Tetris: Experiments with the LP Approach to Approximate DP

Vivek F. Farias Electrical Engineering Stanford University Stanford, CA 94403 vivekf@stanford.edu Benjamin Van Roy Management Science and Engineering and Electrical Engineering Stanford University Stanford, CA 94403 bvr@stanford.edu

Abstract

We study the linear programming (LP) approach to approximate dynamic programming (DP) through experiments with the game of Tetris. Our empirical results suggest that the performance of policies generated by the approach is highly sensitive to how the problem is formulated and the discount factor. Furthermore, we find that, using a state-sampling scheme of the kind proposed in [7], the simulation time required to generate an adequate number of constraints far exceeds the time taken to solve the resulting LP.

As an extension to the standard approximate LP approach, we examine a bootstrapped version wherein a sequence of LPs is solved, with the policy generated by each solution being used to sample constraints for the next LP. Our empirical results demonstrate that this bootstrapped approach can amplify performance.

1 Introduction

There has been significant recent interest in the linear programming (LP) approach to approximate dynamic programming (DP) (e.g., [12, 1, 6, 7, 10, 9]). In this paper we further the study of this approach through experiments with the game of Tetris. In particular, we apply the LP approach to fit a linear combination of basis functions to a value function. We employ the collection of twenty-two basis functions introduced in [3], where a variation of temporal-difference learning was applied to fit their linear combination to a value function. This same collection of basis functions was also used in the study of a policy gradient method in [11]. These prior studies involving the same game of Tetris and the same basis functions offer a context for assessing the LP approach.

We focus on the LP formulation as described in [6]. In the context of our study, this formulation leads to an LP with twenty-two variables, which is easily manageable. However, the objective function involves the summation of an astronomical number of terms and there are an even larger number of constrains. In order to approximate the sum and reduce the number of constraints, we employ the state sampling scheme proposed in [7]. The approach requires us to choose a sampling distribution. To this end, we use as samples, states visited by a heuristic game-playing policy. Some trial-and-error was required to generate a good policy. In particular, we found that the performance of policies generated by the LP approach is highly sensitive to how the problem is formulated and the discount factor. We were not able to produce an effective policy when formulating the problem as one of maximizing the number of rows eliminated during the course of the game. However, formulating the problem as one of minimizing a discounted time-average of Tetris wall height lead to good policies. We also found that performance is sensitive to the choice of discount factor.

In terms of compute time, we are encouraged by the fact that the LP can be solved by commercial software in minutes, even on a low-end personal computer. However, the simulation required by our sampling scheme, which produces data for the LP, calls for hours of computation.

As an extension to the standard approximate LP approach, we examine a bootstrapped version wherein a sequence of LPs is solved. The samples for the first LP are drawn from states visited by a heuristic policy. For each subsequent LP, the samples are generated based on the policy produced by the previous LP. Our empirical results demonstrate that this bootstrapped approach can amplify performance.

The remainder of this paper is organized as follows. In Section 2, we review the LP approach to approximate DP. In Section 3, we describe a dynamic programming formulation for the problem of playing Tetris and a basic implementation of the LP approach for this problem. In Section 4 we present our bootstrapped approach. Finally, in Section 5, we present experimental results, along with a discussion.

2 The LP approach

We consider a discrete-time finite-state stochastic system. The state takes on values in a finite set \mathcal{X} , and in each state $x \in \mathcal{X}$, there is a finite collection \mathcal{A}_x of actions from which we can choose. Given a current state $x \in \mathcal{X}$, a particular choice of action $a \in \mathcal{A}_x$, and any state $y \in \mathcal{X}$, the probability that the system transitions to state y at the next time step is $P_a(x, y)$. We allow $\sum_{y \in \mathcal{X}} P_a(x, y)$ to be less than one, the residual probability being associated with the event that the system terminates. We take P_a to be a matrix whose xyth element is $P_a(x, y)$, noting that this is a substochastic matrix.

A policy is a mapping from state to action. Under a policy u, the system follows a Markov chain with transition probabilities $P_{u(x)}(x, y)$. We denote by P_u the transition matrix associated with this Markov chain, noting that this again is a substochastic matrix.

A cost of g(x, a) is incurred each time the system is in state x and action a is taken. Let $g_u(x) = g(x, u(x))$. The expected cost-to-go $J_u(x)$ associated with a policy u is defined by

$$J_u = \sum_{t=0}^{\infty} \alpha^t P_u^t g_u,$$

where $\alpha \in (0, 1)$ is a discount factor reflecting temporal preferences. We consider the problem of finding a policy u^* that minimizes $J_u(x)$ simultaneously for all $x \in \mathcal{X}$.

Define the dynamic programming operator $T: \Re^{|\mathcal{X}|} \to \Re^{|\mathcal{X}|}$ by

$$(TJ)(x) = \min_{a \in \mathcal{A}_x} \{g(x,a) + \alpha(P_a J)(x)\}.$$

Then, the optimal cost-to-go function $J^*(x) = \min_u J_u(x)$ is the unique solution to Bellman's Equation; that is, $TJ^* = J^*$. Furthermore, any policy u^* satisfying

$$u^*(x) \in \underset{a \in \mathcal{A}_x}{\operatorname{argmin}} \{g(x, a) + \alpha(P_a J)(x)\},\$$

is optimal, in the sense that $J_{u^*} = J^*$.

It can be shown that J^* is the unique optimal solution to the optimization problem

$$\begin{array}{ll} \text{maximize} & c'J\\ \text{subject to} & TJ \ge J, \end{array}$$

given any vector $c \in \Re^{|\mathcal{X}|}$ for which every component is strictly positive. It is well-known that this optimization problem can be converted to an LP [2]. For large-scale problems, this LP is intractable since the numbers of variables and constraints grow proportionately with the number of states $|\mathcal{X}|$. We therefore attempt to approximate J^* by a linear combination of basis functions $\phi_i : \mathcal{X} \to \Re, i = 1, ..., n$. Denoting by Φ the $\Re^{|\mathcal{X}| \times n}$ matrix with *i*th column ϕ_i , we define an "approximate LP" (ALP):

$$\begin{array}{ll} \text{maximize} & c' \Phi r \\ \text{subject to} & T \Phi r \geq \Phi r. \end{array}$$

The ALP may has N variables, but the number of terms being summed in the inner product $c' \Phi$ and the number of constraints both grow proportionately with the number of states $|\mathcal{X}|$. We therefore consider a "reduced LP" (RLP):

 $\begin{array}{ll} \text{maximize} & \sum_{x \in \overline{\mathcal{X}}} (\Phi r)(x) \\ \text{subject to} & (T \Phi r)(x) \geq (\Phi r)(x), \qquad \forall x \in \overline{\mathcal{X}}. \end{array}$

where $\overline{\mathcal{X}}$ a set of states, sampled with replacement from \mathcal{X} , sampled with relative frequencies equal to the components of *c*. Analyses of the ALP and RLP can be found in [6] and [7], respectively.

As discussed in [6], the ALP objective of maximizing $c'\Phi r$ is equivalent to minimizing $||J^* - \Phi r||_{1,c}$, where the norm is defined by

$$||J||_{1,c} = \sum_{x \in \mathcal{X}} c(x) |J(x)|,$$

for any J. Hence, the vector c represents a weighting of approximation errors across states. It is also suggested in [6] that it may be desirable to choose weights such that they reflect relative frequencies of visits to various states under a reasonable policy. In solving the RLP, a natural approach to generating the set of samples $\overline{\mathcal{X}}$ is to collect a list of states visited by a reasonable heuristic policy.

3 Tetris

Tetris is a video game in which falling "bricks" are positioned on a two dimensional grid of width 10 and height 20. A point is received for each row constructed without any holes, and the corresponding row is cleared. The game ends once the height of the wall crosses a particular threshold. The objective is to maximize the expected number of points accumulated over the course of the game. Tetris may be viewed as a stochastic control problem. In the context of the formulation in Section 2, we have:

- The state at a given epoch, x_t , is the current board configuration and shape of the current falling piece.
- The control action *a*_t taken at each time *t*, is the orientation and translation to be applied to the piece before it is placed on the wall.
- The transition probabilities $P_a(x, y)$ are governed by the impact that the piece placing has on the wall, together with the (randomly generated) shape of the next falling piece. Each piece is uniformly sampled from seven possible shapes.

• It is natural to consider the reward (i.e., negative cost) at each time step to be the number of points received, and to consider as the objective maximization of the expected sum of rewards over the course of a game. However, we found that, with this formulation of reward and objective, solutions to the RLP do not offer reasonable policies for playing the game. For this reason, we consider an alternative formulation, in which the cost $g(x_t)$ at each time is taken to be the height of the Tetris wall. Furthermore, we take as our objective an expected discounted sum of of these costs:

$$E\left[\sum_{t=0}^{\infty} lpha^t g(x_t)
ight],$$

where $\alpha \in (0, 1)$ is a discount factor. We also found performance of policies produced by the RLP to be sensitive to α , and we ended up using a discount factor of $\alpha = 0.9$.

Several interesting observations have been made about the game of Tetris. It was shown in [5] that the game terminates with probability one, under any policy. In terms of complexity, it is proven in [8] that for an off-line version of Tetris, where the player is offered knowledge of the shapes of the next K pieces to appear, optimizing various simple objectives is NP-complete, even to approximate. Though there is no formal connection between such results and the on-line model we consider, the results suggest that finding an optimal policy for on-line Tetris might also be difficult.

There are over 2^{180} possible states in our formulation of Tetris. As such, standard dynamic programming methods can not be used to find an optimal policy. Instead, we will approximate the cost-to-go function via a linear combination of basis functions. Previous studies have used the following set of twenty-two basis functions, and we use the same ones to enable a comparison of results:

- Ten basis functions mapping the state to the height h_k of each of the ten columns.
- Nine basis functions, each mapping the state to the absolute difference between heights of successive columns: $|h_{k+1} h_k|, k = 1, ..., 9$.
- One basis function that maps state to the maximum column height: $\max_k h_k$.
- One basis functions that maps state to the number of "holes" in the wall.
- One basis functions that is equal to one at every state.

We denote these twenty-two basis functions by $\phi_1, \ldots, \phi_{22}$ and let Φ be the matrix with twenty-two columns given by these basis functions. We will use the approximate LP approach to compute a parameter vector $r \in \Re^{22}$ so that Φr approximates the cost-to-go function J^* .

4 Sampling and Bootstrapping

Recall that, in the ALP, the number of terms being summed in the objective and the number of constraints both grow proportionately with the number of states $|\mathcal{X}|$. To cope with the enormity of the state space in Tetris, we formulate an RLP:

 $\begin{array}{ll} \text{maximize} & \sum_{x\in\overline{\mathcal{X}}}(\Phi r)(x) \\ \text{subject to} & (T\Phi r)(x) \geq (\Phi r)(x), \qquad \forall x\in\overline{\mathcal{X}}. \end{array}$

where $\overline{\mathcal{X}}$ is a sample of states. To do this, we have to generate the samples that make up $\overline{\mathcal{X}}$. In our most basic set-up, we make use a heuristic policy generated by guessing and adjusting weights for the basis functions until reasonable performance is achieved. The intent is

TT 1 1 1	a ·	· · 1 · · 1	1 .1
Inhia I.	1 omnaricon	with other	algorithme
Table 1.	Companson	with other	argomunns

Algorithm	Performance	Computation Time
TD-Learning	3183	days
Policy Gradient	5500	?
LP w/ Bootstrap	4274	hours

to generate nearly i.i.d. samples of states, distributed according to the relative frequencies with which states are visited by the heuristic policy. To this end, some number N of games are played using the heuristic policy, and for some choice of M, states visited at times that are multiples of M are incorporated in the set $\overline{\mathcal{X}}$. Note that time, here, is measured in terms of the number of time-steps from the start of the first of the N games, rather than from the start of a current game. The reason for selecting states that are observed some M time-steps apart is to obtain samples that are near-independent. When consecutively visited states are incorporated in $\overline{\mathcal{X}}$, samples exhibit a high degree of statistical dependence. Consequently, a far greater total number of samples $|\overline{\mathcal{X}}|$ is required for the RLP to generate good policies. This is problematic, as computer memory limitations become an obstacle in solving linear programs with large numbers of constraints.

In addition to the basic set-up we have described above, we have also experimented with a bootstrapped version of the RLP. To understand the motivation for bootstrapping, suppose that the policy generated as an outcome of the RLP is superior to the initial heuristic used to sample states. Then, it is natural to consider producing a new sample of states based on the improved policy and solving the RLP again with this new sample. But why stop there? This procedure might be iterated to repeatedly amplify performance. This idea leads to our bootstrapping algorithm:

- 1. Begin with a simulator that using a policy u_0 .
- 2. Generate a sample $\overline{\mathcal{X}}_k$ of states using policy u_k .
- 3. Solve the RLP based on the sample $\overline{\mathcal{X}}_k$, to generate a policy u_{k+1} .
- 4. Increment *k* and go to step 2.

Other variants to this may include a more guarded update of the state-sampling distribution, wherein the sampling distribution used in a given iteration is the average of the distribution induced by the latest policy and the sampling distribution used in the previous iteration. That is, in Step 2 we might randomize between using samples generated by the current policy u_k , and the samples used in the generation of the previous collection, $\overline{\mathcal{X}}_{k-1}$.

In the next section, we discuss results generated by the RLP and bootstrapping.

5 Results and Discussion

Our numerical experiments may be summarized as follows. All RLPs were limited to two million constraints, this figure being determined by available physical memory. Initial experiments with the simple implementation helped determine a suitable sampling interval, M. All subsequent RLPs were generated with a sampling interval of M = 90.

For a fixed simulator policy, five RLPs were generated, of which the best was picked for the next bootstrap iteration. Figure 1 summarizes the performance of policies obtained from our experiments with the bootstrapping methodology. The "median" figures are illustrative of the variance in the quality of policies obtained at a given iteration, while the "best policy"



Figure 1: Bootstrapping Performance

figures correspond to the best performing policy at that iteration. Table 1 compares the performance of the best policy obtained in this process to that of other approaches used in the past [3], [11].

We now make some comments on the computation time required for our experiments . As mentioned previously, every RLP in our experiments had two million constraints. For general LPs this is a very large problem size. However, the RLP has special structure in that it has a small number of variables (22 in our case). We take advantage of this structure by solving the dual of the RLP. The dual has number of constraints equal to the number of basis functions (22 in our case) and so is effectively solved using a barrier method whose complexity is dominated by the number of constraints [4]. Using this, we are able to solve an RLP in minutes. Hence, the computation time is dominated by the time taken in generating state samples, which in our case translates to several hours for each RLP. These comments apply, of course, to RLPs for general large scale problems since the number of basis functions is typically several orders of magnitude smaller than the number of sampled states. We have found that solving smaller RLPs at leads to larger variance in policy quality, and lower median policy performance.

Finally, one might expect successive iterations of the bootstrapping methodology to yield continually improving policies. However, in our experiments, we have observed that beyond three to four iterations, median as well as best policy performance degrade severely. Use of a more guarded update to the state sampling distribution as described in Section 4, does not seem to alleviate this problem. We are unable to explain this behavior.

Acknowledgements

The second author thanks Richard Lippman and Linda Kukolich for introducing him to the problem of computing Tetris strategy. This research was supported by NSF CAREER Grant ECS-9985229 and by the ONR under grant MURI-N00014-00-1-0637.

References

- [1] D. Adelman. A price-directed approach to stochastic inventory/routing. Preprint, 2002.
- [2] A.S.Manne. Linear Programming and Sequential Decisions. *Management Science*, 60(3):259–267, 1960.
- [3] D. P. Bertsekas and S. Ioffe. Temporal Differences–Based Policy Iteration and Applications in Neuro–Dynamic Programming. Technical Report LIDS–P–2349, MIT Laboratory for Information and Decision Systems, 1996.
- [4] S. Boyd and L. Vandenberghe. Convex Optimization. Book Draft, 2002.
- [5] H. Burgiel. How to lose at tetris. Mathematical Gazette, page 194, July, 1997.
- [6] D.P. de Farias and B. Van Roy. The linear programming approach to approximate dynamic programming. To appear *Operations Research*, 2001.
- [7] D.P. de Farias and B. Van Roy. On constraint sampling in the linear programming approach to approximate dynamic programming. Conditionally accepted to *Mathematics of Operations Research*, 2001.
- [8] Susan Hohenberger Erik D. Demaine and David Liben-Nowell. Tetris is Hard, Even to Approximate. In *Proceedings of the 9th International Computing and Combinatorics Conference*, 2003.
- [9] G. Gordon. Approximate Solutions to Markov Decision Processess. PhD thesis, Carnegie Mellon University, 1999.
- [10] C. Guestrin, D. Koller, and R. Parr. Efficient solution algorithms for factored MDPs. Submitted to Journal of Artificial Intelligence Research, 2001.
- [11] S. Kakade. A Natural Policy Gradient. In *Advances in Neural Information Processing Systems* 14, Cambridge, MA, 2002. MIT Press.
- [12] P. Schweitzer and A. Seidmann. Generalized polynomial approximations in Markovian decision processes. *Journal of Mathematical Analysis and Applications*, 110:568–582, 1985.