Relaxing Dynamic Programming

Bo Lincoln and Anders Rantzer, *Fellow, IEEE*

***Abstract*—The idea of dynamic programming is general and very simple, but the "curse of dimensionality" is often prohibitive and restricts the fields of application. This paper introduces a method to reduce the complexity by relaxing the demand for optimality. The distance from optimality is kept within prespecified bounds and the size of the bounds determines the computational complexity. Several computational examples are considered. The first is optimal switching between linear systems, with application to design of a dc/dc voltage converter. The second is optimal control of a linear system with piecewise linear cost with application to stock order control. Finally, the method is applied to a partially observable Markov decision problem (POMDP).**

***Index Terms*—Dynamic programming, nonlinear synthesis, optimal control, switching systems.**

I. Introduction

A. Motivation

Since the 1950s, the idea of dynamic programming [3], [5] has propagated into a vast variety of applications. This includes as diverse problems as portfolio theory and inventory control in economics, shortest path problems in network routing and speech recognition, task scheduling in real time programming and receding horizon optimization in process control.

The use of dynamic programming is however limited by the inherent computational complexity. The need for approximative methods was therefore recognized already in the early works by Bellman. Approximative value functions have since been introduced in a variety of ways. One approach is *neuro-dynamic programming*, described in [6]. Also, [10] and [15] present methods where the approximate value function is parameterized as a linear combination of a set of basis functions. Techniques for formal verification of nonlinear or hybrid systems are related to this work. For example, the toolbox HyTech, presented in [11], calculates over- and under-approximations of a reachable set. This can be viewed as calculating upper and lower bounds of a value function. The main contribution of this paper is to give a new simple method to approximate the optimal value function $V^*(x)$, which guarantees that the suboptimal solution is within a *prespecified distance* from the optimal solution. A major part of the paper is devoted to application of this method for some important problem classes where standard value iteration gives a finite description of $V_k(x)$, whose complexity grows rapidly with iteration number k. Using the new method, instances for which dynamic programming has previously been considered hopeless can here be practically solved.

Manuscript received January 15, 2003; revised May 21, 2004 and May 23, 2005. Recommended by Associate Editor A. Bemporad. This work was supported by the Swedish Research Council and by the EU/ESPRIT project Computation and Control (CC).

The authors are with the Department of Automatic Control, LTH, Lund University, SE-221 00 Lund, Sweden (e-mail: lincoln@control.lth.se; rantzer@control.lth.se).

Digital Object Identifier 10.1109/TAC.2006.878720

This paper first introduces some background, followed by the main relaxation method in Section II. Then, three different applications are presented in the following sections. For further details on the theoretical foundations, the reader is referred to [13].

B. Dynamic Programming

Let $x(n) \in X$ be the state of a given system at time n, while $u(n) \in U$ is the value of the control signal. The system evolves as

$$x(n+1) = f\left(x(n), u(n)\right), \qquad x(0) = x_0.$$

For a given cost function

$$\sum_{n=0}^{\infty} l\left(x(n), u(n)\right) \tag{1}$$

such that $l(x,u) \geq 0$ with equality only if $x = 0$, we would like to find an optimal control policy $u = \mu(x)$, such that the cost is minimized from every initial state. For a fixed control policy μ the map from initial state x_0 to the value of (1) is called a *value function* $V^\mu(x_0)$. The optimal value function is denoted

$$V^*(x_0) = \inf_\mu V^\mu(x_0)$$

and is characterized by the "Bellman equation"

$$V^*(x) = \min_u \left\{V^*\left(f(x,u)\right) + l(x,u)\right\}, \qquad V(0) = 0.$$

A common method to find the optimal value function is *value iteration*, i.e., to start at some initial $V_0(x)$, for example $V_0 \equiv 0$, and update iteratively

$$V_{k+1}(x) = \min_u \left\{V_k\left(f(x,u)\right) + l(x,u)\right\} \tag{2}$$

It is well known that value iteration converges under mild conditions. For easy reference, a formal statement of this fact is given as follows.

Proposition 1: (Convergence of Value Iteration) Suppose the inequality $V^*(f(x,u)) \leq \gamma l(x,u)$ holds uniformly for some $\gamma < \infty$ and that $\delta V^* \leq V_0 \leq V^*$. Then, the sequence defined iteratively by (2) approaches V^* according to the inequalities

$$\left[1 + \frac{\delta - 1}{\left(1+\gamma^{-1}\right)^k}\right] V^* \leq V_k \leq V^*. \tag{3}$$

A proof is given in the Appendix. Notice that the constant γ gives a measure on how "contractive" the optimally controlled

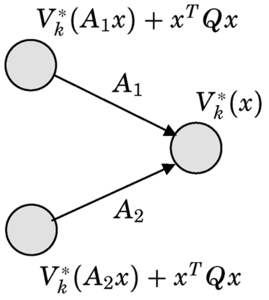


Fig. 1. In value iteration, the cost from the state at one time instant is expressed in terms of the cost from the state at the next time instant.

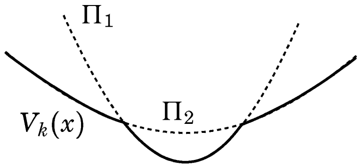


Fig. 2. Piecewise quadratic value function $V_k(x)$.

system is, i.e., how close the total cost is to the cost of a single step. The smaller γ is, the faster convergence.

For a more extensive treatment of basic dynamic programming theory, see, e.g., [5].

C. An Example

Many applications of dynamic programming become intractable because of the fact that the value function gets increasingly hard to represent for each iteration. Consider the following example: Let A_1 and A_2 be two system matrices and let a discrete-time dynamical system evolve as

$$x(n+1) = A_{i(n)}x(n), \qquad i(n) \in \{1,2\}.$$

The goal is to find a control law, that given the current state x assigns an i (i.e., system matrix) so as to minimize

$$\sum_{n=0}^{\infty} x(n)^T Q x(n).$$

Let us consider value iteration for this problem, starting with $V_0(x) = 0$. At each time step, there are two choices; use A_1 or A_2 (see Fig. 1). This leads to a sequence of functions $V_k(x)$ of the form

$$V_k(x) = \min_{\Pi \in P_k} x^T \Pi x$$

where P_k is a set of positive matrices (see Fig. 2 for an illustration). Typically, the number of matrices in P_k grows exponentially with k due to the two choices in each time step. In the following section we present a method of finding an approximate value function for this problem.

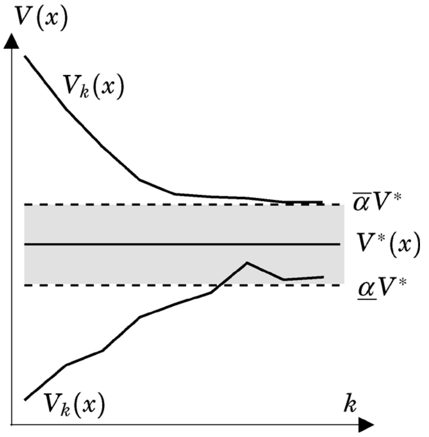


Fig. 3. Illustration of two different value functions $V_k(x)$ converging to the close-to-optimal-set.

II. Relaxed Dynamic Programming

This section will describe a method to find a $V(x)$ which fulfills $V(0) = 0$ and

$$\min_u \{V(f(x,u)) + \underline{l}(x,u)\} \le V(x) \le \min_u \{V(f(x,u)) + \overline{l}(x,u)\} \quad (4)$$

for all x and u (see Fig. 3). Iterative application of the inequalities gives

$$\min_{\{u(n)\}_{n=0}^{\infty}} \sum_{n=0}^{\infty} \underline{l}(x,u) \le V(x) \le \min_{\{u(n)\}_{n=0}^{\infty}} \sum_{n=0}^{\infty} \overline{l}(x,u) \quad (5)$$

for every initial state x such that the minima are finite. Moreover, the control law $u(n) = \mu(x(n))$ with

$$\mu(x) = \arg\min_u \{V(f(x,u)) + \underline{l}(x,u)\}$$

achieves

$$\sum_{n=0}^{\infty} \underline{l}(x(n), \mu(x(n))) \le V(x).$$

In particular, μ is a stabilizing feedback law because the lower bound in (4) implies that V is a Lyapunov function for the closed-loop system.

Usually $\overline{l}$ and $\underline{l}$ are chosen to satisfy $\underline{l}(x,u) \le l(x,u) \le \overline{l}(x,u)$, for example

$$\overline{l}(x,u) = \overline{\alpha} l(x,u), \qquad \overline{\alpha} \ge 1 \quad (6)$$
$$\underline{l}(x,u) = \underline{\alpha} l(x,u), \qquad \underline{\alpha} \le 1. \quad (7)$$

With this relaxation of Bellman's equation, we can search for a solution $V(x)$ which is more easily parameterized than $V^*(x)$.

A. Relaxed Value Iteration

Given $V_{k-1}(x)$ satisfying

$$\min_{\{u(n)\}_{n=0}^{k-1}} \sum_{n=0}^{k-1} \underline{l}(x,u) \le V_{k-1}(x) \le \min_{\{u(n)\}_{n=0}^{k-1}} \sum_{n=0}^{k-1} \overline{l}(x,u) \quad (8)$$

define $\overline{V}_k(x)$ and $\underline{V}_k(x)$ according to

$$\overline{V}_k(x) = \min_u \left\{V_{k-1}\left(f(x,u)\right) + \overline{l}(x,u)\right\} \quad (9)$$

and

$$\underline{V}_k(x) = \min_u \left\{V_{k-1}\left(f(x,u)\right) + \underline{l}(x,u)\right\}. \quad (10)$$

The expressions for $\overline{V}_k(x)$ and $\underline{V}_k(x)$ are generally more complicated than for $V_{k-1}(x)$. From this, a simplified $V_k(x)$ which satisfies

$$\underline{V}_k(x) \le V_k(x) \le \overline{V}_k(x) \quad (11)$$

is calculated. This $V_k(x)$ satisfies

$$\min_{\{u(n)\}_{n=0}^{k}} \sum_{n=0}^{k} \underline{l}(x,u) \le V_k(x) \le \min_{\{u(n)\}_{n=0}^{k}} \sum_{n=0}^{k} \overline{l}(x,u) \quad (12)$$

and the procedure can be iterated. In particular, the lower bound shows that V_k grows with k at least as fast as standard value iteration with the step cost $\underline{l}(x,u)$.

The iteration of (9)–(11), which we call *relaxed value iteration*, can often be used to find a solution of (4).

The α's (and the l's) are chosen as a tradeoff between complexity (time and memory) and accuracy. If $\overline{\alpha}$ and $\underline{\alpha}$ are close to 1, then the iterative condition (11) becomes close to ordinary value iteration (2), which gives high accuracy and high complexity. On the other hand, if the fraction $\overline{\alpha}/\underline{\alpha}$ is very big, then the accuracy drops, but (11) can be satisfied with less complex computations. This will be demonstrated in examples later.

Note that if l is chosen as in (6) and (7), then the relative error in the value function defined by $\overline{\alpha}$ and $\underline{\alpha}$ is independent of the number of iterations.

B. Stopping Criterion

If the value function $V_k(x)$ at iteration k satisfies (4) for $V(x) = V_k(x)$, then it also satisfies (5), and a solution to the infinite horizon problem has been found. If the iteration is stopped before (4) holds, it is possible to calculate $\overline{l}(x,u)$ and $\underline{l}(x,u)$ for which it does hold and thus test for which relaxation the current $V_k(x)$ holds as a solution.

Remark: This method of calculating the slack to optimality can be used *no matter how* $V(x)$ was obtained. For example, a finite-time $V(x)$ obtained by solving a multiparametric QP (see [4]) for a certain horizon N could be used. The upper and lower bound in the inequalities (4) can be calculated by solving the problem for horizon N using $\overline{l}$ and $\underline{l}$ for the first time-step.

C. Bounded Complexity for Modified Algorithm

An alternative to (11) is to use the following implicit upper bound:

$$\underline{V}_k(x) \le V_k(x) \le \min_u \left\{V_k\left(f(x,u)\right) + \overline{l}(x,u)\right\} \qquad V_k(0) = 0 \quad (13)$$

in the value iteration. Iterative application of the second inequality in (13) implies that

$$V_k(x) \le \min_{\{u(n)\}_{n=0}^{\infty}} \sum_{n=0}^{\infty} \overline{l}\left(x(n), u(n)\right).$$

Note that the upper bound (13) is implicit, i.e., cannot be calculated before the search for a simplified $V_k(x)$. In the applications in this paper, the more explicit condition (11) will therefore be used. Nevertheless, (13) defines a convex condition on $V_k(x)$ (since every value of u gives a linear condition on V_k) and is, therefore, computationally tractable. Moreover, unlike the original relaxation (11), the implicit relaxation (13) has a simple criterion for feasibility: Assume there exists a $\hat{V}(x)$ which satisfies

$$\begin{aligned}\min_u \left\{V^*\left(f(x,u)\right) + \underline{l}(x,u)\right\} &\le \hat{V}(x)\\ &\le \min_u \left\{\hat{V}\left(f(x,u)\right) + \overline{l}(x,u)\right\} \qquad \hat{V}(0) = 0.\end{aligned}$$

Then, starting with $V_0(x) \equiv 0$, $\hat{V}(x)$ satisfies inequality (13) at every iteration, so the "complexity" of $\hat{V}(x)$ gives an upper bound on the complexity of V_k that needs to be considered during the iteration. Further discussion of the implicit algorithm is given in [13].

D. Parameterization of V

The value function approximations considered in this paper will all have the form

$$V_k(x) = \operatorname*{select}_{p \in P_k} p(x)$$

where P_k is a set of (simple, e.g. linear or quadratic) functions on x, and the "select" operator selects one of the functions according to some criterion (e.g., "maximum", "minimum", or "feasible region"). Moreover, it is essential that also $\overline{V}_k$ and $\underline{V}_k$ will have the same form.

We define the complexity of the representation as $|P_k|$, i.e., the number of elements in the set P_k. If

$$V_k(x) = \min_{p \in P_k} p(x)$$

we will denote the value function "minimum-type." Note that adding a new function p to P_k *decreases* (or leaves unchanged) $V(x)$ for all x if V is minimum-type. This observation will be used in the next section.

E. A Simple Algorithm to Calculate V

So far, we have not discussed how to calculate $V_k(x)$ to satisfy (11). Doing this in an optimal way with respect to complexity of the optimization may be a very hard problem. In this section, we present a simple and efficient (albeit not optimal) algorithm to obtain $V_k(x)$ for the minimum-type (and conversely maximum-type) parameterization defined in the previous section. The algorithm is described in Procedure 1.

Procedure 1 (From V_{k-1} to V_k, minimum-type)

1. Calculate $\overline{V}_k$ and $\underline{V}_k$ from V_{k-1}:

$$\overline{V}_k(x) = \min_u \left\{V_{k-1}\left(f(x,u)\right) + \overline{l}(x,u)\right\}$$
$$\underline{V}_k(x) = \min_u \left\{V_{k-1}\left(f(x,u)\right) + \underline{l}(x,u)\right\}$$

Define $\overline{P}_k$ and $\underline{P}_k$ such that

$$\overline{V}_k(x) = \min_{p\in\overline{P}_k} p(x) \quad \text{and} \quad \underline{V}_k(x) = \min_{p\in\underline{P}_k} p(x).$$

2. Let $P_k = \emptyset$ and $V_k(x) = \infty$.
3. If possible, find $x_0 \in X$ such that

$$V_k(x_0) > \overline{V}_k(x_0).$$

If not, $V_k(x)$ satisfies (11) $\Rightarrow$ Done.
5. Let

$$\underline{p} = \operatorname*{argmin}_{p\in\underline{P}_k} p(x_0)$$

and add $\underline{p}$ to the set P_k.
4. Define

$$V_k(x) = \min_{p\in P_k} p(x)$$

and go to step 3.

The algorithm simply adds elements from the lower bound $\underline{V}_k(x)$ until the resulting value function $V_k(x)$ satisfies the upper bound $V_k(x) \leq \overline{V}_k(x)$. The resulting value function satisfies (11) by construction. Naturally, the algorithm can be made more efficient by changing details such as removing functions in $\overline{P}_k(x)$ once they have been tested as nonactive. It is, however, crucial that the expressions defining $\underline{V}_k(x)$ and $\overline{V}_k(x)$ give functions of the right form.

Procedure 1 can be iterated until (4) or some other stopping criterion is satisfied. For maximum-type value functions the procedure is simply changed to add from $\overline{V}_k(x)$ until the lower bound holds.

III. Application 1: Switched Linear Systems With Quadratic Costs

In this section, a linear system switching problem will be described. Given a set of alternative system matrices for a linear system, the problem is to find a control law both for the continuous inputs and for switching between system matrices at each time step. All switches are initiated by the control law; there is no autonomous switching.

In other words, for the N triples of system matrices $\{(\Phi_1,\Gamma_1,Q_1),\ldots,(\Phi_N,\Gamma_N,Q_N)\}$ consider the linear system

$$x(n+1) = \Phi_w x(n) + \Gamma_w u(n)$$

and the step cost $l(x,u) = [x^T\ u^T]Q_w[x^T\ u^T]^T$. The problem is to find a feedback control law

$$u = \mu(x) \quad w = \nu(x) \in \{1,2,\ldots,N\}$$

such that the closed-loop system minimizes the accumulated cost.

A. Value Function

We assume $V_{k-1}(x)$ at iteration $k-1$ to be on the form

$$V_{k-1}(x) = \min_{\Pi\in P_{k-1}} x^T\Pi x \tag{14}$$

where P_{k-1} is a set of non-negative, symmetric matrices. To obtain a relaxed value function, the procedure in Section II is used with $\underline{\alpha} = 1$ and $\overline{\alpha} = \alpha$ (and, thus, α is our relaxation parameter). The upper and lower bounds $\overline{V}_k(x)$ and $\underline{V}_k(x)$, with corresponding sets $\overline{P}_k$ and $\underline{P}_k$, are easily calculated as defined in (9) and (10), respectively. These are on the same form as $V_{k-1}(x)$ and, thus, we can theoretically continue the value iteration. The sizes of the sets $|\overline{P}_k|$ and $|\underline{P}_k|$ can however be up to $N\cdot|P_{k-1}|$, so worst-case complexity of $V_k(x)$ grows exponentially with k. This is why the proposed relaxed dynamic programming is needed.

The two sets of matrices $\overline{P}_k$ and $\underline{P}_k$ are stored as ordered sets

$$\overline{P}_k = \{\overline{\Pi}_1,\ldots,\overline{\Pi}_{N_k}\} \quad \underline{P}_k = \{\underline{\Pi}_1,\ldots,\underline{\Pi}_{N_k}\}$$

and sorted so that

$$\operatorname{tr}\overline{\Pi}_1 \leq \ldots \leq \operatorname{tr}\overline{\Pi}_{N_k} \quad \underline{\Pi}_i \leq \overline{\Pi}_i, \qquad \text{for all } i.$$

After that, use Procedure 2 (which is a special case of Procedure 1) to calculate V_k.

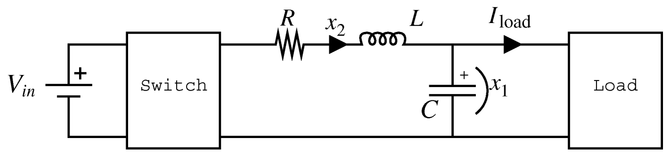


Fig. 4. Setup for the switched dc/dc-converter.

Procedure 2 (Relaxed $V_k(x)$, switched system).

1. Define
$$V_k(x) = \min_{\Pi \in P_k} x^T \Pi x.$$
and take initially $P_k := \emptyset$, $i := 0$.
2. If $i = N_k$, then stop, else let $i := i + 1$.
3. If there exists a convex combination Π of elements in P_k such that $\overline{\Pi}_i \geq \Pi$, then go to step 2. (This is the S-procedure test [16] to check if $x^T \overline{\Pi}_i x \geq V_k(x)$.)

If not, then add $\underline{\Pi}_i$ to P_k and go to 3.

Remark: The sorting of $\overline{P}_k$ on trace ensures that small Π's are added first to P_k. In practice it means that more $\overline{\Pi}$'s can be discarded and, thus, a smaller set P_k is found.

B. Example—A Switched Voltage Controller

A naturally switched control problem is the design of a switched power controller for dc to dc conversion. The idea is to use a set of semiconductor switches to effectively change polarity of a voltage source, and the controller has to decide which polarity to use each time slot (at a high frequency) so that the load voltage and current are kept as constant as possible. See Fig. 4 for the setup. This kind of dc/dc-converter is used in practice, often with a pulse width modulation control for the switch. The problem can also be extended to an ac/dc-converter by making it time-varying.

1) Modeling: Except for the switch, all components in the power system can be viewed as linear. For the purpose of control optimization, the load is modeled as a constant current sink. To make the controller work well for varying loads as well, an integrator is added. The model becomes

$$\dot{x}_1 = \frac{1}{C}(x_2 - I_{\text{load}}) \tag{15}$$
$$\dot{x}_2 = -\frac{1}{L}x_1 - \frac{R}{L}x_2 + \frac{1}{L}s(t)V_{\text{in}} \tag{16}$$

where $s(t)$ is the sign of the switch as set by the controller. To obtain integral action in the controller, a third state is added as the integral of the voltage error

$$\dot{x}_3 = V_{\text{ref}} - x_1.$$

Using the affine extension

$$x_e(n) = \begin{bmatrix} x(n) \\ 1 \end{bmatrix}$$

and a sample period of h s, the model can be described in discrete time as

$$x_e(n+1) = \Phi_i x_e(n), \qquad i \in \{1, 2\}$$

where Φ_i is a 4×4 matrix due to the integral state and the affine extension of the state vector.

2) Cost Function: The objective of the controller is to keep the voltage x_1 as constant as possible at V_{ref}. To avoid constant errors, but also strong harmonics, a combination of average, current and derivative deviations are punished. This is done by using the step cost

$$l(x) = q_P(x_1 - V_{\text{ref}})^2 + q_I x_3^2 + q_D(x_2 - I_{\text{load}})^2.$$

With the extended state vector this can also be written on standard form $l(x) = x_e^T Q x_e$.

The switching controller will never be able to bring the system to a steady state at the set-point. Therefore, the standard cost function will grow indefinitely. In this example, a "forgetting factor" $\lambda \leq 1$ is introduced in the cost function to cope with this problem

$$\sum_{n=0}^{\infty} \lambda^n l(x(n)).$$

3) Finding a Controller: The previous example is now on standard form, and the algorithm in this section can be used to find a controller. Generally, a forgetting factor λ simplifies the problem solution a lot, but also disqualifies $V(x)$ as a Lyapunov function. The example has been tried with λ in the range 0.95 to 1, and for λ's less than 1, the value function complexity stabilizes on a reasonable level. We set the parameters to

$$\begin{array}{lll} R = 1 & C = 4 & L = 0.1 \\ \lambda = 0.95 & \alpha = 1.5 & h = 0.1 \\ V_{\text{in}} = 1 & V_{\text{ref}} = 0.5 & I_{\text{load}} = 0.3 \\ q_P = 1 & q_I = 1 & q_D = 0.3 \end{array}$$

and again note that the I_{load} is only nominal and will change in the simulation later. The value iteration is done for 140 steps. For all steps, the complexity of the value function, i.e., number of quadratic functions, stays below 40, as can be seen in Fig. 5.

The controller defined implicitly by $V_{200}(x)$ is used in the simulation shown in Fig. 7. The explicit controller, evaluated each time step, is

$$s(n) = R\left(\arg\min_{\Pi \in P_{200}} \left\{x(n)^T \Pi x(n)\right\}\right)$$

and $R(\Pi_{\text{min}})$ is simply a table lookup mapping one specific member of P_{200} to a switch position ($+1$ or -1). See Fig. 6. Note that evaluating this expression online is not very computationally intensive, compared to solving the offline dynamic programming.

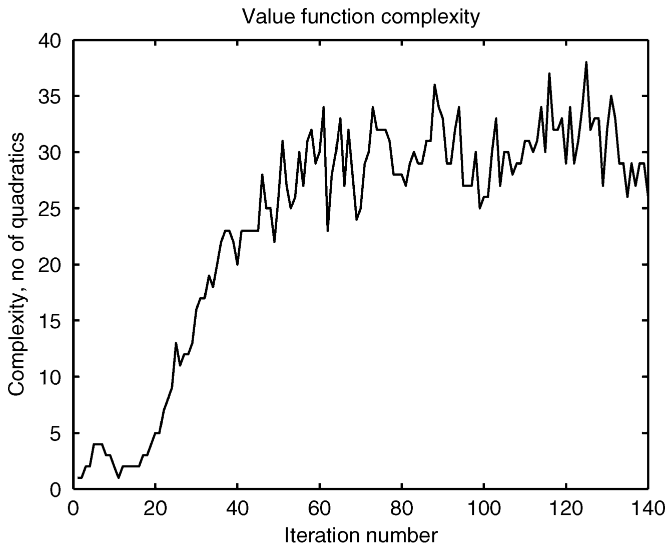


Fig. 5. Complexity of the value function in the power controller example.

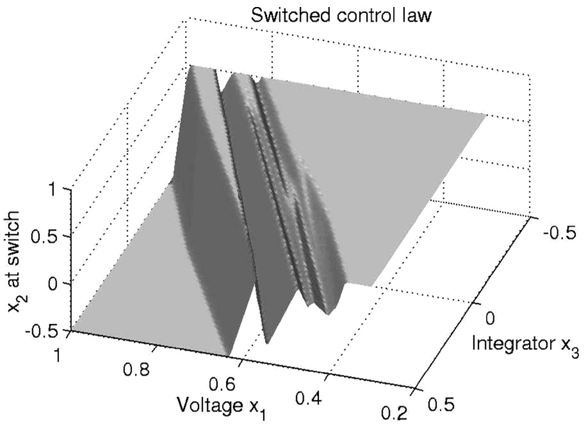


Fig. 6. Resulting switching feedback law is monotonous in the current x_2 (by observation) and, therefore, it can be plotted in 3-D. The plot shows at which current x_2 the switch from $s(n) = +1$ to $s(n) = -1$ takes place for varying voltages x_1 and integral states x_3. Note that the gridding is only for plotting purposes.

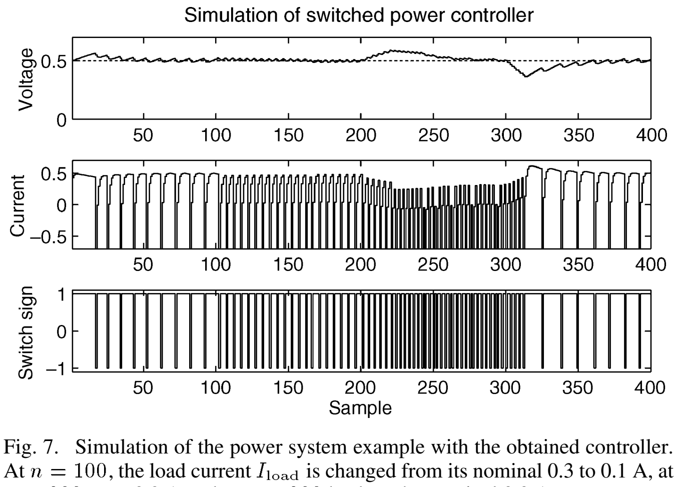


Fig. 7. Simulation of the power system example with the obtained controller. At $n = 100$, the load current I_{load} is changed from its nominal 0.3 to 0.1 A, at $n = 200$, to -0.2 A and at $n = 300$ back to the nominal 0.3 A.

In the simulation in Fig. 7, the load current I_{load} is changed at several points in time, but the controller keeps the voltage well thanks to the integral action.

C. Example

Consider the classical inverted pendulum. Because the system is unstable, the controller needs to give the system constant attention to stabilize it. Assume there are several pendulums to be controlled by one controller simultaneously. If the controller has limited computing or communication resources and can only make one control decision per time unit, an obvious problem is to choose which pendulum to control each time.

We can pose such a problem based on a linearized model of a rotating inverted pendulum ("the Furuta pendulum") from our teaching laboratory in Lund.

$$\frac{dx}{dt}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 31.3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -0.588 & 0 & 0 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ -71.2 \\ 0 \\ 191 \end{bmatrix} u(t).$$

The step cost matrix is set to

$$Q = \operatorname{diag}([1 \quad 0.1 \quad 1 \quad 0.1 \quad 1]).$$

A reasonable sample period for this pendulum is around 20–50 ms. Because several pendulums are to be controlled, the controller "time-slot" is set to $h = 20$ ms. The sampled system matrices are denoted Φ and Γ. We assume that a pendulum which does not get attention in the current time slot holds its previous control signal. This increases the system order.

For the two-pendulum-problem, the system matrices become

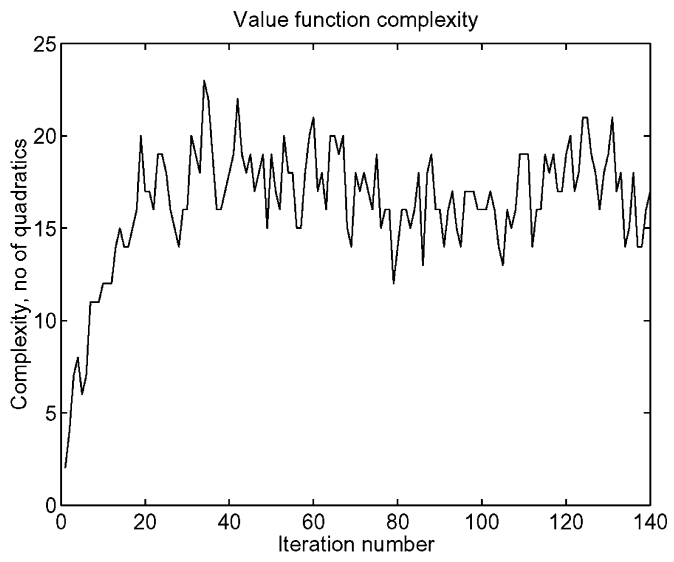
$$\Phi_1 = \begin{bmatrix} \Phi & \Gamma & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \Phi & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \Gamma_1 = \begin{bmatrix} 0 \\ 0 \\ \Gamma \\ 1 \end{bmatrix}$$

$$\Phi_2 = \begin{bmatrix} \Phi & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \Phi & \Gamma \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \Gamma_2 = \begin{bmatrix} \Gamma \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

which is essentially the augmented system consisting of two pendulums plus states for holding the control signal.

For the two pendulum problem, it is possible to do value iteration with $1/\underline{\alpha} = \overline{\alpha} = 1.01$, i.e., with only 1% slack to the optimal solution. The resulting complexity-over-iterations graph can be seen in the upper plot of Fig. 8. As the system is unstable and also sampled quite fast, the value iteration takes about 30 steps to converge. As can be seen, the complexity stays under 25 for all iterations, though. The lower graph is obtained if the demand for accuracy is relaxed even further, to $1/\underline{\alpha} = \overline{\alpha} = 1.5$.

Extending the problem to three pendulums, the dynamic programming gets harder. Setting $1/\underline{\alpha} = \overline{\alpha} = 1.5$, the problem is still solvable, and the resulting complexity graph can be seen in Fig. 9. Note that the state–space for this problem is 15-dimensional (or, effectively, 14-dimensional). The Matlab code for the voltage converter and the pendulum examples is available from [1].



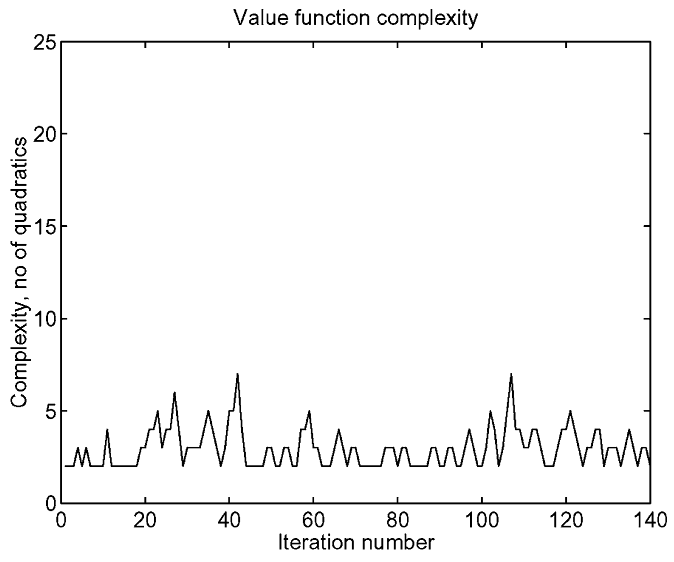


Fig. 8. Complexity of the value function for the example of controlling two Furuta pendulums (ten continuous states). The upper diagram was obtained with $\alpha = 1.01$ using 233 s CPU-time on an Intel Pentium 1600-MHz processor. The lower diagram shows the value iteration complexity with $\alpha = 1.5$ and required 14.7 s.

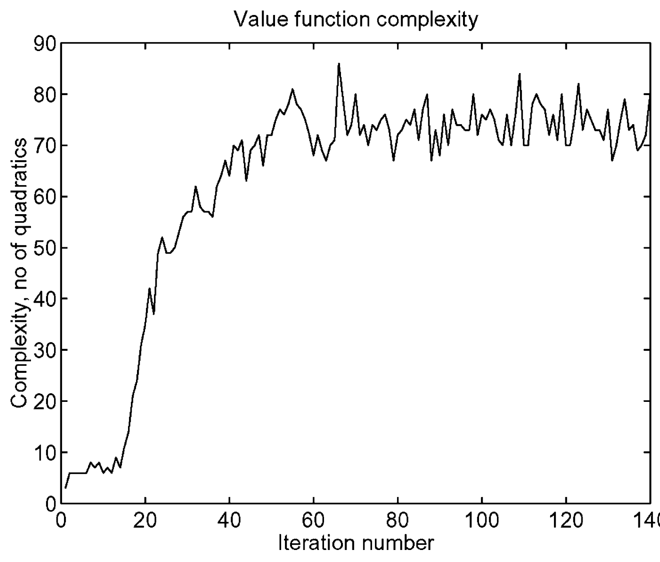


Fig. 9. Complexity of relaxed value iteration for the example of three Furuta pendulums (15 continuous states) with $\alpha = 1.5$. The calculations used 4790 s CPU-time on an Intel Pentium 1600 MHz.

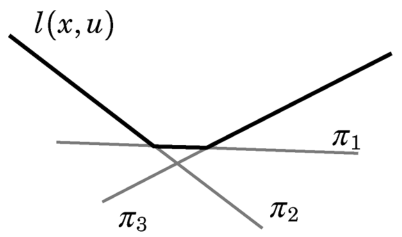


Fig. 10. Piecewise linear cost function $l(x, u)$.

IV. Application 2: Linear System With Piecewise Linear Cost

This section will describe an optimal control problem. The plant to be controlled is a linear time-invariant (LTI) system, and the cost to be minimized is piecewise linear. This makes it possible to punish states in more elaborate ways than the usual quadratic cost. For example, it is possible make the cost asymmetric such that negative states are more costly than positive.

A. Problem Formulation

The controlled system is LTI

$$x(n+1) = Ax(n) + Bu(n)$$

where $x \in X$, $u \in U$ and X and U are polyhedra. The cost function is on the form (1), with

$$l(x,u) = \max_{\pi \in Q} \pi^T \begin{bmatrix} x \\ u \\ 1 \end{bmatrix}$$

where π is a vector and Q is a finite set of vectors. See Fig. 10. Hence, $l(x,u)$ is piecewise linear, convex, and of "maximum-type" (see Section II-D). The goal of the controller is to minimize the cost.

B. Value Function

Assume the value function $V_{k-1}(x)$ at some time $k-1$ is on the form

$$V_{k-1}(x) = \max_{\Pi \in P_{k-1}} \Pi^T \begin{bmatrix} x \\ 1 \end{bmatrix} \tag{17}$$

where P_{k-1} is a finite set of vectors. Bellman's equation is used to calculate $V_k(x)$

$$\begin{aligned} V_k(x) &= \min_{u \in U} \left\{ \max_{\Pi \in P_{k-1}} \Pi^T \begin{bmatrix} Ax + Bu \\ 1 \end{bmatrix} + \max_{\pi \in Q} \pi^T \begin{bmatrix} x \\ u \\ 1 \end{bmatrix} \right\} \\ &= \min_{u \in U} \max_{r \in R} \left\{ F_r^T x + G_r^T u + H_r \right\}. \end{aligned}$$

For the iteration to work, we need to rewrite this on the form (17). This can be done by rewriting $V_k(x)$ as the linear program

$$\begin{aligned} V_k(x) = \min_{u,f} f \quad & \text{s.t.} \\ F_r^T x + G_r^T u + H_r \le f \qquad & \forall r \in R \end{aligned}$$

where the constraints $x \in X$ and $u \in U$ are implicit. Solving this for any initial position x yields a value function that can again be written on the form (17). The solution can either be found by solving a *multiparametric linear programming* (MPLP) problem (see, e.g., [7]), or by doing explicit enumeration of the dual extreme points as is shown later.

Introducing Lagrange multipliers λ, we rewrite V_k as

$$\begin{aligned} V_k(x) &= \min_{u,f} \max_{\lambda_i \ge 0} \left\{ f + \sum_{r\in R} \lambda_k \left(F_r^T x + G_r^T u + H_r - f\right)\right\} \\ &= \max_{\lambda_i \ge 0} \min_{u,f} \left\{ \left(1 - \sum_{r\in R}\lambda_r\right) f + \sum_{r\in R} \lambda_r \left(F_r^T x + H_r\right) \right. \\ &\qquad \left. + \sum_{r\in R} \lambda_r G_r^T u \right\} \end{aligned}$$

where the last equality is due to strong duality for linear programming [9]. From this, we see that

$$\sum_{r\in R} \lambda_r = 1 \quad \sum_{r\in R} \lambda_r G_r^T = 0 \tag{18}$$

and

$$V_k(x) = \max_{\lambda \in L} \sum_{r\in R} \lambda_r \left(F_r^T x + H_r\right)$$

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

As $V_k(x)$ 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 $\{(i,j)|i \ne j \in R\}$, setting all other $\lambda_r = 0$, $r \ne i, r \ne j$. If λ_i and λ_j have positive solutions in (18), 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 of vectors P_k, and again the value function is on the form

$$V_k(x) = \max_{\Pi \in P_k} \Pi^T \begin{bmatrix} x \\ 1 \end{bmatrix}. \tag{19}$$

C. Parsimonious Representation

$V_k(x)$ can usually be represented by a much smaller set than the P_k obtained by the aforementioned extreme point enumeration. A set $P_k = \{\Pi_1, \dots, \Pi_{N_k}\}$ is called a *parsimonious* representation, if only the members of P_k which are ever active (achieve the maximum for some state) are included. Such a set can be obtained from Procedure 3 (which is well known, and, again, a special case of Procedure 1).

Procedure 3 (Parsimonious $V_k(x)$)

1. Let $P_k^{\text{pars}} := \varnothing$ and $i := 0$.
2. If $i = N_k$, then stop, else let $i := i + 1$.
3. If there exists a convex combination Π of elements in P_k^{pars} such that

$$\Pi_i^T \begin{bmatrix} x \\ 1 \end{bmatrix} \ge \Pi \begin{bmatrix} x \\ 1 \end{bmatrix} \qquad x \in X$$

then go to step 2.
If not, then add Π_i to P_k^{pars} and go to 3.

Note that after this procedure, naturally

$$\max_{\Pi \in P_k} \Pi^T \begin{bmatrix} x \\ 1 \end{bmatrix} = \max_{\Pi \in P_k^{\text{pars}}} \Pi^T \begin{bmatrix} x \\ 1 \end{bmatrix} \qquad x \in X.$$

D. Relaxed Value Function

To be able to use the proposed relaxed dynamic programming, we define the relaxed cost

$$\underline{l}(x,u) = \alpha_1 l(x,u) - \alpha_2 \tag{20}$$
$$\overline{l}(x,u) = l(x,u) \tag{21}$$

where $\alpha_1 \le 1$ and $\alpha_2 \ge 0$. Choosing $\alpha_1 < 1$ is of course only meaningful if $l(x,u) \ge 0$, $\forall x, u$. The procedure in Section II will now be used to find a relaxed $V_k(x)$. Assume $V_{k-1}(x)$ satisfies (8). Doing one value iteration from this $V_{k-1}(x)$ the costs

$$\begin{aligned} \overline{V}_k(x) &= \max_{\Pi \in \overline{P}_k} \Pi^T \begin{bmatrix} x \\ 1 \end{bmatrix} = \min_u \left\{ V_k\left(f(x,u)\right) + \overline{l}(x,u)\right\} \\ \underline{V}_k(x) &= \max_{\Pi \in \underline{P}_k} \Pi^T \begin{bmatrix} x \\ 1 \end{bmatrix} = \min_u \left\{ V_k\left(f(x,u)\right) + \underline{l}(x,u)\right\} \end{aligned}$$

are calculated.

Now, finding a V_k in between $\underline{V}_k(x)$ and $\overline{V}_k(x)$ is done by adding one element from $\overline{P}_k$ at a time according to Procedure 4 (which is again a special case of Procedure 1, but for maximum-type).

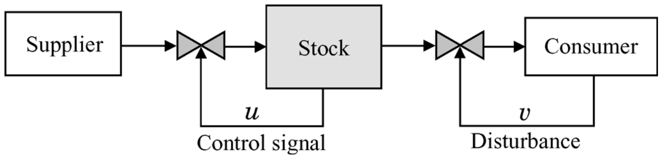


Fig. 11. Stock controlling the orders u to the manufacturer and the consumer controlling orders v from the stock in the example. As seen from the stock, u is the control signal and v is a disturbance.

Procedure 4 (Relaxed $V_k(x)$)

1. Let $P_k = \emptyset$.
2. Pick one $\Pi \in \underline{P}_k$. Find a state x where $\Pi^T \begin{bmatrix} x \\ 1 \end{bmatrix} > \pi_i^T \begin{bmatrix} x \\ 1 \end{bmatrix}, \forall \pi \in P_k$.
3. If such an x exists, find the $\pi \in \overline{P}_k$ with the highest cost $\pi^T \begin{bmatrix} x \\ 1 \end{bmatrix}$.
   Add π to P_k.
4. If no such x exists, remove Π from $\underline{P}_k$.
5. Repeat from 2 until $\underline{P}_k$ is empty.

E. Example—Stock Order Control

To illustrate the capability to have nonsymmetric cost functions, this section presents an example of a stock of some product. The control problem is to meet customer demand while not storing too many products nor running out of products when there is customer demand.

The system is modeled in discrete time, where the sample period h is one day. In one sample period, the stock controller can order from 0 to 0.5 units of the product and anything in between. The order control signal is denoted $u(n)$ at day k. It takes three days for the order to arrive at the stock. See Fig. 11.

After the stock order has been placed, the consumers buy v units of the product from the stock (and the products are removed immediately, without delay). $v(n)$ at day n is random and independent with the following probabilities:

$$v(n) = \begin{cases} 1, & \text{with probability 0.1} \\ 0.3, & \text{with probability 0.2} \\ 0, & \text{with probability 0.7.} \end{cases}$$

The cost of the system is a sum of the backlog cost when the stock is negative and the storing cost when the stock is positive. For each day with negative stock y, the cost is $l = -10y$. For each day with positive stock, the cost is $l = y$ (see Fig. 12).

Problem Formulation: This problem can be written as

$$\begin{aligned} x(n+1) &= \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} x(n) + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u(n) - \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} v(n) \\ y(n) &= [1 \quad 0 \quad 0] x(n) \\ & 0 \le u(n) \le 0.5. \end{aligned}$$

The step cost is

$$l(x,u) = \max\left([-10 \quad 0 \quad 0], [1 \quad 0 \quad 0]x\right)$$

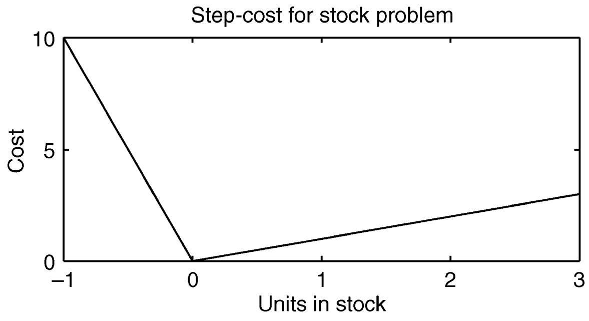


Fig. 12. Step cost for the stock example. Negative stock (backlog) is more expensive than storing.

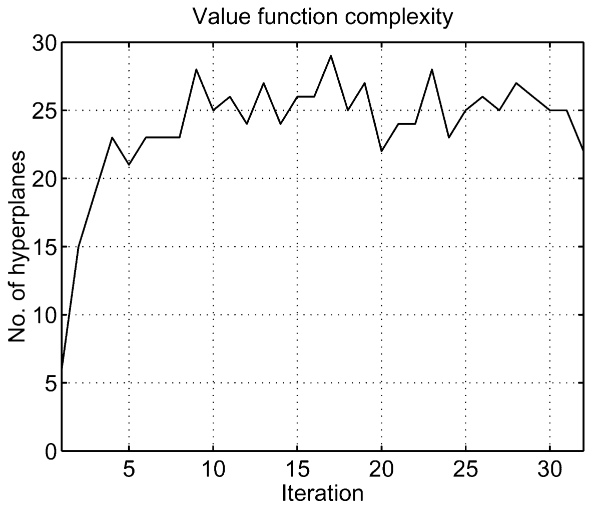


Fig. 13. Complexity of the value function for the stock example over iterations.

and the objective is to minimize the cost

$$\lim_{N\to\infty} \sum_{n=0}^{N} \lambda^n l\left(x(n), u(n)\right)$$

where the "forgetting factor" $\lambda = 0.9$. This factor ensures a finite value function, and, loosely speaking, weighs future versus immediate costs.

Solution: The method in Section IV-D is used to solve for a steady-state value function (and thus a control law). The random action of the consumer is accounted for by starting each iteration by forming

$$\tilde{V}_k(x) = V_{k-1}\left(x - \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}\right).$$

The values $\alpha_1 = 0.5$ and $\alpha_2 = 0.1$ turns out to be a good tradeoff between solution complexity and accuracy for this problem. With these parameters, our solution is guaranteed to have

$$0.5V_k^*(x) - 1 = 0.5V_k^*(x) - \frac{0.1}{1-\lambda} \le V_k(x) \le V_k^*(x)$$

for each iteration k. The value function has been iterated 32 times and the resulting complexity plot is shown in Fig. 13. As can be seen, at this level of accuracy, the value function can be described by less than 30 hyperplanes for all iterations. Using

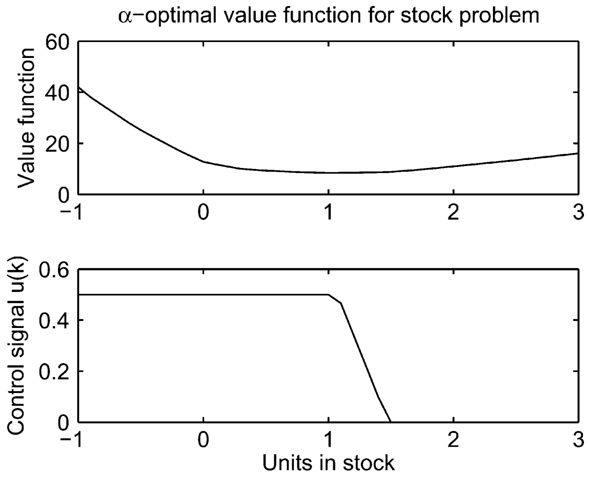


Fig. 14. Relaxed value function and corresponding control law for the stock control problem. This curve corresponds to $x_2 = x_3 = 0$, i.e., there are no units in the "pipeline."

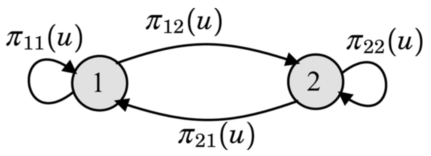


Fig. 15. Example POMDP Markov graph with a set of transition matrices $\pi(u)$, one for each control action u.

$V(x) = V_{32}(x)$ as the infinite-time horizon solution, it can be shown to obey

$$0.5V^*(x) - 1.51 \leq V(x) \leq V^*(x)$$

by iterating the lower bound one more step and checking which α_2 gives $\underline{V}_{k+1}(x) \leq V_k(x), \forall x$ (in this case, $\alpha_2 = 0.151$).

As the problem is 3-D, the resulting value function is somewhat hard to visualize. In Fig. 14, the two delay-states have been set to zero, resulting in a 1-D value function in "units-in-stock" (x_1). Apparently, with no units in the delay pipeline, a stock of about 1 unit is good.

V. Application 3: POMDPs

The problem of partially observable Markov decision processes (POMDPs) has been around for a long time (see e.g. [2], [12]). Lately, it has mostly been investigated in the AI/robotics fields, e.g. robot navigation problems where limited sensor information is available. A POMDP is a control problem where the state–space X is finite, as is the control signal (or action) space U and observation space Y. The dynamic programming procedure for this problem is very similar to the piecewise linear cost application in Section IV.

The state $x(n)$ is a Markov state and, and the dynamics is specified by transition matrix $\pi(u)$, where element m, l denotes the probability to move to state m if the system is currently in state l. This probability can be controlled by u. See Fig. 15 for an illustration.

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

$$P(y(n)|x(n) = m) = \Omega_m(y(n), u(n))$$

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 an MDP and easily solved). To be able to use dynamic programming, the state is changed to the belief state $z(n) : z_m(n) = P(x(n) = m)$. Note that state space is closed as $z_m(n) \geq 0, \forall m$ and $\sum_m z_m(n) = 1$. The dynamics of the belief state is linear

$$z(n+1) = \pi(u)z(n)$$

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

$$\begin{aligned} z_m(k|y(n)) &= P(x(n) = m|y(n)) \\ &= \frac{P(x(n) = m \cap y(n))}{P(y(n))} \\ &= \frac{P(y(n)|x(n) = m) \cdot P(x(n) = m)}{P(y(n))} \\ &= \frac{\Omega_m(y(n), u(n)) \cdot z_m(n)}{P(y(n)).} \end{aligned}$$

Thus, the expected state over all possible observations is

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

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

$$\mathbf{E}\sum_{n=0}^{\infty} \lambda^n l(x(n), u(n)) = \sum_{n=0}^{\infty} \lambda^n R(u(n))^T z(n)$$

where $R(u(n))$ is a vector of rewards of using control signal $u(n)$ for each Markov state $x(n)$, and $\lambda \leq 1$.

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 $u = \mu(z)$ which maximizes the reward for any initial state $z(n)$. As it turns out, again the value function $V_k(z)$ is of the form

$$V_k(z) = \max_{\Pi \in P_k} \Pi^T z(n) \qquad (22)$$

where P_k is a finite index set and Π_k is a vector. Thus, the value function is piecewise linear in the state z.

If the value function $V_{k-1}(z)$ is known and in the form (22), we can calculate the value $V_k(z)$ from Bellman's equation

$$\begin{aligned} V_k(z) &= \max_u \mathbf{E}\left\{V_{k-1}\left(\pi(u)z\right) + l(x,u)\right\} \\ &= \max_u \mathbf{E}\left\{\max_{\Pi\in P_{k-1}} \left(\Pi^T\pi(u) + R(u)^T\right) z\left(k|y(n)\right)\right\} \\ &= \max_u \sum_{y\in Y} \max_{\Pi\in P_{k-1}} \left(\Pi^T\pi(u) + R(u)^T\right)\Omega(y,u)z \\ &= \max_{\Pi\in P_k} \Pi^T z(n). \end{aligned}$$

Note that the "raw" size of P_k is significantly larger than P_{k-1} (actually, $|P_k| = |P_k|^{|Y|}|U|$, where $|U|$ denotes the number of elements in U).

A. Parsimonious Representation

Just like for the control problem in Section IV-C, the set P_k is often unnecessarily large and may be pruned without changing the value of $V_k(x)$. Procedure 3 can be used to obtain a parsimonious representation.

B. Relaxed Value Function

Analogous to Section IV-D, a modified pruning procedure can be used to obtain an α-optimal value function. The relaxed step cost is the same as in (20) and (21).

C. Example

There is a wide variety of reference POMDP problems defined in literature. In this section, we focus on the 4×3 Maze problem found in [8], which is a modified version from [14]. The state x is a position in a square 4×3 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

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

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-SOLV from [8] with incremental pruning and searching for the *optimal solution* fails to return within a reasonable time as the set P grows too fast (after ten iterations and 36 CPU-minutes on a fast PC the complexity is 3393).

Setting $\alpha_1 = 1$ and $\alpha_2 = 0.01$, the algorithm keeps a value function V of complexity (set size) of 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 $\approx$1.7. Using our α and the discount factor λ, we can bound the optimal value $V^*(x)$ by

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

A smaller α_2 produces a larger search and a tighter bound, and *vice versa*.

VI. Conclusion

A novel method for reduction of the computational complexity in dynamic programming has been presented. Applications to three well-known classes of optimal control problems show that the method has a potential for significant improvement compared to other approaches. Most likely, the same is true for many other application areas which still remain to be investigated.

Appendix

Proof of Convergence

A. Proof of Proposition 1

The assumption $V^*(f(x,u)) \le \gamma l(x,u)$ gives

$$\begin{aligned} V_1(x) &= \min_u \left[V_0\left(f(x,u)\right) + l(x,u)\right] \\ &\ge \min_u \left[\delta V^*\left(f(x,u)\right) + l(x,u)\right] \\ &\ge \min_u \left[\left(\delta + \frac{1-\delta}{\gamma+1}\right) V^*\left(f(x,u)\right)\right. \\ &\qquad \left. + \left(1 - \gamma\frac{1-\delta}{\gamma+1}\right) l(x,u)\right] \\ &= \frac{1+\delta\gamma}{\gamma+1} \min_u \left[V^*\left(f(x,u)\right) + l(x,u)\right] \\ &= \left[1 + \frac{\delta-1}{1+\gamma^{-1}}\right] V^*(x). \end{aligned}$$

The lower bound in (3) is obtained by repeating the argument k times.

References

[1] [Online]. Available: http://www.control.lth.se/publications/extra/minquadsolver.zip

[2] K. J. Åström, "Optimal control of Markov processes with incomplete state information I," *J. Math. Anal. Appl.*, vol. 10, pp. 174–205, 1965.

[3] R. E. Bellman*, Dynamic Programming*. Princeton, NJ: Princeton Univ. Press, 1957.

[4] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.

[5] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 2nd ed. Belmonth, MA: Athena Scientific, 2000.

[6] D. P. Bertsekas and J. Tsitsiklis, *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific, 1996, 1-886529-10-8.

[7] F. Borrelli, A. Bemporad, and M. Morari, "A geometric algorithm for multi-parametric linear programming," *J. Optim. Theory Appl.*, vol. 118, no. 3, pp. 525–540, 2003.

[8] A. R. Cassandra, Tony's POMDP page [Online]. Available: http://www.cs.brown.edu/research/ai/pomdp/

[9] G. B. Danzig, *Linear Programming and Extensions*. Princeton, NNJ: Princeton Univ. Press, 1963.

[10] D. P. de Farias and B. Van Roy, "Approximate dynamic programming via linear programming," in *Advances in Neural Information Processing Systems*. Cambridge, MA: MIT Press, 2002, vol. 14.

[11] T. A. Henzinger, H. Pei-Hsin, and H. Wong-Toi, "HYTECH: The next generation," *Proc. 16th IEEE Real-Time Systems Symp.* pp. 55–65, IEEE Comput. Soc. Press, 1995.

[12] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artif. Intell.*, vol. 101, pp. 99–134, 1998.

[13] A. Rantzer, "On relaxed dynamic programming in switching systems," *Proc. Inst. Elect. Eng. Control Theory Appl.*, Sep. 2006, to be published.

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

[15] P. J. Schweitzer and A. Seidmann, "Generalized polynomial approximations in Markovian decision processes," *J. Math. Anal. Appl.*, vol. 110, no. 2, pp. 568–582, Sep. 1985.

[16] V. A. Yakubovich, S-procedure in nonlinear control theory Vestnik Leningrad Univ., pp. 62–77, 1971, (English translation in Vestnik Leningrad Univ. 4:73–93, 1977).

Bo Lincoln received the M.Sc. degree from Linköping University, Linköping, Sweden, in 1999, and the Ph.D. degree in the areas of optimal control of switched systems and control systems with varying delays from the Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, in 2003.

He is with Combra AB, Lund, Sweden.

Anders Rantzer (S'90–M'91–SM'97–F'01) was born in 1963. He received the Ph.D. degree from KTH, Stockholm, Sweden.

After postdoctoral positions at KTH and the University of Minnesota, Minneapolis, he joined Lund University, Lund, Sweden, in 1993. In 1999, he was appointed professor of Automatic Control. His research interests are in modeling, analysis and synthesis of control systems, with particular attention to robustness, optimization and distributed control.

Prof. Rantzer was a winner of the 1990 SIAM Student Paper Competition and the 1996 IFAC Congress Young Author Prize. He has served as Associate Editor of the IEEE TRANSACTIONS ON AUTOMATIC CONTROL and several other journals.