Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

APPROXIMATE DYNAMIC PROGRAMMING IN MULTI-SKILL CALL CENTERS

Ger Koole
Auke Pot

Department of Mathematics
De Boelelaan 1081a
Vrije Universiteit
Amsterdam, 1081HV, The Netherlands

ABSTRACT

We consider a multi-skill call center consisting of specialists and fully cross-trained agents. All traffic is inbound and there is a waiting queue for each skill type. Our objective is to obtain good call routing policies. In this paper we use the so-called policy iteration (PI) method. It is applied in the context of approximate dynamic programming (ADP). The standard PI method requires the exact value function, which is well known from dynamic programming. We remark that standard methods to obtain the value function suffer from the curse of dimensionality, i.e., the number of states grows exponentially with the size of the call center. Therefore, we replace the real value function by an approximation and we use techniques from ADP. The value function is approximated by simulating the system. We apply this method to our call routing problem, yielding very good results.

1 INTRODUCTION

In this paper we consider a multi-skill call center having specialists and generalists and skill- and group-dependent service rates. We define specialists as agents with exactly one skill and generalists as agents with all type of skills. Note that group-dependent service rates are realistic because often specialists work faster than generalists. In addition, different priorities are assigned to call types. A priority is implemented as a cost for holding a call in the queue during one unit of time. The holding costs for high priority calls are considered higher than for low priority calls. By minimizing these costs the queue length of jobs with a high priority will decrease and waiting times will get shorter. We are interested in the associated routing policies for calls. These policies dictate the assignment of calls to agents.

Good routing policies help call centers to reduce the number of calls in the system, which is a way to improve the service level. Consider a multi-skill call centers, with agents having different skill sets. It is common that at an arrival of a call different agents with different skill sets are available that can all handle this call. The decision about the agent that is chosen influences the long-run average number of jobs in the system.

Obtaining optimal routing policies for multi-skill call centers is a difficult problem to solve analytically. Hardly any structural results exist. Some are given in Chevalier, Shumsky, and Tabordon (2004) (in Appendix A). However, they consider a model without queues and without priorities. (They do consider group- and skill-dependent service rates.) Their results are difficult to extend to our model. Therefore, we focus on numerical methods.

The system is modeled as a Markov process and the determination of routing policies becomes a Markov decision problem. Dynamic programming (DP), see Puterman (1994), offers straightforward techniques to solve Markov decision problems, both analytically and numerically. Therefore it is also capable of handling dynamic routing in multi-server queueing systems numerically. However, a shortcoming of DP is the fact that high-dimensional systems such as most call centers are not tractable because the state space becomes too large. This is called the curse of dimensionality, see Bellman (1961). Examples with more than ten agents and two skills can not be solved successfully with the standard techniques such as value iteration and matrix multiplication (see Tijms (1986)). This paper addresses call centers with more agents and more skills. We explore the potential of approximate dynamic programming (ADP) to obtain results for these call centers. We will show that ADP is an efficient and effective way for obtaining dynamic routing policies. This method is able to solve routing policies of call centers that could not be solved before. The advantage of ADP is that it suffers less from the curse of dimensionality, because it allows us to traverse only a subset of all states. It has been shown several times that ADP reduces the complexity without much loss of performance and accuracy. It reduces both the computation time and the memory

usage. The last one becomes almost zero. The game Tetris is an example of a successful implementation and treated in Farias and Van Roy (1994). However also in many cases ADP was not a success, some are mentioned in de Farias and Van Roy (2004). There are several reasons for this, for example the computation times get too long or the value function has a structure that is too difficult to approximate. This will be explained at a later point in this paper, after the different techniques have been discussed. More references to the literature about ADP are also given at a later point in this paper.

The method used for the optimization is one-step policy improvement in conjunction with an approximation method, being discussed in Section 3.1. However we will already give a brief description and introduction. One-step policy improvement starts with calculating the value function, well-known from DP, under an initial policy. It is obtained by solving the so-called Poisson equation. The solution of this equation gives next to the value function also the average cost of the system, which will serve as a performance measure of the system. Next, one-step policy improvement is executed and yields an improved policy. The system performance will be measured under the new policy that results from the one-step policy improvement. Several methods will be discussed.

In the context of ADP, the solution of the Poisson equation is approximated by replacing the value function by a simple structure and fitting the structure as close as possible. The approximation of the value function is used for the one-step policy improvement. The technique that we consider for the approximation is presented in Section 4.

The structure of this paper is as follows. Section 2 describes the model and also discussses the control parameters and the dynamics. In section 3 the call center is modeled as a Markov Decision Process (MDP) and policy improvement is discussed. Section 4 treats ADP, including:

- a numerical analysis in order to find a structure for the value function such that policy improvement is most effective,
- the estimation of the coefficients of the approximation function by minimizing the Bellman-error
- and the numerical results for call centers with two and three skills.

Concluding remarks and plans for future research are given in Section 6.

Despite the fact that the number of skills in our numerical examples are moderate and that there is only one group of generalists, we believe that the results

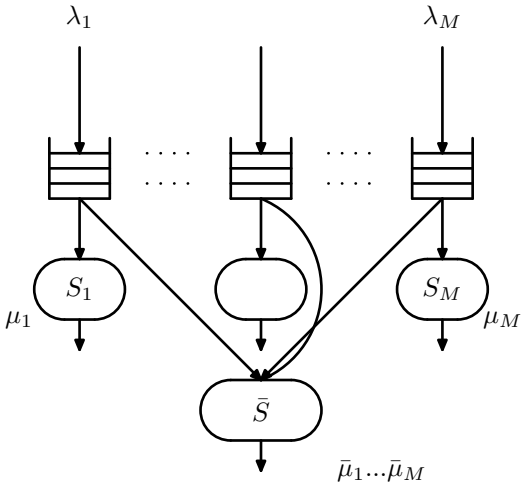

Figure 1: call center model

are still interesting. Chevalier, Shumsky, and Tabordon (2004) show that a bit of flexibility can improve the service level a lot. Their conclusion is that optimal staffing requires that 20 percent of all agents is cross-trained, the remainder is specialist. This shows that, despite our model includes flexibility by only one group of generalists, high performance can still be obtained. Concerning routing, they consider only a specific type of routing policies; calls are first assigned to specialists and only if all specialists are busy to generalists. They prove that this policy is optimal in blocking systems with specialists working faster than generalists and without call priorities. However, the queues and call priorities from our model make routing much more complicated. This paper proposes a method to obtain routing policies for these cases.

2 MODEL

The model, with M skills, is depicted in Figure 1. The jobs of each type arrive according to separate Poisson processes. There is one group of specialists for each type and one group of generalists, who have all M skills. Jobs of each skill type are served according to a first-in-first-out service discipline. Each job is served exactly once, either by a specialist or a generalist. This is determined by the routing policy and is discussed in Section 2.1.

The service times are exponentially distributed and the parameters of the groups with specialists are μ_i, $i \in \{1, 2, ..., M\}$. Generalists work successively on calls of different types, with parameter $\bar{\mu}_i$ in the case of a type i call. The service rates are next to skill-dependent also group-dependent because in reality specialists of-

ten work faster than generalists. But we do not make an assumption on this, according to the model also generalists are allowed to work faster than specialists.

The number of specialists in the group with skill i is denoted with S_i and $\bar{S}$ denotes the total number of generalists. The state space is defined such that it is possible that calls are directed to generalist while keeping specialists idle and vice versa. It consists of $2M$ components:

$$x := (x_1, x_2, ..., x_M, \bar{x}_1, \bar{x}_2, ..., \bar{x}_M)$$

x_i ($i \in \{1, 2, ..., M\}$) is the total number of jobs in the queue and the number of jobs served by specialists of group i. Variable $\bar{x}_i$ is the number of generalists that work on type i jobs. Because there are $\bar{S}$ generalists we have $\bar{x}_1 + \bar{x}_2 + ... + \bar{x}_M \leq \bar{S}$. According to the definition of x_i it is not possible to queue jobs while keeping specialists idle. The state space will be denoted by $\mathcal{X}$ and it contains all feasible states.

There are linear holding costs for each type: $h_i, i \in \{1, 2, ..., M\}$. The objective of our analysis is minimization of the long-run average holding costs of the waiting customers in each queue. The control parameters are described in the next section.

Our model is an extension of the one studied in Örmeci (2003), which treats the case with $M = 2$ without waiting rooms. Instead of holding costs there are different rewards for each type of call included. They show that under certain conditions the optimal policy of the assignment to generalists can be characterized as a monotonic treshold policy. In Örmeci (2003) no answer is given to the question if the optimal policy is work-conserving because a loss model is considered.

2.1 Policy

The performance of our call centers is controlled by a routing policy $\mathcal{R}$. It dictates the assignment of calls to agents and is state-dependent. We call the policy for a particular state an action $\mathcal{R} : x \to a$ and will write $a^x \equiv \mathcal{R}(x)$ when the meaning of R is clearly defined. Each action describes the transition at an arrival or service completion of a specialist or a generalist for each call type. Only the moments of a transition are relevant because the state is memoryless (remind that all distributions are exponential).

For simplicity we define the action set in such a way that at most one job from one queue is assigned to a generalist at the moment of a state transition. This yields an action vector that is composed of three elements. The different components will be discussed next.

The job assignment policy at an arrival of type i in state x is defined as

$$a_i^x := \begin{cases} 0 : \text{generalist} \\ 1 : \text{specialist or queued} \end{cases}.$$

The value 0 means that the job of type i is assigned to a generalist, resulting in a transition to state $x+\bar{e}_i$. If the action is 1 two possibities exist, depending on the state: the job is assigned to a specialist if one is available or queued. Sometimes only one action is possible, for example when $S_1 = 0$ or $\bar{x}_1 + \bar{x}_2 + ... + \bar{x}_M = \bar{S}$. To prevent that the system reaches illegal states, we always take

$$a_i^x = \begin{cases} 1 : \sum_i \bar{x}_i = \bar{S} \\ 0 : S_i = 0, \sum_i \bar{x}_i < \bar{S} \end{cases}.$$

At a service completion of a specialist from type i in state x there are two possibilities

$$s_i^x := \begin{cases} 0 : \text{idle} \\ 1 : \text{type } i \end{cases}.$$

The value 0 indicates idling and in the case of value 1 the specialist receives another job of type i. Note that if a specialist of type i finishes a job and the queue is not empty, the decision about the assignment of the first call in the queue is the same as when this call would arrive instantaneously.

At a service completion of a generalist in state x the generalist can undertake several actions

$$\bar{s}_i^x := \begin{cases} 0 : \text{idle} \\ j : \text{type } j \\ -1 : \text{random type 1,2,.. or } M \end{cases}.$$

This choice depends on the job type i of the finished call. We note that an optimal policy does not necessarily randomize. Randomization is supported because we need that the initial policy is allowed to randomize. In the first case (0) the generalist becomes idle and the new state is $x-\bar{e}_i$, meaning that a generalist that serves a type i call finishes. In the second case (j) a call from queue j is assigned, with a transition to $x-e_j-\bar{e}_i+\bar{e}_j$, meaning that the queue length of type j jobs becomes one job shorter, a generalist that serves a type i job finishes and a generalist starts to serve a job of type j. Finally, the value -1 means a random selection from the job types 1,2,...,M, with equal probability. A special case is

$$\bar{s}_i^x = \begin{cases} 0 : x_i = 0, \forall i \\ 0 : \bar{S} = 0 \end{cases}.$$

3 OPTIMIZATION

The dynamics of the system can be described using dynamic programming. An introduction to Markov Decision Processes is given in Puterman (1994). We refer

to this book for background information on dynamic programming and algorithms for policy improvement.

The value function for a fixed policy can be expressed as the solution of the Poisson equation

$$v(x) + g = c(x) + \sum_y p(x, a^x, y)v(y), \qquad (1)$$

with $p(x, a^x, y)$ the transition probability (after uniformization see Lippman (1975)) of going from state x to state y according to action a^x, $c(x)$ is de immediate cost of the transition, g the average cost and $v(x)$ the value of state x. In more detail, we have (after moving all terms of $v(x)$ to the left-hand side)

$$\begin{aligned}
&\sum_{i=1}^{M}(\lambda_i + \mu_i \min\{x_i, S_i\} + \bar{\mu}_i\bar{x}_i)v(x) + g = c(x) + \\
&\quad \sum_{i=1}^{M}(\lambda_i v(x + e_i)a_i^x + \lambda_i v(x + \bar{e}_i)(1 - a_1^x)) + \\
&\qquad \sum_{i=1}^{M}(\mu_i \min\{x_i, S_i\}v((x - e_i)^+) + \\
&\qquad\quad \bar{\mu}_i\bar{x}_i[v((x - \bar{e}_i)^+)\mathrm{I}\{\bar{s}_i^x = 0\} + \\
&\qquad \sum_{j=1}^{M} v((x - e_j - \bar{e}_i + \bar{e}_j)^+)\mathrm{I}\{\bar{s}_i^x = -1\}/M \\
&\qquad\quad \sum_{j=1}^{M} v((x - e_j - \bar{e}_i + \bar{e}_j)^+)\mathrm{I}\{\bar{s}_i^x = j\}])
\end{aligned}$$

with the cost function $c(x)$ defined as

$$c(x) := \sum_{i=1}^{M} h_i(x_i - S_i)^+,$$

i.e., the holding costs times the queue lengths.

A job of type i arrives with rate λ_i and is assigned to a specialist or a generalist according to action a_i. The group with specialists of type i finishes jobs with rate $\min\{x_i, S_i\}\mu_i$ when there are $\min\{x_i, S_i\}$ agents busy and $(S_i - x_i)^+$ agents idle. Generalists can work on both types of jobs. When they finish a job of type i, a decision has to be made about the next type of job, which is specified by action $\bar{s}_i$.

3.1 Policy Improvement

Before optimization the system is evaluated under an initial policy $\mathcal{R}^0$. The initial policy from the numerical examples is defined as follows. A call is first routed to a specialist. If all specialists of the call type are busy, then it is routed to a generalist. If all generalists are busy too, the call is queued. When a generalist completes a call, the next call is assigned randomly from the non-empty queues.

Given the initial value function v^0, the actions of the one-step improved policy are defined as:

$$a^x := \arg\min_a \{c(x) + \sum_y p(x, a, y)v^0(y)\} \qquad \forall x.$$

The right-hand side contains the relative values for each state if first the optimal action is chosen and next the system evolves according to the initial policy. We will discuss this in more detail for our model.

Concerning arrivals in state $\{x : S_i > 0, \bar{x}_1 + \bar{x}_2 + ... + \bar{x}_M < \bar{S}\}$ the optimal action a_i^x is not trivial, since an assignment to either a specialist or a generalist is possible. If $\mu_i > \bar{\mu}_i$ and the holding costs are equal, we expect that it is optimal to route the arriving call first to a specialist, as shown in Chevalier, Shumsky, and Tabordon (2004) for the blocking model. If no generalist is available the only possible action is to queue the job. We will also mention a few situations in which the routing problem becomes difficult. In the first place, in states that satisfy $x_i < S_i$, $\bar{x}_1 + \bar{x}_2 + ... + \bar{x}_M < \bar{S}$ and $\mu_i < \bar{\mu}_i$ for some i. Then specialist and generalists are available and generalists work faster. In the second place, in states that satisfy $\bar{x}_1 + \bar{x}_2 + ... + \bar{x}_M < \bar{S}$, $\mu_i > \bar{\mu}_i$ for multiple i and different holding costs or service rate per type. Then it is possible that the optimal policy is not work-conserving. The optimal action, with respect to the value function v, is determined according to

$$a_i^x = \begin{cases} 0 : v(x + \bar{e}_i) > v(x + e_i) \\ 1 : v(x + \bar{e}_i) \leq v(x + e_i) \end{cases}.$$

The action is chosen that gives the lowest future costs. The meaning of 0 and 1 was given in Section 2.1.

Concerning the service completions of generalists, the relevant states are $\{x : \bar{x}_i > 0\}$. We introduce

$$v(x, i, j) \equiv v(x - \bar{e}_i + (\bar{e}_j - e_j)) \qquad \forall \bar{x}_i, x_j > 0.$$

It is the value of the new state after x by the transition that a generalist finishes a type i job and starts serving a type j job. The optimal action is determined according to

$$s_i^x = \begin{cases} 0 : & v(x, i, 0) < v(x, i, j) & \forall j : x_j > S_j \\ j : & v(x, i, j) \leq v(x, i, k) & \forall k \in \{0, 1, ..., M\} \end{cases}$$

From a numerical analysis of realistic situations we conclude that if specialists work faster than generalists, then near optimal policies are possible by considering the following class of policies:

- Specialists are kept busy as much as possible.
- If the queue length exceeds a treshold, generalists start working. As a result, generalists are sometimes kept idle while there is work in at least one

queue, but not in all of them. (This type of propagation is also possible by reserving generalists for the most attractive job types, for example via a treshold on the number of busy generalists.)

- When the queue length increases, the priority of that type increases too (for stability).

We also investigated this by applying value iteration (Puterman 1994) to a few small call center instances. The case that generalists work faster than specialists is less realistic, but does occur in our numerical examples. We can expect that the policies become more complicated. But still, we observed that in all realistic cases giving priority to specialists above generalists yields almost optimal policies.

3.2 Performance Evaluation

Evaluating routing policies requires a numerical method. In our analysis the average cost is considered as the main performance measure. The policies of the smaller instances are analyzed by the power series algorithm (PSA). The PSA was introduced in Hooghiemstra, Keane, and van de Ree (1988), see Blanc (1987) for an overview. It is a numerical method to calculate the value function or the stationary probabilities. It considers the value function as a polynomial function of the arrival rates. This makes it possible to calculate the associated coefficients recursively. The examples that are not tractable, due to the curse of dimensionality, are evaluated with simulation.

4 APPROXIMATE DYNAMIC PROGRAMMING

Bertsekas and Tsitsiklis (1996) is essentially concerned with approximate dynamic programming methods. Our focus is on one of them, namely the Bellman-error method. Section 4.1 is devoted to the minimization of the Bellman error. This method requires that an approximation structure is found for v, which we denote by $\tilde{v}$. In Section 4.2 more insight is obtained in the structure of $\tilde{v}$. It is achieved by fitting $\tilde{v}$ to the real value function. As we already mentioned the state space is very big and not computationally tractable. We are forced to consider a finite number of states and to ignore the remaining states. Several ways are explored in Section 4.3. Finally, the algorithm is presented in Section 4.4.

4.1 Bellman-Error Minimization

Bertsekas and Tsitsiklis (1996) describe in chapter 6.10 a method for fitting the value function $\tilde{v}$. It will be described in this section. In summary, with this method we find an approximation for v as in Equation (1). This is accomplished by plugging in the structure from Equation (2) and determining the best coefficients.

In more detail, let us define the Bellman error as

$$D(x,r) := \tilde{v}(x,r) - c(x) + \tilde{g} - \sum_y p(x,a^x,y)\tilde{v}(y,r),$$

derived from Equation (1). Our goal is to minimize

$$\min_r \sum_x w(x) \left(D(x,r)\right)^2,$$

with $w(x)$ a weight that is used for fine-tuning and defined as

$$w(x) := \rho^{xe} \qquad \rho \in (0,1).$$

In our numerical analysis the unknown $\tilde{g}$, representing the average cost, is roughly estimated with a short simulation run. This is called a 2-phase method, see de Farias and Van Roy (2003).

The remainder of this section deals with the estimation of the unknown coefficients from Equation (2). The minimization is achieved with the Gauss-Newton method, based on linearizing $\tilde{v}(x,r)$ around r. In more detail we update r to r' according to

$$\begin{aligned} r' &:= r - \gamma w(x) \sum_x D(x,r) \nabla D(x,r) \\ &= r - \gamma w(x) \sum_x D(x,r) \\ &\quad \left(\sum_y p(x,a^x,y)[\nabla\tilde{v}(y,r) - \nabla\tilde{v}(x,r)] \right) \end{aligned}$$

with γ denoting the stepsize and $\nabla D(x,r)$ the gradient of $D(x,r)$ with respect to r.

This method is related to temporal-difference learning, see Tsitsiklis and Van Roy (1997). The main difference is that temporal-difference learning is an online procedure: the estimates of the unknown coefficients are updated while the system evolves. In contrast, ADP methods require that a sample of states is available in advance, the so-called set of representative states (see Section 4.3).

The expression $D(x,r)$ becomes zero when the structure of Equation (2) matches the real value function. As an exercise and a check, our implementation was verified by implementing ADP for the $M/M/1$ queue. This yielded very accurate results.

For the stepsize γ we take a simple rule. The parameter is initialized with a positive value and the algorithm starts. When no improvement is found or r is oscilating, γ is multiplied with a scalar between 0 and 1. The algorithm goes further until the stop criterium is satisfied again, and so forth. It stops when γ becomes smaller than a certain value or the maximum number of iterations is reached.

4.2 A Sufficient Structure for Approximating The Value Function

We aim to obtain a polynomial structure for the value function that is accurate enough for one step of policy improvement. This is analyzed by fitting the $\bar{v}$ to the real value function and measuring the mean-square error. We prefer a simple structure and consider a value function of the form, which is a *basis function* (see Bertsekas and Tsitsiklis (1996), Section 3.1.1)

$$\tilde{v}(x,r) = \sum_{i=1}^{M} \sum_{k=1}^{K} (r_i^{(k)} x_i^k + \bar{r}_i^{(k)} \bar{x}_i^k) \qquad (2)$$

with $r_i^{(k)}$ the unknown coefficient of order k and associated with variable x_i and $\bar{r}_i^{(k)}$ the unknown coefficient of order k and associated with variable $\bar{x}_i$. We omit more extended structures, for example with products of variables, to prevent the complexity from increasing and thereby resulting in very long computation times. Moreover, our conclusion is that the fit is already very accurate if we take $M = 2$. Alternatively, the variables x_i can be split into two parts: the queue length and the number of specialists that are busy. This would increase the number of basis variables from $2MK$ to $3MK$. Our experience is that, in conjunction with ADP, it does not yield significantly better results and, more important, the computation times get longer. We investigated this by considering a call center with zero specialists. In that case, x_i denotes the queue length.

4.3 Simulating the Set of Representative States

As we mentioned before there are infinitely many states. Even if we truncate the state space at a certain level, the number of states is huge.

ADP does not require that the sum of $D(x,r)$ is minimized over all states. With ADP we are allowed to choose a smaller set of states. It turned out that this is possible without much loss in accuracy. But the choice of representative states is of crucial importance for the performance of ADP. We experienced that to obtain good results it is not necessary to traverse the whole state space, which is inifitely big. Instead, we generate the set of representative states in two different ways:

- Sample path: By means of simulation a sample path is generated with q events. This is a random evolvement of the state during some time interval. The states that are traversed during the sample path are taken as the set of representative states. Every state is picked according to a Bernoulli distributed variable with parameter p. If the variable takes zero it is ignored. If it takes one, the state is included and during the algorithm the Bellman error is calculated for that state. The generated set of states is kept fixed during the algorithm. The algorithm in combination with this method is denoted as ADP1.

- Optimized: The goal is to compose a set of states such that after minimization of the Bellman error of these states a satisfying policy is found. Initially a very small set of representative states is composed randomly, containing only four or five elements. In the algorithm this set is optimized by removing and adding states iteratively, such that the number of elements stays the same. To determine the state to be removed we minimize the Bellman error of the remaining states and evaluate the one-step improved policy by simulation. The addition of a new state, which replaces the removed state, occurs similarly; a number of states is evaluated and the best state is selected. States are generated by means of simulation. We refer to this method by ADP2.

The composition of the set with representative states (the sample/realization) is crucial for success. Therefore we apply ADP with different seeds for the random number generator.

Remind that with ADP1 and ADP2 the set of representative states is composed by means of a simulation run. We define the level of a state as the total number of calls in the system. A property of the model is that states from either high level states or low level states are not frequently visited during these runs. We observed that the approximations are less accurate for these states and can yield bad actions with one-step policy improvement. Therefore, we introduce M^- as the lowest level of states with actions being changed according to the one-step improved policy and the highest level will also be denoted with M^+.

4.4 Algorithm

With the knowledge that we obtained so far we compose the algorithm. We start with the discussion of ADP1, consisting of the following steps:

1. Choose the initial policy and generate the set of representative states as described in Section 4.3. Suitable parameters are given in Section 5.
2. Determine the average cost by simulation.
3. Apply ADP conform Section 4.1.
4. Execute one-step policy improvement, see Section 3.1. Go to step 1 if the outcome is not satisfying.
5. Fine-tune M^- and M^+ afterwards by simulation and try different weights for the states by changing ρ

such that the average cost is further minimized.

ADP2 integrates ADP1 with a method to optimize the set of representative states, which is achieved by local search. Details are already given in Section 4.3. The difference with ADP1 is in the first place a smaller set of representative states and in the second place an extra loop in order to optimize this set.

Each of the algorithms yields a one-step improved policy. Note that instead of calculating and storing the policy explicitly, it can online be derived from $\tilde{v}$.

5 NUMERICAL EXAMPLES

In this section we discuss the numerical examples, categorized into two- and three-skilled call centers. All of them are solved by minimizing the Bellman error and using the gradient method that we described.

In Table 1 results are presented of 6 instances with two skills and three agent groups ($M = 2$). Each column corresponds with one call center instance. The rows are grouped together: the upper block contains the model parameters and the lower part presents the performance measures. If we look at the second column from the left (instance 1), we read that under the initial policy the average cost is 7.8, against 3.6 under the optimal policy. With ADP1 we obtain a policy with an average of 4.1 per unit of time and with ADP2 an average of 3.7.

Table 2 shows the results of three call centers with three skills and four agent groups ($M = 3$). With respect to the initial policy, ADP reduces the average cost with almost fifty percent. Because the state spaces of these examples are huge, it is not possible to obtain and evaluate the optimal policies.

Some parameters are kept constant. With value iteration the state space is truncated at level 125. Concerning the gradient method, the initial stepsize γ is 0.2 and the scalar is 1.15^{-1}. Parameter K is fixed at 2. With ADP1 the values $p = 0.00005$ (the parameter of the Bernoulli distribution) and $q = 2500000$ events gave satisfying results.

Several runs are executed for each example, until the long-run average cost is about 50 percent of the average cost that corresponds with the initial policy. Remind that the resulting policies are evaluated by means of simulation or by using the power series algorithm. Our experience is that the outcomes per run vary a lot and are often even worse than under the initial policy. On average about ten runs were necessary with ADP1 to find the results presented in the table. The computation times are several minutes per instance, consisting of multiple runs.

In Section 3.1 we listed a number of properties that describe the structure of an optimal policy. To validate the quality of the policies found with ADP, we compose manually, by trial and error and using these rules, dynamic routing policies. The quality of the resulting policies was above our expectation. They were almost as good as the results obtained with ADP1.

Table 1: Examples with 2 Skills

	Examples					
	1	2	3	4	5	6
h_1	.5	2	2	2	.5	.5
h_2	2	.5	.5	.2	2	2
λ_1	2.5	1.5	1.8	1.6	4.3	7
λ_2	2.5	1.5	1.8	1.6	4.3	7
μ_1	1	.5	.25	.25	.5	.5
μ_2	1	.5	.25	.25	.5	.5
$\bar{\mu}_1$	.25	.4	.3	.3	.13	.13
$\bar{\mu}_2$	1.8	.6	.2	.2	.38	.38
S_1	2	1	2	6	7	11
S_2	2	1	2	6	7	11
$\bar{S}$	4	6	12	4	14	25
g^0	7.8	1.89	7.4	2.20	5.6	4.7
$\tilde{g}^1$ (adp1)	4.1	1.17	3.8	1.38	3.5	3.0
$\tilde{g}^1$ (adp2)	3.7	1.16	3.8	1.30	3.2	2.4
g (opt)	3.6	1.15	3.6	1.18	2.9	> 2.2

Table 2: Examples with 3 Skills

	Examples		
	1	2	3
$(h_1,..,h_3)$	(.5,2,.5)	(.5,2,.5)	(2,.2,1)
$(\lambda_1,..,\lambda_3)$	(2.5,..,2.5)	(4.3,..,4.3)	(1.6,..,1.6)
$(\mu_1,..,\mu_3)$	(.5,..,.5)	(.5,..,.5)	(.25,..,.25)
$(\bar{\mu}_1,..,\bar{\mu}_3)$	(.13,.9,.13)	(.13,.38,.13)	(.3,.2,.15)
$(S_1,..,S_3)$	(4,..,4)	(7,..,7)	(6,5,4)
$\bar{S}$	12	21	8
g^0	14.0	10.5	14.7
$\tilde{g}^1$ (adp1)	7.0	5.5	8.1
$\tilde{g}^1$ (adp2)	6.6	5.2	8.5

6 CONCLUDING REMARKS AND FUTURE WORK

Methods for analyzing multi-skill call centers suffer from the curse of dimensionality. With approximate dynamic programming the curse of dimensionality does not give any trouble. Dynamic policies are obtained for bigger call centers than was possible before.

Using basic functions of order two (see Section 4.2) is sufficient to obtain numerically near-optimal policies. This was investigated by fitting second-order polynomials to the real value function. However, according to

our experience, the minimization of the Bellman error yields approximations that are less accurate than the one obtained with the real value function. Thus, there exists second-order polynomials that approach the real value function closer than the one we obtained with ADP. It shows that there is space left over for further improvements.

We experienced that the choice/composition of representative states is a crucial factor for success. It is in our opinion interesting for further research, to understand the relation between the choice of representative states and the accuracy of the fit better. This could eventually result in a smaller set of representative states, a lower number of 'trials' and reduce the computation times significantly, by which it would become easier to apply ADP to bigger call centers.

From our experiments it follows that generating sample paths by simulation is an effective way to compose the set of representative states, denoted as ADP1. We run the algorithm several times with different sets of representative states. This is necessary for obtaining good policies because the policies are very sensitive for the composition of the set with representative states.

ADP2 is a totally different approach: the set of representative states is optimized by local search, such that the quality of the policies is maximized. The results are good. In comparison to the initial policy, which assigns calls first to specialists, the weighted-average queue-lengths are reduced with about 50 percent.

The quality of the policies with ADP1 are satisfying, but not good. We experienced that it is not difficult to compose routing policies manually such that the performance is almost as good as with ADP1. We conclude that the algorithm ADP2 is superior to ADP1. Within a reasonable amount of time we solved the examples close to optimality.

In future work we will extend the analysis to the general setting of multi-skill routing in inbound call centers. There are several issues to start with. In the first place, the performance depends to some extend on the type of gradient method. It seems to be worth to study the literature to find more effective methods that reduce the computation times, which may be desired for the analysis of bigger call center. In the second place, another direction is to improve the structure of the basis function or to change its definition by adding more variables. Finally, the state space could be divided into multiple regions and separate functions could be fitted for each region, see for example Poupart, Patrascu, and Schuurmans (2002).

REFERENCES

Bellman, R. 1961. *Adaptive control processes: A guided tour*. Princeton University Press.

Bertsekas, D., and J. Tsitsiklis. 1996. *Neuro-dynamic programming*. Athena Scientific.

Blanc, J. 1987. On a numerical method for calculating state probabilities for queueing systems with more than one waiting line. *Journal of Computational and Applied Mathematics* 20:119–125.

Chevalier, P., R. Shumsky, and N. Tabordon. 2004. Routing and staffing in large call centers with specialized and fully flexible servers. Submitted to Manufacturing and Service Operations Management.

de Farias, D., and B. Van Roy. 2003. Approximate linear programming for average-cost dynamic programming. In *Advances in Neural Information Processing Systems 15*: MIT Press.

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

Farias, V., and B. Van Roy. 1994. Tetris: A study of randomized constraint sampling. to appear in Probabilistic and Randomized Methods for Design Under Uncertainty.

Hooghiemstra, G., M. Keane, and S. van de Ree. 1988. Power series for stationary distributions of coupled processor models. *SIAM Journal on Applied Mathematics* 48:1159–1166.

Lippman, S. 1975. Applying a new device in the optimization of exponential queueing systems. *Operations Research* 23:687–710.

Örmeci, E. 2003. Dynamic admission control in a call center with one shared and two dedicated service facilities. unpublished.

Poupart, P., C. Patrascu, and D. Schuurmans. 2002. Piecewise linear value function approximation for factored MDPs. In *Proceedings of the Eighteenth National Conference on AI*, 292–299.

Puterman, M. 1994. *Markov decision processes*. Wiley.

Tijms, H. 1986. *Stochastic models. an algorithmic approach*. Wiley.

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

AUTHOR BIOGRAPHIES

GER KOOLE is a professor in the Department of Mathematics at the Vrije Universiteit Amsterdam. He graduated in 1992 from Leiden University, and held postdoc position at CWI (Amsterdam) and INRIA

(Sophia Antipolis). He is interested in the theory and applications of controlled queueing systems. His email address is <koole@few.vu.nl>, and his web page is <http://www.math.vu.nl/~koole>.

AUKE POT is a PhD student at the Vrije Universiteit Amsterdam. He is working on call center planning, especially skill-based routing. His e-mail address is <sapot@few.vu.nl>, and his web page is <http://www.math.vu.nl/~sapot>.