Approximate Dynamic Programming for Ambulance Redeployment

Matthew S. Maxwell
School of Operations Research and Information Engineering
Cornell University, Ithaca, NY 14853, USA
msm57@cornell.edu

Mateo Restrepo
Center for Applied Mathematics
Cornell University, Ithaca, NY 14853, USA
mr324@cornell.edu

Shane G. Henderson, Huseyin Topaloglu
School of Operations Research and Information Engineering
Cornell University, Ithaca, NY 14853, USA
{sgh9,ht88}@cornell.edu

May 19, 2009

Abstract

We present an approximate dynamic programming approach for making ambulance redeployment decisions in an emergency medical service system. The primary decision is where we should redeploy idle ambulances so as to maximize the number of calls reached within a delay threshold. We begin by formulating this problem as a dynamic program. To deal with the high-dimensional and uncountable state space in the dynamic program, we construct approximations to the value function that are parameterized by a small number of parameters. We tune the parameters using simulated cost trajectories of the system. Computational experiments demonstrate the performance of the approach on emergency medical service systems in two metropolitan areas. We report practically significant improvements in performance relative to benchmark static policies.

1 Introduction

Rising costs of medical equipment, increasing call volumes and worsening traffic conditions are putting emergency medical service (EMS) managers under pressure to meet performance goals set by regulators or contracts. Ambulance redeployment is one strategy that can potentially help. Ambulance redeployment, also known as relocation, move up, system-status management or dynamic repositioning, refers to any strategy by which a dispatcher repositions idle ambulances to compensate for others that are busy, and hence, unavailable. The increasing availability of geographic information systems and the increasing affordability of computing power have finally created ideal conditions for bringing real-time ambulance redeployment approaches to fruitful implementation.

In this paper, which is an outgrowth of Restrepo (2008), we present an approximate dynamic programming (ADP) approach for making real-time ambulance redeployment decisions. We begin by formulating the ambulance redeployment problem as a dynamic program. This dynamic program involves a high-dimensional and uncountable state space and we address this difficulty by constructing approximations to the value function that are parameterized by a small number of parameters. We tune the parameters through an iterative and simulation-based method. Each iteration of this method consists of two steps. In the first step, we simulate the trajectory of the greedy policy induced by the current value function approximation and collect cost trajectories of the system. In the second step, we tune the parameters of the value function approximation by solving a regression problem that fits the value function approximation to the collected cost trajectories. This yields a new set of parameters that characterize a new value function approximation, and so, we can go back and repeat the same two steps above. In this respect, the idea we use closely resembles the classical policy iteration algorithm in the Markov decision process literature. In particular, the first and second steps are respectively analogous to the policy evaluation and policy improvement step of the policy iteration algorithm.

There are two streams of literature that are related to our work. The first one is the literature on ADP. A generic approach for ADP involves using value function approximations of the form $\sum_{p=1}^{P} r_p \, \phi_p(\cdot)$, where $\{r_p : p = 1, \ldots, P\}$ are tunable parameters and $\{\phi_p(\cdot) : p = 1, \ldots, P\}$ are fixed basis functions; see Bertsekas and Tsitsiklis (1996) and Powell (2007). There are a number of methods to tune the parameters $\{r_p : p = 1, \ldots, P\}$ so that $\sum_{p=1}^{P} r_p \, \phi_p(\cdot)$ yields a good approximation to the value function. For example, temporal-difference learning and Q-learning use stochastic approximation ideas in conjunction with simulated trajectories of the system to iteratively tune the parameters; see Sutton (1988), Watkins and Dayan (1992), Tsitsiklis (1994), Bertsekas and Tsitsiklis (1996), Tsitsiklis and Van Roy (1997) and Si, Barto, Powell and Wunsch II (2004). The linear programming approach for ADP finds a good set of values for the parameters by solving a large linear program whose decision variables are $\{r_p : p = 1, \ldots, P\}$; see Schweitzer and Seidmann (1985), de Farias and Van Roy (2003) and Adelman and Mersereau (2007). Both classes of approaches are aimed at tuning the parameters $\{r_p : p = 1, \ldots, P\}$ so that $\sum_{p=1}^{P} r_p \, \phi_p(\cdot)$ yields a good approximation to the value function. The choice of the basis functions $\{\phi_p(\cdot) : p = 1, \ldots, P\}$, on the other hand, is regarded as more of an art form, requiring substantial knowledge of the problem structure. Applications of ADP include inventory control (Van Roy, Bertsekas, Lee and Tsitsiklis, 1997), inventory routing (Adelman, 2004), option pricing (Tsitsiklis and Van Roy, 2001), game playing (Yan, Diaconis, Rusmevichientong and Van Roy, 2005; Farias and Van Roy, 2006*b*), dynamic fleet management (Topaloglu and Powell, 2006) and network revenue management (Adelman, 2007; Farias and Van Roy, 2007).

The second stream of literature that is related to our work is the literature on ambulance redeployment. One class of redeployment models involves solving integer programs in real-time whenever an ambulance redeployment decision needs to be made; see Kolesar and Walker (1974), Gendreau, Laporte and Semet (2001), Brotcorne, Laporte and Semet (2003), Gendreau, Laporte and Semet (2006), and Nair and Miller-Hooks (2006). The objective function in these integer programs involves a combination of backup coverage for future calls and relocation cost of ambulances. They are usually computationally intensive, since they require solving an optimization problem every time a decision is made. As a result, a parallel computing environment is sometimes used to implement a working real-time system. A second class of models is based on solving integer programs in a preparatory phase. This approach provides a lookup table describing, for each number of available ambulances, where those ambulances should be deployed. Dispatchers attempt to dispatch so as to keep the ambulance configuration close to the one suggested by the lookup table; see Ingolfsson (2006) and Goldberg (2007). A third class of models attempts to capture the randomness in the system explicitly, either through a dynamic programming formulation or through heuristic approaches. Berman (1981*a*), Berman (1981*b*) and Berman (1981*c*) represent the first papers that provide a dynamic programming approach for the ambulance redeployment problem, and this approach was revisited recently by Zhang, Mason and Philpott (2008) to attempt to gain insight. However, these papers follow an exact dynamic programming formulation and, as is often the case, this formulation is tractable only in oversimplified versions of the problem with few vehicles and small transportation networks. Andersson (2005) and Andersson and Vaerband (2007) make the ambulance redeployment decision by using a "preparedness function" that essentially measures the capability of a certain ambulance configuration to cover future calls. The preparedness function is similar in spirit to the value function in a dynamic program, measuring the impact of current decisions on the future evolution of the system. However, the way the preparedness function is constructed is heuristic in nature.

When compared with the three classes of models described above, our approach provides a number of advantages. In contrast to the models that are based on integer programs, our approach captures the random evolution of the system over time since it is based on a dynamic programming formulation of the ambulance redeployment problem. Furthermore, the decisions made by our approach in real-time can be computed very quickly as this requires solving a simple optimization problem that minimizes the sum of the immediate cost and the value function approximation. In lookup table approaches, there may be more than one way to redeploy the ambulances so that the ambulance configuration over the transportation network matches the configuration suggested by the lookup table. Therefore, table lookup approaches still leave some aspects of dispatch decisions to subjective interpretation by dispatchers. Our approach, on the other hand, can fully automate the decision making process, while allowing dispatchers to override recommendations if they wish. In traditional dynamic programming approaches, one is usually limited to very small problem instances, whereas ADP can be used on problem instances with realistic dimensions. Our approach allows working with a variety of objective functions, such as the number of calls that are not served within a threshold time standard or the total response time for the calls. Furthermore, our approach allows the possibility of constraining the frequency and destinations of ambulance relocations. This is important since a relocation scheme should balance improvements in service with the additional redeployment burden imposed on ambulance crews.

In summary, we make the following research contributions. 1) We develop a tractable ADP approach for the ambulance redeployment problem. Our approach employs value function approximations of the form $\sum_{p=1}^{P} r_p \, \phi_p(\cdot)$ and uses sampled cost trajectories of the system to tune the parameters $\{r_p : p = 1, \ldots, P\}$. Since it is based on the dynamic programming formulation of the problem, our approach is able to capture the random evolution of the system over time. 2) We develop a set of basis functions $\{\phi_p(\cdot) : p = 1, \ldots, P\}$ that yield good value function approximations for the ambulance redeployment problem. This opens up the possibility of using other ADP approaches, such as temporal-difference learning and the linear programming approach. 3) We provide computational experiments on EMS systems in two metropolitan areas. Our results indicate that ADP has the potential to obtain good redeployment policies in real systems. They also show that our approach compares favorably with benchmark policies that are similar to those used in practice.

The remainder of this paper is organized as follows. In Section 2, we present a dynamic programing formulation for the ambulance redeployment problem. In Section 3, we describe our ADP approach. In Section 4, we discuss the basis functions that we use in our value function approximations. In Section 5, we report computational results for two metropolitan areas. We conclude in Section 6.

2 Markov Decision Process Formulation

This section presents a dynamic programming formulation of the ambulance redeployment problem. As will shortly be clear, our model involves an uncountable state space. For an excellent account of the basic terminology, notation and fundamental results regarding dynamic programming in uncountable state spaces, we refer the reader to Bertsekas and Shreve (1978).

2.1 State Space

There are N ambulances in the EMS system. To simplify the presentation, we assume that we do not keep more than M waiting calls, possibly by diverting excess calls to another EMS organization. This is not a restriction from a practical perspective since M can be quite large. The two main components in the state of the system are the vectors $A = (a_1, \ldots, a_N)$ and $C = (c_1, \ldots, c_M)$, where a_i contains information about the state of the ith ambulance and c_j contains information about the jth waiting call. Naturally, the state of the ambulances and the calls in the waiting queue evolve over time, but we omit this dependence for brevity. The state of ambulance i is given by a tuple $a_i = (\sigma_i, \ell_i, d_i, t_i)$, where σ_i is the status of the ambulance, ℓ_i and d_i are respectively the origin and destination locations of the ambulance and t_i is the starting time of any ambulance movement. To serve a call, an ambulance first moves to the call scene and provides service at the scene for a certain amount of time. Following this, the ambulance transports the patient to a hospital, and after spending some time at the hospital, the ambulance becomes free to serve another call. Therefore, the status of an ambulance σ_i can take the values "idle at base," "going to scene of call," "serving at scene of call," "going to hospital," "transferring patient to hospital" and "returning to base." If ambulance i is stationary, then we have $\ell_i = d_i$. If ambulance i is in motion, then t_i corresponds to the starting time of this movement. Otherwise, t_i corresponds to the starting time of the current phase in the service cycle. For example, if the status of the ambulance is "transferring patient at hospital," then t_i corresponds to the time at which the ambulance arrived at the hospital. This time is kept in the state variable to give a Markov formulation for the non-Markovian elements in the system, such as nonexponentially distributed service times and deterministic travel times. Similarly, for the jth call in the waiting queue, we have $c_j = (\delta_j, p_j, \zeta_j, \eta_j)$, where δ_j is the status of the call, p_j is the location of the call, ζ_j is the time at which the call arrived into the system and η_j is the priority level of the call. The status of a call δ_j takes one of the values "assigned to ambulance i" and "queued for service." We take a call off the waiting queue C as soon as an ambulance reaches this call and starts serving it.

We model the dynamics of the system as an event-driven process. Events are triggered by changes in the status of the ambulances or by call arrivals. Therefore, the possible event types in the system are "call arrives and is placed in the jth position," "ambulance i departs for scene of call j," "ambulance i arrives at scene of call j," "ambulance i leaves scene of call for hospital," "ambulance i arrives at hospital," "ambulance i finishes at hospital" and "ambulance i arrives at base." We assume that we can make decisions only at the times of these events. Events occur at discrete points in time, so our modeling approach precludes the possibility of making a decision at any time point. This naturally comes at the cost of some loss of optimality as it may be desirable to make decisions between the times of the events. For example, our modeling approach, as stated, does not allow the possibility of rerouting an ambulance before it reaches its destination. It may be possible to incorporate such extensions by defining artificial events such as "consider repositioning" and firing these events while an ambulance is in transit. Nevertheless, we do not consider these extensions here and rely on the fact that the events that we work with occur frequently enough to provide ample decision opportunities.

By restricting our attention to the times of events, we visualize the system as jumping from one event time to another. Therefore, we can use the tuple $s = (\tau, e, A, C)$ to represent the state of the system, where τ corresponds to the current time, e corresponds to the current event type, and A and C respectively correspond to the state of the ambulances and the waiting call queue. In this case, the state trajectory of the system can be written as $\{s_k : k = 1, 2, \ldots\}$, where s_k is the state of the system just after the kth event occurs. Time is rolled into our state variable. Throughout the paper, we use $\tau(s)$ and $e(s)$ to respectively denote the time and the event type when the state of the system is s. In other words, $\tau(s)$ and $e(s)$ are the first two components of the tuple $s = (\tau, e, A, C)$.

2.2 Controls

We assume that calls are served in decreasing order of priority, and within a given priority level, they are served in first-in-first-out order. Furthermore, the closest available ambulance is dispatched to a call. This is not an exact representation of reality, but it is close enough for our purposes.

We use $\mathcal{R}(s)$ to denote the set of ambulances that are available for redeployment when the state of the system is s. In our implementation, if the state of the system is s and the event $e(s)$ is of the form "ambulance i finishes at hospital," then we let $\mathcal{R}(s) = \{i\}$. Otherwise, we let $\mathcal{R}(s) = \varnothing$. As a result, we consider an ambulance as available for redeployment only immediately after it finishes transferring a patient to a hospital. Ambulances that are idle at the bases or moving to different locations are not considered for redeployment. The appealing aspect of this redeployment policy is that it minimizes disturbance to the crews, but exploring the benefit of additional redeployments is important and we do so in our computational experiments. To capture the decisions, we let $x_{ib}(s) = 1$ if we redeploy ambulance i to base b when the state of the system is s, and 0 otherwise. Letting $\mathcal{B}$ be the set of ambulance bases and $x(s) = \{x_{ib}(s) : i \in \mathcal{R}(s),\ b \in \mathcal{B}\}$, the set of feasible decisions can be written as

$$\mathcal{X}(s) = \big\{x(s) \in \{0,1\}^{|\mathcal{R}(s)|\times|\mathcal{B}|} : \sum_{b\in\mathcal{B}} x_{ib}(s) = 1 \quad \forall\, i \in \mathcal{R}(s)\big\}.$$

The constraints in this definition simply state that the ambulance considered for redeployment has to be redeployed to one base. If the event $e(s)$ is not of the form "ambulance i finishes at hospital," then we have $\mathcal{R}(s) = \varnothing$, which implies that $\mathcal{X}(s) = \varnothing$ as well. In this case, we simply allow the system to evolve naturally without any interference from the decision-maker.

An important implication of our definition of $\mathcal{R}(s)$ is that the cardinality of $\mathcal{X}(s)$ is small so that an optimization problem that takes place over this feasible set can be solved by enumerating over all feasible decisions. In other words, since we consider only one ambulance at a time for deployment, we can try each base one by one to find out to which base an ambulance should be redeployed. In contrast, if we considered K ambulances simultaneously for redeployment, then the number of possible redeployment decisions would be $|\mathcal{B}|^K$, which can get quite large for moderate values of K and $|\mathcal{B}|$. This is a simplification that we make to avoid the combinatorial aspects of the problem and focus more on its dynamic and stochastic nature. Having said that, $K = 2$ is easily within our computational grasp if we restrict attention to sensible relocations of short range, and that will be the subject of future research.

Not considering the ambulances that are in transit as available for redeployment may have some undesirable affects. For example, we may decide to redeploy an ambulance at the northeast corner of the city to a base at the southwest corner. As soon as this redeployment starts, an ambulance in the southwest corner of the city may be available for deployment and this ambulance may be redeployed to a base at the northeast corner. It may be better to reroute the first ambulance to the northeast base and redeploy the second ambulance to the southwest base, although some results of Zhang et al. (2008) suggest that this is not universally true. The controls that we adopt in this paper miss this kind of opportunity to improve performance. Nevertheless, our computational experiments indicate that significant improvements are possible even when we ignore such opportunities, and one would expect that dispatchers monitoring the EMS system would take advantage of any obvious opportunities.

The set $\mathcal{B}$ may include additional locations, other than ambulance bases, at which ambulance crews can park and rest. As a result, our approach allows parking and resting at arbitrary locations as long as the list of possible locations is not too large. In our computational experiments, work with as many as 88 locations.

2.3 Fundamental Dynamics

Call arrivals are generated across the region $R \subset \mathbb{R}^2$ according to a Poisson point process with a known arrival intensity $\{\Lambda(t, x, y) : t \geq 0,\ (x, y) \in R\}$. As mentioned above, we have a fixed policy for serving calls, but our general approach does not depend on the particular form of this policy. If there are no available ambulances to serve a call, then the call is placed into a waiting queue. An ambulance serving a call proceeds through a sequence of events, including arriving at the scene, treating the patient, transporting and handing over the patient at the hospital. The main source of uncertainty in this call service cycle are the times spent between events. We assume that probability distributions for all of the activity durations are known.

Besides these sources of randomness, the major driver of dynamics is dispatching. As a result, the complete trajectory of the system is given by $\{(s_k, x_k) : k = 1, 2, \ldots\}$, where s_k is the state of the system at the time of the kth event and x_k is the decision (if any) made by the dispatcher when the state of the system is s_k. We capture the dynamics of the system symbolically by

$$s_{k+1} = f(s_k, x_k, \omega(s_k, x_k)),$$

where $\omega(s_k, x_k)$ is a random element of an appropriate space encapsulating all the sources of randomness described above and $f(\cdot, \cdot, \cdot)$ is the transfer function. One way to visualize $\omega(s_k, x_k)$ is that there is a stochastic process corresponding to call arrivals and each ambulance. The stochastic process corresponding to call arrivals keeps track of when calls arrive, along with the location and priority of each call. The stochastic processes corresponding to ambulances that are in transit keep track of the residual travel times. Similarly, the stochastic processes corresponding to ambulances that are serving calls at call scenes or at a hospital keep track of the residual service times. The state s_k and action x_k contain all of the information necessary to deduce the probability distributions for residual travel and service times. Therefore, $\omega(s_k, x_k)$ captures the time and type of the first event in the superposition of all stochastic processes.

2.4 Transition Costs

Along with a transition from state s_k to s_{k+1} through decision x_k, we incur a cost $c(s_k, x_k, s_{k+1})$. In our implementation, letting Δ be a fixed threshold response time that is on the order of 8 minutes, we use the transition cost function

$$c(s_k, x_k, s_{k+1}) = \begin{cases} 1 & \text{if the event } e(s_{k+1}) \text{ is of the form ``ambulance } i \text{ arrives at scene} \\ & \qquad\qquad \text{of call } j,\text{'' call } j \text{ is urgent and the response time exceeds } \Delta \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

This cost function counts the number of high-priority calls whose response times exceed Δ. We are interested in the performance of the system over the finite planning horizon $[0, T]$, so let $c(s, \cdot, \cdot) = 0$ whenever $\tau(s) > T$. In our implementation, T corresponds to two weeks. When the state of the system is s_k and we make the decision x_k, the transition cost $c(s_k, x_k, s_{k+1})$ is still a random variable since its value depends on s_{k+1}. This does not create any difficulty since the Markov decision process framework allows transition costs that depend on the states before and after the transition.

An appealing aspect of our transition cost function is its simplicity. Furthermore, most contracts and performance benchmarks in the EMS industry are formulated in terms of the percentage of calls that are reached within a time standard, and our transition cost function is in alignment with this tradition. However, this transition cost function does not distinguish between an urgent call whose response time exceeds Δ by one second or by one hour. As a result, it is conceivable that a call might be left unattended for a long period of time. This is not a huge concern in our approach since high-priority calls are served in first-in-first-out order within a priority level, but there may still be low-priority calls that are left unattended for a long period of time. Recent research by Erkut, Ingolfsson and Erdoğan (2008) incorporates medical outcomes into the objective function, and while we do not take that path here, our framework is general enough to do so.

2.5 Objective Function and Optimality Equation

A policy is a mapping from the state space to the action space, prescribing which action to take for each possible state of the system. Throughout the paper, we use $\mu(s) \in \mathcal{X}(s)$ to denote the action prescribed by policy μ when the state of the system is s. In other words, $\mu(s)$ is the action taken in state s given that we use policy μ. If we follow policy μ, then the state trajectory of the system $\{s_k^\mu : k = 1, 2, \ldots\}$ evolves according to $s_{k+1}^\mu = f(s_k^\mu, \mu(s_k^\mu), \omega(s_k^\mu, \mu(s_k^\mu)))$ and the discounted total expected cost incurred by starting from initial state s is given by

$$J^\mu(s) = \mathbb{E}\Big[\sum_{k=1}^{\infty} \alpha^{\tau(s_k^\mu)}\, c(s_k^\mu, \mu(s_k^\mu), s_{k+1}^\mu) \,|\, s_1^\mu = s\Big], \tag{2}$$

where $\alpha \in [0, 1)$ is a fixed discount factor. The expectation in the expression above involves the random variables $\{s_k^\mu : k = 1, 2, \ldots\}$ and $\tau(s_k^\mu)$ is the time at which the system visits state s_k^μ. The policy μ^* that minimizes the discounted total expected cost can be found by computing the value function through the optimality equation

$$J(s) = \min_{x \in \mathcal{X}(s)} \left\{ \mathbb{E}\Big[c(s, x, f(s, x, \omega(s, x))) + \alpha^{\tau(f(s,x,\omega(s,x))) - \tau(s)}\, J(f(s, x, \omega(s, x)))\Big] \right\} \tag{3}$$

and letting $\mu^*(s)$ be the minimizer of the right side above; see Bertsekas and Shreve (1978). The difficulty with the optimality equation above is that the number of possible values for the state variable is uncountable. Even if we are willing to discretize the state variable to obtain a finite state space, the state variable is still a high-dimensional vector and solving the optimality equation (3) through classical dynamic programming approaches is computationally very difficult. In the next two sections, we propose a method to construct tractable approximations to the value function.

The discount factor in (3) implies that we minimize the total expected *discounted* number of calls that are not reached within the time threshold, though one is more likely to be interested in the total expected *undiscounted* number of calls. The discount factor is an important computational device when we move to the ADP framework in the next section. In particular, our value function approximations invariably have some error, and the role of the discount factor is to put less emphasis on the value of a future state, as predicted by the value function approximation, when compared with the immediate transition cost. It is quite common in the ADP literature to use a discounted cost formulation when an undiscounted cost formulation may reflect the objectives of the decision-maker more accurately. For example, Farias and Van Roy (2004) and Farias and Van Roy (2006*a*) use ADP to construct strategies for playing Tetris. The authors use a discounted cost formulation, but the objective of the player is clearly to maximize the undiscounted total expected score. Crites and Barto (1996) use a continuous-time discounted cost formulation in an application targeted at minimizing elevator wait times, when the real performance measure of interest is the average wait time per person. Singh and Bertsekas (1996) develop an ADP method for channel allocation in cellular telephone systems. The objective of the decision-maker is to minimize the rate of blocked calls per unit time, whereas the authors use a formulation that minimizes the discounted number of blocked calls.

3 Approximate Dynamic Programming

The ADP approach that we use to construct approximations to the value function is closely related to the traditional policy iteration algorithm in the Markov decision processes literature. We begin with a brief description of the policy iteration algorithm. Throughout the rest of the paper, the greedy policy induced by an arbitrary function $\hat{J}(\cdot)$ refers to the policy that takes a decision in the set

$$\operatorname*{argmin}_{x\in\mathcal{X}(s)}\left\{\mathbb{E}\Big[c(s,x,f(s,x,\omega(s,x)))+\alpha^{\tau(f(s,x,\omega(s,x)))-\tau(s)}\,\hat{J}(f(s,x,\omega(s,x)))\Big]\right\} \tag{4}$$

whenever the state of the system is s. Finding the optimal solution to the problem above requires computing potentially difficult expectations. We use Monte Carlo simulation to compute such expectations and we carefully elaborate on this issue at the end of this section. If the state variable took finitely many values, then the optimal policy could be obtained by using the following policy iteration algorithm.

Policy Iteration

Step 1. Initialize the iteration counter n to 1 and initialize $J^1(\cdot)$ arbitrarily.

Step 2. (Policy improvement) Let μ^n be the greedy policy induced by $J^n(\cdot)$.

Step 3. (Policy evaluation) Let $J^{n+1}(\cdot)=J^{\mu^n}(\cdot)$, where $J^{\mu^n}(s)$ denotes the expected discounted cost incurred when starting from state s and using policy μ^n, as given in (2).

Step 4. Increase n by 1 and go to Step 2.

Even if we were willing to discretize the state variable to obtain a finite state space, since the state variable is a high-dimensional vector, the number of possible values for the state variable would be far too large to apply the policy iteration algorithm above directly. We try to overcome this difficulty by using value function approximations of the form

$$J(s,r)=\sum_{p=1}^{P} r_p\,\phi_p(s). \tag{5}$$

In the expression above, $r = \{r_p : p = 1, \ldots, P\}$ are tunable parameters and $\{\phi_p(\cdot) : p = 1, \ldots, P\}$ are fixed basis functions. The challenge is to construct the basis functions and tune the parameters so that $J(\cdot, r)$ is a good approximation to the solution to the optimality equation (3). To achieve this, each basis function $\phi_p(\cdot)$ should capture some essential information about the solution to the optimality equation. In Section 4, we describe one set of basis functions that work well for our application. Once a good set of basis functions is available, we can use the following approximate version of the policy iteration algorithm to tune the parameters $\{r_p : p = 1, \ldots, P\}$.

Approximate Policy Iteration

Step 1. Initialize the iteration counter n to 1 and initialize $r^1 = \{r_p^1 : p = 1, \ldots, P\}$ arbitrarily.

Step 2. (Policy improvement) Let μ^n be the greedy policy induced by $J(\cdot, r^n)$.

Step 3. (Policy evaluation through simulation) Simulate the trajectory of policy μ^n over the planning horizon $[0, T]$ for Q replications. Let $\{s_k^n(q) : k = 1, \ldots, K(q)\}$ be the state trajectory of policy μ^n in replication q and $C_k^n(q)$ be the discounted cost incurred by starting from state $s_k^n(q)$ and following policy μ^n in replication q.

Step 4. (Projection) Compute the tunable parameters at the next iteration as

$$r^{n+1} = \underset{r \in \mathbb{R}^P}{\operatorname{argmin}} \left\{ \sum_{q=1}^{Q} \sum_{k=1}^{K(q)} \left[C_k^n(q) - J(s_k^n(q), r) \right]^2 \right\}.$$

Step 5. Increase n by 1 and go to Step 2.

In Step 3 of approximate policy iteration, we use simulation to evaluate the expected discounted cost incurred by policy μ^n. Therefore, $\{C_k^n(q) : k = 1, \ldots, K(q),\ q = 1, \ldots, Q\}$ are the sampled cost trajectories of the system under policy μ^n. In Step 4, we tune the parameters $r = \{r_p : p = 1, \ldots, P\}$ so that the value function approximation $J(\cdot, r)$ provides a good fit to the sampled cost trajectories. We use the LAPACK linear algebra package in Netlib repository to solve the least squares regression problem in Step 4; see Netlib (2004). For our data sets, solving the least squares regression problem takes less than three seconds.

There is still one computational difficulty in the approximate policy iteration algorithm. When simulating the trajectory of policy μ^n in Step 3, we need to solve an optimization problem of the form (4) to find the action taken by the greedy policy induced by $J(\cdot, r^n)$. This optimization problem involves an expectation that is difficult to compute. As mentioned at the beginning of this section, we use Monte Carlo simulation to overcome this difficulty. In particular, if the state of the system is s and we want to find the action taken by the greedy policy induced by $J(\cdot, r^n)$ in this state, then we enumerate over all decisions in the feasible set $\mathcal{X}(s)$. Enumerating over all feasible decisions in the set $\mathcal{X}(s)$ is possible since the cardinality of this set is equal to the number of ambulance bases, which is at most 88 in our experiments. Starting from state s and taking decision x, we simulate the trajectory of the system until the next event and this provides a sample of $f(s, x, \omega(s, x))$. Since this simulation is only until the time of the next event, it is very quick to run. In particular, sampling a realization of $\omega(s, x)$ involves sampling the residual interarrival time for the next call, and at most N residual travel times and N residual service times. In this case, we obtain a sample of $f(s,x,\omega(s,x))$ by generating the state of the system at the next event time. By simulating multiple samples, we estimate the expectation

$$\mathbb{E}\Big[c(s,x,f(s,x,\omega(s,x))) + \alpha^{\tau(f(s,x,\omega(s,x)))-\tau(s)}\, J(f(s,x,\omega(s,x)),r^n)\Big]$$

through a sample average. Once we estimate the expectation above for all $x \in \mathcal{X}(s)$, we choose the decision that yields the smallest value and use it as the decision taken by the greedy policy induced by $J(\cdot,r^n)$ when the state of the system is s. This approach is naturally subject to sampling error, but it provides good performance in practice. In our computational experiments, we use 10 to 25 replications to estimate the expectation above. This is a small number of replications, but we use common random numbers when estimating the expectation for different actions. This allows us to quickly identify whether the expectation corresponding to a particular action is smaller than the expectation corresponding to another action.

Proposition 6.2 in Bertsekas and Tsitsiklis (1996) provides a performance guarantee for the approximate policy iteration algorithm. Their result is for finite state spaces, but it is easily extended to infinite state spaces. This result provides theoretical support for the approximate policy iteration algorithm, but its conditions are difficult to verify in practice. In particular, the result assumes that we precisely know the error induced by using regression to estimate the discounted total expected cost of a policy, and it assumes that expectations are computed exactly rather than via sampling as in our case. For this reason, we do not go into the details of this result and refer the reader to Bertsekas and Tsitsiklis (1996) for further details.

4 Basis Functions

In this section, we describe the basis functions $\{\phi_p(\cdot) : p = 1,\dots,P\}$ that we use in our value function approximations. We use six basis functions, some of which are based on the queueing insights developed in Restrepo, Henderson and Topaloglu (2008).

1. Baseline The first basis function is $\phi_1(s) = 1$. When multiplied by r_1 in (5), this basis function shifts the value function approximation to any desired level.

2. Unreachable Calls The second basis function computes the number of calls in the call waiting queue for which an ambulance assignment has been made, but the ambulance will not reach the call within the threshold response time. This quantity is easily computable when the travel times are deterministic, which is the case in our implementation. To see this, let $\mathbf{1}(\cdot)$ be the indicator function and $d(\ell_1,\ell_2)$ be the travel time between locations ℓ_1 and ℓ_2. Given that the state of the system is s, we can count the number of ambulances that are assigned to the jth call in the call waiting queue but will not reach the call within the threshold response time via

$$G_j(s) = \sum_{i=1}^{N} \mathbf{1}(\sigma_i = \text{“going to scene of call } j\text{”})\, \mathbf{1}(t_i + d(\ell_i,d_i) - \zeta_j \geq \Delta).$$

The expression above first checks each ambulance to see if there is one whose status is "going to scene of call j." If there is one such ambulance, then it checks whether the time at which this ambulance arrives at its destination exceeds the arrival time of the jth call by more than Δ time units. The second basis function can now be written as

$$\phi_2(s) = \sum_{j=1}^{M} G_j(s).$$

3. Uncovered Call Rate The third basis function captures the rate of call arrivals that cannot be reached on time by any available ambulance. To define this basis function precisely, we need additional notation. Recall that calls arrive across the region $R \subset \mathbb{R}^2$ according to a Poisson point process with arrival intensity $\{\Lambda(t, x, y) : t \geq 0,\ (x, y) \in R\}$. Partition the region R into L subregions and associate a representative point or center of mass location ρ_l with each subregion l.

The coverage of subregion l is the number of available ambulances that can reach the center of mass within the threshold time standard. We let $\mathcal{A}(s)$ be the set of available ambulances when the state of the system is s. This set includes the ambulances whose status is either "idle at base" or "returning to base." Given that the system is in state s, we let $\hat{\ell}_i(s)$ be the location of ambulance i at time $\tau(s)$. In this case, the coverage of subregion l can be written as

$$N_l(s) = \sum_{i \in \mathcal{A}(s)} \mathbf{1}(d(\hat{\ell}_i(s), \rho_l) \leq \Delta).$$

Using $\Lambda_l(t)$ to denote the total rate of call arrivals in subregion l at time t, we can compute the rate of call arrivals that are not covered by any available ambulance by

$$\phi_3(s) = \sum_{l=1}^{L} \Lambda_l(\tau(s))\, \mathbf{1}(N_l(s) = 0).$$

The values $\{\Lambda_l(t) : t \geq 0\}$ in the expression above are part of the problem data and can be computed in advance.

4. Missed Call Rate The previous two basis functions respectively capture calls already received that we know we cannot reach on time and the rate of arriving calls that cannot be reached on time because they are too far from any available ambulance. We could also fail to reach a call on time due to queueing effects from ambulances being busy with other calls. The fourth basis function represents an attempt to capture this effect. This basis function is of the form

$$\phi_4(s) = \sum_{l=1}^{L} \Lambda_l(\tau(s))\, P_l(s),$$

where $P_l(s)$ is the probability that all ambulances that could reach a call in subregion l on time are busy with other calls. We estimate $\{P_l(s) : l = 1, \ldots, L\}$ by treating the call service processes in different subregions as Erlang loss systems. In Erlang loss systems, calls arriving when all servers are busy are lost. In particular, in an Erlang loss system with arrival rate λ, service rate μ and n servers, the steady state probability of losing a call is given by

$$\mathcal{L}(\lambda, \mu, n) = \frac{(\lambda/\mu)^n/n!}{\sum_{k=0}^{n}(\lambda/\mu)^k/k!}.$$

In our EMS system, a call that arrives when all ambulances are busy is queued and served as ambulances become free, but the time threshold is almost always missed for such calls, so counting them as lost seems reasonable. The issue that such calls impose some load on the true system, but are discarded in an Erlang loss system creates a slight mismatch between our EMS system and the Erlang loss system, but our computational experiments show that this basis function is still highly effective.

To characterize the Erlang loss system for subregion l, given that the state of the system is s, we need to specify the number of servers, along with the arrival and service rates. Let $\mathcal{N}_l(s)$ be the set of available ambulances that can serve a call in subregion l within the threshold response time so that

$$\mathcal{N}_l(s) = \{i \in \mathcal{A}(s) : d(\hat{\ell}_i(s), \rho_l) \leq \Delta\}.$$

We use $|\mathcal{N}_l(s)|$ as the number of servers in the Erlang loss system for subregion l. Let $\mu_l(t)$ be the service rate in the loss system. This is the rate at which an ambulance can serve a call at time t in subregion l. It is difficult to come up with a precise value for $\mu_l(t)$. It primarily depends on the time spent at the scene of a call and any transfer time at the hospital, since the travel times are usually small relative to these quantities. In our implementation, we use historical data to estimate the time spent at the call scenes and the hospital, and add a small padding factor to capture travel times. Finally, let $\lambda_l(s)$ be the rate of call arrivals that should be served by ambulances in the set $\mathcal{N}_l(s)$. Coming up with a value for $\lambda_l(s)$ is even more difficult than devising a value for $\mu_l(t)$. One option is to let $\lambda_l(s) = \Lambda_l(\tau(s))$, which is the rate of call arrivals at time $\tau(s)$ in subregion l. However, ambulances in the set $\mathcal{N}_l(s)$ serve calls other than those in subregion l. To attempt to capture this, let

$$\lambda_l(s) = \sum_{i \in \mathcal{N}_l(s)} \sum_{k=1}^{L} \Lambda_k(\tau(s))\, \mathbf{1}(d(\hat{\ell}_i(s), \rho_k) \leq \Delta), \tag{6}$$

so that $\lambda_l(s)$ reflects the total call arrival rate in subregions that are close to any of the ambulances in the set $\mathcal{N}_l(s)$. We then use the approximation $P_l(s) \approx \mathcal{L}(\lambda_l(s), \mu_l(\tau(s)), |\mathcal{N}_l(s)|)$.

There are several shortcomings in the approximation that we use for $P_l(s)$. To begin with, there is double counting in the estimate of $\lambda_l(s)$. In particular, if two ambulances $i_1, i_2 \in \mathcal{N}_l(s)$ can both reach subregion l' within the time threshold, then the summation for $\lambda_l(s)$ counts $\Lambda_{l'}(\tau(s))$ twice. In addition, $\Lambda_{l'}(\tau(s))$ could be counted in the demand rates for multiple subregions. To be more precise, if there are three subregions l_1, l_2, l' and two ambulances $i_1 \in \mathcal{N}_{l_1}(s)$, $i_2 \in \mathcal{N}_{l_2}(s)$ such that both i_1 and i_2 can reach subregion l' within the time threshold, then the summations for $\lambda_{l_1}(s)$ and $\lambda_{l_2}(s)$ both count $\Lambda_{l'}(\tau(s))$. Therefore, we typically have $\sum_{l=1}^{L} \lambda_l(s) > \sum_{l=1}^{L} \Lambda_l(\tau(s))$ and the call arrival rates that we use in our approximation for $P_l(s)$ may not be accurate. In addition to this shortcoming, the number of servers that we use in the Erlang loss system may also not be accurate. In particular, it is not realistic to expect that the ambulances in the set $\mathcal{N}_l(s)$ exclusively serve the calls in subregion l. Finally, as mentioned above, the service rate $\mu_l(\tau(s))$ may not be accurate since it is obtained by adding a small padding factor to the time spent at the call scenes and the hospital. To overcome these shortcomings, we scale the call arrival rates by a factor κ. In particular, we use the call arrival rate $\kappa\,\Lambda_l(\tau(s))$ in (6) instead of $\Lambda_l(\tau(s))$. We find a good value for κ through preliminary experimentation. Noting that there are potential inaccuracies in the estimates of the call arrival rates, number of servers and service rates, the best choice of κ can be smaller or larger than one.

5. Future Uncovered Call Rate We do not consider ambulances that are already in transit as available for redeployment. Therefore, from the perspective of covering future calls, the destinations of moving ambulances are as important as their current locations. This is the motivation underlying the fifth and sixth basis functions.

Our fifth basis function parallels the third one, but it replaces the current locations of ambulances by their destinations. In other words, the definition of this basis function is identical to that of $\phi_3(\cdot)$, but the configuration of the ambulances that we use to compute $N_l(s)$ is not the current one, but an estimated future configuration that is obtained by letting all ambulances in transit reach their destinations and all stationary ambulances remain at their current locations. Given that the system is in state $s = (\tau, e, A, C)$ with $A = (a_1, \ldots, a_N)$ and $a_i = (\sigma_i, \ell_i, d_i, t_i)$, we define a new state $\vec{s}(s) = (\tau + 1/\sum_{l=1}^{L} \Lambda_l(\tau), e, \vec{A}, C)$ with $\vec{A} = (\vec{a}_1, \ldots, \vec{a}_N)$ and $\vec{a_i} = (\vec{\sigma}_i, d_i, d_i, \tau + 1/\sum_{l=1}^{L} \Lambda_l(\tau))$, where $\vec{\sigma}_i$ is the status of ambulance i when it reaches its destination d_i. In this case, the fifth basis function is

$$\phi_5(s) = \sum_{l=1}^{L} \Lambda_l(\tau(\vec{s}(s)))\,\mathbf{1}(N_l(\vec{s}(s)) = 0).$$

In the expression above, the time $\tau(\vec{s}(s)) = \tau + 1/\sum_{l=1}^{L} \Lambda_l(\tau)$ is used as an approximation for the expected time of the next call arrival. The next call may arrive before or after the ambulances actually reach their destinations, but we heuristically use the time $\tau(\vec{s}(s))$ simply to look into the future. The idea is that the estimated future configuration of the ambulances $\vec{A}$ is more likely to hold at the future time $\tau(\vec{s}(s))$ than at the current time $\tau(s)$.

We plug $J(\cdot, r)$ into the right side of (4) to find the action taken by the greedy policy induced by $J(\cdot, r)$. Since $J(\cdot, r)$ is computed at the future state $f(s, x, \omega(s, x))$, we need to compute $\phi_5(\cdot)$ at this future state as well. To compute $\phi_5(\cdot)$ at $f(s, x, \omega(s, x))$, we need to compute the state $\vec{s}(f(s, x, \omega(s, x)))$ and this state peeks even further ahead of the future state $f(s, x, \omega(s, x))$.

6. Future Missed Call Rate The sixth basis function parallels $\phi_4(\cdot)$ in that it captures the rate of calls that will likely be lost due to queueing congestion. As with the fifth basis function, it uses an estimated future configuration of the ambulances. It is defined as

$$\phi_6(s) = \sum_{l=1}^{L} \Lambda_l(\tau(\vec{s}(s)))\,P_l(\vec{s}(s)).$$

Our basis functions are relatively complex and this complexity, coupled with the expectation in problem (4), could make computing the greedy policy induced by the value function approximation

$J(\cdot, r)$ quite difficult. Nevertheless, since the cardinality of the feasible set $\mathcal{X}(s)$ is small, we can solve an optimization problem that takes place over this feasible set simply by enumerating over all possible decisions. For very large feasible sets, this approach would not be practical. We close this section with a brief note on the complexity of computing our basis functions at a particular state s. The effort required to compute $\phi_1(s)$ is clearly $O(1)$. We can compute $G_j(s)$ in $O(N)$ time, which implies that the effort required to compute $\phi_2(s)$ is $O(MN)$. It is possible to compute $N_l(s)$ in $O(N)$ time so that we can compute $\phi_3(s)$ in $O(NL)$ time. We can compute $\mathcal{N}_l(s)$ in $O(N)$ time. We always have $|\mathcal{N}_l(s)| \leq N$, so we can compute $\lambda_l(s)$ in $O(NL)$ time. Since the computation of the Erlang loss function with $|\mathcal{N}_l(s)|$ servers can be done in $O(N)$ time, we can compute $P_l(s)$ in $O(N + NL) = O(NL)$ time. Therefore, the effort required to compute $\phi_4(s)$ is $O(NL^2)$. The effort for computing the future state $\vec{s}(s)$ is small when compared to the basis functions. Therefore, we can compute $\phi_5(s)$ and $\phi_6(s)$ in essentially the same time as $\phi_3(s)$ and $\phi_4(s)$. As a result, all of our basis functions can be computed in $O(MN + NL^2)$ time. We give specific timing results in our computational results.

5 Computational Results

We now present computational results for two EMS systems. The first EMS system belongs to the city of Edmonton, Alberta in Canada and was also studied in Ingolfsson, Erkut and Budge (2003). We are not able to disclose the identity of the second EMS system due to confidentiality agreements.

5.1 Experimental Setup

In this section, we give a description of the data sets along with our assumptions.

1. The City of Edmonton The data we use for the city of Edmonton corresponds to the data set used in the computational experiments in Ingolfsson et al. (2003). The city has a population of 730,000 and covers an area of around 40×30 km^2. The EMS system has 16 ambulances, 11 bases and 5 hospitals. We assume that all ambulances are of the same type and operate all day. An ambulance, upon arriving at a call scene, treats the patient for an exponentially distributed amount of time with mean 12 minutes. After treating the patient at the call scene, the ambulance transports the patient to a hospital with probability 0.75. The probability distribution for the hospital chosen is inferred from historical data. The time an ambulance spends at the hospital has a Weibull distribution with mean 30 minutes and standard deviation 13 minutes. The turn-out time is assumed to be 45 seconds so that if an ambulance crew is at a base when notified of a call, then it takes 45 seconds to get on the road. An ambulance crew already on the road does not incur the turn-out time. A call that is not served within 8 minutes is interpreted as missed. The road network that we use models the actual road network on the avenue level. There are 252 nodes and 934 arcs in the road network. Travel times are deterministic and do not depend on the time of day.

The data we had access to were not sufficient to develop a detailed model of call arrivals, so we proceed with a representative model that does not exactly correspond to the actual distribution of calls over the city of Edmonton. The model maintains a constant overall arrival rate, but the distribution of the location of calls changes in time. We divide the city into 20×17 subregions and assume that the rate of call arrivals in subregion l at time t is given by

$$\Lambda_l(t) = \Lambda\big[\gamma_l + \beta_l \sin(2\pi t/24)\big],$$

where t is measured in hours. In the expression above, Λ, γ_l and β_l are fixed parameters that satisfy $\Lambda \geq 0$, $\gamma_l \geq |\beta_l|$, $\sum_{l=1}^{340} \gamma_l = 1$ and $\sum_{l=1}^{340} \beta_l = 0$. We have $\sum_{l=1}^{340} \gamma_l + \sum_{l=1}^{340} \beta_l \sin(2\pi t/24) = 1$ so that we can interpret Λ as the total call arrival rate into the system and $\gamma_l + \beta_l \sin(2\pi t/24)$ as the probability that a call arriving at time t falls in subregion l. If $\beta_l > 0$, then the peak call arrival rate in subregion l occurs at hours $\{6, 30, 54, \ldots\}$, whereas if $\beta_l < 0$, then the peak call arrival rate in subregion l occurs at hours $\{18, 42, 66, \ldots\}$. The average call arrival rate over a day in subregion l is $\Lambda\gamma_l$. We estimated Λ and γ_l using historical data and Λ came out to be about 4 calls per hour. We chose appropriate values of β_l so that we have higher call arrival rates in the business subregions early in the day and higher call arrival rates in the residential subregions later in the day.

2. The Second City The population of the second city is more than five times that of the city of Edmonton and its size is around 180×100 km^2. The EMS system includes up to 97 ambulances operating during peak times, 88 bases and 22 hospitals. The turn-out times, call-scene times and hospital-transfer times are comparable to those in Edmonton, but are chosen to be representative rather than realistic to protect confidentiality. The destination hospital for a call depends on the location of the call. Calls originating at a given location are transported to any of a small set of hospitals, usually no more than two or three out of the 22 hospitals in the system. The corresponding probabilities are inferred from historical data. The road network that we use models the actual network on the avenue level and there are 4,955 nodes and 11,876 arcs.

Our call arrival model is quite realistic. The data were collected from one year of operations of the EMS system and consist of aggregated counts of calls for each hour of the week and for each of 100×100 geographic zones in the city. Due to the irregular shape of the metropolitan area, roughly 80% of these zones have zero total call counts and do not intervene in the dynamics. From the remaining 20%, a significant number of zones have very low hourly counts of at most five calls. Therefore, it was necessary to apply a smoothing procedure for the lowest intensity zones so as to reduce the sampling noise. We classified the zones into a few groups according to their average intensity over the week. For the lowest intensity groups, we computed a total intensity for each hour and then distributed this total intensity uniformly among the zones in this group. In this way, we were able to obtain an intensity model that combined a uniform low intensity background with accurate counts on the highest intensity zones. In the end, the average call arrival rate is 570 calls per day and fluctuates on any day of the week from a low of around 300 calls per day to a high of around 750 calls per day. These figures represent a modest departure from the true figures to protect confidentiality.

For both data sets, the simulation horizon is 14 days. We use a discount factor of $\alpha = 0.8$, but our results are relatively insensitive to the choice of the discount factor as long as it is not too close to one. We initialize r^1 to zero in Step 1 of the approximate policy iteration algorithm and use $Q = 30$ replications in Step 3. A few setup runs indicated that setting $Q = 30$ provides a reasonable balance between computational burden and stable performance. We also tried using different values for κ in the fourth and sixth basis functions. We varied κ over the interval $[0, 3]$ and setting $\kappa = 0.1$ gave the best results for the city of Edmonton, whereas setting $\kappa = 1.6$ gave the best results for the second city. These setup runs also indicated that tuning the value of κ is quite important for obtaining good performance from our ADP approach.

5.2 Baseline Performance

The goal of our first set of computational experiments is to give a feel for how our ADP approach compares with several benchmark strategies. In Figure 1, we begin by showing the performance of our ADP approach on the city of Edmonton over 25 iterations. The horizontal axis in this figure gives the iteration number in the approximate policy iteration algorithm, whereas the vertical axis gives the expected percentage of calls not reached within the threshold response time. In other words, each data point in Figure 1 gives the expected percentage of calls missed by the greedy policy induced by the value function approximation at a particular iteration. We compute the expected percentage of missed calls by using the undiscounted numbers of calls. Since each iteration of the approximate policy iteration algorithm requires simulating the performance of the greedy policy for 30 replications, it is straightforward to estimate the expected percentage of missed calls at each iteration by using a sample average. From Figure 1, we observe that the apparent-best policy is obtained at the third iteration and this policy yields an expected percentage of missed calls of about 25.6%. To test the performance of this policy more carefully, we simulate its performance for an independent set of 400 replications. From these 400 replications, the expected percentage of missed calls is estimated to be $25.5\% \mp 0.1\%$, where $\mp 0.1\%$ is a 95% confidence interval.

We use two benchmark strategies. The first benchmark strategy is the myopic policy, which is obtained by letting $r_p = 0$ for all $p = 1, \ldots, P$ in (5) and using the greedy policy induced by this value function approximation. Step 1 of the approximate policy iteration algorithm initializes r^1 to zero, so the first data point Figure 1 naturally gives the performance of the myopic policy. The expected percentage of calls missed by the myopic policy is about 30.2%.

The second benchmark strategy that we use is the static policy, which preassigns a base to each ambulance and redeploys an ambulance back to its preassigned base whenever it becomes free after serving a call. We find a good static policy by simulating the performance of the system under a large number of possible base assignments and choosing the base assignment that gives the best performance. The horizontal line in Figure 1 shows the performance of the best static policy we found. The expected percentage of calls missed by the best static policy is $29.5\% \mp 0.1\%$.

The best policy obtained by our ADP approach improves on the myopic and static policies respectively by 4.7% and 4.0%. These improvements are obtained without adding any extra resources and EMS managers would be very interested in obtaining these kinds of improvements by simply using their existing resources more carefully. To put the improvement figures in perspective, a quick investigation into the data reveals that 18.6% of the overall call volume occurs in locations that are at least 8 minutes away from the ambulance bases. This implies that 18.6% of the calls would be missed

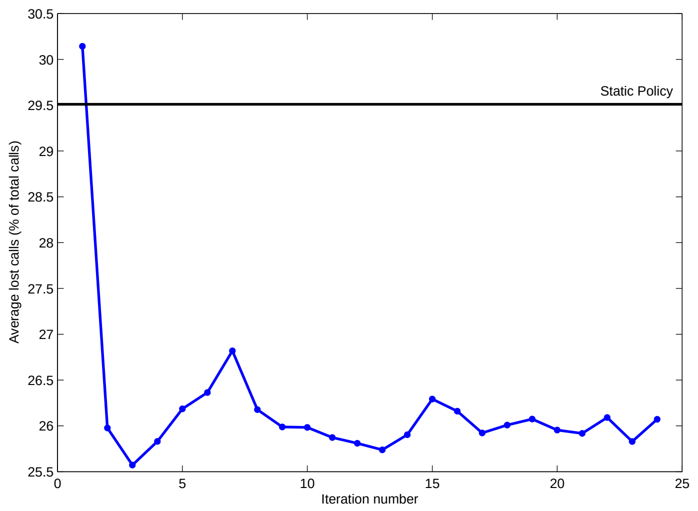

Figure 1: Performance of our ADP approach on the city of Edmonton.

even if there were always at least one ambulance available at every base. Naturally, this line of reasoning ignores the fact that a call may be served by an ambulance that is already on the road, but it provides a sense of a lower bound on what is achievable. Our ADP approach makes a significant step towards achieving this lower bound.

The CPU time for each iteration of our approximate policy iteration algorithm is 22 minutes. Such runtimes are acceptable given that we run the approximate policy iteration algorithm in an offline fashion to search for a good value function approximation. Once we have a good value function approximation, it takes about 45 milliseconds to make one redeployment decision by solving an optimization problem of the form (4). This CPU time includes enumerating over all feasible decisions and estimating the expectations through Monte Carlo samples, and is far faster than necessary for real-time operation.

Figure 2 shows the empirical cumulative distributions of the response times for the static (solid line) and ADP (dashed line) policies. Figure 2 indicates that our ADP approach not only decreases the expected percentage of missed calls, but it also shifts the entire distribution of call response times to the left. It is encouraging that the improvement in the expected percentage of missed calls is not obtained by letting a few calls wait for a very long time.

As mentioned at the end of Section 3, we use Monte Carlo simulation to estimate the expectation on the right side of (4). This requires simulating the trajectory of the system starting from time $\tau(s)$ until the next event to obtain a sample of $f(s, x, \omega(s, x))$. For certain states s, the next event after time $\tau(s)$ is always very close to time $\tau(s)$. For example, the residual travel time of an ambulance may be so small that the next event after time $\tau(s)$ is almost always the arrival of the ambulance at its destination. In this case, it is conceivable that $f(s, x, \omega(s, x))$ is not very sensitive to the action we choose and we lose our ability to differentiate the effectiveness of different redeployment decisions. To test the significance of this potential shortcoming, we tried simulating the trajectory of the system

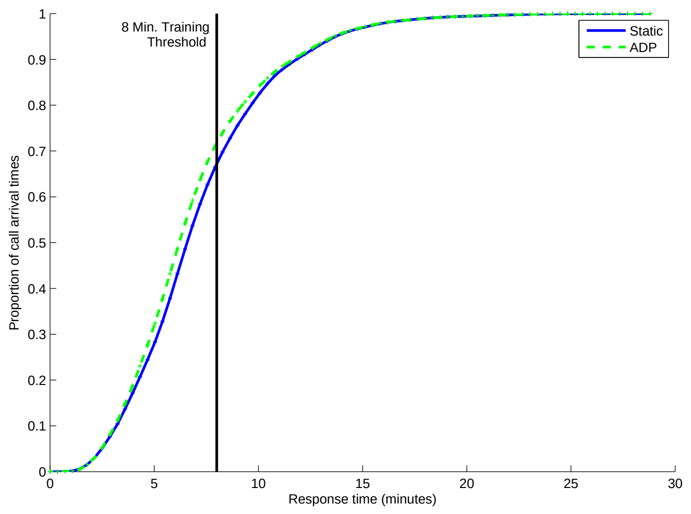

Figure 2: Empirical cumulative distribution of the response times for the city of Edmonton.

not until the next event, but until the time of the next redeployment decision. Figure 3 shows the performance of our ADP approach when we simulate the trajectory of the system until the next event (thick line) and until the time of the next redeployment decision (thin line). It appears that we do slightly better when we simulate until the next redeployment decision rather than until the next event. However, this also increases the CPU time by more than a factor of two. The performance gap between the two approaches is small, so we choose to simulate only until the next event.

Figure 4 shows the performance of our ADP approach on the second city. The observations from Figure 4 are very similar to those from Figure 1. The best policy obtained by our ADP approach misses 26.9% $\mp$ 0.1% of the calls, whereas the myopic and static policies respectively miss 29.3% and 28.8% $\mp$ 0.1% (as estimated through independent runs of 400 replications). The improvements are smaller for the second city than for Edmonton. Nevertheless, our ADP approach still improves on the static policy by about 2%.

5.3 Contributions of Different Basis Functions

Computation time can potentially be reduced if we can satisfactorily use a subset of the basis functions rather than all six of them. Therefore, it is natural to ask whether all six of the basis functions are really needed. We repeated the computational experiments described in the previous subsection by using only subsets of the basis functions. For the city of Edmonton, we are able to drop all but the fifth and sixth basis functions. By using only these two basis functions, our ADP approach identifies a policy that misses 25.4% $\mp$ 0.3% of the calls. Recall that the best policy obtained by our ADP approach with all of the six basis functions misses 25.5% $\mp$ 0.1% of the calls. Therefore, the performance of our ADP approach with only the fifth and sixth basis functions is pretty close to the performance with all of the six basis functions. Dropping either the fifth or sixth basis function provides policies that perform

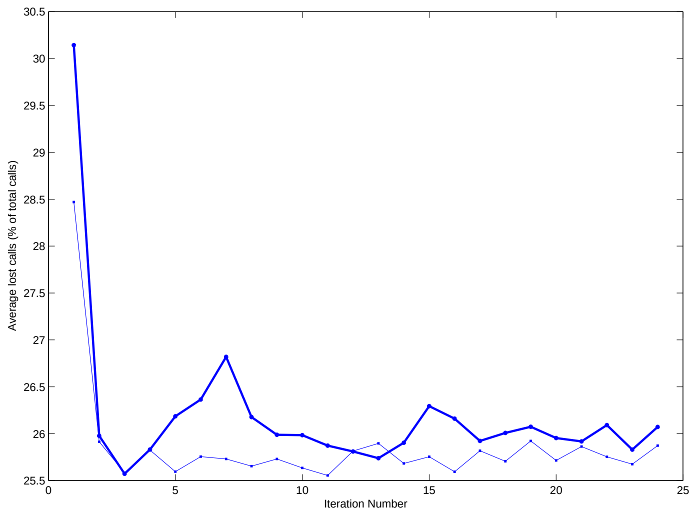

Figure 3: Performance of our ADP approach on the city of Edmonton when we simulate the trajectory of the system until the time of the next redeployment decision.

substantially worse than the myopic policy.

When we tried to carry out a similar set of experiments on the second city, the results were somewhat mixed. For example, when we dropped the first basis function, our ADP approach immediately settled on a sequence of policies that miss about 30.1% of the calls. Noting the computational complexity analysis at the end of Section 4, the computational burden for the fifth and sixth basis functions is at least as large as the computational burden for the others. Given that the fifth and sixth basis functions appear to be crucial for the success of our ADP approach, we decided to keep the other basis functions in our approximation architecture as well.

To conserve space, the remainder of this section reports on our computational results only for the city of Edmonton. We carried out similar computational experiments on the second city as well and we obtained very similar results.

5.4 Comparison with Random Search

For a fixed set of basis functions $\{\phi_p(\cdot) : p = 1, \ldots, P\}$, a set of values for the tunable parameters $r = \{r_p : p = 1, \ldots, P\}$ characterize a value function approximation $J(\cdot, r)$ and this value function approximation induces a greedy policy. Therefore, a brute-force approach for finding a good set of values for the tunable parameters is to carry out a random search over an appropriate subset of $\mathbb{R}^P$, and use simulation to test the performance of the greedy policies induced by the different sets of values for the tunable parameters.

To implement this idea, we first use our ADP approach to obtain a good set of values for the tunable parameters. Letting $\{\hat{r}_p : p = 1, \ldots, P\}$ be this set of values, we sample $r = \{r_p : p = 1, \ldots, P\}$

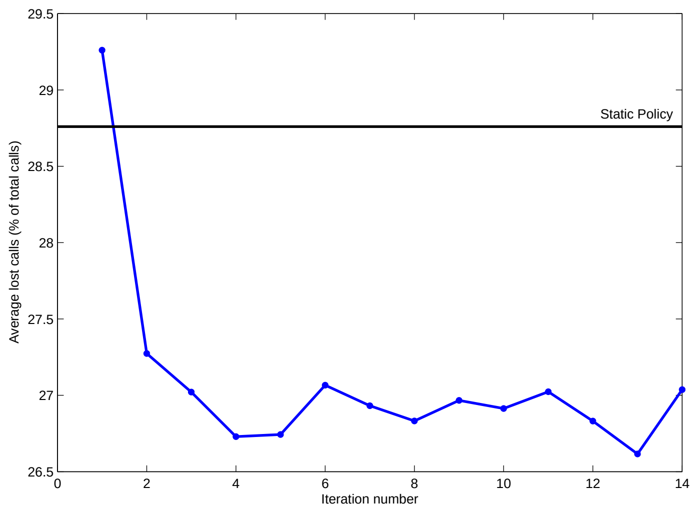

Figure 4: Performance of our ADP approach on the second city.

uniformly over the box $\left[\hat{r}_1 - \frac{1}{2}\,\hat{r}_1, \hat{r}_1 + \frac{1}{2}\,\hat{r}_1\right] \times \ldots \times \left[\hat{r}_P - \frac{1}{2}\,\hat{r}_P, \hat{r}_P + \frac{1}{2}\,\hat{r}_P\right]$ and use simulation to test the performance of the greedy policy induced by the value function approximation $J(\cdot, r)$. We sampled 1,000 sets of values for the tunable parameters and this, in turn, provides 1,000 value function approximations. Figure 5 gives a histogram for the expected percentage of calls missed by the greedy policies induced by these 1,000 value function approximations. The vertical lines correspond to the expected percentage of calls missed by the best policy obtained by our ADP approach and the static policy. The figure indicates that only 1.2% of the sampled sets of values for the tunable parameters provide better performance than the best policy obtained by our ADP approach. On the other hand, 42.3% of the samples provide better performance than the static policy.

The random search procedure we use is admittedly rudimentary and one could use more sophisticated techniques to focus on the more promising areas of the search space. Nevertheless, our results indicate that when one looks at a large number of possible values for the tunable parameters, ADP is quite effective in identifying good parameters. Furthermore, our search effort is focused on a neighborhood of the tuneable parameters that are identified by ADP. Therefore, the random search procedure is rather focused and it is remarkable that even an extensive local search can find very few policies that outperform the one found by ADP. Finally, we note that the computation time required by the random search procedure is on the order of several days, whereas we can carry out ADP in a few hours.

5.5 Additional Redeployments

The computational experiments up to this point allow redeployments only when an ambulance becomes free after serving a call. We now explore the possibility of improving performance by allowing additional ambulance redeployments. We define an extra event type "consider redeployment" and schedule an event of this type with a certain frequency that is detailed below. Whenever an event of this type is triggered,

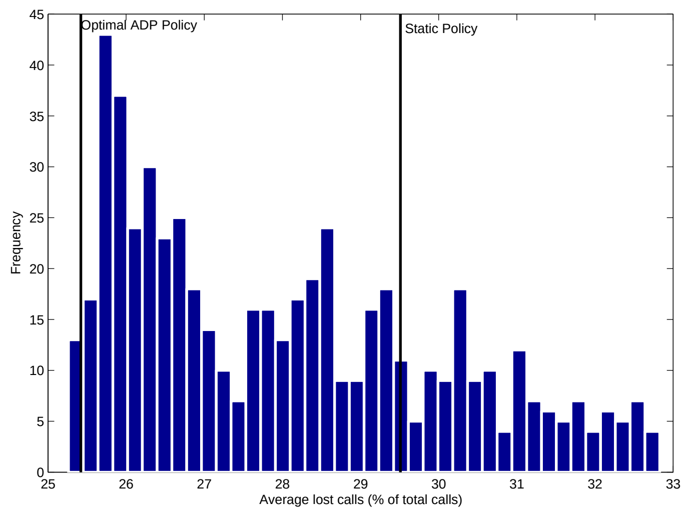

Figure 5: Performance of the 1,000 greedy policies obtained through random search.

we consider redeploying any ambulance that is either at a base or returning to a base, so that $\mathcal{R}(s)$ can contain multiple ambulances at such times. The set $\mathcal{R}(s)$ continues to be a singleton when $e(s)$ corresponds to an ambulance becoming free after serving a call, and at all other events, $\mathcal{R}(s) = \varnothing$.

We use two methods to vary the redeployment frequency. In the first method, we equally space consider-redeployment events to obtain frequencies between 0 and 10 per hour. In the second method, the frequency of consider-redeployment events is fixed at 30 per hour, but we make a redeployment only when the estimated benefit from making the redeployment exceeds the estimated benefit from not making the redeployment by a significant margin. More precisely, letting $\epsilon \in [0, 1)$ be a tolerance margin and using $\bar{0}(s)$ to denote the $|\mathcal{R}(s)| \times |\mathcal{B}|$ dimensional matrix of zeros corresponding to the decision matrix of not making a redeployment, if we have

$$\begin{aligned}\underset{x \in \mathcal{X}(s)}{\operatorname{argmin}} \Big\{ \mathbb{E}\Big[c(s, x, f(s, x, \omega(s, x))) + \alpha^{\tau(f(s,x,\omega(s,u)))-\tau(s)}\, J(f(s, x, \omega(s, x)), r) \Big] \Big\} \\ \leq (1-\epsilon)\, \mathbb{E}\Big[c(s, \bar{0}, f(s, x, \omega(s, \bar{0}))) + \alpha^{\tau(f(s,\bar{0},\omega(s,\bar{0})))-\tau(s)}\, J(f(s, \bar{0}, \omega(s, \bar{0})), r) \Big],\end{aligned}$$

then we make the redeployment decision indicated by the optimal solution to the problem on the left-hand side. Otherwise, we do not make a redeployment. Larger values of ϵ decrease the frequency of redeployments. We vary ϵ between 0.1 and 0.001.

Figure 6 shows the performance improvement obtained by the additional redeployments. The horizontal axis gives the frequency of the redeployments measured as the number of redeployments per ambulance per day. The vertical axis gives the percentage of missed calls. The solid (dashed) data series corresponds to the first (second) method of varying the redeployment frequency. Recall from Figure 1 that we miss 25.5% of the calls without making any additional redeployments. By making about six additional redeployments per ambulance per day, we can decrease the percentage of missed

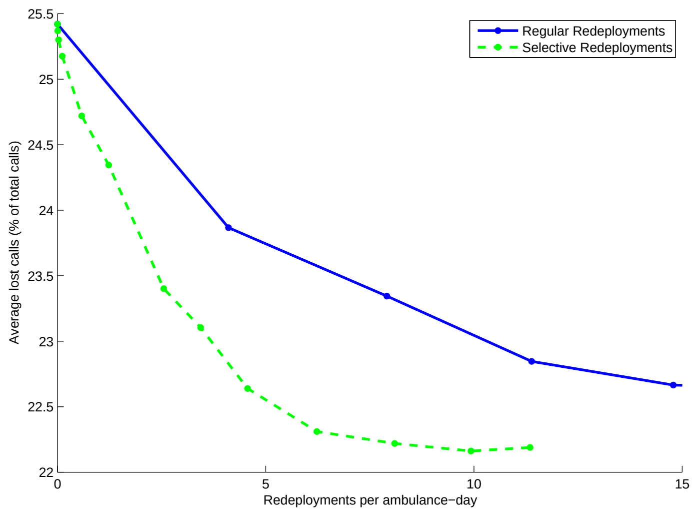

Figure 6: Performance of our ADP approach as a function of the frequency of additional redeployments.

calls to 22.3%. Beyond this range, we reach a plateau and additional redeployments do not provide much improvement. Another important observation is that the second method tends to provide significantly better performance improvements with the same frequency of additional redeployments. For example, the second method reduces the percentage of missed calls to 23.3% with three additional redeployments per ambulance per day, whereas the first method needs eight additional redeployments to reach the same level. Therefore, it appears that making redeployments only when the value function approximation signals a significant benefit is helpful in avoiding pointless redeployments.

5.6 Varying Fleet Sizes

In this section, we explore the effect of the fleet size on the performance of our ADP approach. Figure 7 summarizes our results. The horizontal axis in this figure gives the number of ambulances in the fleet, whereas the vertical axis gives the expected percentage of missed calls. For each fleet size, we find the best policy obtained by our ADP approach by rerunning the approximate policy iteration algorithm, and the best static policy by enumerating over a large number of possible base assignments. In Figure 7, the dashed data series correspond to our ADP approach, whereas the solid data series correspond to the static policy.

Our ADP approach performs consistently better than the static policy. The performance gaps between the two approaches diminish when there are too few or too many ambulances. Intuitively speaking, if the fleet size is small, then ambulances are always busy and there is little opportunity for repositioning. In this case, using a more intelligent repositioning strategy does not make much difference. On the other hand, if the fleet size is large, then there is almost always an available ambulance to respond to a call, and again, using a more intelligent repositioning strategy does not make much difference. Another observation from Figure 7 is that if our goal is to keep the percentage of missed calls below a

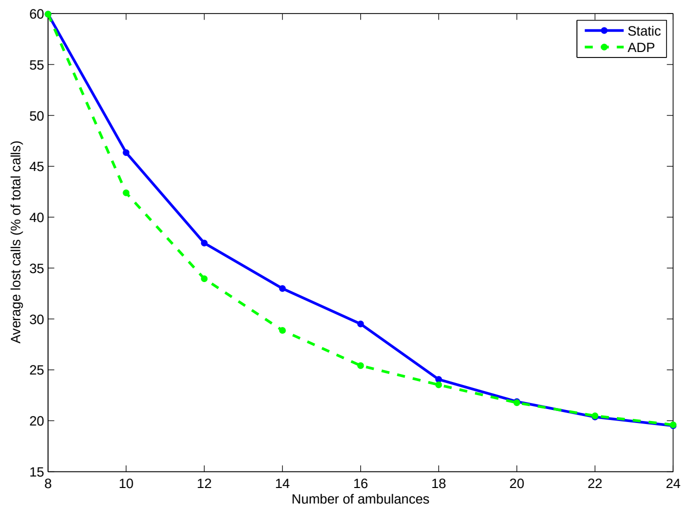

Figure 7: Performance of our ADP approach and the static policy for different fleet sizes.

given threshold, say 30%, then our ADP approach allows us to reach this goal with one or two fewer ambulances than the static policy. This translates into significant cost savings in an EMS system.

5.7 Varying Call Arrival Rates

In this section, we explore the sensitivity of the policies obtained by our ADP approach to changes in the call arrival rate. Recall that the original call arrival rate we used in Edmonton is about 4 calls per hour. Under this call arrival rate, we run the approximate policy iteration algorithm to find the best policy obtained by our ADP approach. We then vary the call arrival rate over the interval $[3.2, 4.8]$, but continue making the redeployment decisions by using the policy that was obtained under the call arrival rate of 4 per hour. Our goal is to show that the policy that was obtained under the call arrival rate of 4 per hour continues to provide good performance when we perturb the call arrival rates. As a benchmark strategy, we find the best static policy under the call arrival rate of 4 per hour and use this static policy as we vary the call arrival rate over the interval $[3.2, 4.8]$.

The results are summarized in Figure 8. The horizontal axis in this figure gives the call arrival rate, whereas the vertical axis gives the expected percentage of missed calls. The results indicate that our ADP approach performs consistently better than the benchmark policy. Also, the expected percentage of calls missed by the static policy under the call arrival rate of 3.2 per hour is still larger than the expected percentage of calls missed by our ADP approach under the call arrival rate of 4.8 per hour.

5.8 Effect of Turn-out Time

Recall that if the ambulance crew is stationed at a base when it is notified of a call, then it takes 45 seconds to get ready. This duration is referred to as the turn-out time. On the other hand, an ambulance crew that is already on the road does not incur turn-out time. A potential argument against

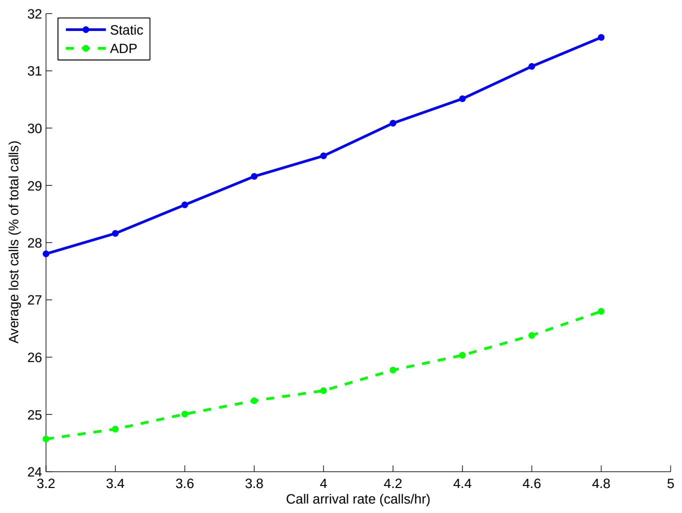

Figure 8: Performance of our ADP approach and the static policy for different call arrival rates.

ambulance redeployment is that any gains are simply due to ambulance crews being on the road more often, and therefore incurring less turn-out time delays.

To check the validity of this argument, we applied our ADP approach under the assumption that turn-out time is zero. In this case, the expected percentage of missed calls for our ADP approach turned out to be $21.08\% \mp 0.1\%$, whereas the expected percentage of calls missed by the static policy turned out to be $23.93\% \mp 0.1\%$. As expected both the ADP approach and the static policy perform better when turn-out time is zero. However, our ADP approach continues to provide practically significant improvements over the static policy. This indicates that the performance improvement of our ADP approach comes not only from incurring less turn-out time delays, but also from making more careful ambulance redeployment decisions.

6 Conclusions

In this paper, we formulated the ambulance redeployment problem as a dynamic program and used an approximate version of the policy iteration algorithm to deal with the high-dimensional and uncountable state space. Computational experiments on two realistic problem scenarios show that our ADP approach can provide high-quality redeployment policies. The basis functions that we construct open up the possibility of using other approaches, such as temporal-difference learning and the linear programming approach, to tune the parameters $\{r_p : p = 1, \ldots, P\}$.

Other future research will incorporate additional degrees of realism into our model. We plan to include stochastic travel times, multiple call priorities, other cost functions and more realistic ambulance dynamics that involve multiple ambulances serving certain calls. Incorporating these complexities may require constructing additional basis functions.

Acknowledgments

We thank the editors and two anonymous referees for their useful comments that substantially improved the paper. We thank Armann Ingolfsson and Andrew Mason for advice, support and data. This work was partially supported by National Science Foundation Grants DMI 0400287, DMI 0422133 and CMMI 0758441.

References

Adelman, D. (2004), 'A price-directed approach to stochastic inventory routing', *Operations Research* **52**(4), 499–514.

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

Adelman, D. and Mersereau, A. J. (2007), 'Relaxations of weakly coupled stochastic dynamic programs', *Operations Research* . to appear.

Andersson, T. (2005), Decision support tools for dynamic fleet management, PhD thesis, Department of Science and Technology, Linkoepings Universitet, Norrkoeping, Sweden.

Andersson, T. and Vaerband, P. (2007), 'Decision support tools for ambulance dispatch and relocation', *Journal of the Operational Research Society* **58**, 195–201.

Berman, O. (1981*a*), 'Dynamic repositioning of indistinguishable service units on transportation networks', *Transportation Science* **15**(2).

Berman, O. (1981*b*), 'Repositioning of distinguishable urban service units on networks', *Computers and Operations Research* **8**, 105–118.

Berman, O. (1981*c*), 'Repositioning of two distinguishable service vehicles on networks', *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-11**(3).

Bertsekas, D. and Shreve, S. (1978), *Stochastic Optimal Control: The Discrete Time Case.*, Academic Press, New York.

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

Brotcorne, L., Laporte, G. and Semet, F. (2003), 'Ambulance location and relocation models', *European Journal of Operations Research* **147**(3), 451–463.

Crites, R. H. and Barto, A. G. (1996), Improving elevator performance using reinforcement learning, *in* 'Advances in Neural Information Processing Systems 8', MIT Press, pp. 1017–1023.

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

Erkut, E., Ingolfsson, A. and Erdoğan, G. (2008), 'Ambulance deployment for maximum survival', *Naval Research Logistics* **55**, 42–58.

Farias, V. F. and Van Roy, B. (2004), Tetris: Experiments with the LP approach to approximate DP, Technical report, Stanford University.

Farias, V. F. and Van Roy, B. (2006*a*), *Probabilistic and Randomized Methods for Design Under Uncertainty*, Springer-Verlag, chapter Tetris: A Study of Randomized Constraint Sampling.

Farias, V. F. and Van Roy, B. (2006*b*), Tetris: A study of randomized constraint sampling, *in* G. Calafiore and F. Dabbene, eds, 'Probabilistic and Randomized Methods for Design Under Uncertainty', Springer-Verlag.

Farias, V. F. and Van Roy, B. (2007), An approximate dynamic programming approach to network revenue management, Technical report, Stanford University, Department of Electrical Engineering.

Gendreau, M., Laporte, G. and Semet, S. (2001), 'A dynamic model and parallel tabu search heuristic for real time ambulance relocation', *Parallel Computing* **27**, 1641–1653.

Gendreau, M., Laporte, G. and Semet, S. (2006), 'The maximal expected coverage relocation problem for emergency vehicles', *Journal of the Operational Research Society* **57**, 22–28.

Goldberg, J. B. (2007). Personal Communication.

Ingolfsson, A. (2006), 'The impact of ambulance system status management'. Presentation at 2006 INFORMS Conference.

Ingolfsson, A., Erkut, E. and Budge, S. (2003), 'Simulation of single start station for Edmonton EMS', *Journal of the Operational Research Society* **54**(7), 736–746.

Kolesar, P. and Walker, W. E. (1974), 'An algorithm for the dynamic relocation of fire companies', *Operations Research* **22**(2), 249–274.

Nair, R. and Miller-Hooks, E. (2006), 'A case study of ambulance location and relocation'. Presentation at 2006 INFORMS Conference.

Netlib (2004), 'Linear Algebra PACKage', `http://www.netlib.org/lapack/`.

Powell, W. B. (2007), *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, John Wiley & Sons, Hoboken, NJ.

Restrepo, M. (2008), Computational methods for static allocation and real-time redeployment of ambulances, PhD thesis, Cornell University, Ithaca NY, USA.

Restrepo, M., Henderson, S. G. and Topaloglu, H. (2008), 'Erlang loss models for the static deployment of ambulances', *Health Care Management Science* . To appear.

Schweitzer, P. and Seidmann, A. (1985), 'Generalized polynomial approximations in Markovian decision processes', *Journal of Mathematical Analysis and Applications* **110**, 568–582.

Si, J., Barto, A. G., Powell, W. B. and Wunsch II, D., eds (2004), *Handbook of Learning and Approximate Dynamic Programming*, Wiley-Interscience, Piscataway, NJ.

Singh, S. and Bertsekas, D. (1996), Reinforcement learning for dynamic channel allocation in cellular telephone systems, *in* 'Proceedings of Advances in Neural Information Processing Systems', pp. 974–980.

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

Topaloglu, H. and Powell, W. B. (2006), 'Dynamic programming approximations for stochastic, time-staged integer multicommodity flow problems', *INFORMS Journal on Computing* **18**(1), 31–42.

Tsitsiklis, J. N. (1994), 'Asynchronous stochastic approximation and Q-learning', *Machine Learning* **16**, 185–202.

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

Tsitsiklis, J. and Van Roy, B. (2001), 'Regression methods for pricing complex American-style options', *IEEE Transactions on Neural Networks* **12**(4), 694–703.

Van Roy, B., Bertsekas, D. P., Lee, Y. and Tsitsiklis, J. N. (1997), A neuro dynamic programming approach to retailer inventory management, *in* 'Proceedings of the IEEE Conference on Decision and Control'.

Watkins, C. J. C. H. and Dayan, P. (1992), 'Q-learning', *Machine Learning* **8**, 279–292.

Yan, X., Diaconis, P., Rusmevichientong, P. and Van Roy, B. (2005), ‘Solitaire: Man versus machine’, *Advances in Neural Information Processing Systems* **17**, 1553–1560.

Zhang, O., Mason, A. J. and Philpott, A. B. (2008), Simulation and optimisation for ambulance logistics and relocation. Presentation at the INFORMS 2008 Conference.