Tetris:
A Study of Randomized Constraint Sampling

Vivek F. Farias and Benjamin Van Roy

Stanford University

1 Introduction

Randomized constraint sampling has recently been proposed as an approach for approximating solutions to optimization problems when the number of constraints is intractable – say, a googol or even infinity. The idea is to define a probability distribution ψ over the set of constraints and to sample a subset consisting of some tractable number N of independent identically distributed constraints. Then, a *relaxed problem*, in which the same objective function is optimized but only the sampled constraints are imposed, is solved.

An immediate question raised is whether solutions to the relaxed problem provide meaningful approximations to solutions of the original optimization problem. This question is partially addressed by recent theory developped first in the context of linear programming [8] and then convex programming [6]. In particular, it has been shown that, for a problem with K variables, given a number of samples

$$N = O\left(\frac{1}{\epsilon}\left(K \ln \frac{1}{\epsilon} + \ln \frac{1}{\delta}\right)\right),$$

with probability at least $1 - \delta$, any optimal solution to the relaxed problem violates a set of constraints $\mathcal{V}$ with measure $\psi(\mathcal{V}) \leq \epsilon$. Hence, given a reasonable number of samples, one can ensure that treating the relaxed problem leads to an "almost feasible" solution to the original problem. One interesting aspect of this result is that N does not depend on the number of constraints associated with the original optimization problem.

The aforementioned theoretical result leads to another question:

> In order that a solution to the relaxed problem be useful, does it suffice to know that the measure of the set set $\mathcal{V}$ of constraints it violates, $\psi(\mathcal{V})$, is small?

It is not possible to address this question without more specific context. In some problems, every constraint is critical. In others, violating a small fraction

of the constraints may be acceptable. Further, the context should influence the relative importance of constraints and therefore how the distribution ψ is selected.

Approximate dynamic programming offers one context in which randomized constraint sampling addresses a pressing need. The goal is to synthesize a suboptimal control policy for a large scale stochastic control problem. One approach that has received much recent attention entails solving a linear program with an intractable number of constraints [13, 7, 14]. For certain special cases, the linear program can be solved exactly [10, 12] while [15, 14] study constraint generation heuristics for general problems. Most generally, constraint sampling can be applied [8]. The linear programming approach to approximate dynamic programming provides a suitable context for assessing the effectiveness of constraint sampling. In particular, violation of constraints can be translated to a tangible metric – controller performance. The relationship is studied in [8], which offers motivation for why violation of a small fraction of constraints should not severely degrade controller performance. However, the theory is inconclusive. In this chapter, we present experimental results that further explore the promise of constraint sampling in this context.

Our study involves a specific stochastic control problem: the game of Tetris. In principle, an optimal strategy for playing Tetris might be computed via dynamic programming algorithms. However, because of the enormity of the state space, this is computationally infeasible. Instead, one might synthesize a suboptimal strategy using methods of approximate dynamic programming, as has been done in [16, 2, 11]. In this chapter, we experiment with the linear programming approach, which differs from others that have previosly been applied to Tetris. This study sheds light on the effectiveness of both the linear programming approach to approximate dynamic programming as a means of producing controllers for hard stochastic control problems, and randomized constraint sampling as a way of dealing with an intractable number of constraints.

The remainder of this chapter is organized as follows: In section 2, we make precise the notion of a stochastic control problem and present Tetris as an example of such a problem. In section 3, we introduce the linear programming approach to dynamic programming. In the following section, we discuss how this linear programming approach might be extended to approximate dynamic programming and in doing so, discuss results from [7, 8] on the quality of approximation such an approach might achieve, and a practically implementable constraint sampling scheme. Finally in section 5 we describe how a controller for Tetris was constructed using the LP approach for approximate dynamic programming along with constraint sampling.

2 Stochastic Control and Tetris

Consider a discrete-time dynamic system which, at each time t, takes on a state $x_t \in \mathcal{S}$ and takes as input an action $a_t \in \mathcal{A}_{x_t}$. We assume that the state space $\mathcal{S}$ is finite and that for each $x \in \mathcal{S}$, the set of actions $\mathcal{A}_x$ is finite. Let $p_a(x, y)$ denote the probability that the next state is y given that the current state is x and the current action is a.

A *policy* is a mapping $\pi : \mathcal{S} \to \mathcal{A}$ from state to action. A cost function $g : \mathcal{S} \times \mathcal{A} \to \Re$ assigns a cost $g(x, a)$ to each state-action pair (x, a). We pose as the objective to select a policy π that minimizes the expected sum of discounted future costs:

$$E\left[\sum_{t=0}^{\infty} \alpha^t g(x_t, a_t) \middle| x_0 = x, a_t = \pi(x_t)\right], \tag{1}$$

where $\alpha \in (0, 1)$ is the discount factor.

Tetris is a popular video game in which falling pieces are positioned by rotation and translation as they fall onto a wall made up of previously fallen pieces. Each piece is made up of four equally-sized bricks, and the Tetris board is a two-dimensional grid, ten-bricks wide and twenty-bricks high. Each piece takes on one of seven possible shapes. A point is received for each row constructed without any holes, and the corresponding row is cleared. The game terminates once the height of the wall exceeds 20. The objective is to maximize the expected number of points accumulated over the course of the game. A representative mid-game board configuration is illustrated in Figure 1.

Fig. 1. A representative Tetris board configurtaion

Indeed, Tetris can be formulated as a stochastic control problem:

- The state x_t encodes the board configuration and the shape of the falling piece.
- The action a_t encodes the rotation and translation applied to the falling piece.

- It is natural to consider the reward (i.e., negative cost) associated with a state-action pair to be the number of points received as a consequence of the action, 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, our approach (section 5) did not yield reasonable policies. We found that a different cost function, together with discounting, lead to effective policies. In particular, we set the cost $g(x_t, a_t)$ to be the height of the current Tetris wall, and let the objective be to minimize the expected sum of discounted future costs 1, with a discount factor $\alpha = 0.9$. Further, we set the cost of a transition to a termination state at $\frac{20}{1-\alpha}$ which is a trivial upper bound on the cost-to-go for a state under any policy. With this formulation, an optimal policy maximizes the number of rows cleared prior to termination with a greater emphasis on the immediate future, due to discounting.

Several interesting observations have been documented in the literature on Tetris. It was shown in [5] that the game terminates with probability one, under any policy. In terms of complexity, it is proven in [9] 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.

3 Dynamic Programming

For each policy π, define a cost-to-go function,

$$J_\pi(x) = E\left[\sum_{t=0}^{\infty} \alpha^t g(x_t, a_t) \Big| x_0 = x, a_t = \pi(x_t)\right].$$

Given the optimal cost-to-go function

$$J^*(x) = \min_\pi J_\pi(x),$$

an optimal policy can be generated according to

$$\pi(x) \in \text{argmin}_{a \in \mathcal{A}_x} \left(g(x,a) + \alpha \sum_{y \in \mathcal{S}} p_a(x,y) J^*(y) \right).$$

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

$$(TJ)(x) = \min_{a \in \mathcal{A}_x} \left(g(x,a) + \alpha \sum_{y \in \mathcal{S}} p_a(x,y) J(y) \right).$$

It is well-known that the optimal cost-to-go function J^* is the unique solution to Bellman's equation: $TJ = J$. Dynamic programming offers a suite of algorithms for solving this equation. One example involves a linear program:

$$\begin{array}{ll} \max_J & c'J \\ \text{s.t.} & TJ \geq J \end{array}$$

Note that, as written above, the constraints are nonlinear. However, they can be converted to linear constraints since each constraint $(TJ)(x) \geq J(x)$ is equivalent to a set of linear constraints:

$$g(x,a) + \alpha \sum_{y \in \mathcal{S}} p_a(x,y) J(y) \geq J(x) \quad \forall a \in \mathcal{A}_x.$$

It is well-known that for any $|\mathcal{S}|$-dimensional vector $c > 0$, J^* is the unique optimal solution to this linear program (see, e.g., [1]).

In principle, stochastic control problems like Tetris can be solved by dynamic programming algorithms. However, the computational requirements are prohibitive. For example, the above linear program involves one variable per state and one constraint per state-action pair. Clearly, Tetris presents far too many states ($\sim 2^{1400}$!) for such a solution method to be viable. One must therefore resort to approximations.

4 Approximate Dynamic Programming

In order to deal with an intractable state space, one might consider approximating the optimal cost-to-go function J^* by fitting a parameterized function approximator, in a spirit similar to statistical regression. A number of methods for doing this are surveyed in [3]. We will consider here cases where the approximator depends linearly on the parameters. Such an approximator can be thought of as a linear combination of pre-selected basis functions $\phi_1, \ldots, \phi_K : \mathcal{S} \mapsto \Re$, taking the form $\sum_{k=1}^K r_k \phi_k$, where the parameters are weights $r_1, \ldots, r_K \in \Re$. Generating such an approximation involves two steps:

1. Selecting basis functions $\phi_1, \ldots, \phi_K$.
2. Computing weights $r_1, \ldots, r_K$ so that $\sum_{k=1}^K r_k \phi_k \approx J^*$.

In our study of Tetris we will select basis functions based on problem specific intuition and compute weights by solving a linear program that with a reasonably small number of parameters but an intractable number of constraints. In this section, we discuss this linear program approach and the use of randomized constraint sampling in this context.

4.1 A Linear Program for Computing Basis Function Weights

It is useful to define a matrix $\Phi \in \Re^{|\mathcal{S}| \times K}$ by

$$\Phi = \begin{bmatrix} | & & | \\ \phi_1 & \cdots & \phi_K \\ | & & | \end{bmatrix}$$

The linear program presented in Section 3, which computes J^*, motivates another linear program for computing weights $r \in \Re^K$:

$$\begin{array}{ll} \max_r & c'\Phi r \\ \text{s.t.} & T\Phi r \geq \Phi r. \end{array}$$

To distinguish the two, we will call this linear program the ALP (approximate linear program) and the one from Section 3 the ELP (exact linear program). Note that, while the ELP involves one variable per state, the ALP only has one variable per basis function. However, the ALP has as many constraints as the ELP. We will later discuss the use of constraint sampling to deal with this. For now, we will discuss results from [7] that support the ALP as a reasonable method for approximating the optimal cost-to-go function.

Let $\tilde{r}$ be an optimal solution to the ALP, and $\|J\|_{1,c} = \sum_x c(x)J(x)$ denote a weighted ℓ_1-norm. One result from [7] asserts that $\tilde{r}$ attains the minimum of $\|J^* - \Phi r\|_{1,c}$ within the feasible region of the ALP.

Lemma 1. *A vector r solves*

$$\begin{array}{ll} \max_r & c'\Phi r \\ \text{s.t.} & T\Phi r \geq \Phi r \end{array}$$

if and only if it solves

$$\begin{array}{ll} \min_r & \|J^* - \Phi r\|_{1,c} \\ \text{s.t.} & T\Phi r \geq \Phi r. \end{array}$$

Recall that J^* is the optimal solution to the ELP for any $c > 0$. In the case of the ALP, however, the choice of c determines relative emphasis among states, with states corresponding to higher values of c likely to benefit from smaller approximation errors.

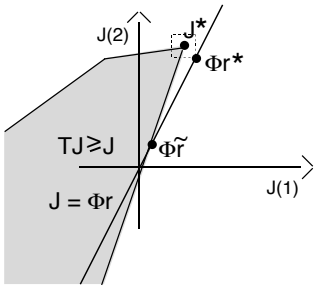

Fig. 2. Graphical interpretation of the ALP

It is easy to see that if J^* is in the range of Φ then $\Phi\tilde{r} = J^*$. One might hope that if J^* is close to the range of Φ then the $\Phi\tilde{r}$ will be close to J^*. This is not promised by the above result, because of the restriction to the feasible region of the ALP. In particular, as illustrated in Figure 2 from [7], one might imagine $\Phi\tilde{r}$ being close to or far from J^* even though there is some (infeasible) r^* for which $\Phi r^* \approx J^*$. The following theorem (Theorem 4.1 from [7]) offers some assurance through a bound on how far $\Phi\tilde{r}$ can be from J^* in terms of the proximity of J^* to the range of Φ. The result requires that e, the vector with every component equal to 1, is in the range of Φ.

Theorem 1. *Let e be in the range of Φ and let c be a probability distribution. Then, if $\tilde{r}$ is an optimal solution to the approximate LP,*

$$\|J^* - \Phi\tilde{r}\|_{1,c} \leq \frac{2}{1-\alpha} \min_r \|J^* - \Phi r^*\|_\infty.$$

As discussed in [7], this bound is rather loose. In particular, for large state-spaces, $\|J^* - \Phi r^*\|_\infty$ is likely to be very large. Further, the bound does not capture the fact, that the choice of c has a significant impact on the error $\|J^* - \Phi\tilde{r}\|_{1,c}$. More sophisticated results in [7] address these issues by refining the above bound. To keep the discussion simple, we will not present those results here.

After computing a weight vector $\tilde{r}$, one might generate decisions according to a policy

$$\tilde{\pi}(x) = \text{argmax}_{a \in \mathcal{A}_x} \left(g(x,a) + \alpha \sum_{y \in \mathcal{S}} p_a(x,y)(\Phi\tilde{r})(y) \right). \tag{2}$$

Should this policy be expected to perform reasonably? This question is addressed by another result, adapted from Theorem 3.1 in [7].

Theorem 2. *Let J be such that $TJ \geq J$. Then,*

$$\nu^T(J_{\tilde{\pi}} - J^*) \leq \frac{1}{1-\alpha}\|J - J^*\|_{1,c},$$

where $\nu(y) = \frac{1}{1-\alpha}(c(y) - \alpha \sum_x c(x) p_{\pi(x)}(x,y))$.

For each state x, the difference $J_{\tilde{\pi}}(x) - J^*(x)$ is the excess cost associated with suboptimal policy $\tilde{\pi}$ if the system starts in state x. It is easy to see that ν sums to 1. However, $\nu^T(J_{\tilde{\pi}} - J^*)$ is not necessarily a weighted average of excess costs associated with different states. Depending on the choice of c, individual elements of ν may be positive or negative. As such, the choice of c influences performance in subtle ways. [7] motivates choosing c to reflect the relative frequencies with which states are visited by good policies.

4.2 Randomized Constraint Sampling

If there are a reasonably small number of basis functions, the ALP involves a manageable number of variables but an intractable number of constraints. To deal with these constraints, we will use randomized constraint sampling, as proposed in [8]. In particular, consider the following relaxed linear program (RLP):

$$\begin{aligned} &\max\ c'\Phi r \\ &\text{s.t.}\ \ g_a(x) + \alpha \textstyle\sum_{y\in\mathcal{S}} P_a(x,y)(\Phi r)(y) \geq (\Phi r)(x), \quad \forall (x,a) \in \mathcal{X}, \end{aligned}$$

where $\mathcal{X}$ is a set of N constraints each sampled independently according to a distribution ψ.

The use of constraint sampling is motivated to some extent by the following result from [7]. (An important generalization that applies to convex programs has been established in [6].)

Theorem 3. *Consider a linear program with K variables and any number of constraints. Let ψ be a probability distribution over the constraints and let $\mathcal{X}$ be a set of N constraints sampled independently according to ψ, with $N \geq \frac{4}{\epsilon}\left(K \ln \frac{12}{\epsilon} + \ln \frac{2}{\delta}\right)$, $\epsilon \in (0,1)$ and $\delta \in (0,1)$. Let $r \in \Re^K$ be an optimal solution to the linear program with all constraints relaxed except for those in $\mathcal{X}$, and let $\mathcal{V}$ be the set of constraints violated by r. Then, $\psi(\mathcal{V}) \leq \epsilon$ with probability at least $1-\delta$.*

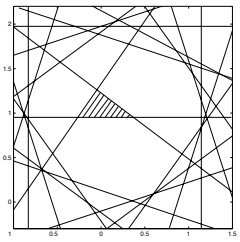

Fig. 3. Graphical interpretation of the ALP

In spite of the above result, it is not clear whether the RLP will yield solutions close to those of the ALP. In particular, it might be the case that a few constraints affect the solution dramatically as Figure 3 (which is from [8]) amply illustrates. Fortunately, the structure of the ALP precludes such undesirable behavior, and we have the following result, which is adapted from [8].

Theorem 4. *Let ϵ and δ be scalars in $(0,1)$. let π^* be an optimal policy and $\mathcal{X}$ be a random set of N state-action pairs sampled independently according to the distribution*

$$\psi_\alpha^*(x) = (1-\alpha)E\left[\sum_{t=0}^{\infty} \alpha^t \mathbf{1}\{x_t = x\} \Big| x_0 \sim c, a_t = \pi^*(x_t)\right].$$

Let $\hat{r}$ be a solution to the RLP. If

$$N \geq \frac{16\|J^* - \Phi\hat{r}\|_\infty}{(1-\alpha)\epsilon c^T J^*}\left(K \ln \frac{48\|J^* - \Phi\hat{r}\|_\infty}{(1-\alpha)\epsilon c' J^*} + \ln\frac{2}{\delta}\right),$$

then, with probability at least $1-\delta$, we have

$$\|J^* - \Phi\hat{r}\|_{1,c} \leq \|J^* - \Phi\tilde{r}\|_{1,c} + \epsilon\|J^*\|_{1,c}$$

In the proof of this error bound, sampling according to ψ_α^* ensures that with high probability, $\|J^* - \Phi\hat{r}\|_{1,c} \approx \|J^* - \Phi\tilde{r}\|_{1,c}+$ a term that can be made arbitrarily small with N large. As such this is a weakness; sampling according to ψ_α^* requires knowledge of the optimal policy. Nevertheless, one might hope that for a distribution sufficiently "close" to ψ_α^*, the bound of Theorem 4 still holds for a reasonable value of N. In any case, Theorem 4 offers some hope that the RLP is a tractable means for finding a meaningful approximation to J^*.

5 Synthesis of a Tetris Strategy

We've already seen that playing Tetris optimally is an example of a stochastic control problem with an intractable state-space. As a first step to coming up with a near-optimal controller for Tetris we select a linear approximation architecture for the tetris cost-to-go function. In particular, we will attempt to approximate the cost-to-go for a state using a linear combination of the following $K = 22$ basis functions:

- Ten basis functions, $\phi_0, \ldots, \phi_9$, mapping the state to the height h_k of each of the ten columns.
- Nine basis functions, $\phi_{10} \ldots \phi_{18}$, each mapping the state to the absolute difference between heights of successive columns: $|h_{k+1} - h_k|$, $k = 1, \ldots, 9$.
- One basis function, ϕ_{19} that maps state to the maximum column height: $\max_k h_k$.
- One basis function, ϕ_{20} that maps state to the number of 'holes' in the wall.
- One basis function, ϕ_{21} that is equal to one at every state.

Such an approximation architecture has been used with some success in [2, 11]. For example, in [2], the authors used an approximate dynamic programming technique – approximate policy iteration – to generate policies that averaged 3183 points a games which is comparable to an expert human player. The controller presented surpasses that performance.

In the spirit of the program presented in section 4.2, we formulate the following RLP:

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

where $\overline{\mathcal{X}}$ is a sample of states. Observe that in the above RLP, the sampling distribution takes on the role of c.

In our most basic set-up, we make use of a heuristic policy generated by guessing and adjusting weights for the basis functions until reasonable performance is achieved. The intent is 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.

Now recall that in light of the results of sections 4.1 and 4.2, we would like c to mimic the state distribution induced by the optimal policy as closely as possible. Thus, 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 uses 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, $\overline{\mathcal{X}}_{k-1}$.

Table 1. Comparison with other algorithms

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

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

5.1 Numerical results

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' 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 [2], [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

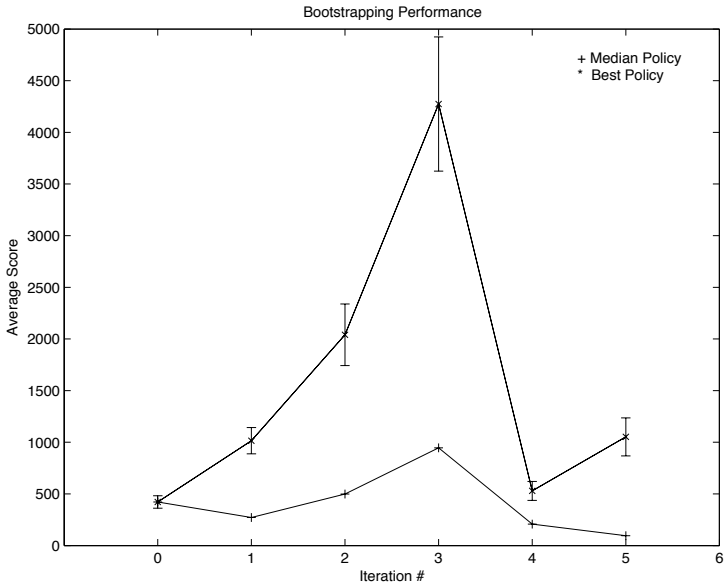

Fig. 4. Bootstrapping Performance

sampling distribution as described in Section 4, does not seem to alleviate this problem. We are unable to explain this behavior.

6 Concluding remarks

We have presented what we believe is a successful application of an exciting new technique for approximate dynamic programming that uses a 'constraint sampling' technique in a central way. Our experiments accentuate the importance of the question asked in the introduction, namely, what is the effect of the (small) number of violated constraints? The approximate LP of section 4.1 provided an interesting setting in which we attempted to answer this question, and we concluded that the answer (in the case of approximate dynamic programming via the LP method) was intimately related to the sampling distribution used for sampling constraints. This connection was strongly borne out in our Tetris experiments; naive sampling distributions led to relatively poor policies.

As such, theorems in the spirit of Theorem 3, while highly interesting represent only a first step in the design of an effective constraint sampling scheme; they need to be complemented by results and schemes along the lines of those in section 4.2 that assure us that the violated constraints cannot hurt us much. In the case of approximate dynamic programming, strengthening those

results and developing an understanding of schemes that sample constraints effectively is an immensely interesting direction for future research.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant ECS-9985229 and by the Office of Naval Research under Grant MURI N00014-00-1-0637.

References

1. D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, MA, 1995.
2. 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.
3. D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 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. G. Calafiore and M. C. Campi. Uncertain Convex Programs: Randomized Solutions and Confidence Levels. unpublished manuscript, 2003.
7. D. P. de Farias and B. Van Roy. The linear programming approach to approximate dynamic programming. *Operations Research*, 51(6):850–865, 2003.
8. D.P. de Farias and B. Van Roy. On constraint sampling in the linear programming approach to approximate dynamic programming. to appear in *Mathematics of Operations Research*, 2001.
9. 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.
10. C. Guestrin, D. Koller, and R. Parr. Efficient Solution Algorithms for Factored MDPs. *Journal of Artificial Intelligence Research*, 19:399–468, 2003.
11. S. Kakade. A Natural Policy Gradient. In *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.
12. D. Schuurmans and R. Patrascu. Direct value-approximation for factored MDPs. In *Advances in Neural Information Processing Systems*, volume 14, 2001.
13. P. Schweitzer and A. Seidmann. Generalized polynomial approximations in Markovian decision processes. *Journal of Mathematical Analysis and Applications*, 110:568–582, 1985.
14. M. Trick and S. Zin. A linear programming approach to solving dynamic programs. Unpublished manuscript, 1993.
15. M. Trick and S. Zin. Spline approximations to value functions: A linear programming approach. *Macroeconomic Dynamics*, 1, 1997.
16. J. N. Tsitsiklis and B. Van Roy. Feature–Based Methods for Large Scale Dynamic Programming. *Machine Learning*, 22:59–94, 1996.