Suboptimal dynamic programming with error bounds

Lincoln, Bo; Rantzer, Anders

Published in:
Proceedings of the 41st IEEE Conference on Decision and Control, 2002

DOI:
10.1109/CDC.2002.1184885

2002

Proceedings of the 41st IEEE
Conference on Decision and Control
Las Vegas, Nevada USA, December 2002

Suboptimal dynamic programming with error bounds

Bo Lincoln, Anders Rantzer

Department of Automatic Control, LTH
Box 118, 221 00 Lund, Sweden
{ lincoln | rantzer } @ control.lth.se

Abstract

This paper presents a method to relax Dynamic Programming. The method makes it possible to find suboptimal solutions with known error bounds to hard problems. The bounds are chosen by the user, who can then effectively trade-off between solution time and accuracy. Several examples from different domains where the method is highly useful are presented.

1. Introduction

For many optimal control problems, Dynamic Programming could have been used to solve the problem if there had been an efficient way to parameterize the value function V^*. Unfortunately, this is most often not the case. This paper presents a way of relaxing Dynamic Programming (DP) slightly to make it possible to find a more easily parameterized suboptimal value function V^α within a strict, user-defined, distance α from the optimal V^*.

The paper will briefly present the method, and then focus on a couple of optimal control problems where it is applicable.

2. Problem formulation

Let $x(n) \in X$ be the state of some system at time n. $u(n) \in U$ is the control decision. Given these, the state evolves as

$$x(n+1) = f(x(n), u(n)) \tag{1}$$

Given a time-additive cost function

$$J(n) = \sum_{k=0}^{N} \varphi(x(k), u(k)). \tag{2}$$

We would like to find an optimal policy $u(n) = \mu(x, n)$, such that the cost J is minimized from any initial state at any initial time n. The resulting cost (value function) from time n to N is denoted $V^*(x, n)$. In this setting, DP can be used to find $V^*(x, n)$ and $\mu(x, n)$.

$$V^*(x, n) = \min_u \Big\{ V^*(f(x, u), n) + \varphi(x, u) \Big\} \tag{3}$$

For a reference on DP, see e.g. [2].

For most problems, though, DP is not usable as the value function cannot be parameterized or described in an efficient manner. In this paper we focus on problems where V^* has a finite description which grows rapidly (typically exponentially) is size with N. The contribution of the paper is a simple algorithm to find a sub-optimal value function V^α which is within a user-definable distance α from the true V^*. Using this method, some problems for which DP has been considered hopeless can now be practically solved.

The method was introduced for a very specific problem in [6]. This paper generalizes the idea and presents some other types of problems where the method is very usable.

3. Suboptimal dynamic programming

Let $V^*(x, n+1)$ have a finite parameterization. The $V^*(x, n)$ is calculated using (3). Except for some special cases (like quadratic cost and linear dynamics), this value function at n will generally have a larger parameterization than at $n+1$.

The aim of this method is to find a more easily parameterized $V^\alpha(x, n)$ which fulfills

$$V^*(x, n) \le V^\alpha(x, n) \le V^{\alpha*}(x, n) \tag{4}$$

where $V^{\alpha*}(x, n)$ corresponds to the is a slightly "larger" step cost $\varphi^\alpha(x, u)$ such as e.g.

$$\varphi^\alpha(x, u) = \begin{cases} \varphi(x, u) + \alpha & (*) \text{ or} \\ \alpha\varphi(x, u) & (**) \end{cases} \tag{5}$$

and

$$V^{\alpha*}(x, n) = \min_u \sum_{k=n}^{N} \varphi^\alpha(x(k), u(k)). \tag{6}$$

From the two choices of φ^α above, having found a $V^\alpha(x, n)$, we get the following bounds on (the unknown) $V^*(x, n)$

$$\frac{1}{\alpha} V^\alpha(x, n) \le V^*(x, n) \le V^\alpha(x, n) \tag{7}$$

0-7803-7516-5/02/$17.00 ©2002 IEEE

for $(*)$ and

$$V^{\alpha}(x,n) - N\alpha \leq V^{*}(x,n) \leq V^{\alpha}(x,n) \qquad (8)$$

for $(**)$.

3.1 Finding V^{α}

Assume we have a $V^{\alpha}(x,n)$ which satisfies (4). Upper and lower bounds on $V^{\alpha}(x,n-1)$ are calculated as

$$\underline{V}(x,n-1) = \min_{u}\Big\{V^{\alpha}(f(x,u),n) + \varphi(x,u)\Big\} \qquad (9)$$

and

$$\overline{V}(x,n-1) = \min_{u}\Big\{V^{\alpha}(f(x,u),n) + \varphi^{\alpha}(x,u)\Big\}. \qquad (10)$$

From this, $V^{\alpha}(x,n-1)$ is calculated as a simplified version of either $\overline{V}(x,n-1)$ or $\underline{V}(x,n-1)$ which satisfies

$$\underline{V}(x,n-1) \leq V^{\alpha}(x,n-1) \leq \overline{V}(x,n-1) \qquad (11)$$

By construction, also $V^{\alpha}(x,n-1)$ satisfies

$$V^{*}(x,n-1) \leq V^{\alpha}(x,n-1) \leq V^{\alpha*}(x,n-1) \qquad (12)$$

and thus the procedure can be repeated. Starting from $V^{\alpha}(x,N) = 0$ and iterating the above procedure produces a set $V^{\alpha}(x,n)$, $n \in [0,N]$ which satisfies (4) by induction.

3.2 The role of α

α and (and φ^{α}) is chosen by the user as a trade-off between time- and memory constraints and the bound on the optimal solution V^{*}.

4. Problem 1: Linear dynamics, piecewise linear cost

This section will describe an optimal control problem where the method in Section 3 can be applied. The plant to be controlled is an LTI system, and the cost to be minimized is piecewise linear. This makes it possible to create more elaborate cost functions than the usual quadratic cost. For example, it is possible to make the cost asymmetric such that negative states are more costly than positive.

4.1 Problem formulation

The controlled system is LTI

$$x(n+1) = Ax(n) + Bu(n) \qquad (13)$$

where $x \in X$, $u \in U$ and X and U are polyhedra. The cost function is on the form (2), with

$$\varphi(x,u) = \max_{q \in Q} q^{T} \begin{pmatrix} x \\ u \\ 1 \end{pmatrix} \qquad (14)$$

where q is a vector and Q is a finite set of vectors. $\varphi(x,u)$ is thus piecewise linear and convex. The goal of the controller is to minimize the cost (2).

4.2 Value function

Assume the value function $V(x,n+1)$ at some time $n+1$ is on the form

$$V(x,n+1) = \max_{k \in \kappa(n+1)} k^{T} \begin{pmatrix} x \\ 1 \end{pmatrix}. \qquad (15)$$

Bellman's equation is used to calculate $V(x,n)$

$$V(x,n) = \min_{u \in U}\left\{ \max_{k \in \kappa(n+1)} k^{T} \begin{pmatrix} Ax+Bu \\ 1 \end{pmatrix} + \max_{q \in Q} q^{T} \begin{pmatrix} x \\ u \\ 1 \end{pmatrix} \right\} = \min_{u \in U} \max_{r \in R}\Big\{F_r^{T}x + G_r^{T}u + H_r\Big\}. \qquad (16)$$

where R is an index set. The value function can be rewritten as the linear program

$$V(x,n) = \min_{u,f} f \quad \text{s.t} \quad F_r^{T}x + G_r^{T}u + H_r \leq f \quad \forall r \in R$$

where the $x \in X$ and $u \in U$ constraints have been removed for clarity (they can easily be added). Solving this for any initial position x yields a value function that can again be written on the form (15). The solution can either be found by solving a *Multi-Parametric Linear Programing* (MPLP) problem (see e.g. [3]), or by doing explicit enumeration of the dual extreme points as is shown below.

Introducing Lagrange multipliers λ, we rewrite it as

$$\begin{aligned} V(x,n) = & \\ & \min_{u,f} \max_{\lambda_i \geq 0}\Big\{f + \sum_{r \in R} \lambda_r\big(F_r^{T}x + G_r^{T}u + H_r - f\big)\Big\} \\ & = \max_{\lambda_i \geq 0} \min_{u,f}\Big\{\big(1 - \sum_{r \in R}\lambda_r\big)f + \sum_{r \in R}\lambda_r\big(F_r^{T}x + H_r\big) + \\ & \qquad \sum_{r \in R}\lambda_r G_r^{T}u\Big\} \end{aligned} \qquad (17)$$

where the last equality is due to strong duality for linear problems. From this we see that

$$\sum_{r \in R}\lambda_r = 1 \qquad (18)$$

$$\sum_{r \in R}\lambda_r G_r^{T} = 0 \qquad (19)$$

and

$$V(x,n) = \max_{\lambda \in L} \sum_{r \in R} \lambda_r (F_r^T x + H_r) \qquad (20)$$

where the set L is defined by $\lambda_r \geq 0$ and (18) and (19).

As $V(x,n)$ is linear in λ, the maximum can be found in one of the extreme points of L. L is a $|R|-2$ dimensional set bounded by $|R|$ hyperplanes, and thus we can find at most $|R|(|R|-2)/2$ extreme points. This can be done by selecting all pairs of $\{(i,j)|i \neq j \in R\}$, setting all other $\lambda_r = 0$, $r \neq i$, $r \neq j$. If λ_i and λ_j have positive solutions in (18) and (19), the resulting λ is an extreme point in L. Note that the extreme points do not depend on the state x.

The extreme points of L form the new set $\kappa(n)$, and again the value function is on the form

$$V(x,n) = \max_{k \in \kappa(n)} k^T \begin{pmatrix} x \\ 1 \end{pmatrix}. \qquad (21)$$

4.3 Parsimonious representation

$V(x,n)$ can usually be represented by a much smaller set than the $\kappa(n)$ obtained by the above expansion. A set κ is called a *parsimonious* representation, if only the members of $\kappa(n)$ which are ever active (maximum for some state) are included. Such a set can be obtained from Procedure 1.

PROCEDURE 1—COST PRUNING

1. Let $\kappa^{\text{pars}}(n) = \{-\infty\}$.
2. Pick one $k \in \kappa(n)$. Find a state x where $k^T \begin{pmatrix} x \\ 1 \end{pmatrix} > p^T \begin{pmatrix} x \\ 1 \end{pmatrix}$, $\forall p \in \kappa^{\text{pars}}(n)$.
3. If such a x exists, find the $p \in \kappa(n)$ with the largest reward $p^T \begin{pmatrix} x \\ 1 \end{pmatrix}$.
   Add p to $\kappa^{\text{pars}}(n)$.
   Remove p from $\kappa(n)$.
4. If no such x exists, remove k from $\kappa(n)$.
5. Repeat from 2 until $\kappa(n)$ is empty.

□

Note that after this procedure, naturally

$$\max_{k \in \kappa(n)} k^T \begin{pmatrix} x \\ 1 \end{pmatrix} = \max_{k \in \kappa^{\text{pars}}(n)} k^T \begin{pmatrix} x \\ 1 \end{pmatrix} \quad \forall x. \qquad (22)$$

4.4 α-optimal value function

To be able to use the proposed suboptimal DP, we define a discounted cost

$$\varphi^\alpha(x,u) = \alpha_1 \varphi(x,u) - \alpha_2 \qquad (23)$$

where $\alpha_1 \leq 1$ and $\alpha_2 \geq 0$. Choosing $\alpha_1 < 1$ is of course only meaningful if $\varphi(x,u) \geq 0, \forall x,u$. Note that this cost is smaller than $\varphi(x,y)$ contrary to (5).

The goal for our α-optimal algorithm is to find a smaller set $\kappa^\alpha(n) \subseteq \kappa^{\text{pars}}(n)$ with corresponding value function $V^\alpha(n)$ for every n, while guaranteeing that

$$V^*(x,n) \geq V^\alpha(x,n) \geq V^{\alpha *}(x,n) = \alpha_1 V^*(x,n) - (N-n)\alpha_2 \qquad (24)$$

(note that this is (4) with opposite inequalities). The procedure in Section 3.1 is can be used to do this.

Assume $V^\alpha(x,n+1)$ satisfies (24) for $n+1$. Doing backward expansion the upper and lower bounds

$$\max_{k \in \overline{\kappa}(n)} k^T \begin{pmatrix} x \\ 1 \end{pmatrix} = \min_u \Big\{ V^\alpha(f(x,u),n) + \varphi(x,u) \Big\}. \qquad (25)$$

and

$$\max_{k \in \underline{\kappa}(n)} k^T \begin{pmatrix} x \\ 1 \end{pmatrix} = \min_u \Big\{ V^\alpha(f(x,u),n) + \varphi^\alpha(x,u) \Big\}. \qquad (26)$$

are calculated.

Now, the pruning of $\overline{\kappa}(n)$ to $\kappa^\alpha(n)$ is done by replacing the the cost pruning procedure by Procedure 2.

PROCEDURE 2—COST α PRUNING

1. Let $\kappa^\alpha(n) = \{-\infty\}$.
2. Pick one $k \in \underline{\kappa}(n)$. Find a state x where $k^T \begin{pmatrix} x \\ 1 \end{pmatrix} > p^T \begin{pmatrix} x \\ 1 \end{pmatrix}$, $\forall p \in \kappa^\alpha(n)$.
3. If such a x exists, find the $p \in \overline{\kappa}(n)$ with the largest reward $p^T \begin{pmatrix} x \\ 1 \end{pmatrix}$.
   Add p to $\kappa^\alpha(n)$.
4. If no such x exists, remove k from $\underline{\kappa}(n)$.
5. Repeat from 2 until $\underline{\kappa}(n)$ is empty.

□

Having found the corresponding $V^\alpha(x,n)$, the true value function can be bounded by

$$V^\alpha(x,n) \leq V^*(x,n) \leq \frac{1}{\alpha_1}(V^\alpha(x,n) + (N-n)\alpha_2). \qquad (27)$$

4.5 Example

In this example a controller for a linearized double water tank process is constructed. The process (see

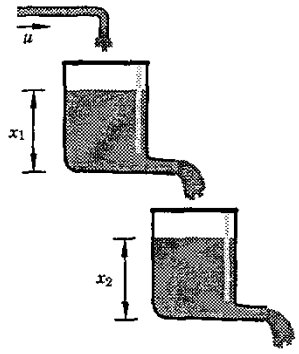

Figure 1: The double tank process in the example.

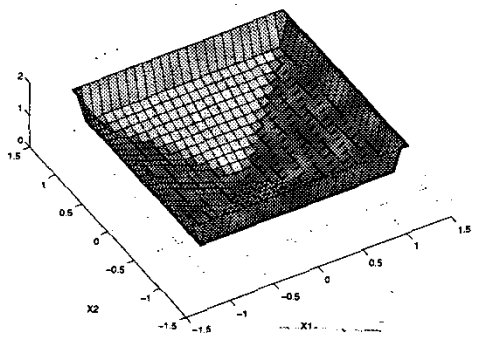

Figure 2: The $\varphi(x)$ function in the example.

Figure 1) is given by

$$x(n+1) = \begin{pmatrix} 0.95 & 0 \\ 0.1 & 0.95 \end{pmatrix} x(n) + \begin{pmatrix} 1 \\ 0 \end{pmatrix} u(n) \quad (28)$$

where x_1 and x_2 is the water heights in the upper and lower tanks, respectively. The control signal u is limited by $|u| \leq 1$. We want the controller to drive the state to zero while avoiding too large states ($|x_i| > 1$). Therefore the cost function is defined by eight hyperplanes, four that punishes states $|x_i| > 1$ with slope 10, and four that punish states $|x_i| \leq 1$ with slope 1. We let the time horizon $N = 10$.

Using DP without slack, the size $|\kappa|$ explodes within a couple of time steps and is impossible to solve for longer horizons.

Setting $\alpha_1 = 0.6$, $\alpha_2 = 0.1$, the algorithm finds $V^a(x,0)$ while having to keep a set $\kappa(n)$ of size less than 65 each time step. From this, we know the true value function within

$$V^a(x,n) \leq V^*(x,y) \leq \frac{1}{0.6}(V^a(x,n)+1). \quad (29)$$

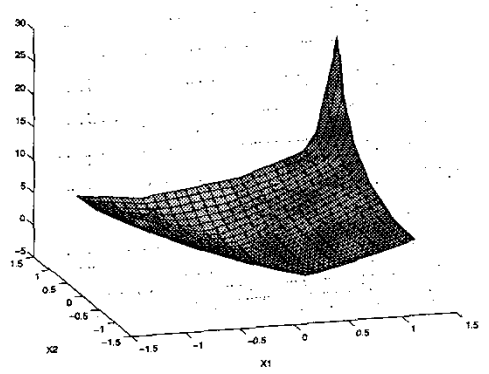

Figure 3: $V^a(x,0)$ in the example. Note that the state space is not gridded in the solution, but only for plotting.

The found $V^a(x)$ is shown for states $|x_i| \leq 1.2$ is shown in Figure 3.

5. Problem 2: POMDPs

The problem of Partially Observable Markov Decision Processes (POMDPs) has been around for a long time (see e.g. [1, 5]). Lately, it has mostly been investigated in the AI/robotics fields, where robot navigation problems where limited sensor information is available. A POMDP basically is a dual control problem where the state-space X is finite, as is the control signal (or action) space U and observation space Y.

The state $x(n)$ is a Markov state and, and the dynamics are specified by transition matrix $\pi(u)$, where element k,l denotes the probability to move to state k if the system is currently in state l. This probability can be controlled by u. See Figure 4 for an illustration.

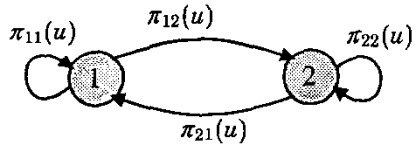

Figure 4: An example POMDP Markov graph with a set of transition matrices $\pi(u)$, one for each control action u.

A specific observation $y(n) \in Y$ will be obtained with probability

$$P(y(n) \mid x(n) = k) = \Omega_k(y(n), u(n)), \quad (30)$$

where Ω is the observation probability vector. Thus, the controller never really knows exactly in which

state the process is (if it was, the problem would be a Markov Decision Process and easily solved). To be able to use DP, the state is changed to the belief state $z(n)$: $z_k(n) = P(x(n) = k)$. Note that state space is closed as $z_k(n) \geq 0$, $\forall k$ and $\sum_k z_k(n) = 1$. The dynamics of the belief state is linear,

$$z(n+1) = \pi(u)z(n). \tag{31}$$

and for each observation $y \in Y$, our belief state is changed according to Bayes' rule:

$$\begin{aligned} z_k(n \mid y(n)) = P(x(n) = k \mid y(n)) = \\ \frac{P(x(n) = k \cap y(n))}{P(y(n))} = \\ \frac{P(y(n) \mid x(n) = k) \cdot P(x(n) = k)}{P(y(n))} = \\ \frac{\Omega_k(y(n), u(n)) \cdot z_k(n)}{P(y(n))}. \end{aligned} \tag{32}$$

Thus, the expected state over all possible observations is

$$\begin{aligned} \mathop{\mathbf{E}}_{y(n)}\Big\{z(n)\Big\} = \sum_{y(n)} P(y(n)) \frac{\Omega(y(n), u(n))z(n)}{P(y(n))} = \\ \sum_{y(n)} \Omega(y(n), u(n))z(n) \end{aligned} \tag{33}$$

The cost in a POMDP problem is usually replaced by a reward, so we will stick to that. The reward J is defined as

$$J(n) = \mathbf{E} \sum_{k=n}^{N} \varphi(x(k), u(k)) = \sum_{k=n}^{N} R(u(k))^T z(k), \tag{34}$$

where $R(u(k))$ is a vector of rewards of using control signal $u(k)$ for each Markov state $x(k)$.

For each time step, the controller has to make a control decision $u(n)$ based on the current belief state $z(n)$. After the control decision, an observation $y(n)$ based on $z(n)$ and $u(n)$ is obtained. We would like to find an optimal control policy $u = \mu(z, n)$ which maximizes the reward $J(n)$ for any initial state $z(n)$. As it turns out, the value function $V(z, n)$ is of the form

$$V(z, n) = \max_{k \in \kappa(n)} k^T z(k), \tag{35}$$

where $\kappa(n)$ is a finite set and k is a vector. Thus the value function is piecewise linear in the state z.

If the value function $V(z, n+1)$ is known and in the form (35), we can calculate the value $V(z, n)$ from Bellman's equation:

$$\begin{aligned} V(z, n) = \\ \max_u \mathbf{E}\Big\{V(\pi(u)z, n+1) + \varphi(x, u)\Big\} = \\ \max_u \mathbf{E}\Big\{\max_{k \in \kappa(n+1)} (k^T \pi(u) + R(u)^T) z(n \mid y(n))\Big\} = \\ \max_u \sum_{y \in Y} \max_{k \in \kappa(n+1)} (k^T \pi(u) + R(u)^T) \Omega(y, u) z = \\ \max_{k \in \kappa(n)} k^T z(n) \end{aligned} \tag{36}$$

Note that the "raw" size of $\kappa(n)$ is significantly larger than $\kappa(n+1)$ (actually $|\kappa(n)| = |\kappa(n)|^{|Y|}|U|$, where $|U|$ denotes the number of elements in U).

5.1 Parsimonious representation

Just like for the control problem problem in Section 4.3, the set $\kappa(n)$ is often unnecessarily large and may be pruned without changing the value of $V(x, n)$. Procedure 1 can be used without modification to obtain a parsimonious representation.

5.2 α-optimal value function

Analogous to Section 4.4, a modified pruning procedure can be used to obtain an α-optimal value function. The modified cost $\varphi^\alpha(x, u)$ can be chosen as

$$\varphi^\alpha(x, u) = \varphi(x, y) - \alpha. \tag{37}$$

5.3 Example

There is a wide variety of reference POMDP problems defined in literature. In this section we focus on the 4x3 Maze problem found in [4], which is a modified version from [7]. The state x is a position in a square 4x3 Maze where one state is inaccessible, and therefore the state space X has size 11 (z is 11-dimensional). Y consists of six observations, and there are four actions in U. The immediate reward is

$$\varphi(x) = \begin{cases} +1 & \text{if } x = \text{good} \\ -1 & \text{if } x = \text{bad} \\ -0.04 & \text{otherwise} \end{cases}. \tag{38}$$

After reaching the "good" or "bad" state, the state is reset. The problem is solved over an infinite horizon using value iteration with a discount factor $\lambda = 0.95$.

Running POMDP-SOLVE from [4] with incremental pruning and $\alpha = 0$ fails to return a solution within a reasonable time as the set κ grows too fast.

Setting $\alpha = 0.01$, the algorithm keeps an set κ^α of size about 150 after reaching steady state. The algorithm was run with a finite horizon of 50 time steps, and the resulting average value (for random initial states) is

≈ 1.7. Using our α and the discount factor λ, we can bound the optimal value $V^*(x)$ by

$$V^\alpha(x) \leq V^*(x) \leq V^\alpha(x) + \sum_{i=0}^{\infty} \lambda^i \alpha = V^\alpha(x) + 0.2 \quad (39)$$

A smaller α produces a larger search and a tighter bound, and vice versa.

6. Other problems

Two problems where the pruning method is highly useful has been presented in the previous sections. There are many more problem where the method is applicable, and some will be briefly mentioned here.

6.1 Linear Quadratic Switching

In [6], a control problem with linear switched dynamics and quadratic immediate cost

$$\varphi(x,u) = \begin{pmatrix} x \\ u \end{pmatrix}^T Q \begin{pmatrix} x \\ u \end{pmatrix}. \quad (40)$$

Using the relaxed

$$\varphi^\alpha(x,u) = \alpha\varphi(x,u) \quad (41)$$

with $\alpha \geq 1$, many problems can be practically solved.

6.2 Verification

The problem Section 4 may be immediately interpreted as a verification problem. By defining the safe set Ω by

$$\Omega = \{x \mid V(x,N) = \max_{k \in \kappa(n)} P_k(n)^T \leq 0\} \quad (42)$$

and using a step cost $\varphi(x) = 0$ and $\varphi^\alpha(x) = -\alpha$ for $\alpha \geq 0$, we can obtain upper and lower bounds on the set of states from which we can reach the safe set.

7. Conclusions

This paper has presented a method to relax Dynamic Programming to find suboptimal solutions with known error bounds to hard problems. The bounds are chosen by the user, who can then effectively trade-off between solution time and accuracy. Several examples where the method is useful have been presented.

8. References

[1] K. J. Åström. "Optimal control of Markov processes with incomplete state information I." *Journal of Mathematical Analysis and Applications*, **10**, pp. 174–205, 1965.

[2] D. P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, 2nd edition, 2000.

[3] F. Borrelli, A. Bemporad, , and M. Morari. "A geometric algorithm for multi-parametric linear programming." *Journal of Optimization Theory and Applications*, 2002. To appear.

[4] A. R. Cassandra. "Tony's pomdp page." `http://www.cs.brown.edu/research/ai/pomdp/`.

[5] A. R. Cassandra. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes.* PhD thesis, Brown University, Department of Computer Science, Providence, RI, 1998.

[6] B. Lincoln and A. Rantzer. "Optimizing linear system switching." In *Proceedings of the 40th Conference on Decision and Control*, 2001.

[7] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, Eaglewood Cliffs, NJ, 1994.