Heuristic Refinements of Approximate Linear Programming for Factored Continuous-State Markov Decision Processes

Branislav Kveton and **Milos Hauskrecht**
Intelligent Systems Program and Department of Computer Science
University of Pittsburgh
{*bkveton,milos*}*@cs.pitt.edu*

Abstract

Approximate linear programming (ALP) offers a promising framework for solving large factored Markov decision processes (MDPs) with both discrete and continuous states. Successful application of the approach depends on the choice of an appropriate set of feature functions defining the value function, and efficient methods for generating constraints that determine the convex space of the solution. The application of the ALP in continuous state-space settings poses an additional challenge – the number of constraints defining the problem is infinite. The objective of this work is to explore various heuristics for selecting a finite subset of constraints defining a good solution policy and for searching the space of such constraints more efficiently. The heuristics that we developed rely upon: (1) the structure of the factored model and (2) stochastic state simulations to generate an appropriate set of constraints. The improvements resulting from such heuristics are illustrated on three large factored MDP problems with continuous states.

Introduction

Markov decision processes (MDPs) offer an elegant mathematical framework for representing and solving decision problems in the presence of uncertainty. While standard solution techniques, such as value or policy iteration scale-up well in terms of the number of states, the state space of more realistic MDP problems is factored and thus becomes exponential in the number of state components. This has prompted the development of efficient algorithmic solutions that fit well the factored models. Approximate linear programming (ALP) has emerged recently as one of the most promising methods for solving complex factored MDPs with discrete state components. The method relies on a value function model that consists of a linear combination of local feature functions (Van Roy 1998), such that every feature function is defined over a small number of state components. A number of refinements of the ALP approach have been developed over past few years. These include the work by (Guestrin, Koller, & Parr 2001), (de Farias & Van Roy 2002; 2001), (Schuurmans & Patrascu 2002), and others (Poupart *et al.* 2002).

Copyright © 2004, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

The ALP solutions are not limited to MDPs with discrete state spaces. Recently, (Hauskrecht & Kveton 2003) showed how one can extend the ALP approach to factored high dimensional continuous-state MDPs (CMDPs). The approach incorporates some aspects of the linear programming method developed by (Trick & Zin 1993). However, solutions by (Trick & Zin 1993) rely on the state-space discretization that is used to simplify both the value function and constraints. Moreover, examples that are addressed by (Trick & Zin 1993) consists of two-dimensional continuous-state spaces only. In contrast, the ALP problem formulation of (Hauskrecht & Kveton 2003) does not rely on the explicit state space discretization, and the value function model is optimized over the complete high dimensional continuous-state space.

The ALP formulation of the factored CMDP problem as proposed by (Hauskrecht & Kveton 2003) comes with a limitation: the number of constraints in the linear program (LP) is infinite. However, for the linear value function model only a finite number of active constraints define the optimal solution. The challenge is to identify these constraints or at least their good substitutes. Interestingly, (Hauskrecht & Kveton 2003) have shown that the ALP solution for factored CMDPs with constraints placed on a random grid is much better, both in terms of the running time efficiency and the solution quality, than the corresponding grid-based MDP (GMDP) approximation. Since grid-based approximations are the most common methods to solve large CMDPs, this result illustrates potential benefit and impact of the ALP on the solution of large CMDPs.

Heuristic constraint generation. A number of improvements and speed-ups of the basic ALP algorithm are possible. Random generation of constraints is blind and does not take any advantage of the problem definition. Since we want to identify a finite subset of constraints that give a very good policy as quickly as possible, the design of heuristics that tend to select constraints with a positive impact on the quality of the solution is desirable. In this work, we develop and test a heuristic constraint generation method based on the Monte Carlo simulation of states. In the ALP, every constraint is associated with a point in a multidimensional state space, so the rational behind the Monte Carlo simulation is to focus on the points and their corresponding constraints that are likely to be visited more often.

Heuristic constraint filtering. It is unlikely that one would be able to identify a good set of constraints in one shot. So some search process may be necessary to assure a good constraint coverage. In such a case, the efficiency with which we can generate constraints and solve ALPs defined upon such constraints affect tremendously the efficiency of our final solution. While heuristic constraint generation may help us to identify potentially useful constraints, many of the generated constraints can still be redundant in that they do not contribute to the solution. Since the running time complexity of the linear program solver depends on the number of constraints, it is equally important, for the sake of efficiency, to filter out as many redundant constraints as possible. In this work we develop and present a simple but very powerful constraint filtering approach called greedy constraint selection (filtering) that can be easily combined with incremental and iterative search approaches. The filtering is a heuristic since it focuses only on constraints that can improve the solution directly in one step and ignores other useful constraints without immediate effect on the solution.

Exploiting the structure. The factored CMDP model offers a great deal of additional structure that can be used to speed-up the computations in addition to various constraint-coverage heuristics. One structure-based refinement that we have devised and implemented takes advantage of the local effect of actions in factored CMDP settings and leads to significant computational savings on our test problems. The approach optimizes the order of computation of structurally related subtasks that occur, for example, during constraint and policy evaluations, so that the overlapping calculations are tied and performed together.

In the following, we first review continuous-state MDPs (CMDPs), their factored refinements and efficient approximations that one can apply to solve them. After that we focus on the ALP approach and describe its application to factored CMDPs. Next we develop two different heuristics for speeding up the calculations of the ALP policy and for improving its quality. Finally, we test the effects of the heuristics on three CMDP problems.

Continuous-state MDPs

A **continuous-state MDP (CMDP)** is defined by a 4-tuple $(\mathbf{x}, A, T, R)$, where $\mathbf{x}$ is a state space defined on $\mathbb{R}^n$, A is a finite set of actions, T a transition model defining the transition density $p(\mathbf{x}'|\mathbf{x}, a)$ between states $\mathbf{x}$ and $\mathbf{x}'$ under an action a, and R defines a reward function $R(\mathbf{x}, a)$ that maps state-action pairs to real-valued rewards.

Factored CMDPs. The factored version of the CMDP simplifies the definition of the decision process and decomposes transition and reward models along individual state dimensions $(x_1, x_2, \ldots, x_n)$. In particular, the transition function $p(\mathbf{x}'|\mathbf{x}, a)$ is assumed to be factored as:

$$p(\mathbf{x}'|\mathbf{x}, a) = \prod_{x'_j} p(x'_j|\mathbf{x}_{j,a}, a),$$

where $\mathbf{x}_{j,a}$ are state components influencing the next value of x'_j under an action a. Similarly, we assume that rewards are factored over smaller subsets of states into:

$$R(\mathbf{x}, a) = \sum_{i=1}^{m} R_{a,i}(\mathbf{x}_{a,i}, a).$$

Optimization criterion. Given a CMDP, our objective is to find a control policy $\pi^* : \mathbf{x} \rightarrow A$ that maximizes the infinite-horizon, discounted reward $E(\sum_{i=0}^{\infty} \gamma^i r_i)$, where $\gamma \in [0, 1)$ is a discount factor, and r_i is a reward obtained in time step i.

Bellman equation. The value function $V(\mathbf{x})$ of the optimal policy satisfies the Bellman fixed point equation (Bellman 1957):

$$V(\mathbf{x}) = \max_a \left[R(\mathbf{x}, a) + \gamma \int_{\mathbf{x}'} p(\mathbf{x}'|\mathbf{x}, a) V(\mathbf{x}') d\mathbf{x}' \right].$$

The main problem in solving CMDPs is that in most cases the recursive integral problem cannot be solved in a closed form, and there exists no finite support for the description of the optimal value function. To solve the problem, either the value function or the optimal policy is replaced with a finite approximation.

Grid-based MDP (GMDP) discretizations. A typical solution is to discretize the state space to a set of grid points and approximate value functions over such points. Unfortunately, classic grid algorithms scale-up exponentially with the number of state factors n (Chow & Tsitsiklis 1991). New approximation algorithms based on random and pseudorandom grids (Rust 1997) offer more flexibility together with a good accuracy-confidence guarantees. Let $G = \{\mathbf{x}^1, \mathbf{x}^2, \ldots \mathbf{x}^N\}$ be a set of grid points of the state space $[0, 1]^n$. Then the Bellman operator H can be be approximated with an operator H_G that is restricted to grid points G. One such operator has been studied by (Rust 1997) and is defined as:

$$V_G(\mathbf{x}^i) = \max_a \left[R(\mathbf{x}^i, a) + \gamma \sum_{j=1}^{N} P_G(\mathbf{x}^j|\mathbf{x}^i, a) V_G(\mathbf{x}^j) \right],$$

where $P_G(\mathbf{x}^j|\mathbf{x}^i, a) = \psi_a(\mathbf{x}^i) p(\mathbf{x}^j|\mathbf{x}^i, a)$ defines a normalized transition probability such that $\psi_a(\mathbf{x}^i)$ is a normalizing constant. The equation applied to grid points G defines a finite state MDP with $|G|$ states. [1] The solution, $V_G = H_G V_G$, approximates the original CMDP. Convergence properties of the grid-approximation scheme for random or pseudo-random samples were analyzed (Rust 1997).

Parametric function approximations. An alternative way to solve a continuous-state MDP is to approximate the optimal value function $V(\mathbf{x})$ with an appropriate parametric function model (Bertsekas & Tsitsiklis 1996). The parameters of the model are fitted iteratively by applying one step Bellman backups to a finite set of state points arranged on a fixed grid or obtained through Monte Carlo sampling. Least squares criterion is used to fit the parameters of the model.

[1] The operator H_G defines a special convex approximator for grids with a fixed point solution (see (Hauskrecht 1997) or (Gordon 1999)). Other examples include the nearest neighbor or barycentric interpolators (Munos & Moore 1999)

In addition to parallel updates and optimizations, on-line update schemes based on gradient descent (Bertsekas & Tsitsiklis 1996; Sutton & Barto 1998) are very popular and can be used to optimize the parameters. The disadvantage of the methods is their instability and possible divergence (Bertsekas 1994).

Approximate linear programming

Recently, (Hauskrecht & Kveton 2003) have proposed the approximate linear programming (ALP) approach as an alternative method for solving large factored continuous-state MDPs. Similarly to factored discrete-state MDP settings (Koller & Parr 1999; Guestrin, Koller, & Parr 2001), the new approach builds upon the linear model of the value function (Van Roy 1998):

$$V(\mathbf{x}) = \sum_i w_i f_i(\mathbf{x}_i),$$

where $f_i(\mathbf{x}_i)$s denote feature functions defined over subsets of state variables $\mathbf{x}_i$, and w_is are weights that are fit by the model. In terms of optimization, the benefit of the linear value function model is that it allows one to convert the optimization of the value function over a very complex state space to the optimization over a small set of weights.

Assuming that the state space of a CMDP is bounded to the region $[0,1]^n$, (Hauskrecht & Kveton 2003) showed that the optimization of the value function over the complete state space can be expressed in terms of the following approximate linear program (ALP):

$$\begin{aligned} \text{minimize}_{\mathbf{w}}: \quad & \sum_i w_i \int_{\mathbf{x}_i} f_i(\mathbf{x}_i) d\mathbf{x}_i \\ \text{subject to:} \quad & \sum_i w_i F_i(\mathbf{x}, a) - R(\mathbf{x}, a) \geq 0, \\ & \forall \mathbf{x} \in \mathbf{X}, a \in A \end{aligned}$$

where

$$F_i(\mathbf{x}, a) = f_i(\mathbf{x}_i) - \gamma \int_{\mathbf{x}'_i} \left(\prod_{x'_j \in \mathbf{x}'_i} p(x'_j | \mathbf{x}_{j,a}, a) \right) f_i(\mathbf{x}'_i) d\mathbf{x}'_i.$$

Conjugate choices. The ALP formulation of the CMDP assumes that all integrals in the objective function and constraints are proper integrals. Also important is the existence of analytical solutions of integrals in the objective function and constraints. To assure both of these conditions, (Hauskrecht & Kveton 2003) have devised conjugate classes of feature functions and transition models. The matching pairs include transitions based on beta or mixture of betas densities, where beta density is defined as:

$$p(x_j | \mathbf{x}_{j,a}, a) = Beta(x_j | g^1_{j,a}(\mathbf{x}_{j,a}), g^2_{j,a}(\mathbf{x}_{j,a})), \qquad (1)$$

and basis functions that are products of factors:

$$f_i(\mathbf{x}_i) = \prod_{x_j \in \mathbf{x}_i} x_j^{m_{j,i}}.$$

In such a case the integrals in the objective function simplify to (Hauskrecht & Kveton 2003):

$$\int_{\mathbf{x}_i} f_i(\mathbf{x}_i) d\mathbf{x}_i = \int_{\mathbf{x}_i} \prod_{x_j \in \mathbf{x}_i} x_j^{m_{j,i}} d\mathbf{x}_i = \prod_{x_j \in \mathbf{x}_i} \frac{1}{m_{j,i}+1},$$

and the integrals in constraints simplify to:

$$\int_{\mathbf{x}'_i} \left(\prod_{x'_j \in \mathbf{x}'_i} p(x'_j | \mathbf{x}_{j,a}, a) \right) f_i(\mathbf{x}'_i) d\mathbf{x}'_i = \prod_{x'_j \in \mathbf{x}'_i} \frac{\Gamma(g^1_{j,a}(\mathbf{x}_{j,a}) + g^2_{j,a}(\mathbf{x}_{j,a}))\Gamma(g^1_{j,a}(\mathbf{x}_{j,a})) + m_{j,i})}{\Gamma(g^1_{j,a}(\mathbf{x}_{j,a}) + g^2_{j,a}(\mathbf{x}_{j,a}) + m_{j,i})\Gamma(g^1_{j,a}(\mathbf{x}_{j,a}))},$$

where $\Gamma(.)$ is a gamma function.

The new ALP formulation for factored CMDPs is similar to the ALP formulation for factored discrete-state MDPs (Schuurmans & Patrascu 2002). In particular, the ALP optimizes the weights of the linear model and the objective function and constraints decompose over state subspaces associated with individual feature functions. The fact that we optimize over the finite set of weights means that the number of active constraints defining the optimal solution is finite. So our ultimate objective is to identify active constraints or, at least, they good surrogates.

The main difference between the ALPs for the two models is that the linear program built for a CMDP has infinite number of constraints; one for each state $\mathbf{x}$ and action a, while the number of constraints in a factored discrete-state MDPs is finite, though exponential in the number of state variables. Existing methods for solving ALPs for factored finite-state MDPs (Guestrin, Koller, & Parr 2001; Schuurmans & Patrascu 2002) take advantage of local constraint decompositions and various heuristics to search for the set of active constraints. However, at the end, all of these methods depend on the fact that the decompositions are defined on a finite state subspace that can be enumerated. Unfortunately, constraints in the CMPD model decompose over smaller but still continuous subspaces, so the existing solutions for the finite-state MDPs cannot be applied directly. To address this problem, (Hauskrecht & Kveton 2003) applied and tested constraint generation methods based on sampling. They showed a very good performance of the approach when compared to two standard approaches for solving CMDPs: the grid state-space discretization and approximate dynamic programming with the least squares fit.

Searching the space of constraints

The objective of this research is to develop methods for solving ALPs for CMDPs that can replace random constraint sampling and identify a finite subset of constraints defining a good-quality solution in a more principled way. Unfortunately, in general, it is very hard to come up with a good set of constraints in one shot. Typically, a much better option is to generate the set gradually in multiple steps, such that in every step, the procedure takes advantage of the previously found solution, and applies it to build a new set of constraints. In the following, we describe incremental and iterative forms of such a procedure.

function **ALP-incremental**($model$)
  $\mathcal{C} \leftarrow$ initialize constraints
  $\mathbf{w} \leftarrow$ ALP($\mathcal{C}$)
  while a stopping criterion is not met
    $\mathcal{C}_{new} \leftarrow$ generate new constraints from $model$ and $\mathbf{w}$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}_{new}$
    $\mathbf{w} \leftarrow$ ALP($\mathcal{C}$)
  return $\mathbf{w}$

Figure 1: ALP with incremental generation of constraints. The most recent solution of the ALP, represented by weights $\mathbf{w}$, is used together with the model to generate a new set of constraints $\mathcal{C}_{new}$. In every iteration, the existing set of constraints $\mathcal{C}$ is extended with the set $\mathcal{C}_{new}$.

function **ALP-iterative**($model$)
  $\mathcal{C} \leftarrow$ initialize constraints
  $\mathbf{w} \leftarrow$ ALP($\mathcal{C}$)
  while a stopping criterion is not met
    $\mathcal{C} \leftarrow$ generate new constraints from $model$ and $\mathbf{w}$
    $\mathbf{w} \leftarrow$ ALP($\mathcal{C}$)
  return $\mathbf{w}$

Figure 2: ALP with iterative generation of constraints. The most recent solution of the ALP, represented by weights $\mathbf{w}$, is used together with the model to generate a new set of constraints $\mathcal{C}_{new}$. In every iteration, the existing set of constraints $\mathcal{C}$ is replaced with the set $\mathcal{C}_{new}$.

Incremental approach. In the incremental approach (Figure 1), an initial set of constraints is gradually expanded such that the feedback from the previous-stage solution is actively used to generate a new set of constraints. The new constraints are added to the existing set of constraints, and the procedure is repeated until a stopping criterion is met.

Iterative approach. In the iterative approach (Figure 2), the set of constraints is build from an empty set in every step. The procedure starts with an initial set of constraints, uses feedback from the most recent solution to generate new constraints, and finally replaces the set of existing constraints with the new constraints. The process is repeated until a stopping criterion is met.

Both iterative and incremental search procedures can come in different guises depending on the methods employed to generate constraints. In the following, we propose and explore a number of heuristic approaches for generating constraints that can improve tremendously the quality of the resulting solution as well as the efficiency of the search process.

Heuristics for constraint generation

State space simulation

The ALP methods for solving CMDPs presented by (Hauskrecht & Kveton 2003) use randomly generated constraints. Surprisingly, even in this case, the solutions were much better (in terms of both the quality and the efficiency) than solutions obtained for the corresponding grid-based approaches. An open issue is whether it is possible to choose constraints that would reflect better the underlying CMDP model and its dynamics.

In the ALP, randomly generated points lead to a uniform coverage of the state space by constraints. This may not reflect the differences in the importance among different regions of the state space. Intuitively, we would like to have a better coverage of the state space regions that affect the solution the most. To address this problem, we propose a Monte Carlo method, in which the constraints are selected by simulating the state space according to the currently available policy π. The method prefers points (and subsequently constraints) in the regions that are visited with a higher probability. The intuition behind this heuristic is that the regions that are visited more likely are also more important for the quality of the solution.

The state space simulation heuristic can be easily incorporated into the incremental and iterative approaches, and used to generate new constraints. In one scenario, one can pick randomly a starting point in the state space and simulate a trajectory of k steps according to the currently available policy π. The point reached after k steps is selected as the point that defines a new set of constraints. As there is a constraint associated with every point and action, by adopting this Monte Carlo strategy, a total number of $|A|$ constraints can be generated. The simulation can be repeated multiple times using trajectories of the same or different length. Alternatively, one can use the same simulation trajectory to choose more than one point.

Greedy constraint selection

Every constraint in the ALP is associated with an n-dimensional point $\mathbf{x}$ and an action a, so we can use a pair $(\mathbf{x}, a)$ to identify a constraint. A finite set of constraints defines a convex space, a simplex, and the solution of the corresponding linear program is in one of its corners. Constraints that are critical for the solution are denoted as active constraints. Assuming that $\mathbf{w}^*$ is the optimal solution, all active constraints satisfy the condition

$$\sum_i w_i^* F_i(\mathbf{x}, a) - R(\mathbf{x}, a) = 0.$$

Other constraints in the linear program are inactive.

An addition of a new constraint $(\mathbf{x}, a)$ changes the existing solution $\mathbf{w}^*$ only if the new constraint violates the solution, which means that

$$\sum_i w_i^* F_i(\mathbf{x}, a) - R(\mathbf{x}, a) < 0. \quad (2)$$

This condition offers a simple but in practice very powerful way of filtering constraints from a set of candidate constraints. Simply, only a constraint that violates the current

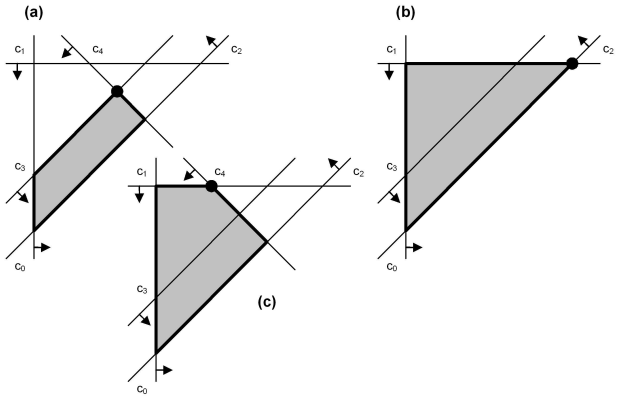

Figure 3: Graphical representation of simplexes and ALP solutions for (a) constraints $\{c_0, c_1, c_2, c_3, c_4\}$, (b) constraints $\{c_0, c_1, c_2\}$, and (c) constraints $\{c_0, c_1, c_2, c_4\}$. The simplexes are represented by gray polygons, and the corresponding solutions of the ALPs by black circles. The arrows at end of constraints point to the subspaces of solutions that are not violated by the constraints.

solution $\mathbf{w}^*$, and thus is guaranteed to immediately improve the solution, is chosen. We refer to such a method as to *greedy constraint selection* since it greedily focuses only on the constraints that assure immediate improvement of the current solution $\mathbf{w}^*$. We note that a similar idea for MDPs with much simpler state spaces was investigated by (Trick & Zin 1993). The method gained significant computational savings as compared to linear programs with all constraints.

Unfortunately, the greedy constraint selection is only a heuristic since the constraint that does not violate the condition can become active later in a context of newly added constraints. To illustrate this, let us consider a CMDP problem with two continuous state variables. Let us assume that its ALP approximation relies on five constraints $\{c_0, c_1, c_2, c_3, c_4\}$. The solution simplex defined by such constraints is illustrated in Figure 3a, and the solution $\mathbf{w}^*$ is represented by a black circle. The active constraints, which determine the solution $\mathbf{w}^*$, are c_3 and c_4. Now let us consider the case in which only the first three constraints $\{c_0, c_1, c_2\}$ are used to solve the ALP. The solution $\mathbf{w}'$ of the ALP is shown in Figure 3b. It is clear that the ALP solution $\mathbf{w}'$ for this subset differs from the optimal solution $\mathbf{w}^*$ obtained for all five constraints, which implies that at least one of the constraints $\{c_3, c_4\}$ is critical for the solution $\mathbf{w}^*$. Unfortunately, if the next constraint tested by the greedy criterion is c_3, it fails the greedy test, since it cannot change the solution $\mathbf{w}'$ alone. Thus, the greedy test fails to pick a constraint that is important for the final solution. However, note that the constraint c_3 becomes active (and passes the greedy test) if the constraint c_4 is added to the set of constraints $\{c_0, c_1, c_2\}$ before c_3 is tested. This is illustrated in Figure 3c. In general, if there exists a better solution, there is at least one constraint that passes the greedy test.

Using greedy constraint selection to solve the ALP. Greedy constraint filtering is especially useful when combined with incremental or iterative search approaches. This filtering is also at the heart of the method that we call *greedy incremental ALP (GI-ALP)*, which can be viewed as a heuristic speed-up of the ALP solver.

function **GI-ALP**($\mathcal{C}$)
  partition $\mathcal{C}$ into $\mathcal{C}_1, \ldots, \mathcal{C}_m$
  initialize linear program lp
  for $(\mathbf{x}, a) \in \mathcal{C}_1$
    add $(\mathbf{x}, a)$ to lp
  $\mathbf{w} \leftarrow$ solve lp
  for $j \leftarrow 2$ to m
    for $(\mathbf{x}, a) \in \mathcal{C}_j$
      if $(\sum_i w_i F_i(\mathbf{x}, a) - R(\mathbf{x}, a) < 0)$
        add $(\mathbf{x}, a)$ to lp
    $\mathbf{w} \leftarrow$ solve lp
  return $\mathbf{w}$

Figure 4: Greedy incremental ALP (GI-ALP).

Greedy incremental ALP (GI-ALP). Let $\mathcal{C} = \{c_1, c_2, \ldots, c_N\}$ be a set of N candidate constraints we consider adding to the ALP. In the first step, the procedure tests constraint c_1 whether it passes the greedy test. If the constraint passes, it is added to the linear program, and the solution of the LP is recomputed. In the next step, constraint c_2 is tested, and this process of filtering constraints continues until all constraints are scanned.

In more a general setting, the GI-ALP method tests whole sets of constraints. If $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m$ is a partitioning of $\mathcal{C}$ into m mutually exclusive constraint sets, the procedure starts by testing constraints in the first partition $\mathcal{C}_1$. Those constraints that pass the greedy test are added to the linear program, and the LP is resolved. In the next step, constraints from the set $\mathcal{C}_2$ are tested, and the algorithm continues until all partitions are scanned. An outline of this procedure is presented in Figure 4.

Partitioning and ordering. It is easy to see that the time complexity of the algorithm and the quality of its solution depends on the partitioning of $\mathcal{C}$ and the order in which the partitions are examined. If the number of partitions m is set to 1, the algorithm turns into the standard ALP. Setting $m = N$ corresponds to the case when only one constraint is added at time. In such a case, the linear program is recomputed only if a new constraint violates the current solution. Another interesting partitioning divides constraints into partitions of unequal cardinality, such as $|\mathcal{C}_i| = 2^i$. The rational behind this choice is that if the probability of detecting an active constraint decreases exponentially with time step i, the total number of added constraints to the LP in every partition $\mathcal{C}_i$ stays constant. The results presented in the experimental section are obtained for this partitioning.

The GI-ALP is a heuristic in terms of the optimal solution $\mathbf{w}^*$ and its corresponding policy π^*. However, in highly distributed environments, our experimental results show that the solutions obtained by GI-ALP are very close to those of

the ALP. Moreover, in terms of computation time, several-fold speed-up is observed due to a smaller number of applied constraints.

Speed-ups due to the local effect of actions

CMDPs built for highly distributed environments tend to exhibit local effect of actions. We say that an action a has a local effect if it directly influences only a small subset of state components. This feature can be used to speed up the operations that require iterations over all possible actions, while the state component $\mathbf{x}$ is fixed. Example of such an operation is the evaluation of constraints in the ALP or the computation of a policy π. In the following, we show how the evaluation of a constraint factorizes due to local actions.

If an action a_1 affects factors $\mathbf{x}_{a_1}$, and an action a_2 affects factors $\mathbf{x}_{a_2}$, the evaluation of the constraint $(\mathbf{x}, a_2)$ can be rewritten as:

$$\sum_i w_i F_i(\mathbf{x}, a_2) - R(\mathbf{x}, a_2) =$$

$$\sum_{i \in \mathbf{t}_{a_1,a_2}} w_i F_i(\mathbf{x}, a_2) + \sum_{i \notin \mathbf{t}_{a_1,a_2}} w_i F_i(\mathbf{x}, a_1) - R(\mathbf{x}, a_2),$$

where

$$\mathbf{t}_{a_1,a_2} = \{i : \exists j : (j \in (\mathbf{x}_{a_1} \cup \mathbf{x}_{a_2})) \wedge (x_j \in \mathrm{dom}(f_i))\}$$

represents the set of feature function indices whose domain contains at least one of the factors from $\mathbf{x}_{a_1} \cup \mathbf{x}_{a_2}$. Due to the factorization above, the computation of the terms $F_i(\mathbf{x}, a)$ can be divided into two components: $i \notin \mathbf{t}_{a_1,a_2}$, which can be effectively cached from the same computation performed for the action a_1; and $i \in \mathbf{t}_{a_1,a_2}$, which are affected by either of the actions, and have to be recomputed. Thus, the computational savings are realized on subexpressions that are cached from previous computations.

This result is especially important if the terms $F_i(\mathbf{x}, a)$ are hard to compute because integrals do not have closed-form solutions. The savings are even larger if the constraints are evaluated for all actions. In such a case, one scans sequentially all actions while caching the terms from the most recent constraint, and reuses the results for the next constraint. We note that the computation of the reward part $R(\mathbf{x}, a_2)$ can be decomposed in a similar manner. We have optimized our ALP solvers and took advantage of both factorizations in all experiments.

Experiments

Experimental setup

To analyze the heuristics and their performance, we use three CMDP problems, each of them defined by one of the computer network topologies in Figure 5. The computer network examples were originally proposed for factored finite-state MDPs by (Guestrin, Koller, & Parr 2001). Continuous versions of these examples were introduced in (Hauskrecht & Kveton 2003).

A network consists of n connected computers, one of which is a server, and the remaining computers are workstations. The state of the jth computer x_j is represented by a real number between 0 and 1, which reflects the reliability of the computer to process tasks. The state of the network

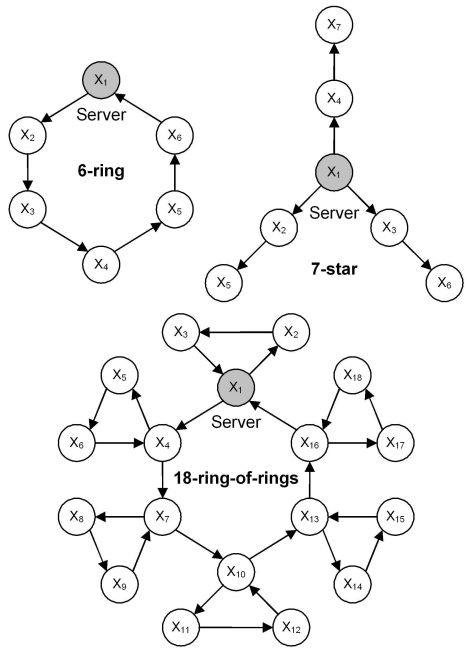

Figure 5: Example of computer network topologies 6-ring, 7-star, and 18-ring-of-rings. Server is marked by a gray circle.

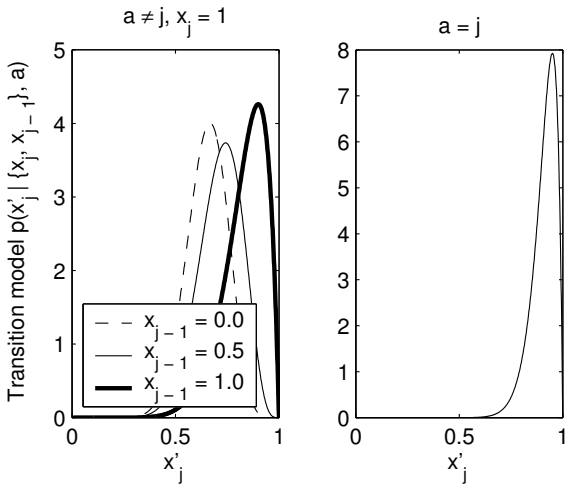

Figure 6: Example of transition functions for the ring network topology. If $a = j$, the transition function simplifies to Beta(20, 2). If $a \neq j$, the transition model is defined by Beta($2 + 13x_j - 5x_j x_{j-1}$, $10 - 2x_j - 6x_j x_{j-1}$).

$\mathbf{x}$ is defined by a reliability vector $(x_1, x_2, \ldots, x_n)$ of individual computers, such that x_1 is the state of the server. The topology of a network plays an important role in the operation of the network. In particular, an unreliable computer may affect the reliability of computers connected to it.

To improve the performance of the network and its processing power, we employ an administrator who maintains the computers, and is able to improve their reliability. There are $(n+1)$ actions that the administrator can perform. The action $j \in \{1, 2, \ldots, n\}$ means that the administrator attends the jth computer. The effect of the action is improved reliability of the attended computer. The action $(n+1)$ is a dummy action and represents the scenario when the administrator does nothing. Transition functions defining the dynamics of the network state over time are factored and defined using beta densities as presented in Equation 1. Figure 6 illustrates various transitions employed by the model.

The quality of the network operation is measured in terms of the reward function:

$$R(\mathbf{x}, a) = 2x_1^2 + \sum_{j=2}^{n} x_j^2,$$

that is given by a weighted sum of computer states. In our model, the highest weight is assigned to the server. The discount factor is $\gamma = 0.95$.

To approximate the optimal value function, a combination of linear (one per node of the network) and quadratic (one per link) feature functions is used. As discussed earlier, these are the conjugate choices that lead to closed-form solutions to integrals in the factored ALP.

Results

The comparison of the ALP method with random constraints to alternative CMDP solutions on computer network problems was performed by (Hauskrecht & Kveton 2003). The results presented in that work illustrate the benefit of the ALP approach both in terms of the running time and the quality of the resulting approximation. In this work, we focus primarily on the heuristics and use the ALP solution with random constraints as the baseline method.

Figures 7, 8, and 9 summarize the results of the experiments and comparisons. The graphs capture the quality of solution policies and running times of different methods while varying the number of state-space samples M. Since every sample defines $|A|$ different constraints and all are used in the ALP, the total number of constraints is $M|A|$. To evaluate the quality of every method we use averages over 30 different runs of the algorithm. The quality of the policy generated by each algorithm is estimated via simulation. To compute the estimate we use average of cummulative discounted rewards obtained for 100 simulation trajectories, each of length 50.

We run three sets of experiments. The objective of the first experiment was to evaluate the benefit of the state-simulation heuristic. Figure 7 shows the results for the 24-ring problem. The results obtained for other two network topologies (25-star and 24-ring-of-rings) are very similar

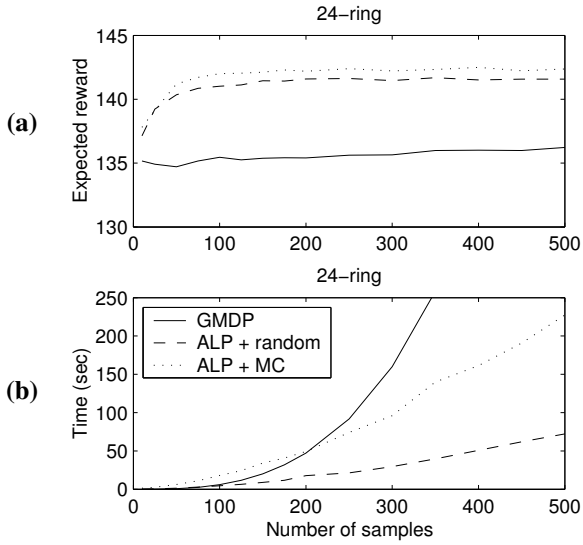

Figure 7: Comparison of the ALP with random constraints (ALP + random), the ALP-iterative with the state-simulation heuristic (ALP + MC), and the grid-based MDP (GMDP). Panel (a) shows the estimate of expected rewards of policies found by different methods and panel (b) their running times.

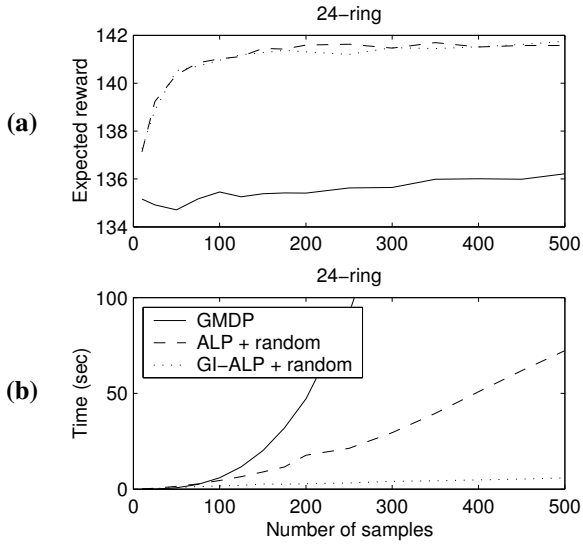

Figure 8: Comparison of the ALP with random constraints (ALP + random), the greedy incremental ALP (GI-ALP) with random constraints (GI-ALP + random), and the grid-based MDP (GMDP). Panel (a) shows the estimate of expected rewards of policies found by different methods and panel (b) their running times.

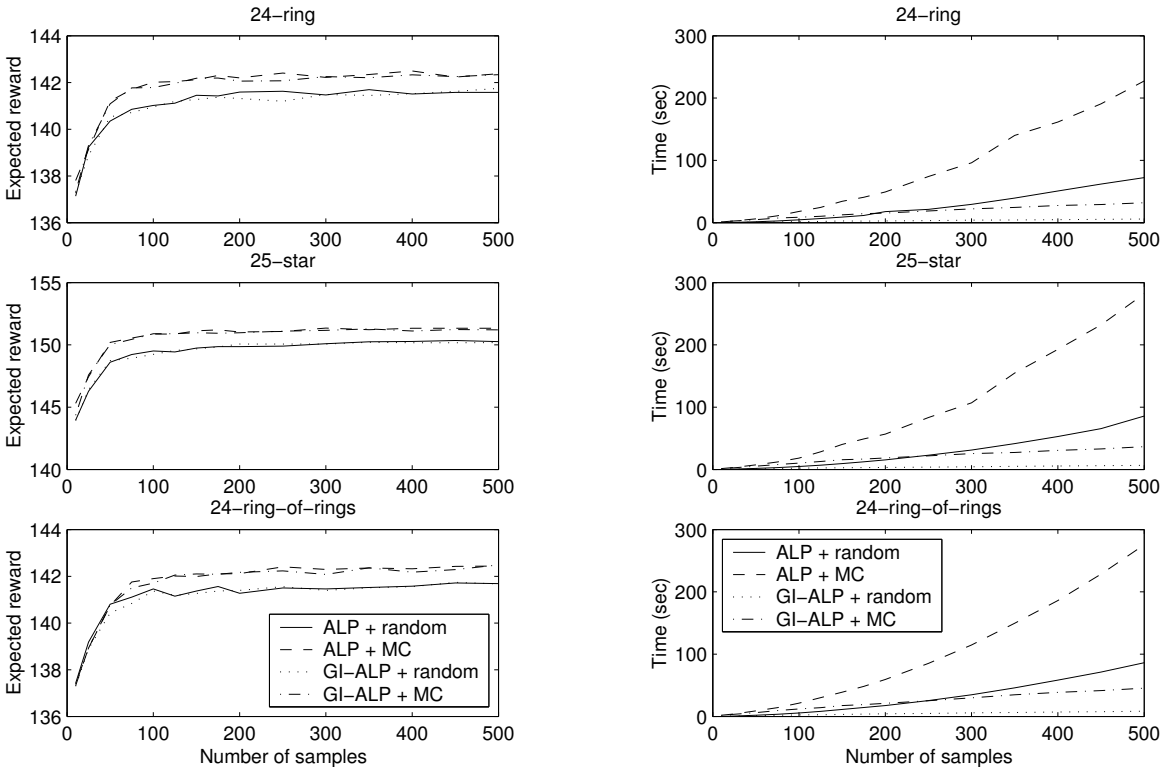

Figure 9: Comparison of estimated expected rewards of policies obtained through three heuristic methods and their running times on three network architectures. The ALP with random constraints is used as the baseline of the comparison.

and are not shown. Figure 7a compares the quality of policies obtained by the ALP with random constraint generation and the ALP with the iterative constraint search (ALP-iterative) powered with the state simulation heuristic. As a baseline we also include the results of the grid-based approximation (GMDP). The comparison shows that the ALP-iterative with the state simulation heuristic is able to improve over the ALP with random constraints. In terms of the quality of generated policies, both ALP methods outperform the GMDP approach. Figure 7b compares running times that are needed to solve the problem for different sample sizes. We see that the running time performance of the ALP-iterative with the state simulation heuristic is worse than the performance of the ALP with random constraints. This behavior is expected since the iterative method solves a sequence of ALPs, each of which is of the same size as the ALP solved by the random constraint generation. The GMDP is inferior to the ALP methods in terms of both the quality and the running times.

The objective of the second set of experiments was to compare the greedy incremental implementation of the ALP (GI-ALP) with random constraints to the standard ALP with random constraints. The GI-ALP uses exponentially increasing partitions as described earlier in the paper. Again, the corresponding grid-based approximation (GMDP) is used for the comparison. Figures 8a and 8b show the quality of generated policies and running times for all three algorithms. The results show that the GI-ALP is slightly worse in terms of solution quality when compared to the ALP, but at the significant savings in the running times. This evidence supports the intuition that the greedy constraint filtering is able to eliminate a large portion of unimportant constraints, and thus, to speed up the algorithm, while the solution quality drop due to the elimination of useful constraints is relatively small. Once again the GMDP is inferior both in terms of the solution quality and the running times.

The first experiment clearly shows the benefit of the state simulation in terms of the solution quality, but at the expense of the increase in the running time. On the other hand, the GI-ALP demonstrates the potential of a tremendous speed-up at the expense of the solution quality. An immediate question that arises is whether the combination of the two heuristics would be able to offset their individual deficiencies. To explore this issue, we have built an iterative version of the ALP powered with the state simulation heuristic such that every ALP to be solved inside the iterative procedure is solved using the GI-ALP method. Figure 9 compares the results of the new heuristic method with the two previous heuristics and the standard ALP with random constraints. The benefit of the new heuristic is evident. The quality of the solution on all three network topologies is very close to the one of the ALP with the state simulation heuristic. The running time of the new heuristic drops below the running times of ALPs without greedy constraint filtering.

Conclusions and future work

We have developed and tested two different heuristics for improving the performance of the ALP solvers for factored continuous-state MDPs. The improvements were achieved by appropriate selections of constraints: in the first case through Monte Carlo state simulations, which leads to a better coverage of more likely regions of the state space; and in the second case by rapid filtering of a large number of inactive constraints.

Despite of the recent progress in solving factored CMDPs, a number of issues remain open and have to be further explored. One promising direction is the application of local search methods that can be used to gradually optimize a fixed number of constraints. In such a case, each constraint is parameterized, and the parameters are optimized. Local search techniques based on gradient ascent represent one possible approach. The proof of sample complexity bounds for the ALP with random constraints remains another important open issue. We expect that the proof along the lines of the proof by (de Farias & Van Roy 2001) for discrete MDPs may be possible.

References

Bellman, R. E. 1957. *Dynamic programming*. Princeton, NJ: Princeton University Press.

Bertsekas, D. P., and Tsitsiklis, J. N. 1996. *Neuro-dynamic Programming*. Athena Scientific.

Bertsekas, D. P. 1994. A counter-example to temporal differences learning. *Neural Computation* 7:270–279.

Chow, C., and Tsitsiklis, J. 1991. An optimal one-way multigrid algorithm for discrete-time stochastic control. *IEEE Transactions on Automatic Control* 36:898–914.

de Farias, D. P., and Roy, B. V. 2001. On constraint sampling for the linear programming approach to approximate dynamic programming. *Mathematics of Operations Research* submitted.

de Farias, D., and Roy, B. V. 2002. Approximate dynamic programming via linear programming. In Dietterich, T. G.; Becker, S.; and Ghahramani, Z., eds., *Advances in Neural Information Processing Systems 14*. Cambridge, MA: MIT Press.

Gordon, G. 1999. *Approximate Solutions to Markov Decision Processes*. Ph.D. Dissertation, Carnegie Mellon University.

Guestrin, C.; Koller, D.; and Parr, R. 2001. Max-norm projections for factored MDPs. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, 673–682.

Hauskrecht, M., and Kveton, B. 2003. Linear program approximations for factored continuous-state Markov decision processes. In *the Proceedings of the 17th Annual Conference on Neural Information Processing Systems*.

Hauskrecht, M. 1997. *Planning and control in stochastic domains with imperfect information*. Ph.D. Dissertation, Massachusetts Institute of Technology.

Koller, D., and Parr, R. 1999. Computing factored value functions for policies in structured MDPs. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1332–1339.

Munos, R., and Moore, A. 1999. Variable resolution discretizations for high accuracy solutions of optimal control problems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1348–1355.

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

Roy, B. V. 1998. *Learning and value function approximation in complex decision problems*. Ph.D. Dissertation, Massachussetts Institute of Technology.

Rust, J. 1997. Using randomization to break the course of dimensionality. *Econometrica* 65:487–516.

Schuurmans, D., and Patrascu, R. 2002. Direct value-approximation for factored MDPs. In Dietterich, T. G.; Becker, S.; and Ghahramani, Z., eds., *Advances in Neural Information Processing Systems 14*. Cambridge, MA: MIT Press.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An introduction*. Cambridge, Mass: MIT Press.

Trick, M., and Zin, E. 1993. A linear programming approach to solving stochastic dynamic programs.