could be extended to allow failure of an installed edge with some probability. For the computational results in this paper, we assume that installation of an edge takes one time step.

2.2 Basis functions and the control step

The basis function value assigned to an installable edge is a function of the current sample ψ_t of the scenario level stochastic process, but does not depend on a particular realization of the outcome level outages. It is the probability that at least one edge will fail in the path most likely to remain intact, with the path constructed from the original edges $\mathcal{A}$. For a given installable edge i, let $F(i) \subseteq \mathcal{A}$ be the set of edges in this path. Thus the basis function ϕ_i is the probability of failure of at least one edge in the path:

$$\phi_i(\psi_t) := \pi(F(i)) = 1 - \prod_{kl \in F(i)} (1 - \psi_{tkl}) = 1 - \prod_{kl \in \mathcal{A}} (1 - \psi_{tkl})^{x_{kl}} \tag{6}$$

where the binary variable x_{kl} indicates whether edge $(k, l) \in F(i)$ and ψ_{tkl} is the forecast outage datum for edge (k, l) in the original network. The set $F(i)$ can be determined by solving a shortest

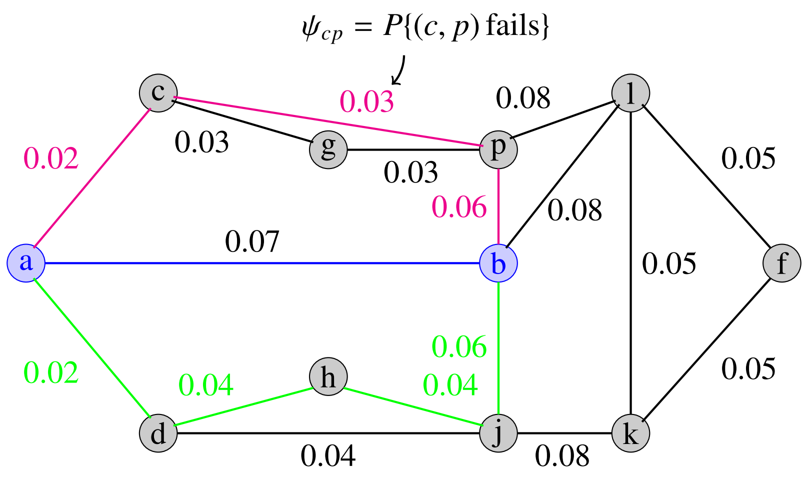


Figure 2: Two paths between nodes a **and** b**. P(path** $adhjb$ **fails)** ≈ 0.151**, P(path** $acpb$ **fails)** ≈ 0.106

path linear program:

$$F(i) := \arg\min_{kl \in \mathcal{A}} \quad -\sum_{kl \in \mathcal{A}} \ln(1 - \psi_{tkl})\, x_{kl} \tag{7a}$$

$$\text{s.t.} \quad \sum_{l} (x_{kl} - x_{lk}) = \begin{cases} 1, & \text{if } k = \sigma \\ -1, & \text{if } k = \tau \\ 0, & \text{otherwise} \end{cases}, \tag{7b}$$

$$x \geq 0. \tag{7c}$$

The value function approximation at time $t+1$ consists of a simple linear dependence on the actions at time t. We let $\mathcal{Z}$ denote the set of installation decisions at time t. The basis functions do not depend on the capacities of the edges, so we will describe a refinement to them in Section 6. An illustration of path length calculation using logs can be found in Figures 2 and 3.

Now that we have closed and explicit forms for all entities in the objective, we are ready to write the full control step. The set of origin nodes is denoted $\mathcal{O}$ with the supply available denoted b, the set of demand nodes is denoted $\mathcal{D}$ with demand denoted d, and the set of transshipment nodes is denoted $\mathcal{T}$. The capacities of the original edges $\mathcal{A}$ are denoted u and those of the installable edges $\mathcal{A}'$ by u', with flows on these two classes of edges denoted by x and y respectively. Installation decisions are captured using the binary variables ζ. We assume both demand and supply nodes are outgoing-terminal and incoming-terminal, respectively, and that all measurable quantities (flows, capacities, etc.) shall be non-negative. In this discussion, we allow for reinforcement of existing edges and we have an installation budget of Λ at each time t, and there are no installable nodes.

The one step cost $C(S_t, a_t)$ is approximated using a sample average approximation of the unmet demand z when no additional edges are installed over K outage outcomes, with different scenarios indicated through the use of the subscript k. The VFA coefficients θ are adjusted over time, and we

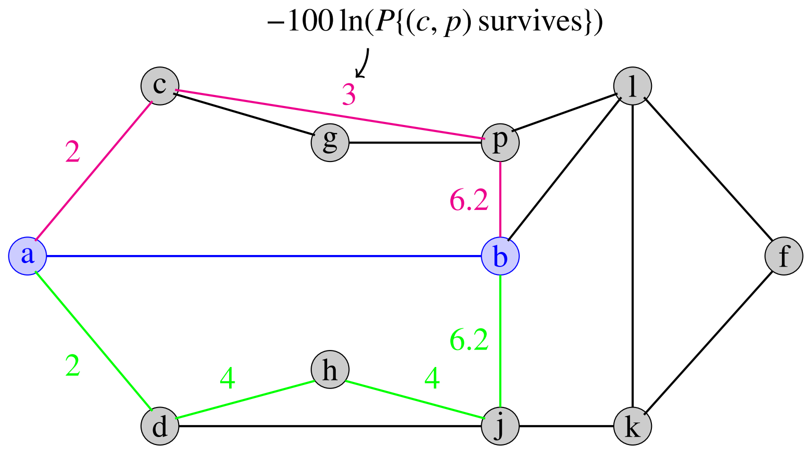


Figure 3: Path length calculation using logs of probabilities. Path *adhjb* length 16.2, path *acpb* length 11.2

represent the iterations of this process using the superscript n. We interpret the terms in the VFA to represent the reduction in expected unmet demand that is attributable to the installation of edges, with a base cost that represents the expected cost if no installations are performed. Thus, we split the control step into two subproblems, resulting in a disjoint control step:

$$\tilde{\zeta}_t^n := \arg\min_{\zeta_t \in \mathbb{B}^{|\mathcal{A}'|}} \quad -\sum_{q=1}^{|\mathcal{Z}|} \theta_{qt}^n \phi_q\left(\psi_t^n\right) \zeta_{qt} \tag{8a}$$

$$\text{s.t.} \quad \zeta_t \geq \tilde{\zeta}_{t-1}^n, \tag{8b}$$

$$\sum_{i'j' \in \mathcal{A}'} \left(\zeta_{ti'j'} - \tilde{\zeta}_{t-1,i'j'}^n\right) \leq \Lambda \tag{8c}$$

$$v_{Ct}^n := \min_{x_t, y_t, z_t} \quad \frac{1}{K} \sum_{k=1}^{K} \sum_{j=1}^{|\mathcal{D}|} z_{tkj} \tag{8d}$$

$$\text{s.t.} \quad x_{tk} \leq u_{tk}^n \qquad \forall k = 1, \dots, K \tag{8e}$$

$$y_t \leq u'^{\mathrm{T}} \tilde{\zeta}_{t-1}^n \qquad \forall k = 1, \dots, K \tag{8f}$$

$$\sum_{\substack{ij \in \mathcal{A} \\ ij' \in \mathcal{A}'}} \left(x_{tkij} + y_{tij'}\right) \leq b_i \qquad \forall i \in \mathcal{O},\ k = 1, \dots, K \tag{8g}$$

$$\sum_{\substack{ij \in \mathcal{A} \\ i'j \in \mathcal{A}'}} \left(x_{tkij} + y_{ti'j}\right) + z_{tkj} = d_j \qquad \forall j \in \mathcal{D},\ k = 1, \dots, K \tag{8h}$$

$$\sum_{\substack{il, lj \in \mathcal{A} \\ i'l, lj' \in \mathcal{A}'}} \left(x_{tkil} + y_{ti'l} - x_{tklj} - y_{tlj'}\right) = 0 \qquad \forall l \in \mathcal{T},\ k = 1, \dots, K \tag{8i}$$

$$x,\ y,\ z \geq 0 \tag{8j}$$

The base cost v^n_{Ct} for the current time step is calculated by solving a linear program (LP) when no additional edges are installed in (8d)–(8j), and a discount (the objective value for the solution $\tilde{\zeta}^n_t$ in (8a)) is determined by edge selection. We shall call the sum of the two objectives (8a) and (8d) the *discounted cost* of unmet demand for installing edges dictated by the current policy. The VFA coefficients are restricted to be nonnegative, since the installation of any edge can only reduce the expected unmet demand; nonnegativity is maintained in the prediction step, using the methods outlined in Section 3.2.

2.3 The overall algorithm

At each iteration n, the algorithm takes a control step, performs a policy evaluation, and takes a prediction step to update the values of θ. We initialize the VFA by assigning values to the coefficients θ^0_{it} $\forall t = 0, \dots, T$, given by

$$\theta^0_{it} = \begin{cases} \frac{\sum_{j=1}^{|\mathcal{D}|} d_j}{2|\mathcal{Z}|}, & t = 0, \dots, T-1 \\ 0, & t = T, \end{cases} \tag{9}$$

the rationale here being that we want coefficients to take on values commensurate with potential cost values. The coefficients are zero at time T because there is no future installations to process.

At each iteration n, we sample a forecast scenario ψ^n_t from $\hat{\Psi}$ that will dictate our stochastically determined outage outcomes. Once we have this, we can evaluate our basis functions (6). At time 0, there are no installed edges, so we are ready to proceed.

We now run the control step as described in Section 2.2, proceeding forward through time T. This system is linear-in-the-parameters, linear-in-the-resource, and “separable-in-the-parameters”. The overall algorithm is given by Algorithm 1.

The transfer function δ in Step 1.4.3 indicates that we may employ an off-policy heuristic. We used a soft on-policy ϵ-greedy heuristic, which took the form of installing each chosen edge with probability $1 - \epsilon$ or instead installing an entirely random edge or no edge with probability ϵ. This is to ensure diversification in the exploration of the state space, so all states are visited with probability one if the process is continued for long enough.

The target value and the prediction step are discussed in the next section.

3 The prediction step

We identify six distinct components involved in determining θ^{n+1}_t: the previous VFA coefficient values, a target value function, a direction, a step size parameter, the VFA for the current control step evaluation, and the vector of basis functions of future states as functions of the actions made in the current control step. The target value function is discussed in Section 3.1, possible directions in Sections 3.2 and 3.3, and the step size parameter in Section 3.4.

```
0. Initialize VFA coefficients $\theta_{ft}\ \forall f \in \mathcal{F},\ t = 0, \dots, T$
1. For $n$ in $1, \dots, N$:
    1.1. Sample $\psi_t$ from $\hat{\Psi}_t\ \forall t = -1, \dots, T-1$.
    1.2. Set $S_0$ having taken no actions.
    1.3. Calculate $\phi_{ft}\ \forall f \in \mathcal{F},\ t = 0, \dots, T-1$.
    1.4. For $t$ in $0, \dots, T$:
        1.4.1. Control step: Calculate actions $\tilde{\zeta}_t$ by solving the disjoint control step (8).
        1.4.2. Set $\tilde{V}_t = \sum_{f \in \mathcal{F}} \theta_{ft}\phi_{ft}\tilde{\zeta}_{ft}$
        1.4.3. Advance to $S_{t+1} = \delta\left(S_t, \tilde{\zeta}_t, u_{t+1}\right)$
        1.4.4. $t = t + 1$; If $t \leq T$, go to 1.4.1.
    1.5. For $t$ in $0, \dots, T$:
        1.5.1. Calculate target value $V^{\pi}(S_t^n)$ and corresponding target actions $\overline{\zeta}_t$.
        1.5.2. Prediction step $\theta_{ft} = U\left(\theta_t, \phi_t, \tilde{\zeta}_t, \overline{\zeta}_t, C\left(S_t, \tilde{\zeta}_t\right), \tilde{V}_t, V^{\pi}(S_t^n)\right)\ \forall f \in \mathcal{F}$
        1.5.3. $t = t + 1$; if $t \leq T$, go to 1.5.1.
    1.6. $n = n + 1$; If $n \leq N$, go to 1.1
```

Algorithm 1: An ADP algorithm for determining installation decisions

3.1 Target value considerations

The target value is a desirable quantity that accurately reflects the value for being in a particular state. In traditional RL applications, this value is directly observable by the system after the event is resolved. In our case, this is not possible because we do not know what the actual outages will be as our problem lies entirely pre-event. We therefore must consider a way to gauge an installation's value as accurately as we can. We use an SAA method to get a more accurate value for the expected behavior. In this model, an optimal installation decision is calculated for a sample of scenarios at time T, given the installation decisions made up to time $t-1$. In order to create a scenario, we need to sample the forecast and then, from the forecast at time T, we sample the damage. The corresponding installation decisions are denoted $\bar{\zeta}$ in Step 1.5.1 of Algorithm 1.

$$
\begin{aligned}
V^{\pi}(S_t^n) := \min_{\zeta_t, x_t, y_t, z_t} \quad & \frac{1}{K} \sum_{k=1}^{K} \sum_{j=1}^{|\mathcal{D}|} z_{tkj} && && \text{(10a)}\\
\text{s.t.} \quad & x_{tk} \le u_{Tk}^n && \forall k = 1, \ldots, K, && \text{(10b)}\\
& y_t \le u'^{\mathrm{T}} \zeta_t, && && \text{(10c)}\\
& \zeta_t \ge \tilde{\zeta}_{t-1}^n, && && \text{(10d)}\\
& \sum_{\substack{ij \in \mathcal{A} \\ ij' \in \mathcal{A}'}} \left(x_{tkij} + y_{tij'} \right) \le b_i && \forall i \in \mathcal{O},\ \forall k = 1, \ldots, K, && \text{(10e)}\\
& \sum_{\substack{ij \in \mathcal{A} \\ i'j \in \mathcal{A}'}} \left(x_{tkij} + y_{ti'j} \right) + z_{tkj} = d_j && \forall j \in \mathcal{D},\ \forall k = 1, \ldots, K, && \text{(10f)}\\
& \sum_{\substack{il, lj \in \mathcal{A} \\ i'l, lj' \in \mathcal{A}'}} \left(x_{tkil} + y_{ti'l} - x_{tklj} - y_{tlj'} \right) = 0 && \forall l \in \mathcal{T},\ \forall k = 1, \ldots, K, && \text{(10g)}\\
& \sum_{i'j' \in \mathcal{A}'} \left(\zeta_{ti'j'} - \tilde{\zeta}_{t-1,i'j'}^n \right) \le \Lambda, && && \text{(10h)}\\
& x,\ y,\ z \ge 0,\ \zeta \in \mathbb{B}^{|\mathcal{A}'|}. && && \text{(10i)}
\end{aligned}
$$

3.2 Descent directions

We consider minimizing the mean-squared error (MSE)

$$
MSE(\theta_t) = \sum_{s \in S_t} \left(V^{\pi}(s) - \tilde{V}_t(s) \right)^2 \tag{11}
$$

to determine a descent direction via gradient descent, with the goal of converging to the target value. In the disjoint control step, the total discount and the discounted cost must be nonnegative since the alternatives are nonsensical in the physical world. This leads to the following modified prediction step equation:

$$
\theta_t^{n+1} = \theta_t^n - \alpha_n \left\{ V^{\pi}(S_t^n) - \max \left[v_{Ct}^n - \max \left(\sum_{i=1}^{|\mathcal{Z}|} \theta_{it}^n \phi_i \left(\psi_t^n \right) \tilde{\zeta}_{it}^n,\ 0 \right),\ 0 \right] \right\} \phi \left(\psi_t^n \right) \tilde{\zeta}_{it}^n, \tag{12}
$$

where α_n is a step size parameter and $V^\pi(S_t^n)$ is the target value function. Here θ and ϕ without the subscript f are respectively the vector of coefficients and basis functions of length $\mathcal{F}$.

For the disjoint control step, the recursive least-squares formula for updating θ is given by

$$\theta_t^{n+1} = \theta_t^n - H_n \phi_{S_t}^n \hat{\epsilon}_t, \tag{13}$$

where

$$H_n = \frac{B_{n-1}}{\gamma_n}, \tag{14a}$$

$$\gamma_n = \lambda + \left(\phi_{S_t}^n\right)^{\mathrm{T}} B_n \phi_{S_t}^n, \tag{14b}$$

$$B_n = \frac{1}{\lambda}\left(B_{n-1} - \frac{1}{\gamma_{n-1}} B_{n-1}\phi_{S_t}^n \left(\phi_{S_t}^n\right)^{\mathrm{T}} B_{n-1}\right), \tag{14c}$$

$$\hat{\epsilon}_t = \max\left[v_{Ct}^n - \max\left(\sum_{f\in\mathcal{F}} \theta_{ft}^n \phi_{f,S_{t+1}}(\tilde{\zeta}_t^n),\ 0\right),\ 0\right] - V^\pi(S_t^n). \tag{14d}$$

and λ is a discount factor to discount older observations (those already incorporated into the regression matrix B_n). The dependence on actions of the basis functions were suppressed above for the sake of clarity. It has been shown that it is effective to relate the discount factor to the step size for the gradient descent method, at least for the case of a constant basis function $\phi = 1$, so we use the following formula:

$$\lambda_n = \alpha_{n-1}\frac{1-\alpha_n}{\alpha_n}. \tag{15}$$

3.3 Incorporating target decision variables into prediction

As we have already established, selecting an edge for installation means that its corresponding VFA coefficient will change. Using the disjoint control step, higher coefficient values indicate a larger discount and thus a more desirable edge to install. Whether or not a coefficient's value goes up or down depends on the relative behavior of the target value function and the value function approximation, however.

Let us describe the desired behavior first and derive a new descent direction from it. Say an edge is chosen for installation by the VFA but it is not chosen in the target value determination. We should then consider decreasing the coefficient's value since more importance has been placed in the policy's installation decision than is physically appropriate. If the discounted cost is higher than the target value then this increased importance is erroneous and so the coefficient's value should definitely decrease. If however the discounted cost is lower than the target value then the increased importance is justified, realistically or not, and so the coefficient value should increase. After all, the eventual aim is to find the optimal policy, not just to converge to the target value. Similar motivation can be made in other situations.

Using the overline and tilde notation of Algorithm 1 for the target value quantities and the basis function notation and the disjoint calculations of Section 2.2, we can now algorithmically describe our modified prediction step:

$$\theta_{it}^{n+1} = \theta_{it}^{n} - \alpha_n \left\{ V^{\pi}(S_t^n) - \max\left[v_{Ct}^n - \max\left(\sum_{i=1}^{|\mathcal{Z}|} \theta_{it}^n \phi_i\left(\psi_t^n\right) \tilde{\zeta}_{it}^n,\ 0 \right),\ 0 \right] \right\} \phi\left(\psi_t^n\right) \left(2\overline{\zeta}_{it}^n - \tilde{\zeta}_{it}^n \right). \tag{16}$$

The descent direction is now dictated by both the target installation decisions $\overline{\zeta}_{tk}^n$ and the control step decisions $\tilde{\zeta}_{tk}^n$ rather than just the latter. The determination and calculation of the discount term remains the same. In other words, the control step uses the original basis function description $\phi_{k,q}\left(\psi_t^n\right) \zeta_{tk}^n$ in its formulation and the discount term in the prediction step is calculated only with its solution. This is a significant departure from the traditional RL procedure, as basis functions are open to interpretation by the designer, but the descent direction is directly derived from its structure. This method proposes to use a particular basis function structure but alter it to suit a different descent direction in the prediction step. As we will see in Section 5, the improvement in performance is notable. Developing the theory for its efficacy as well as empirically gauging the performance of a generalized descent direction alteration (such as experimenting with the numeric coefficients to $\overline{\zeta}_{tk}^n$ and $\tilde{\zeta}_{tk}^n$) could both potentially be deep and rewarding areas of further research.

3.4 Step size considerations

Step size is an issue of exploration versus exploitation versus convergence. Convergence to an optimal value of the standard RL model is guaranteed only for step sizes that satisfy the conditions specified in [42]; that is, the step size must satisfy

$$\sum_{n=1}^{\infty} \alpha_n = \infty, \quad \text{and} \quad \sum_{n=1}^{\infty} \alpha_n^2 < \infty. \tag{17}$$

These conditions may be too restrictive: if the step size diminishes too rapidly, the contributions of future iterations might induce insufficient exploration to find better states. Previous work has shown that theoretical convergence is not strictly required for the process to settle on a solution [45]. Based on extensive experimentation, we selected a McClain step size of the form

$$\alpha_n = \frac{\alpha_{n-1}}{1 + \alpha_{n-1} - \alpha}, \tag{18}$$

where $\alpha \in [0, 1)$ is some constant. For the beginning iterations, the McClain step size emulates a harmonic progression, and as $n \to \infty$, $\alpha_n \to \alpha$. Further, the closer α is to 0, the longer α_n takes to flatten out to the constant. This step size is also used when determining the discount factor λ in the recursive least-squares method.

4 Two stochastic programming models

Our model is a multi-stage stochastic programming problem. In order to evaluate the performance of our ADP model, we compare it to two two-stage stochastic programming approximations. In the first approximation, we assume we can delay installation decisions until the final set of outage probabilities are known, see Section 4.1. This is not practicable, since we would not have time to carry out the installation decisions, so the optimal value of this stochastic program would provide a lower bound on the expected unmet demand for the full stochastic program. The second approximation in Section 4.2 takes the opposite approach: we assume all installation decisions are based on the initial forecast of outage probabilities, so we do not use the additional information available as the event draws closer. Ideally, the ADP approach should result in lower expected unmet demand than that returned by this second stochastic programming approximation. The stochastic program models use a set of M timelines, with N outage scenarios drawn for each timeline. These timelines and scenarios are our "testing" timelines and scenarios, so they are not used in the training of the ADP approach.

4.1 Scaled-up optimality and run time performance testing

Given perfect information about a timeline, we can calculate optimal installation decisions for sampled outage outcomes. Thus, we assume perfect information about the Dirichlet process and then sample from the Bernoulli process to create an SAA. The mean of these M optima is then taken to be the ideal lower bound on cost for the purposes of this study. This lower bound is unrealizable because the installation decisions are optimized separately for each timeline, so they violate nonanticipativity.

$$\begin{aligned}
\tilde{V}^m_{IP@T} := \min_{x^m_T, y^m_T, z^m_T} \quad & \frac{1}{N}\sum_{n=1}^{N}\sum_{j=1}^{|\mathcal{D}|} z^m_{Tnj} && && (19a)\\
\text{s.t.} \quad & x^m_{Tn} \le u^m_{Tn} && \forall n = 1,\dots,N, && (19b)\\
& y^m_T \le u'^{\mathrm{T}} \zeta^m_T, && && (19c)\\
& \sum_{\substack{ij\in A\\ ij'\in A'}} \left(x^m_{Tnij} + y^m_{Tij'}\right) \le b_i && \forall i \in \mathcal{O},\ n = 1,\dots,N, && (19d)\\
& \sum_{\substack{ij\in A\\ i'j\in A'}} \left(x^m_{Tnij} + y^m_{Ti'j}\right) + z^m_{Tnj} = d_j && \forall j \in \mathcal{D},\ n = 1,\dots,N, && (19e)\\
& \sum_{\substack{il,\,lj\in A\\ i'l,\,lj'\in A'}} \left(x^m_{Tnil} + y^m_{Ti'l} - x^m_{Tnlj} - y^m_{Tlj'}\right) = 0 && \forall l \in \mathcal{T},\ n = 1,\dots,N, && (19f)\\
& \sum_{i'j'\in A'} \zeta^m_{Ti'j'} \le T\Lambda, && && (19g)\\
& x,\ y,\ z \ge 0,\ \zeta \in \mathbb{B}^{|A'|}. && && (19h)
\end{aligned}$$

4.2 A stochastic approximation using the initial information

These tests present an additional useful basis of comparison: integer program (IP) SAAs for each forecast timeline subject to the outage outcomes sampled at time 0. In other words, it is exactly like the problem given by equations (19) except with the slight subscript specification here:

$$\begin{aligned}
\tilde{V}^m_{IP@0} := \min_{\zeta^m_0, x^m_0, y^m_0, z^m_0} \quad & \frac{1}{N}\sum_{n=1}^{N}\sum_{j=1}^{|\mathcal{D}|} z^m_{0nj} && && \text{(20a)}\\
\text{s.t.} \quad & x^m_{0n} \le u^m_{0n} && \forall n = 1,\dots,N, && \text{(20b)}\\
& y^m_0 \le u'^{\mathrm{T}}\zeta^m_0, && && \text{(20c)}\\
& \sum_{\substack{ij\in A\\ ij'\in A'}} \left(x^m_{0nij} + y^m_{0ij'}\right) \le b_i && \forall i \in \mathcal{O},\ n = 1,\dots,N, && \text{(20d)}\\
& \sum_{\substack{ij\in A\\ i'j\in A'}} \left(x^m_{0nij} + y^m_{0i'j}\right) + z^m_{0nj} = d_j && \forall j \in \mathcal{D},\ n = 1,\dots,N, && \text{(20e)}\\
& \sum_{\substack{il,\, lj\in A\\ i'l,\, lj'\in A'}} \left(x^m_{0nil} + y^m_{0i'l} - x^m_{0nlj} - y^m_{0lj'}\right) = 0 && \forall l \in \mathcal{T},\ n = 1,\dots,N, && \text{(20f)}\\
& \sum_{i'j'\in A'} \zeta^m_{0i'j'} \le T\Lambda, && && \text{(20g)}\\
& x,\ y,\ z \ge 0,\ \zeta \in \mathbb{B}^{|A'|}. && && \text{(20h)}
\end{aligned}$$

Note that it is still allowed to make the cumulative number of decisions.

5 Experiments on variations and adaptations

5.1 Random network generation

The generated network is a generalized and modified hub-and-spoke design. There are four types of nodes in the graph and correspondingly four types of edges. Supply nodes have nothing but outgoing supply lines, hubs have both outgoing and incoming main lines to other hubs, and termini (the demand nodes) have nothing but incoming feed lines. The fourth type of node/edge pair is somewhat different, and is what distinguishes the graph from a standard hub-and-spoke graph: relay nodes and switch paths. Relay nodes are intermediary, or transshipment, nodes that indirectly connect one hub to another through switch path lines. They also may have feed lines to demand nodes and supply lines from supply nodes. Balancing the demand takes the sum of supply amounts from all supply nodes and distributes them evenly across all demand nodes. Hubs connect to all termini within a specified radius. Next the algorithm creates switch paths. The algorithm finds the shortest distances between any hub and relay, connects them, then connects relays to relays in nearest-neighbor fashion until the path length is reached. The algorithm then connects the final relay to the closest hub. The edge placement procedure wraps up by connecting supply and

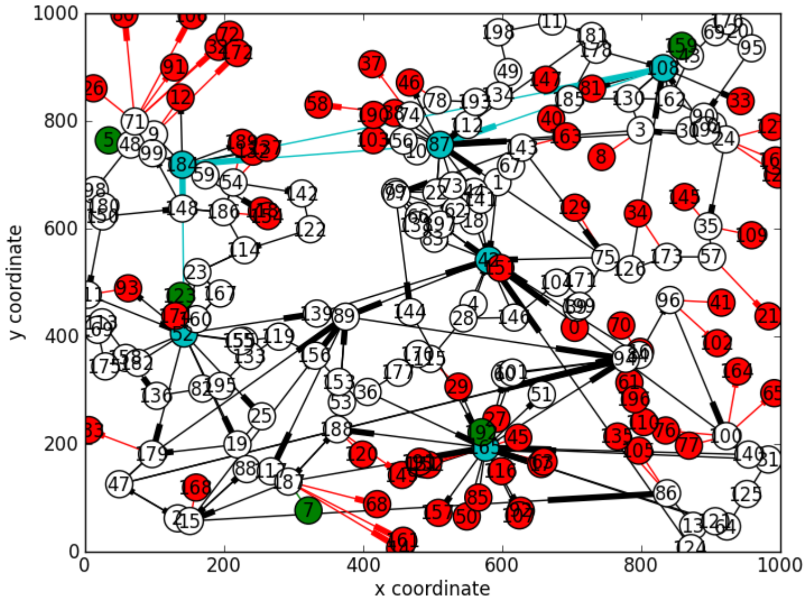


Figure 4: Large-scale randomly generated network. Supply nodes are green, demand nodes are red, hubs are cyan, and the remaining nodes are relay nodes. Edges are directed towards the thicker end.

demand nodes to the established network. Once node and edge placement are finished, the network generator does some final tests to ensure connectedness and feasibility.

A similar temporary network is built for the model, representing the set of installable edges.

The network is comprised of 200 nodes, only 71 of which are demands and 5 are suppliers. There are 317 edges, but only 128 installable edges. A map of the network is given in Figure 4. Since there is more than 1 hub, there are main lines which are colored cyan.

5.2 Monte Carlo forecast timeline simulation

Generating a forecast for a single time period is straightforward: given an epicenter, a set of radii, and an upper and lower bound for each radius, the MC simulation assigns to every edge a random outage probability between the bounds for the smallest concentric circle centered at the epicenter to which it "belongs". An edge belongs to a circle if a user-specified percent of the edge lies within its borders.

There are five parameters involved in the dynamics of evolving the forecast from one time step to the next. The epicenter moves randomly, biased toward reaching the edge, as the new node's distance away from the current epicenter is sampled from a triangular distribution. This encourages greater variability in an otherwise restricted system. In addition to the shift radius, the four other parameters that change the forecast are multipliers that geometrically increase or decrease the next innermost region of effect, the upper and lower bounds of outage percentages, and the shift radius.

The radial means by which the probabilities are placed are perhaps less likely to model a hurricane's path and are more reminiscent of e.g. a targeted terrorist attack. To provide a more accurate model for a hurricane, one might implement a "bar" based probability assignment scheme, where probabilities are placed corresponding to concentric strips. By whichever means, these are all possible, as any user may easily substitute his or her own procedures for generating these components of the model. However, for the forecast simulator it is important to maintain an aspect of refinement, that as time progresses the failure chances have smaller and smaller ranges to some extent. It would appear to be desirable to have a Dirichlet process that is not memoryless for the approximate dynamic programming approach to converge to an optimal solution.

5.3 Computational results

The following tests gauge solution quality only. While introducing one or more of the variations of §3 can slow down run time greatly, it is not of central concern since we recognize that the training algorithm may be run off-line; that is, well before the event and therefore with plenty of time to spare. These experiments are executed on a currently high performance computer, but not a supercomputer-level machine, so speeds should remain roughly similar to any further use of the program. It uses the Ubuntu server OS version 10.10 with two six-core processors and 64 GB RAM. The slowest run time for a variant configuration is approximately 7 hours for training and 18 hours for testing 300 iterations, and so even if one wanted to run the training algorithm on-line, it would not be unreasonable to do so. In the context of planning for an impending hurricane, predictions occur starting about 3–5 days before the event and we would probably want an installation plan for each day.

There are several factors to which all tests adhere. All of the tests on the random network are run with 3 time steps and an installation budget of 7 making for 21 edges possible to install over the course of the trials. Based on preliminary testing [17], we chose to use $\epsilon = 0.03$ in the transfer function of Step 1.4.3. In a preliminary comparison of constant, harmonic, and McClain step sizes, the McClain step sizes were superior, so we henceforth use the McClain step size with $\alpha = 0.03$. When employing the disjoint control step, we set the number of outcomes for the base cost at $K = 10$. All tests which use the target value calculations given in Section 3.1 similarly specify the SAA parameter as $K = 10$.

For the random network, we have

$$\sum_{m=1}^{M} \tilde{V}^m_{IP@T} \approx 6,200 \quad \text{and} \quad \sum_{m=1}^{M} \tilde{V}^m_{IP@0} \approx 15,100.$$

It is obvious to see that the true stochastic optimum lies somewhere between these two values, and so we can calculate a crude optimality gap to measure the algorithm's performance:

$$opt\_gap := \frac{1}{M} \sum_{m=1}^{M} \left(\frac{\tilde{V}^m_{LP} - \tilde{V}^m_{IP@T}}{\tilde{V}^m_{IP@0} - \tilde{V}^m_{IP@T}} \right), \tag{21}$$

where $\tilde{V}^m_{LP}$ denotes the value of the application of the policy derived from our training scenarios to each of the M test scenarios. Hopefully $\tilde{V}^m_{LP} < \tilde{V}^m_{IP@0}$ (i.e. it lies in the gap), but as we will see this is not necessarily the case. If the LP value is greater than the IP value at time 0, this indicates that our algorithm is unable to make proper decisions given the description of the network that we provide. This is a big motivator behind why we experiment with the different modifications presented in section 3.

To get a better feel for the performance of the algorithm, it was tested against a portion of the power infrastructure of New Hanover County in North Carolina. The network consists of 290 nodes, 261 of which are demand nodes, and 8 are suppliers. The network has 292 edges and its installable network is comprised of the same edges. This configuration is quite different in terms of demand nodes as a percentage of all nodes and the edge-to-node ratio than the network previously used. This causes some interesting behavior. For this run, 8 edges were allowed to be installed per time step, to account for the increase in total number of edges. Because run time is a function of the number of installable edges, only the first 80 iterations of training were considered, and the tests evaluated on even numbered iterations (run time performance will be discussed later). The following tests were performed across 20 forecast timelines with the LP and IP SAAs optimizing over 50 scenarios for each. The LP SAA is the calculation of the base cost as given in equations (8d)–(8j).

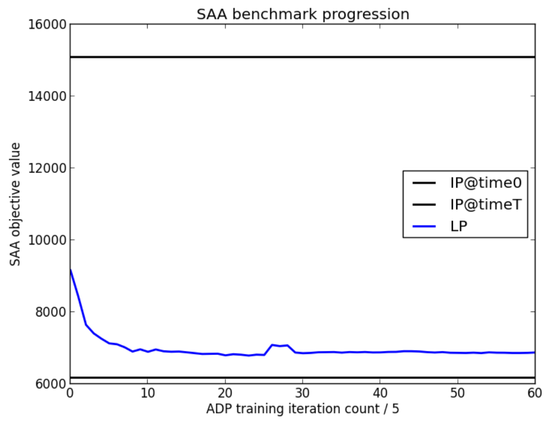
We apply some of our tests to a new network. This network, which we shall call the CLARC county network (or simply CLARC), is a subset of a network specifically designed to represent an interdependent services infrastructure, developed in part by Loggins [24] (see also [26, 25]). CLARC is even bigger and more complex than the NHC network, consisting of 545 edges among 535 nodes, of which 477 are demand nodes and 4 are supply nodes. The set of installable edges is the same as the edges of the original network. The network, while artificial, was designed to closely model a conglomeration of interdependent service networks: power, water, communications, and vital roads.

Among the different cases, the same forecast timeline scenarios and outage outcomes are used in training the policy, and likewise a single separate set of scenarios and outcomes is used in calculating the LP and IP test values.

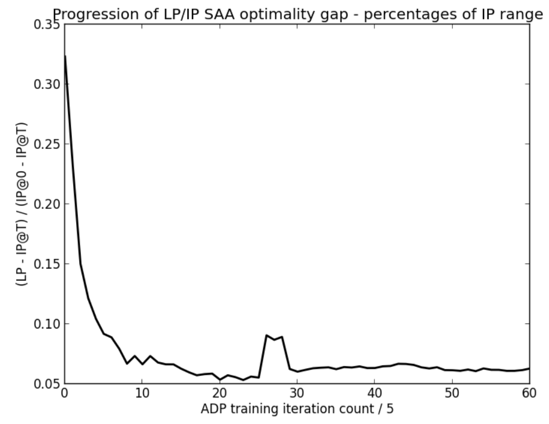
In the disjoint control step edges are selected based off of their discount $\tilde{\zeta}_t^n$ alone. Therefore, we impart some influence of the network's characteristics into the decision by incorporating both the base cost v_{Ct}^n and the target's installation decisions – decisions that are subject to network constraints – into the prediction step.

Early experiments [17] using the gradient descent direction of equation (12), showed an *opt_gap* of about 50%. The earlier results are not included in this paper, because implementing prediction with equation (16) showed a drastic improvement of performance. Figures 5(a) and 5(b) show the performance for the random network. At its best, it shows a 44.5% *opt_gap* reduction from the disjoint method with the gradient descent direction. Convergence to steady-state occurs at approximately iteration 125 after which we do see a noticeable jump up in the graph. These jumps can never entirely be ruled out due to the random nature of scenario and outage selection, epsilon-greedy exploration, and a non-convergent step size in the prediction step. Luckily these factors do not hinder the algorithm's ability to converge to a solution.

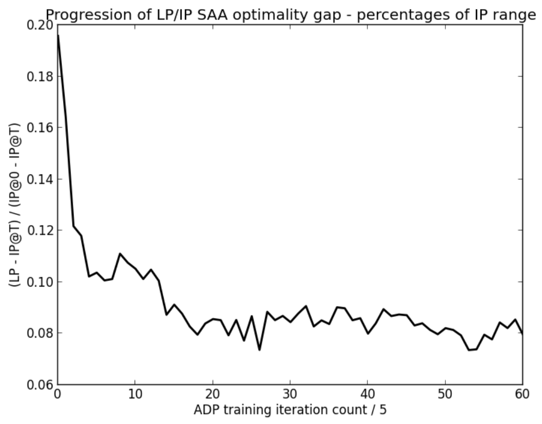
The performance of the same configuration using the NHC network was not as strong. Convergence occurred roughly at iteration 80, with some noisy intermittently increasing performance



(a) Objective value with target decisions



(b) Improvement in gap with target decisions



(c) Improvement in gap with recursive least squares

Figure 5: Results on randomly generated network. (a) Performance of disjoint control with gradient descent prediction including target decisions. (b) *opt_gap* performance with configuration of Figure 5(a). (c) Random network *opt_gap* performance: recursive least-squares prediction.

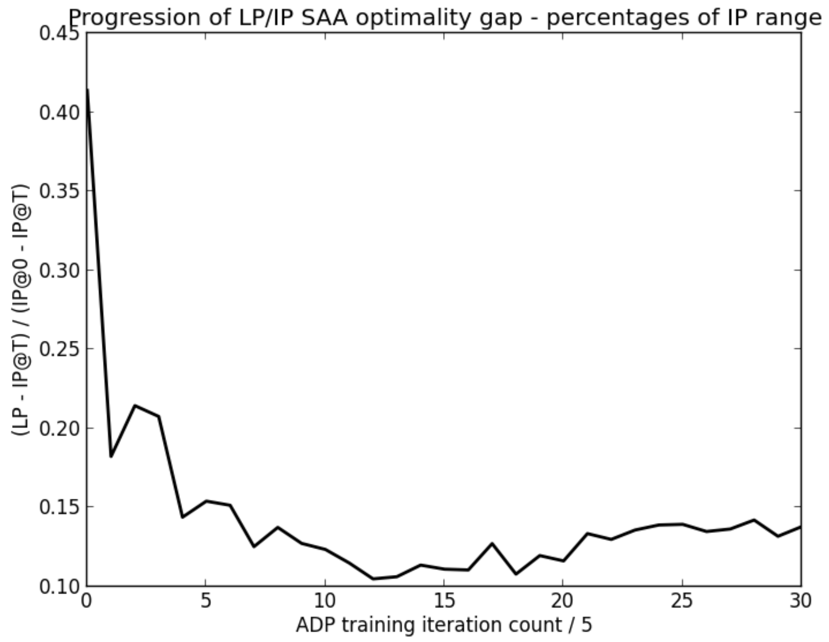


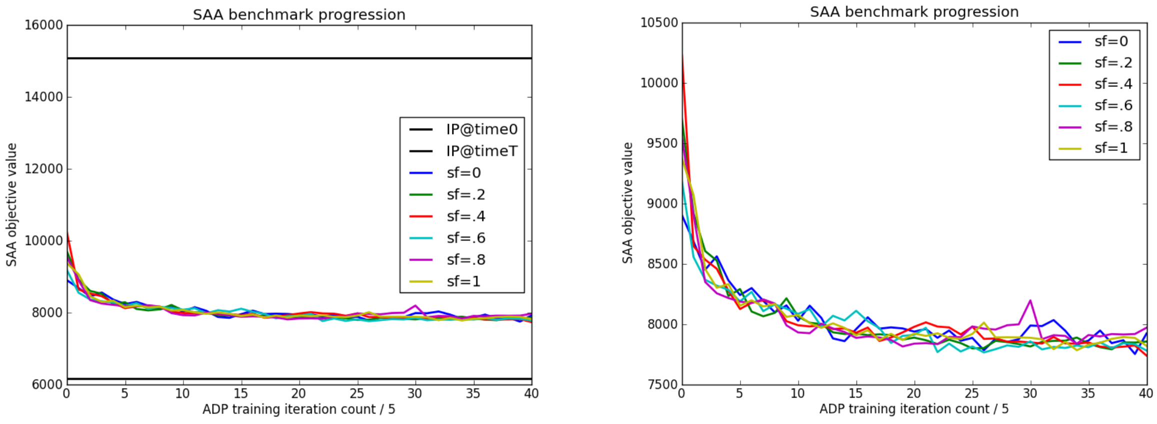
Figure 6: CLARC network *opt_gap* performance: standard basis, recursive least-squares prediction

thereafter. However, the algorithm actually performs worse than the IP at time 0. In the next section we show how we can improve upon this by using the bottleneck capacity ratio. Between these and the results of the random network so far, we are inclined to say that one should train the policy for no longer than 150 iterations to achieve the best performance, but this of course must depend on the structure of the network, scenario distributions, etc.

Results with the recursive least-squares prediction method for the random network are in Figure 5(c) and for CLARC in Figure 6. In more extensive testing, we found that in general recursive least-squares is better than the gradient descent method. It is intuitive that recursive least-squares performs at least as well as the gradient descent method, since both are addressing the same goal in the same manner. The only difference is that the recursive least-squares method can alter VFA coefficients individually based on the system's response in an iteration whereas the gradient method crudely changes all installed edges' coefficients uniformly.

6 Bottleneck capacity ratio

For each edge $i \in \mathcal{A}'$, the basis function calculation in (6) and (7) involves only the probability of failure of the shortest path between the two endpoints of the edge. In order to take into account the capacity of this path and to better capture the existence of alternative paths, we introduced a bottleneck capacity ratio. We calculate the ratio of the capacities of the lowest capacity edge in $F(i)$



(a) Full-scale (b) Zoomed in

Figure 7: Scaling factor performances of random network using disjoint control and least-squares prediction. (a) Full vertical scale. (b) Zoomed in

to the highest capacity edge in $\mathcal{A}'$, giving:

$$\rho_{ni} := \frac{\min_{k \in F(i)} u^n_{tk}}{\max_{l \in \mathcal{A}'} u^n_{tl}}. \tag{22}$$

The importance of this term is weighted by a parameter $\sigma \in [0, 1]$. The modified basis function is

$$\varphi_i \left(\psi^n_t\right) = \pi \left(F(i)\right) \left(1 - \sigma \left(1 - \rho_{ni}\right)\right), \tag{23}$$

where $\pi(F(i))$ is defined in (6). Figures 7(a) and 7(b) show the relative performances for the random network and Figure 8 that of the NHC network for varying values of the scale factor σ. Both were run using the disjoint control method and recursive least-squares prediction. For NHC, incorporating 60% of the bottleneck capacity ratio is best (using 100% was actually the worst!), while for the random network the differences were rather insignificant. The mere fact that there was a larger variation in solution performance for various scaling factors in the NHC network than for the random network suggests that there is some aspect of NHC's network that influences the policy's decision making process. Figure 9 shows the performance on CLARC with the incorporation of the bottleneck capacity ratio, with $\sigma = 1$. The standard basis function calculation works best for the random network but using the bottleneck capacity ratio in its calculations is better for the NHC network; in CLARC we see roughly the same results between the two functions. We speculate that different basis functions yield better results for the networks because of the network structures themselves: the random network has much more redundancy (a higher edge-to-node ratio) and a lower demand density (ratio of demand nodes to the total number of nodes). Changing the basis function specifically by using the bottleneck capacity ratio causes the basis functions to place more emphasis on the flow the installable edge can provide, further imposing elements of the network dynamics into the prediction calculation.

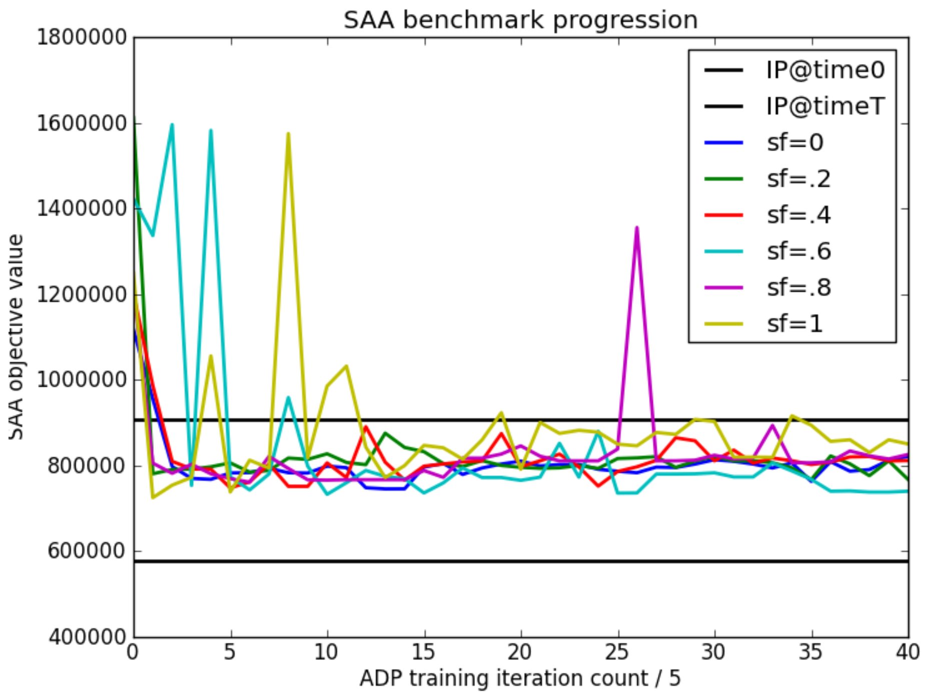


Figure 8: Scaling factor performances of NHC network with configuration of Figure 7(a)

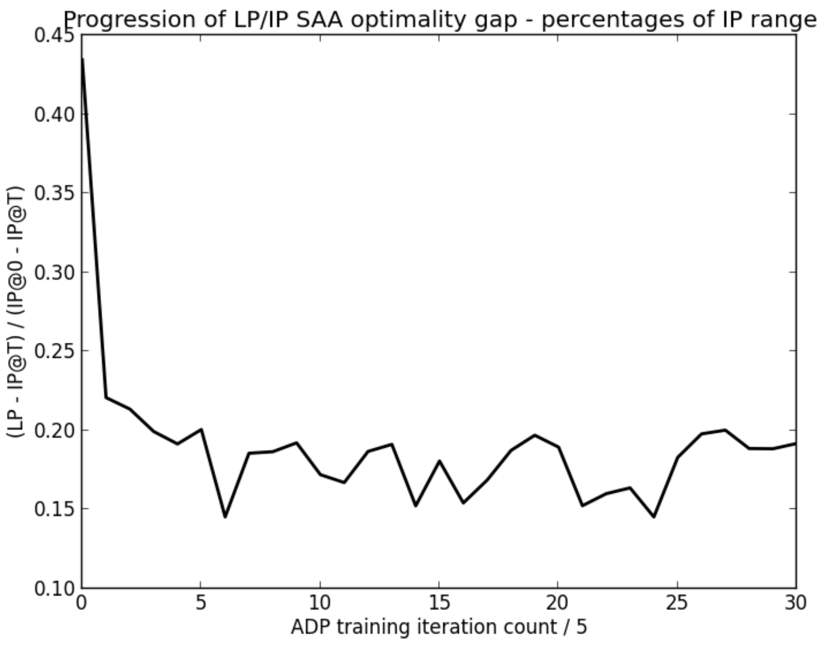


Figure 9: CLARC network *opt_gap* performance: bottleneck capacity ratio basis, recursive least-squares prediction

7 Conclusions and future work

We experimented on three different infrastructure networks: a randomly generated network, a network based on New Hanover County in North Carolina, and a larger-scale realistic network denoted as CLARC county. Our approach allows the modification of an installation plan as the event draws closer. In each case, we improved dramatically over the strategy of determining all installation decisions based on an initial forecast. In particular, for the random network and CLARC county, our approach was able to close 80% or more of the gap between a plan based on just the initial information and a plan based on perfect information (which would not be implementable in practice); the corresponding figure for New Hanover County was around 50%, which is still notable. Further, our solution is within 10–30% of the idealistic plan based on perfect information.

Our approach used approximate dynamic programming. We tackled preventing unmet demand in anticipation of network disruptions rather than minimizing costs after the fact. We furthermore extended the problem to a multistage model to weigh the concerns of resource availability against installation effectiveness. The former issue was handled by considering an installation budget per time step, and the latter by having an evolving distribution of uncertainty over time. Our work stands out within the reinforcement learning community because our model changes over time. In a typical network application, node demand is considered uncertain according to a fixed probability distribution over time. This amounts to altering the constants in the constraints of the control step. In our problem, the constraints themselves change over time, since installed edges become incorporated into flow constraints and different existing edges get removed depending on the changing probability distribution. The changing forecasts over time represent a merging of the two fields: the distributions themselves are uncertain and are sampled from a Monte Carlo simulation, then these forecasts provide actual sample outcomes when considering policy improvement.

With the problem statement formulated and the model constructed, we applied approximate dynamic programming to arrive at a solution, using a linear-in-the-parameters value function approximation. The basis functions correspond to the individual VFA coefficients, which in turn correspond to individual installation decisions. The basis function calculation is a complex product of an amalgamation of network data rather than using state data element-by-element. We further embellished this calculation to include static network data (such as edge capacities).

We also examined several options in the second stage of the reinforcement learning process: policy evaluation. We tried adaptations of gradient descent and recursive least-squares updates. For the gradient descent and recursive least-squares methods, we altered the descent direction (essentially redefining the basis function calculation) to include solution decisions of the target value.

Finally, we have developed various tests to gauge performance for these methods. For pseudo-random and real-world networks, we evaluated performance by making several comparisons to alternate solution methods using varying levels of perfect information; that is, we gauged performance relative to stochastic programs that were with respect to forecast distributions at different time steps or with respect to different forecast distributions within the same time step. We also measured the changes to VFA coefficient values and runtime efficiency over iterations. With these tests, we were able to corroborate convergence with theorems proving convergence for more simplistic algorithms.

Further algorithmic study mainly pertains to basis functions, policy evaluation (the prediction step), and policy improvement (the control step). There is much variability in performance relative to different forms of basis functions. With regards to the prediction step, another potential extension to our work involves finding a generalization for establishing a descent direction. One way to do this is to investigate different weightings between the target value decisions and the control step decisions (rather than using values 2 and 1 for $\bar{\zeta}$ and $\tilde{\zeta}$ in equation (16), respectively), but of course other combinations could be considered. As with prediction, one can work toward refining the present variations or adapting other traditional methods to the given control methods outlined here.

References

[1] A. Atamtürk and M. Zhang, *Two-stage robust network flow and design under demand uncertainty*, Operations Research **55** (2007), no. 4, 662–673.

[2] R. Bellman, *Dynamic programming*, Dover Publications, Mineola, NY, 2003.

[3] O. Berman, E. Ianovsky, and D. Krass, *Optimal search path for service in the presence of disruptions*, Computers and Operations Research **38** (2011), no. 11, 1562–1571.

[4] D. P. Bertsekas and J. Tsitsiklis, *Neuro-dynamic programming*, Athena Scientific, Nashua, NH, USA, 1996.

[5] S. Bhatnagar and M. S. Abdulla, *A reinforcement learning based algorithm for finite horizon Markov decision processes*, Proceedings of the 45th IEEE Conference on Decision and Control, 2006, pp. 5519–5524.

[6] B. Cavdaroglu, E. Hammel, J. E. Mitchell, T. C. Sharkey, and W. A. Wallace, *Integrating restoration and scheduling decisions for disrupted interdependent infrastructure systems*, Annals of Operations Research **203** (2013), 279–294.

[7] S. Cho, P. Gordon, H. W. Richardson, J. E. Moore, and M. Shinozuka, *Analyzing transportation reconstruction network strategies: a full cost approach*, Review of Urban and Regional Development Studies **12** (2000), no. 3, 212–227.

[8] C. Coffrin and P. Van Hentenryck, *A linear-programming approximation of AC power flows*, INFORMS Journal on Computing **26** (2014), no. 4, 718–734.

[9] C. Coffrin, P. Van Hentenryck, and R. Bent, *Last-mile restoration for multiple interdependent infrastructures*, Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, 2012, pp. 455–463.

[10] T. G. Dietterich and X. Wang, *Batch value function approximation via support vectors*, Advances in Neural Information Processing Systems 14, MIT Press, 2001, pp. 1491–1498.

[11] M. Dror, *Modeling vehicle routing with uncertain demands as a stochastic program: properties of the corresponding solution*, European Journal of Operational Research **64** (1993), no. 3, 432–441.

[12] L. Du and S. Peeta, *Pre-disaster investment planning decisions to enhance transportation network survivability and reduce post-disaster response times*, Networks and Spatial Economics **14** (2014), no. 2, 271–295.

[13] A. Erdelyi and H. Topaloglu, *Approximate dynamic programming for dynamic capacity allocation with multiple priority levels*, IIE Transactions **43** (2010), no. 2, 129–142.

[14] A. L. Erera, J. C. Morales, and M. Savelsbergh, *Robust optimization for empty repositioning problems*, Operations Research **57** (2009), no. 2, 468–483.

[15] D. P. de Farias and B. Van Roy, *The linear programming approach to approximate dynamic progamming*, Operations Research **51** (2003), no. 6, 850–865.

[16] G. A. Godfrey and W. B. Powell, *An adaptive dynamic programming algorithm for dynamic fleet management*, Transportation Science **36** (2002), no. 1, 21–39.

[17] E. Hammel, *Using reinforcement learning to improve network durability*, Ph.D. thesis, Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180, May 2013.

[18] P. Van Hentenryck and C. Coffrin, *Transmission system repair and restoration*, Mathematical Programming **151** (2015), no. 1, 347–373.

[19] P. Kall and S. W. Wallace, *Stochastic programming*, John Wiley, Chichester, UK, 1994.

[20] E. E. Lee, J. E. Mitchell, and W. A. Wallace, *Restoration of services in interdependent infrastructure systems: a network flows approach*, IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews **37** (2007), no. 6, 1303–1317.

[21] T. Levina, Y. Levin, J. McGill, and M. Nediak, *Network cargo capacity management*, Operations Research **59** (2011), no. 4, 1008–1023.

[22] Y.-K. Lin, *System reliability evaluation for a multistate supply chain network with failure nodes using minimal paths*, IEEE Transactions on Reliability **58** (2009), no. 1, 34–40.

[23] ———, *A novel algorithm to evaluate the performance of stochastic transportation systems*, Expert Systems with Applications **37** (2010), no. 2, 968–973.

[24] R. Loggins, *Improving the resilience of social infrastructure systems to an extreme event*, Ph.D. thesis, Rensselaer Polytechnic Institute, Troy, NY 12180, 2015.

[25] R. Loggins, J. E. Mitchell, T. C. Sharkey, and W. A. Wallace, *Improving the recovery of interdependent social infrastructure systems after an extreme event: model and application*, Tech. report, Industrial and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180, USA, September 2015.

[26] R. Loggins and W. A. Wallace, *Rapid assessment of hurricane damage and disruption to interdependent civil infrastructure systems*, Journal of Infrastructure Systems **online first** (2015), 14 May.

[27] M. S. Maxwell, M. Restrepo, S. G. Henderson, and H. Topaloglu, *Approximate dynamic programming for ambulance redeployment*, INFORMS Journal on Computing **22** (2010), no. 2, 266–281.

[28] L. Nagar and K. Jain, *Supply chain planning using multi-stage stochastic programming*, Supply Chain Management: An International Journal **13** (2008), no. 3, 251–256.

[29] S. G. Nurre, B. Cavdaroglu, J. E. Mitchell, T. C. Sharkey, and W. A. Wallace, *Restoring infrastructure systems: An integrated network design and scheduling (INDS) problem*, European Journal of Operational Research **223** (2012), no. 3, 794–806.

[30] V. Pillac, P. Van Hentenryck, and C. Even, *A conflict-based path-generation heuristic for evacuation planning*, Transportation Research Part B: Methodological **83** (2016), 136–150.

[31] W. B. Powell, *Approximate dynamic programming: solving the curses of dimensionality*, Wiley-Blackwell, Hoboken, NJ, 2007.

[32] ______, *Merging AI and OR to solve high-dimensional stochastic optimization problems using approximate dynamic programming*, INFORMS Journal on Computing **22** (2010), no. 1, 2–17.

[33] W. B. Powell, J. A. Shapiro, and H. P. Simão, *An adaptive dynamic programming algorithm for the heterogeneous resource allocation problem*, Transportation Science **36** (2002), no. 2, 231–249.

[34] B. Rottkemper, K. Fischer, A. Blecken, and C. Danne, *Inventory relocation for overlapping disaster settings in humanitarian operations*, OR Spectrum **33** (2011), 721–749.

[35] T. Santoso, S. Ahmed, M. Goetschalckx, and A. Shapiro, *A stochastic programming approach for supply chain network design under uncertainty*, European Journal of Operational Research **167** (2005), no. 1, 96–115.

[36] N. Secomandi, *A rollout policy for the vehicle routing problem with stochastic demands*, Operations Research **49** (2001), no. 5, 796–802.

[37] A. Shapiro, D. Dentcheva, and A. Ruszczyński, *Lectures on stochastic programming: Modeling and theory*, MPS/SIAM Series on Optimization, vol. 9, SIAM, 2009.

[38] T. C. Sharkey, B. Cavdaroglu, H. Nguyen, J. Holman, J. E. Mitchell, and W. A. Wallace, *Interdependent network restoration: On the value of information-sharing*, European Journal of Operational Research **244** (2015), no. 1, 309–321.

[39] L. V. Snyder, M. P. Scaparra, M. S. Daskin, and R. L. Church, *Planning for disruptions in supply chain networks*, TutORials 2006 (M. P. Johnson, B. Norman, and N. Secomandi, eds.), INFORMS Tutorials in O.R. Series, INFORMS, 2006.

[40] I. Sungur, F. Ordóñez, and M. Dessouky, *A robust optimization approach for the capacitated vehicle routing problem with demand uncertainty*, IIE Transactions **40** (2008), no. 5, 509–523.

[41] R. S. Sutton, *Learning to predict by the methods of temporal differences*, Machine Learning **3** (1988), no. 1, 9–44.

[42] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, MIT Press, Cambridge, MA, 1998.

[43] H. Topaloglu and S. Kunnumkal, *Approximate dynamic programming methods for an inventory allocation problem under uncertainty*, Naval Research Logistics **53** (2006), no. 8, 822–841.

[44] H. Topaloglu and W. B. Powell, *Dynamic programming approximations for stochastic time-staged integer multicommodity-flow problems*, INFORMS Journal on Computing **18** (2006), no. 1, 31–42.

[45] J. N. Tsitsiklis and B. Van Roy, *An analysis of temporal-difference learning with function approximation*, IEEE Transactions on Automatic Control **42** (1997), no. 5, 674–690.

[46] B. Verweij, S. Ahmed, A. Kleywegt, G. L. Nemhauser, and A. Shapiro, *The sample average approximation method applied to stochastic routing problems: A computational study*, Computational Optimization and Applications **24** (2003), no. 2, 289–333.

[47] R. Wollmer, *Removing arcs from a network*, Operations Research **12** (1964), no. 6, 934–940.

[48] J. J. Wu, H. J. Sun, and Z. Y. Gao, *Capacity assignment model to defense cascading failures*, International Journal of Modern Physics C: Computational Physics and Physical Computation **20** (2009), no. 7, 991–999.