Projected Equation and Aggregation-based Approximate Dynamic Programming Methods for Tetris

by

Daw-sen Hwang

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2011

Author . .
Department of Electrical Engineering and Computer Science
May 24, 2011

Certified by. .....
Dimitri P. Bertsekas
Professor of Engineering Lab. for Information and Decision Systems
Thesis Supervisor

Accepted by ....
Professor Leslie A. Kolodziejski
Chair of the Committee on Graduate Students

Projected Equation and Aggregation-based Approximate Dynamic Programming Methods for Tetris

by

Daw-sen Hwang

Submitted to the Department of Electrical Engineering and Computer Science
on May 24, 2011, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

In this thesis, we survey approximate dynamic programming (ADP) methods and test the methods with the game of Tetris. We focus on ADP methods where the cost-to-go function J is approximated with Φr, where Φ is some matrix and r is a vector with relatively low dimension. There are two major categories of methods: projected equation methods and aggregation methods. In projected equation methods, the cost-to-go function approximation Φr is updated by simulation using one of several policy-updated algorithms such as LSTD(λ) [BB96], and LSPE(λ) [BI96]. Projected equation methods generally may not converge. We define a pseudometric of policies and view the oscillations of policies in Tetris.

Aggregation methods are based on a model approximation approach. The original problem is reduced to an aggregate problem with significantly fewer states. The weight vector r is the cost-to-go function of the aggregate problem and Φ is the matrix of aggregation probabilities. In aggregation methods, the vector r converges to the optimal cost-to-go function of the aggregate problem. In this thesis, we implement aggregation methods for Tetris, and compare the performance of projected equation methods and aggregation methods.

Thesis Supervisor: Dimitri P. Bertsekas
Title: Professor of Engineering Lab. for Information and Decision Systems

Acknowledgments

I offer my sincerest gratitude to my thesis advisor, Professor Dimitri P. Bertsekas, whose encouragement and guidance from the beginning to the end has empowered me to develop an understanding of the material. His deep knowledge and enthusiasm for research has impacted me greatly. I am heartily thankful to Professor Patrick Jaillet, whose financial support enabled me to finish this thesis. I thank Professor John N. Tsitsiklis for his inspirational teachings and discussions in the Dynamic Programming class and group meetings.

I thank God for being faithful. In my daily life, I have been blessed with cheerful new friends from GCF, ROCSA, and LIDS as well as sincere old friends. I express my deep thanks for their prayers and thoughts. Especially, I would like to give my special thanks to Yin-Wen Chang, Chia-Hsin Owen Chen, Yu-Chung Hsiao, Justin Miller, Jagdish Ramakrishnan, and Huizhen Janey Yu for their suggestions in coding and writing.

Last but not least, I would like to express my gratitude to my family for their unconditional love, support and encouragement in every possible way. Finally, I offer my regards and blessings to all who supported me in any respect during my master study.

Contents

List of Figures

Chapter 1

Introduction

Dynamic Programming (DP) is an approach used to solve Markov Decision Processes with controls that affect the transition probability distribution to the next state [Ber05]. The goal is to minimize the expected accumulated cost starting from a given state. The transition cost $g(i,u)$ is the cost between two consecutive states i and the next state $j = f(i,u)$ with the control u, where f is the state transition function. Provided we use the policy π after the first step, the best control we can find for the first step is

$$\mu(i) = \arg\min_{u \in U(i)} \sum_j p_{ij}(u) \left[g(i,u) + J_\pi(j) \right],$$

where J_π is the cost-to-go function for policy π. The policy with the controller μ is called one-step lookahead with respect to J_π. Theoretically, when the number of states and the size of $U(i)$ are finite, we can use policy iteration or value iteration methods to find the optimal cost-to-go function $J^\star$, and hence an optimal policy. However, when the number of states is prohibitively large, one cannot use the DP algorithm to find a good controller due to the computational burden. Therefore Approximate DP algorithms have been developed [BB96], [BI96]. In this thesis, we survey some methods where the vector representation of the cost-to-go function J_π is approximated by the matrix-vector product $\bar{J}_k = \Phi r_k$, where r_k is updated based on approximate policy iteration methods. This approximation approach has

been widely studied [BT96], [SB98], [Gos03], [Mey07], [CFHM07], [Pow07], [Bor08], [Cao09], [Ber10b], [Ber10a], [Ber11]. The update of r_k is based on one of the two linear transformations

$$r_{k+1} = A_{\mu_k} r_k + b_{\mu_k}, \tag{1.1}$$

$$r_{k+1} = A_{\mu_k} r_{k+1} + b_{\mu_k}, \tag{1.2}$$

where both A_{μ_k} and b_{μ_k} depend only on the policy suggested by the cost-to-go function $\bar{J}_k = \Phi r_k$. The matrix A_{μ_k} is a contraction mapping.

Since A_{μ_k} is a contraction mapping, Eq. 1.2 can be viewed as iterating Eq. 1.1 infinite number of times. These methods involve solving a linear equation and are called *matrix inversion* methods.

One major ADP approach involves assigning several relevant feature functions of the state and approximating the cost-to-go function by a linear combination of those features. In this case, a row in Φ represents the corresponding features and r is the vector of weights of these features. The approximator learns and updates r through simulation. In this thesis, we discuss the projected equation methods where a projection is considered when we update the vector r. We consider the LSTD(λ) and LSPE(λ) methods in particular. LSTD(λ) is a *matrix inversion* method and LSPE(λ) is an *iterative* method. These two methods are not guaranteed to converge because the A_{μ_k} matrix is not necessarily monotone. As a result, these methods may not converge to the optimal policy. As shown by Bertsekas, these methods may end up oscillating with more than one policy [Ber10b].

Another major ADP approach is problem approximation, where the original DP problem is associated with an aggregate problem with significant fewer states [SJJ94], [SJJ95], [Gor95], [VR06], [Ber10b], [Ber10a], [Ber11]. The aggregate problem has significantly fewer number of states so that we can calculate the optimal policy for the aggregate problem by policy iteration. The vector r_k is the cost-to-go function of each iteration. The update of r_k is of the form in Eq. 1.2 with a monotonic A_{μ_k}. As a result, $\{r_k\}$ converges to $r^\star$, which is the optimal cost-to-go for the aggregate

problem. The cost-to-go $r^\star$ then specifies a policy $\mu_{\bar{J}}$ with $\bar{J} = \Phi r^\star$, where Φ is a matrix which relates the aggregate problem with the original problem. One of the big differences between the projected equation methods and the aggregation methods is convergence. The policies converge in the aggregation methods, and hence the performance may be more regular.

We will test those methods in Tetris, a video game originally invented by A. Pajitnov in 1984. It is NP-hard to find the optimal policy even when we know the sequence of tetrominoes in the beginning of the game [DHLN03]. On the other hand, the game terminates with probability 1 [Bur96], which allows the methodology of stochastic shortest path DP problems to be used. Because solving Tetris is hard and can be formulated into a standard stochastic shortest path DP problem, this game has become a benchmark problem of large scale DP problems. There are several different versions of Tetris. A study of different variations of Tetris can be found in [TS09]. Tetris has been well tested with methods where feature functions are involved. Bertsekas and Ioffe used a 10×20 board and a set of 22 features and got an average score of 3,183 with the LSPE(λ) method [BI96]. This score might be underrated because they considered a slightly different version of Tetris [TS09]. Later on, Kakade used the same 22 features with a natural policy gradient method and got an average score around 5,000-6,000 [Kak02]. Farias and Van Roy also used the same 22 features with a relaxed linear program and achieved average score 4,274 [FVR06]. All the methods above were trying to update the policy by letting it be a better approximator. Szita and Lorincz used the cross-entropy (CE) method, a random search methods in the space of weight vectors r with special strategy to update the distribution function, and got an average score exceeding 800,000 with the same 22 features [SL06]. It is clear that the performance of the controller depends on the feature functions. Therefore, research also focused on different feature functions. For example, Thiery and Scherrer used a set of 8 features with the cross-entropy method and achieved an average score of 35,000,000 on a simplified version of the game. This controller won the 2008 Tetris domain Reinforcement Learning Competition [TS09]. However, the 8 features they used are not functions of state, but functions of both

state and controller. In this thesis, we are interested in formulating Tetris into a DP problem and testing the behavior of different ADP methods. As a result, we are going to use the 22 features proposed in [BI96].

This thesis organizes as follows. In Chapter 2, we introduce our notation for the DP problem we aim to solve. In Chapter 3, we introduce theoretical background for projected equation methods: LSTD(λ) and LSPE(λ). In Chapter 4, we introduce theoretical background for the aggregation methods. In Chapter 5, we formulate Tetris into a DP problem. We then consider Tetris problems with smaller board sizes so that we can apply exact projected equation and aggregation methods. Finally, we compare the performance of the aggregation methods and the projected equation methods.

Chapter 2

Approximate Dynamic Programming

2.1 Approximate Dynamic Programming (ADP) and Linear Architecture

In this thesis, we consider the stochastic shortest problem with a termination state denoted 0. We use the following transition model:

$$x_{k+1} = f(x_k, u_k, y_k), \tag{2.1}$$

where x_k and x_{k+1} are the current and the next state. The control u_k is selected after an uncontrollable random forecast y_k. The control u_k can be described as $u_k = \mu(x_k, y_k) \in U(x_k, y_k)$. Here $U(x_k, y_k)$ denotes the set of admissible controls and μ denotes a controller which maps (x_k, y_k) to u_k. The number of possible states of x_k, the number of possible values of y_k, and the number of elements of $U(x_k, y_k)$ are finite. In this thesis, we only consider stationary problems and policies, and therefore we may interchangeably refer to μ as either a controller or a policy. We consider the case where the probability distribution function of y_k is stationary. When the state changes, a transition cost $g(x_k, u_k, y_k)$ is accumulated. Bellman's Equation for this

problem is

$$J^\star(x) = \sum_y p(y) \min_{u \in U(x,y)} \left[g(x,u,y) + J^\star(f(x,u,y)) \right], \text{ with all } x, \tag{2.2}$$

where the solution $J^\star(x)$ is the optimal cost-to-go for x. Any optimal policy $\mu^\star$ satisfies

$$\mu^\star(x,y) = \arg \min_{u \in U(x,y)} \left[g(x,u,y) + J^\star(f(x,u,y)) \right]. \tag{2.3}$$

In this thesis, we consider the stochastic shortest problem where the game starts at a special state $x_0 = 1$ and ends at a cost-free state 0. Thus, the process terminates when the state is at 0 and the cost-to-go for the state 0 is always 0. We use a linear architecture to calculate the approximate cost-to-go function $\bar{J}_k = \Phi r_k$[1]. When the optimal cost-to-go vector $J^\star$ in 2.3 is replaced with this approximation, the associated policy is defined to be

$$\mu_k(x,y) \in \arg \min_{u \in U(x,y)} \left[g(x,u,y) + \bar{J}_k(f(x,u,y)) \right]. \tag{2.4}$$

When the minimum is achieved by more than one control in $U(x,y)$, we choose u_k from those controls according to some fixed but arbitrary rule. (In our implementations, we have chosen u_k randomly from the set of controls attaining the minimum according to a uniform distribution.) In order to investigate further, we first introduce some notation.

2.2 Matrix Representation

We will enumerate all the possible states and assume the number of states is n (not including the termination state 0). Most matrices or vectors below depend only on the policy μ but not on the vector r. When there is no confusion, we do not write μ

[1]The integer k here refers to different iteration of policies, which is different from the k used in 2.1.

explicitly.

2.2.1 Approximate Cost-to-Go Function

We denote the approximate cost-to-go function $\bar{J}_k = \Phi r_k$.

$$\bar{J}_k = \begin{bmatrix} \bar{J}_k(1) \\ \vdots \\ \bar{J}_k(n) \end{bmatrix} \triangleq \Phi r_k,$$

where Φ is some matrix, and the value is calculated only when accessed. The definition of Φ varies in different methods. Because 0 is the cost free state, we define $\bar{J}_k(0) \triangleq 0$.

2.2.2 Transition Probabilities

The matrix of transition probabilities for a stationary policy μ is defined to be

$$P = P_\mu \triangleq \begin{bmatrix} \sum_y p(y)p_{11}(\mu(1,y),y) & \cdots & \sum_y p(y)p_{1n}(\mu(1,y),y) \\ \vdots & \ddots & \vdots \\ \sum_y p(y)p_{n1}(\mu(n,y),y) & \cdots & \sum_y p(y)p_{nn}(\mu(n,y),y) \end{bmatrix},$$

where $p_{ij}(\mu(i,y),y)$ is the transition probability with control $u = \mu(x,y)$ and forecast y. We then simplify the notation by using $\bar{p}_{ij}(\mu)$ to denote the transition probability from state i to state j with policy μ:

$$\bar{p}_{ij}(\mu) \triangleq \sum_y p(y)p_{ij}(\mu(i,y),y).$$

Notice that the summation of row i, $\sum_{j=1}^{n} \bar{p}_{ij}(\mu)$, may be less than 1 because there may be positive probability to terminate at state i under μ.

2.2.3 Expected Transition Cost Function

With the stationary policy μ, the expected transition cost from state i to the next state is $\sum_y p(y)g(i,\mu(i,y),y)$. As a result, we can define the expected transition cost vector as

$$g = g_\mu = \begin{bmatrix} g_\mu(1) \\ \vdots \\ g_\mu(n) \end{bmatrix} \triangleq \begin{bmatrix} \sum_y p(y)g(1,\mu(1,y),y) \\ \vdots \\ \sum_y p(y)g(n,\mu(n,y),y) \end{bmatrix}.$$

2.2.4 Occurrence Probabilities

The occurrence probability in the stochastic shortest problem is similar to the steady state probability for the infinite-horizon discounted problem. Denote

$$q^t(\bar{x},\bar{y}) = q^t_\mu(\bar{x},\bar{y}) \triangleq P_\mu(x_t = \bar{x}, y_t = \bar{y}),$$

for all $\bar{x} \neq 0$ and let

$$q(\bar{x},\bar{y}) = \frac{\sum_t q^t(\bar{x},\bar{y})}{\sum_{x,y}\sum_t q^t(x,y)},$$

then $q(\bar{x},\bar{y})$ is the expected frequency at which the state and forecast pair $(\bar{x},\bar{y})$ occurs. Similarly, denote

$$\bar{q}^t(\bar{x}) = \bar{q}^t_\mu(\bar{x}) \triangleq P_\mu(x_t = \bar{x}),$$

for all $\bar{x} \neq 0$ and let

$$\bar{q}(\bar{x}) = \frac{\sum_t \bar{q}^t(\bar{x})}{\sum_x \sum_t \bar{q}^t(x)},$$

then $\bar{q}(\bar{x})$ is the frequency at which the state $\bar{x}$ occurs. We can denote $\bar{q}^t$ and $\bar{q}$ as vectors

$$\bar{q}^t \triangleq \begin{bmatrix} \bar{q}^t(1) \\ \vdots \\ \bar{q}^t(n) \end{bmatrix}, \ \bar{q} \triangleq \begin{bmatrix} \bar{q}(1) \\ \vdots \\ \bar{q}(n) \end{bmatrix}.$$

Let e_1 be the vector $[1, 0, 0, \cdots, 0, 0]'$. We have $\bar{q}^t = P'^t e_1$ and therefore

$$\bar{q} = \alpha \sum_{t=0}^{\infty} \bar{q}^t = \alpha \sum_{t=0}^{\infty} P'^t e_1 = \alpha (I - P')^{-1} e_1,$$

where α is chosen to normalize $\bar{q}$ so that $\sum_i \bar{q}(i) = 1$. We will denote by Ξ the matrix that has $\bar{q}$ as its diagonal,

$$\Xi = \Xi_\mu \triangleq \begin{bmatrix} \bar{q}_\mu(1) & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \bar{q}_\mu(n) \end{bmatrix}.$$

The matrix Ξ_μ defines in turn a weighted Euclidean norm by

$$\| J \|^2_{\Xi_\mu} \triangleq \sum_{i=1}^{n} \bar{q}_\mu(i)(J(i))^2 = J' \Xi_\mu J.$$

2.2.5 T and $T^{(\lambda)}$ Mappings

We denote by T_μ the one-step look-ahead mapping with the policy μ [Ber05]. The vector $T_\mu(J)$ is the cost-to-go vector where the controller μ is applied at the first stage followed by a policy with cost-to-go vector J.

$$T_\mu J(x) \triangleq \sum_y p(y) \left[g(x, \mu(x, y), y) + J(f(x, \mu(x, y), y)) \right], \text{ with all } x.$$

We notice that $\sum_y p(y) g(x, \mu(x, y), y) = g_\mu(x)$ and $\sum_y p(y) J(f(x, \mu(x, y), y)) = \sum_j \bar{p}_{ij}(\mu) J(j)$. As a result, the matrix representation of $T(J)$ satisfies

$$T(J) = T_\mu(J) \triangleq \begin{bmatrix} T_\mu J(1) \\ \vdots \\ T_\mu J(n) \end{bmatrix} = g_\mu + P_\mu J.$$

With this definition, T_μ can be applied to any vector J, not necessarily a valid cost-to-go vector. We will similarly denote $T^{(\lambda)}$ as the infinite-step look-ahead discounted mapping [Ber11]

$$T^{(\lambda)} = T_\mu^{(\lambda)} \triangleq (1-\lambda)\sum_{l=1}^{\infty} \lambda^l T^{l+1}.$$

After some math calculation, we get

$$T^{(\lambda)}(J) = g^{(\lambda)} + P^{(\lambda)} J,$$

where

$$g^{(\lambda)} \triangleq \sum_{l=0}^{\infty} \lambda^l P^l g = (I-\lambda P)^{-1} g,$$
$$P^{(\lambda)} \triangleq (1-\lambda)(I-\lambda P)^{-1} P.$$

Chapter 3

Projected Equation Methods

3.1 Projected Equation-Based Linear Architecture

In projected equation methods such as LSTD and LSPE, the cost-to-go $J(x)$ is approximated by $\bar{J}_k$(x), the inner product of s feature functions $\phi^1(x), \phi^2(x), \cdots, \phi^m(x)$, and a vector r_k:

$$\begin{aligned}
\phi(x) &\triangleq [\phi^1(x), \phi^2(x), \cdots, \phi^m(x)]', \\
\phi(0) &\triangleq [0, 0, \cdots, 0]', \\
r_k &\triangleq [r_k(1), r_k(2), \cdots r_k(m)]', \\
\bar{J}_k(x) &\triangleq \phi'(x) r_k.
\end{aligned}$$

The matrix Φ is defined to be

$$\Phi \triangleq \begin{bmatrix} \phi^1(1) & \cdots & \phi^m(1) \\ \phi^1(2) & \cdots & \phi^m(2) \\ \vdots & \ddots & \vdots \\ \phi^1(n) & \cdots & \phi^m(n) \end{bmatrix},$$

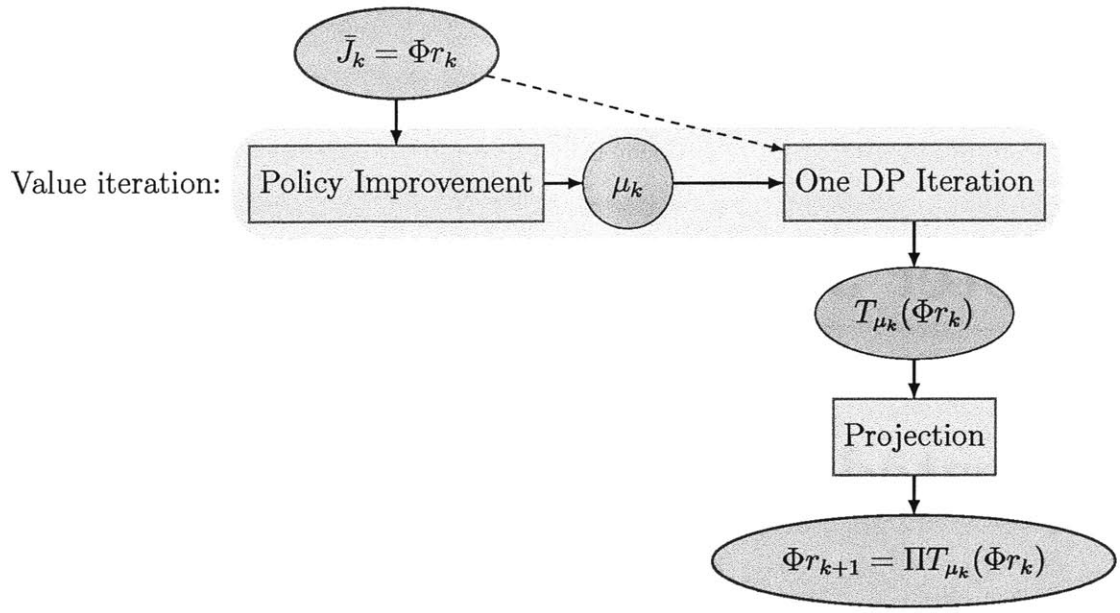


Figure 3-1: This is an illustration of the LSPE(0) method. Each iteration can be viewed as a value iteration followed by a projection Π. The policy μ_k is defined in Equation 2.4.

so that $\bar{J}_k = \Phi r_k$. Because 0 is the cost-free state, we define $\phi(0) = 0$ so that $\bar{J}_k(0) = 0$ for all k. This definition also simplifies the notation for the simulation-based methods.

3.2 Least-Squares Policy Evaluation λ Method

In the Least-Squares Policy Evaluation λ method (LSPE(λ)), each iteration can be viewed as a value iteration followed by a projection as shown in Fig. 3-1 when $\lambda = 0$. If there were no projection, the method would become the value iteration method and the generated policies would converge to an optimal policy. However, in order to represent the next approximation in term of $\bar{J}_{k+1} = \Phi r_{k+1}$, it is necessary to project the cost-to-go function onto the subspace spanned by the columns of Φ. The projection Π on Φ is the projection with respect to the norm defined by Ξ_{μ_k}. The norm defined by Ξ_{μ_k} is critical because we then can update the vector r_k by simulation for the LSPE methods. When the one-step look-ahead mapping T_{μ_k} is replaced by a more general infinite-step look-ahead discounted mapping $T^{(\lambda)}_{\mu_k}$, it becomes the LSPE(λ) method. In summary, the LSPE(λ) method updates the vector r_k by the following

equation:

$$\Phi r_{k+1} = \Pi T_{\mu_k}^{(\lambda)}(\Phi r_k).$$

We can also get r_{k+1} from the following equation:

$$\begin{aligned} r_{k+1} &= \arg\min_r \|\Phi r - g^{(\lambda)} - P^{(\lambda)}\Phi r_k\|_\Xi \\ &= \arg\min_r (\Phi r - g^{(\lambda)} - P^{(\lambda)}\Phi r_k)'\Xi(\Phi r - g^{(\lambda)} - P^{(\lambda)}\Phi r_k). \end{aligned}$$

Taking derivative, r_{k+1} is the solution of the following equation in r:

$$\frac{\partial(\Phi r - g^{(\lambda)} - P^{(\lambda)}\Phi r_k)'\Xi(\Phi r - g^{(\lambda)} - P^{(\lambda)}\Phi r_k)}{\partial r} = 0,$$

or equivalently

$$(\Phi'\Xi\Phi)\Xi(\Phi r - g^{(\lambda)} - P^{(\lambda)}\Phi r_k) = 0.$$

Therefore, r_{k+1} satisfies

$$r_{k+1} = r_k - G(C^{(\lambda)} r_k - d^{(\lambda)}), \tag{3.1}$$

where

$$\begin{aligned} C^{(\lambda)} &\triangleq \Phi'\Xi(I - P^{(\lambda)})\Phi, \\ d^{(\lambda)} &\triangleq \Phi'\Xi g^{(\lambda)}, \\ G &\triangleq (\Phi'\Xi\Phi)^{-1}. \end{aligned}$$

Note that LSPE(λ) is an iterative method of the form Eq. 1.1 with $A_{\mu_k} = I - GC^{(\lambda)}$ and $b_{\mu_k} = Gd^{(\lambda)}$.

3.3 Least-Squares Temporal Difference λ Method

The Least-Squares Temporal Difference λ method is a matrix inversion method. Similar to LSPE, we can view LSTD as iterating over a combination of value iteration and a projection as shown in Fig. 3-1 when $\lambda = 0$. The difference is that in the LSTD method, r_{k+1} is updated after an infinite number of iterations under the same policy μ_k. In practice, we do a matrix inversion calculation to find the fixed point for the iteration. If there were no projection and $\lambda = 0$, the method would become the policy iteration method and the policies will converge to an optimal policy. However, similar to LSPE, in order to represent the next approximation in term of $\bar{J}_{k+1} = \Phi r_{k+1}$, it is necessary to project the cost-to-go function onto the subspace spanned by the columns of Φ. The projection Π is with respect to the norm defined by Ξ_{μ_k}. In summary, the LSTD(λ) method updates the vector r_k based on the projected following equation [Ber11]:

$$\Phi r_{k+1} = \Pi T_{\mu_k}^{(\lambda)}(\Phi r_{k+1}).$$

We can fin r_{k+1} by applying Eq. 1.2 with $A_{\mu_k} = I - GC^{(\lambda)}$ and $b_{\mu_k} = Gd^{(\lambda)}$ from Section 3.2.

$$r_{k+1} = A_{\mu_k} r_{k+1} + b_{\mu_k} = \left(I - GC^{(\lambda)}\right) r_{k+1} + Gd^{(\lambda)}.$$

Therefore, the updated vector r_{k+1} satisfies

$$C^{(\lambda)} r_{k+1} = d^{(\lambda)}. \tag{3.2}$$

The vector r_{k+1} can be obtained by the following matrix inversion calculation:

$$r_{k+1} = \left(C^{(\lambda)}\right)^{-1} d^{(\lambda)}. \tag{3.3}$$

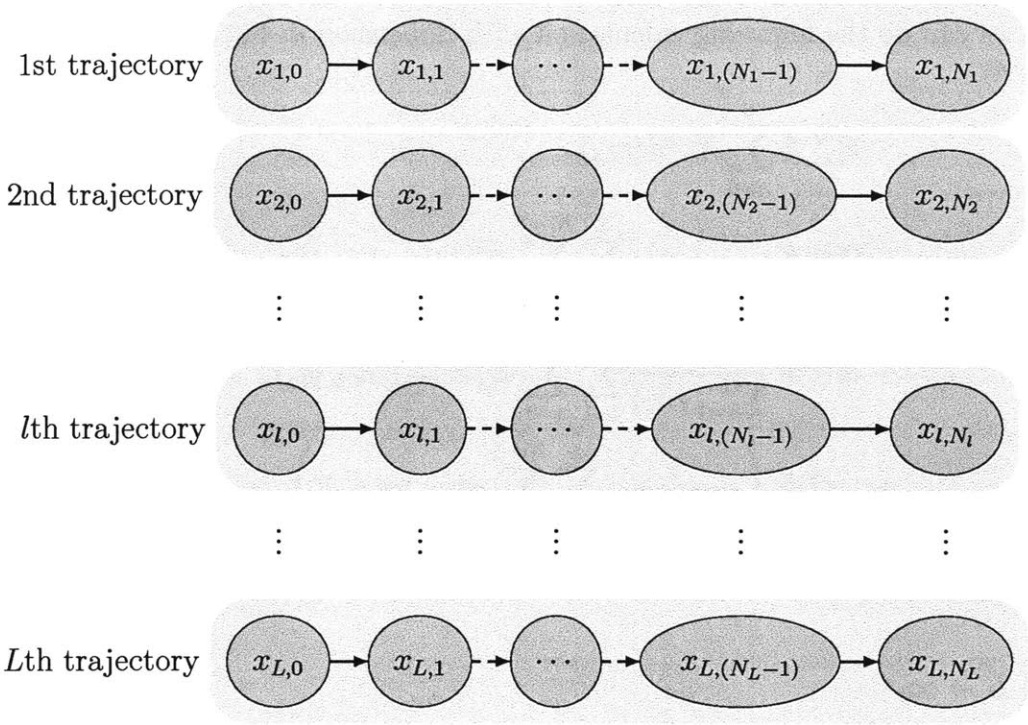


Figure 3-2: This is an illustration of simulation trajectories. Each trajectory starts at state 1: $x_{l,0} = 1$ for all l. At state $x_{l,i}$, forecast $y_{l,i}$ happens and control $u_{l,i}$ is applied. Therefore, the next state is $x_{l,i+1} = f(x_{l.i}, u_{l.i}, y_{l.i})$ and the transition cost is $g(x_{l.i}, u_{l.i}, y_{l.i})$. Trajectory l first enters the termination state 0 at state $x_{l,N_l} = 0$.

3.4 Simulation-Based Projected Equation Methods

In general, the number of states is huge and so the calculation for the matrices $C^{(\lambda)}$, G are prohibitive. Fortunately, the definition of those matrices involves the matrix Ξ, and therefore they can be approximately calculated through simulation. Assume we simulate the DP problem L times and obtain L trajectories as shown in Fig. 3-2.

Then, we can do the following calculation with dimension m but not n:

$$\bar{z}_{l,i}^{(\lambda)} \triangleq \sum_{j=0}^{i} \lambda^{i-j} \phi(x_{l,j}).$$

$$\bar{C}^{(\lambda)} \triangleq \frac{1}{\sum_{l=1}^{L}(N_l)} \sum_{l=1}^{L} \sum_{i=0}^{N_l-1} \bar{z}_{l,i}^{(\lambda)} \left[\phi(x_{l,i}) - \phi(x_{l,i+1})\right]'.$$

$$\bar{d}^{(\lambda)} \triangleq \frac{1}{\sum_{l=1}^{L}(N_l)} \sum_{l=1}^{L} \sum_{i=0}^{N_l-1} \bar{z}_{l,i}^{(\lambda)} g\left(x_{l,i}, u_{l,i}, y_{l,i}\right).$$

$$\bar{G} \triangleq \left(\frac{1}{\sum_{l=1}^{L}(N_l)} \sum_{l=1}^{L} \sum_{i=0}^{N_l-1} \phi(x_{l,i})\phi(x_{l,i})'\right)^{-1}.$$

When $L \to \infty$,

$$\bar{C}^{(\lambda)} \to C^{(\lambda)}.$$

$$\bar{d}^{(\lambda)} \to d^{(\lambda)}.$$

$$\bar{G} \to G.$$

Thus, we can calculate r_{k+1} approximately with the simulation-based LSPE(λ) method by using

$$r_{k+1} = r_k - \bar{G}(\bar{C}^{(\lambda)} r_k - \bar{d}^{(\lambda)}). \tag{3.4}$$

Alternatively, we can use the simulation-based LSTD(λ) method by solving

$$C^{(\lambda)} r_{k+1} = \bar{d}^{(\lambda)}. \tag{3.5}$$

3.5 LSPE(λ, c) Method

In this section, we consider an LSPE(λ)-like method for the case where the last feature $\phi^s(x)$ is 1 and the termination state 0 is not favorable. We write $r_k = [\bar{r}_k' \; c_k]'$, where

$\bar{r}_k$ is an $(m-1)$ dimension vector and c_k is a constant. When the termination state is not favorable, we can avoid termination by allowing a termination decision only when necessary. In this case, the policy μ_k depends on $\bar{r}_k$ but not on c_k. We are interested in the case where c_k remains a constant c for every vector r_k. Denote Π_c to be the projection to the affine set $\left\{ \Phi \begin{bmatrix} r \\ c \end{bmatrix} | r \in \Re^{m-1} \right\}$ with a fixed constant c and the norm Ξ_{μ_k}. LSPE(λ, c) updates r_k with the following equation:

$$\Phi r_{k+1} = \Pi_c T_{\mu_k}^{(\lambda)}(\Phi r_k).$$

Although the policy μ_k does not depend on c, the constant c makes a difference when r_k is updated. Let e be $[1, 1, \cdots, 1]'$ and the matrix $\bar{\Phi}$ be the $n \times (m-1)$ matrix with the first $(m-1)$ columns of Φ. Then, the projected equation can be calculated to be

$$\begin{aligned}
\bar{r}_{k+1} &= \arg \min_{r \in \Re^{m-1}} \|\bar{\Phi} r + ce - g^{(\lambda)} - P^{(\lambda)}(\bar{\Phi}\bar{r}_k + ce)\|_{\Xi_{\mu_k}} \\
&= \arg \min_{r \in \Re^{m-1}} (\bar{\Phi} r + ce - g^{(\lambda)} - P^{(\lambda)}(\bar{\Phi}\bar{r}_k + ce))' \Xi (\bar{\Phi} r + ce - g^{(\lambda)} - P^{(\lambda)}(\bar{\Phi}\bar{r}_k + ce))
\end{aligned}$$

Taking derivative, $\bar{r}_{k+1}$ is the solution of the following equation in r:

$$\frac{\partial (\bar{\Phi} r + ce - g^{(\lambda)} - P^{(\lambda)}(\bar{\Phi}\bar{r}_k + ce))' \Xi (\bar{\Phi} r + ce - g^{(\lambda)} - P^{(\lambda)}(\bar{\Phi}\bar{r}_k + ce))}{\partial r} = 0,$$

or equivalently

$$\bar{\Phi}' \Xi (\bar{\Phi} r + ce - g^{(\lambda)} - P^{(\lambda)}(\bar{\Phi}\bar{r}_k + ce)) = 0.$$

Therefore, $\bar{r}_{k+1}$ satisfies

$$\bar{r}_{k+1} = \bar{r}_k - G_c(C_c^{(\lambda)} \bar{r}_k - d_c^{(\lambda)}),$$

where

$$C_c^{(\lambda)} \triangleq \bar{\Phi}'\Xi(I - P^{(\lambda)})\bar{\Phi},$$
$$d_c^{(\lambda)} \triangleq \bar{\Phi}'\Xi(g^{(\lambda)} - (I - P^{(\lambda)})ce),$$
$$G_c \triangleq \left(\bar{\Phi}'\Xi\bar{\Phi}\right)^{-1}.$$

LSPE(λ, c) is an iterative method of form Eq. 1.1 with $A_{\mu_k} = I - G_c C_c^{(\lambda)}$ and $b_{\mu_k} = G_c d_c^{(\lambda)}$. If there are only the first $(m-1)$ features for the matrices C and G, C_c and G_c happen to be exactly equal to those matrices, respectively. Furthermore, when $c = 0$, if there are only the first $(m-1)$ features for the vector d, d_c happens to be exactly equal to this vector. As a result, the LSPE(λ, c) method is equivalent to the LSPE(λ) methods with only the first $(m-1)$ features.

3.6 LSTD(λ, c) Method

In this section, we consider an LSTD(λ)-like method for the same case discussed in Section 3.5. The LSTD(λ, c) method is similar to the LSPE(λ, c) method. However, LSTD(λ, c) is a matrix-inversion method which finds the fixed point instead of doing one iteration. The LSTD(λ, c) method updates r_k using Eq. 1.2 with the same A_{μ_k} and b_{μ_k} as the LSPE(λ, c) method. Therefore, r_{k+1} is updated by the following equation:

$$\bar{r}_{k+1} = \bar{r}_{k+1} - G_c(C_c^{(\lambda)}\bar{r}_{k+1} - d_c^{(\lambda)}),$$

where $C_c^{(\lambda)}$, $d_c^{(\lambda)}$, and G_c are defined in Section 3.5. Therefore, the vector r_{k+1} satisfies

$$C_c^{(\lambda)}\bar{r}_{k+1} = d_c^{(\lambda)}. \tag{3.6}$$

The updated vector $\bar{r}_{k+1}$ can be calculated by matrix inversion.

3.7 Policy Oscillations

From Eq. 3.2, we know that r_{k+1} depends only on the current policy μ_k. Furthermore, there is only finitely many policies. As a result, for the exact LSTD(λ) method, after several iterations, we will have $\mu_i = \mu_j$ for some $i < j$. When this happens, we will have $r_{i+1} = r_{j+1}$ and thus $\mu_{i+1} = \mu_{j+1}$. By mathematical induction, $r_k = r_{k+j-i}$ for all $k \geq i$. As a result, we will end up oscillating between a subset of policies periodically. The trajectory of the oscillating policies depends on λ and the starting vector r_0 [Ber10b]. On the other hand, from Eq. 3.5, both $\bar{C}^{(\lambda)}$ and $\bar{d}^{(\lambda)}$ are random variable and the distribution depends on L, the number of simulation trajectories. Therefore, in the simulation-based methods, r_{k+1} is also random and it is not necessarily oscillating periodically. In Chapter 6, we will discuss policy oscillations for the exact projected equation methods.

Chapter 4

Aggregation Methods

In aggregation methods, the original DP problem is approximated by a simpler problem. The idea is to reduce the original problem to a problem with significantly fewer states. We call the new problem the *aggregate problem*, and a state in the new problem an *aggregate state*. We use policy iteration methods to find the optimal policy for the aggregate problem. This optimal policy may suggest a good policy for the original DP problem.

4.1 Aggregation Methods Transition Probabilities

We associate the original problem with an aggregate problem by aggregation and disaggregation probabilities. For each pair of aggregate state a and original state i, we specify the aggregation probability ϕ_{ia} and the disaggregation probability d_{ai}. The aggregation and disaggregation probabilities satisfy equations $\sum_i d_{ai} = 1$ and $\sum_a \phi_{ia} = 1$. These probabilities can be defined arbitrarily as long as the associated aggregate problem is a stochastic shortest path problem which terminates with probability 1. When this is not so, the optimal cost-to-go function goes to minus infinity for some states and policy iteration need not converge. Therefore, in practice, we do not have to check for this condition before applying policy iteration. When aggregation and disaggregation probabilities are specified, the aggregate problem is then well-defined, as illustrated in Fig. 4-1. The transition of an aggregate state involves

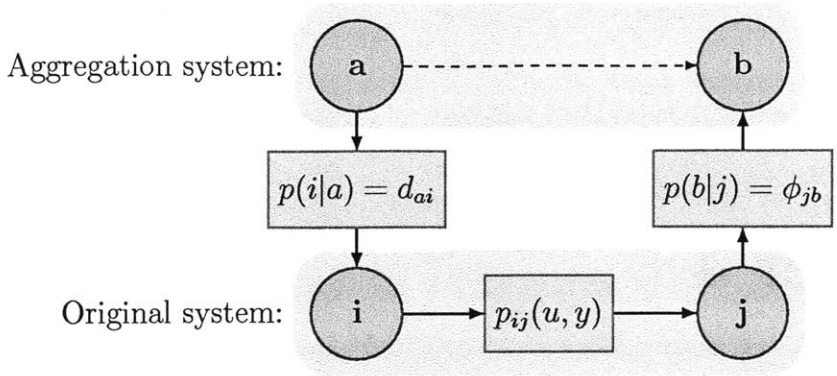


Figure 4-1: Aggregation state transition model. The transition probability from a to b with control u and forecast y is $\widehat{p}_{ab}(u, y) \triangleq \sum_{i=1}^{n} d_{ai} \sum_{j=1}^{n} p_{ij}(u, y)\phi_{jb}$. The transition cost $\widehat{g}(a, u, y)$ is $\sum_{i=1}^{n} d_{ai} g(i, u, y)$

the following 3 steps:

1. Start from the aggregate state a, an original state i is generated with probability $p(i|a) = d_{ai}$.

2. With a control u, state i then changes to state j of the original system with transition probability $p_{ij}(u, y)$. During the transition between state i and j, the transition cost $g(i, u, y)$ is accumulated.

3. Start from the state j, an aggregate state b is generated with aggregation probability $p(b|j) = \phi_{jb}$.

Based on this model, we then define the aggregate system rigorously with transition probability $\widehat{p}_{ab}(u, y)$ and transition-cost $\widehat{g}(a, u, y)$ as follows,

$$\widehat{p}_{ab}(u, y) \triangleq \sum_{i=1}^{n} d_{ai} \sum_{j=1}^{n} p_{ij}(u, y)\phi_{jb}. \tag{4.1}$$

$$\widehat{g}(a, u, y) \triangleq \sum_{i=1}^{n} d_{ai} g(i, u, y). \tag{4.2}$$

Then, the associated Bellman's equation is

$$r^\star(a) = \sum_i d_{ai} \sum_y p(y) \min_{u \in U(i,y)} \left[g(i,u,y) + \sum_j p_{ij}(u,y) \sum_b \phi_{jb} r^\star(b) \right], \text{ for all } a, \tag{4.3}$$

where the solution $r^\star(a)$ is the optimal cost-to-go for the aggregate state a. Any optimal policy $\widehat{\mu}^\star$ satisfies:

$$\widehat{\mu}^\star(i,y) = \arg \min_{u \in U(i,y)} \left[g(i,u,y) + \sum_j p_{ij}(u,y) \sum_b \phi_{jb} r^\star(b) \right]. \tag{4.4}$$

The policy $\bar{\mu}_k$ specified by the approximated cost-to-go function r_k is then defined as follows,

$$\widehat{\mu}_k(i,y) = \arg \min_{u \in U(i,y)} \left[g(i,u,y) + \sum_j p_{ij}(u,y) \sum_b \phi_{jb} r_k(b) \right]. \tag{4.5}$$

For further discussion, we introduce some notation in the next section.

4.2 Matrix Representation

We want to use matrix representation to simplify notation. We will use the same notation for the original system described in Chapter 2. The number m denotes the number of aggregate states except for the termination state. Most matrices or vectors below depend only on the policy μ but not the vector r. When there is no confusion, we do not write μ explicitly.

4.2.1 Aggregation and Disaggregation Probabilities

The matrix of aggregation probabilities is

$$\Phi \triangleq \begin{bmatrix} \phi_{11} & \cdots & \phi_{1m} \\ \phi_{21} & \cdots & \phi_{2m} \\ \vdots & \ddots & \vdots \\ \phi_{n1} & \cdots & \phi_{nm} \end{bmatrix}.$$

The matrix of disaggregation probabilities is

$$D \triangleq \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{m1} & d_{m2} & \cdots & d_{mn} \end{bmatrix}.$$

4.2.2 Approximate Cost-to-Go Function

The vector of the cost-to-go values of the aggregate problem is defined as

$$r_k \triangleq \begin{bmatrix} r_k(1) \\ \vdots \\ r_k(m) \end{bmatrix}.$$

Using the definition of Φ and r_k, and comparing Eq. 2.4 with Eq. 4.5, we see that the policy $\widehat{\mu}_k$ is exactly the same as μ_k. We will use the notation μ_k in the subsequent discussion.

4.2.3 Transition Probabilities

The transition probability from aggregate state a to b with the control $u = \mu(i, y)$ is

$$\sum_i d_{ai} \sum_j \sum_y p(y) p_{ij}(\mu(i, y), y) \phi_{jb}.$$

We define the matrix of transition probabilities for the stationary policy μ to be

$$\begin{aligned}
\widehat{P} = \widehat{P}_\mu \triangleq & \begin{bmatrix} \sum_i d_{1i} \sum_j \sum_y p(y) p_{ij}(\mu(i,y),y) \phi_{j1} & \cdots & \sum_i d_{1i} \sum_j \sum_y p(y) p_{ij}(\mu(i,y),y) \phi_{jm} \\ \vdots & \ddots & \vdots \\ \sum_i d_{mi} \sum_j \sum_y p(y) p_{ij}(\mu(i,y),y) \phi_{j1} & \cdots & \sum_i d_{mi} \sum_j \sum_y p(y) p_{ij}(\mu(i,y),y) \phi_{jm} \end{bmatrix} \\
= & \begin{bmatrix} \sum_i d_{1i} \sum_j \bar{p}_{ij}(\mu) \phi_{j1} & \cdots & \sum_i d_{1i} \sum_j \bar{p}_{ij}(\mu) \phi_{jm} \\ \vdots & \ddots & \vdots \\ \sum_i d_{mi} \sum_j \bar{p}_{ij}(\mu) \phi_{j1} & \cdots & \sum_i d_{mi} \sum_j \bar{p}_{ij}(\mu) \phi_{jm} \end{bmatrix} \\
= & D P_\mu \Phi,
\end{aligned}$$

where $\bar{p}_{ij}(\mu)$ and P_μ are defined in Section 2.2.2.

4.2.4 Expected Transition Cost Function

With the stationary policy μ, the expected transition cost from the aggregation state a to the next aggregate state is $\sum_i d_{ai} \sum_y p(y) g(i, \mu(i, y), y)$. As a result, we can define the vector of expected transition cost values to be

$$\widehat{g} = \widehat{g}_\mu \triangleq \begin{bmatrix} \sum_i d_{1i} \sum_y p(y) g(i, \mu(i,y), y) \\ \vdots \\ \sum_i d_{mi} \sum_y p(y) g(i, \mu(i,y), y) \end{bmatrix} = \begin{bmatrix} \sum_i d_{1i} g_\mu(i) \\ \vdots \\ \sum_i d_{mi} g_\mu(i) \end{bmatrix} = D g_\mu,$$

where g_μ and $g_\mu(i)$ are defined in Section 2.2.3.

4.2.5 $\widehat{T}$ Mapping

Similar to Section 2.2.5, we denote $\widehat{T}_\mu$ as the one-step look-ahead mapping for the aggregate problem with policy μ [Ber11]. The vector $\widehat{T}_\mu(r)$ is the vector of the cost-to-go values where the controller μ is applied at the first stage followed by a policy with cost-to-go vector r:

$$\widehat{T}_\mu r(a) \triangleq \sum_i d_{ai} \sum_y p(y) \left[g(i, \mu(i,y), y) + \sum_j p_{ij}(u,y) \sum_b \phi_{jb} r(b) \right], \text{ with all } a.$$

We notice that

$$\sum_i d_{ai} \sum_y p(y) g(i, \mu(i,y), y) = \widehat{g}_\mu(a),$$

and

$$\sum_i d_{ai} \sum_y p(y) \sum_j p_{ij}(u,y) \sum_b \phi_{jb} r(b) = \widehat{P}_\mu r(a).$$

As a result, the matrix representation of $\widehat{T}(r)$ satisfies

$$\widehat{T}(r) = \widehat{T}_\mu(r) \triangleq \begin{bmatrix} \widehat{T}_\mu r(1) \\ \vdots \\ \widehat{T}_\mu r(m) \end{bmatrix} = \widehat{g}_\mu + \widehat{P}_\mu r = D g_{\mu_k} + D P_{\mu_k} \Phi r.$$

With this definition, $\widehat{T}_\mu$ can be applied to any vector r, not necessarily a valid cost-to-go vector.

4.3 Policy Iteration

Policy iteration is an iterated method which terminates in a finite numbers of iterations with an optimal policy in stochastic shortest problems which terminate with probability 1. There are two phases in policy iteration: *policy improvement* and *policy evaluation*. We will update r_k through policy iteration for the aggregate system. In the beginning of an iteration, we have an approximate cost-to-go function r_k. In the policy improvement phase, we get the improved policy μ_k from Eq. 4.5.

In the policy evaluation phase, we update r_{k+1} to be the cost-to-go vector of the policy μ_k. As a result, r_{k+1} is the fixed point of the mapping $\widehat{T}_{\mu_k}$.

$$r_{k+1} = \widehat{T}_{\mu_k} r_{k+1}$$

With matrix representation, the equation is simplified,

$$r_{k+1} = Dg_{\mu_k} + DP_{\mu_k}\Phi r_{k+1}.$$

Therefore, the cost-to-go vector r_{k+1} can be calculated with a matrix inversion $(I - DP\Phi)^{-1}Dg$. We can write this inversion in the form of Eq. 1.2 with $A_{\mu_k} = DP_{\mu_k}\Phi$ and $b_{\mu_k} = Dg_{\mu_k}$. One of the major differences between the aggregation methods and the projected equation methods is that here the matrix A_{μ_k} is monotone and hence vectors $\{r_k\}$ will converge [Ber10b]. In fact, policy iteration terminates when $r_{k+1} = r_k$, both are the optimal cost-to-go $r^\star$ for the aggregation problem, after a finite number of iterations. This indicates that the aggregation methods provide a more regular behavior.

Chapter 5

Experiments with Tetris

5.1 Problem Description

As mentioned in [TS09], there are several variations of Tetris games. In this thesis, we consider a simplified version of Tetris, used by most researchers, where horizontal transitions and rotations are not allowed once the tetromino starts falling down [LKS93]. In this section, we formulate this version of Tetris into a DP problem of the type introduced in Chapter 2.

Tetris is a game where tetrominoes are placed one-by-one into a board with a width W and a height H. Each cell can be either *occupied* or *empty*. The configuration of the board is the state in the DP problem, denotes x_k. At the beginning of a game, every cell is empty. The empty board is the start state of the stochastic shortest DP problem and is denoted 1.

In a regular Tetris game, $W \times H = 10 \times 20$, and there are 7 different types of tetrominoes as shown in Fig. 5-1. At the beginning of each stage, a tetromino y_k is generated randomly with uniform distribution. The controller then specifies where to put the tetromino y_k by a rotational orientation o_k and a horizontal position s_k. Different tetrominoes have different possible number of orientations. We enumerate the rotational orientation o. The complete configurations of pairs (y, o) are shown in Fig. 5-2. The tetromino y_k falls down and stops right before overlapping with any occupied cell. The position where the tetromino stops is denoted $POS(x_k, y_k, o_k, s_k)$.

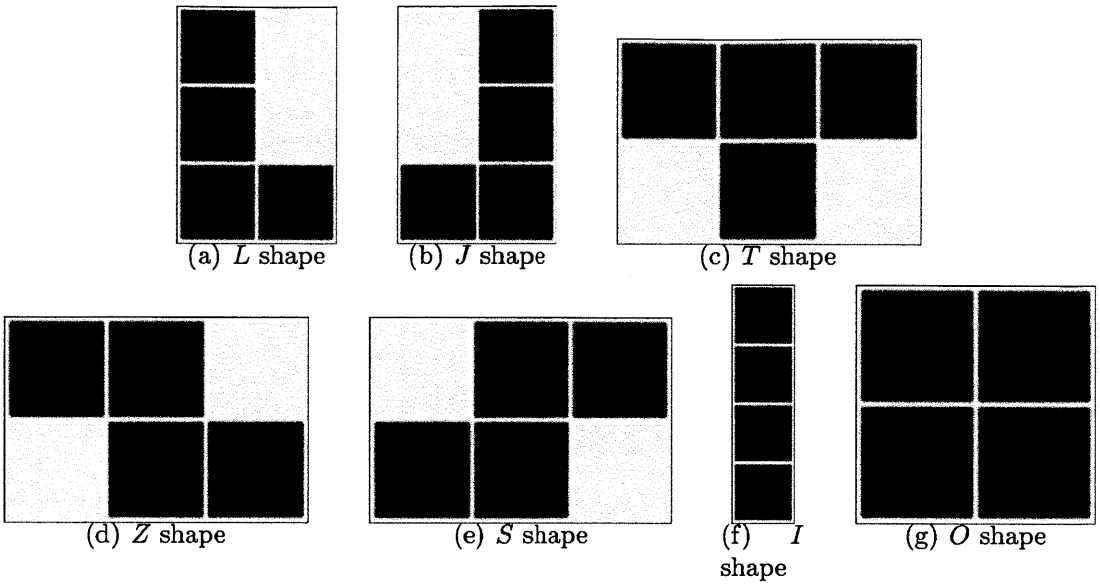


Figure 5-1: The tetrominoes

The corresponding cells of this position become occupied. When there is a row with all W cells occupied, we say the row is *full*. When a row is full, it is eliminated and the rows above all drop by 1 row. The top row is then replaced by empty cells. More than one row can be eliminated when a tetromino is dropped. The number of rows eliminated is added to the score, which starts at 0 when the game starts. We want to maximize the score by minimizing the cumulative cost of a game. Therefore, the cost $g(i, POS(i, y, o, s), y)$ is defined to be (-1) times the number of rows eliminated when y is put at $POS(i, y, s, o)$.

It is possible that for some pairs of orientation and position the tetromino does not fit in the board when dropped. In this case, the game is over. The state where the game is over is the cost free state of the stochastic shortest DP problem and is denoted 0. The set of admissible controls is the set of possible positions where tetrominoes can be put. When no choice of $POS(x_k, y_k, o_k, s_k)$ can avoid the termination of the game, we define the only admissible control to be an artificial termination decision t.

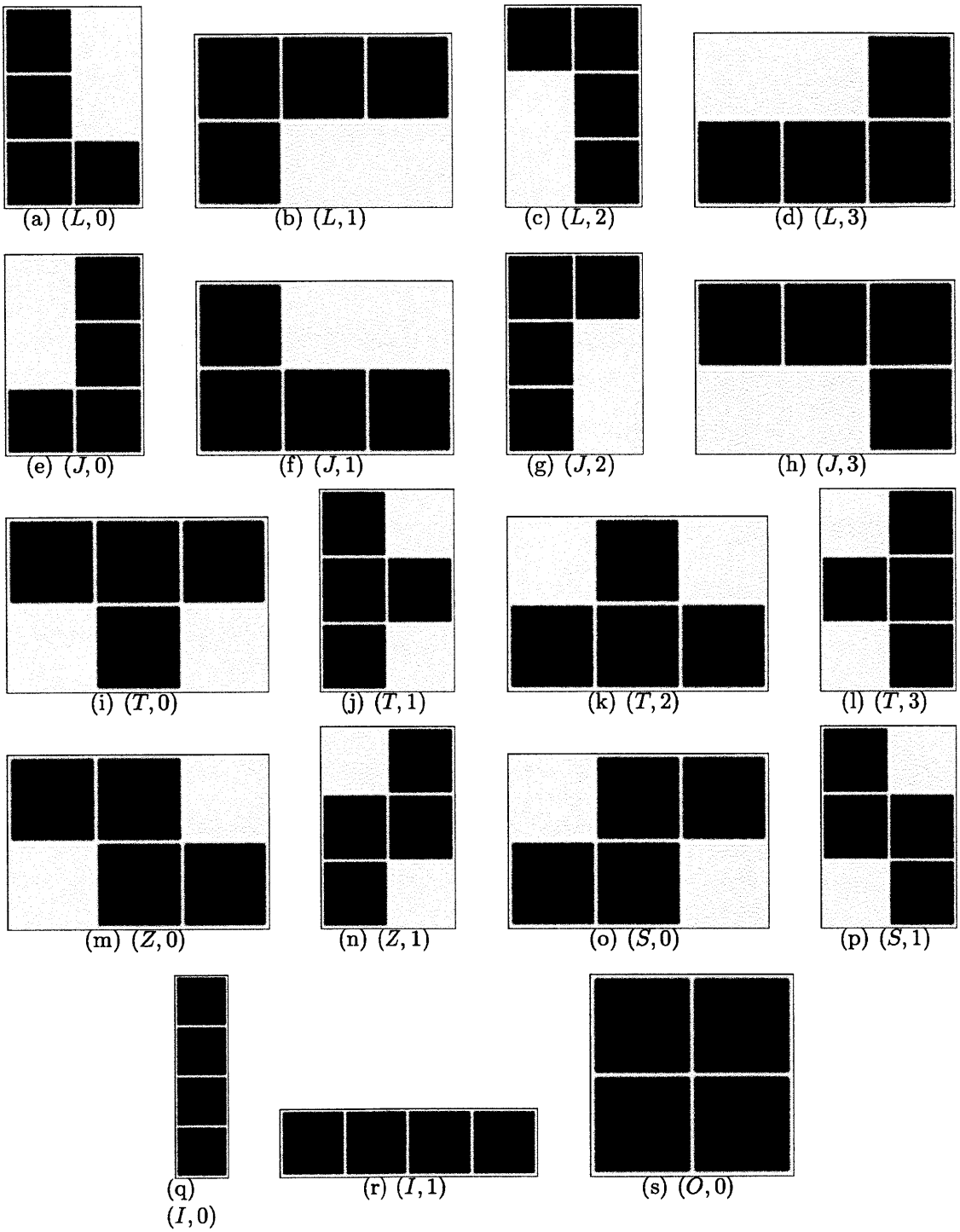


Figure 5-2: The configurations for pairs (tetromino, orientation)

$$\text{State}: x_k = \begin{cases} 0 & \text{, if gameover.} \\ \text{The configuration of board} & \text{, otherwise.} \end{cases}$$

We denote the configuration of the board as a matrix $B = [b_{ij}]$

$$\text{with } b_{ij} = \begin{cases} 1, \text{ if the cell at position } (i,j) \text{ is occupied.} \\ 0, \text{ otherwise.} \end{cases}$$

We know that a Tetris board state contains no full row and no occupied cells are above an empty row if the state is obtained through playing a game. We refer to these states *good* states and other states *bad* states.[1]

Forecast: $y_k \sim \{L, J, T, Z, S, I, O\}$ with uniform distribution.

Admissible

$$\text{Controls: } U(x_k, y_k) = \begin{cases} \{POS(x_k, y_k, o_k, s_k) \text{ which fits in the board}\} & \text{, if possible.} \\ \{t\} & \text{, otherwise.} \end{cases}$$

o_k = The rotational orientation of y_k.

s_k = The left-most horizontal position where y_k is put.

POS = The function mapping (x_k, y_k, o_k, s_k) to the occupied cells.

Control: $u_k = \mu(x_k, y_k) \in U_k(x_k, y_k)$.

State Transition: $x_{k+1} = f(x_k, u_k, y_k)$

$$= \begin{cases} 0 & \text{, if } x_k = 0 \text{ or } u_k = t. \\ \text{The configuration of} & \\ \text{board after putting } y_k \text{ at } u_k & \text{, otherwise.} \end{cases}$$

Transition cost: $g(x_k, u_k, y_k) = (-1) \times$ (Number of rows eliminated instantaneously).

[1]Not every good state can be arrived at through playing a game but every bad state cannot.

An example is shown in Fig. 5-3. The original state x_k is shown in Fig. 5-3(a). The forecast tetromino is $y_k = I$. All possible locations to put the tetromino y_k are shown as the light green cells in Fig. 5-3(f) to Fig. 5-3(i). As a result, the set of admissible controls is $U(x_k, y_k) = \{u^1, u^2, u^3, u^4\}$. The possible next states x_{k+1} with different controls are shown in Fig. 5-3(f) to Fig. 5-3(i). In Fig. 5-3(f), the transition cost g is -2. In all other figures, the transition cost g is 0. Another example is in Fig. 5-4. When the state x_k is as the figure shows and $y_k = O$, there is no location to place the tetromino. As a result, the only admissible control is the termination decision t.

5.2 Small Board Version

We will test the behavior of the projected equation methods in Tetris with a small board where $W = 3$ and $H = 4$ in order to apply exact calculation. We will also test aggregation methods with smaller boards. When we are playing with a smaller board, the tetrominoes may be relatively too big. As a result, we introduce 3 different types of smaller tetrominoes: $\{v, i, o\}$, as illustrated in Fig. 5-5. Similar to regular tetrominoes, different smaller tetrominoes have different numbers of possible rotational orientations. A complete list of configurations is shown in Fig. 5-6. The probabilities of the forecast y_k is defined to be: with probability p_s, $y_k \sim \{v, i, o\}$ with uniform distribution. With probability $(1 - p_s)$, $y_k \sim \{L, J, T, Z, S, I, O\}$ with uniform distribution. In summary, the probability distribution for y_k is replaced with

$$\text{Forecast: } y_k \sim \begin{cases} \{L, J, T, Z, S, I, O\} \text{ each with probability } \frac{1-p_s}{7}, \\ \{v, i, o\} \text{ each with probability } \frac{p_s}{3}. \end{cases}$$

It can be proved that when $p_s = 1$, the optimal policy does not terminate for a 3×4 board.

Theorem 5.2.1. *There exists a policy for a* 3×4 *board such that the costs-to-go of*

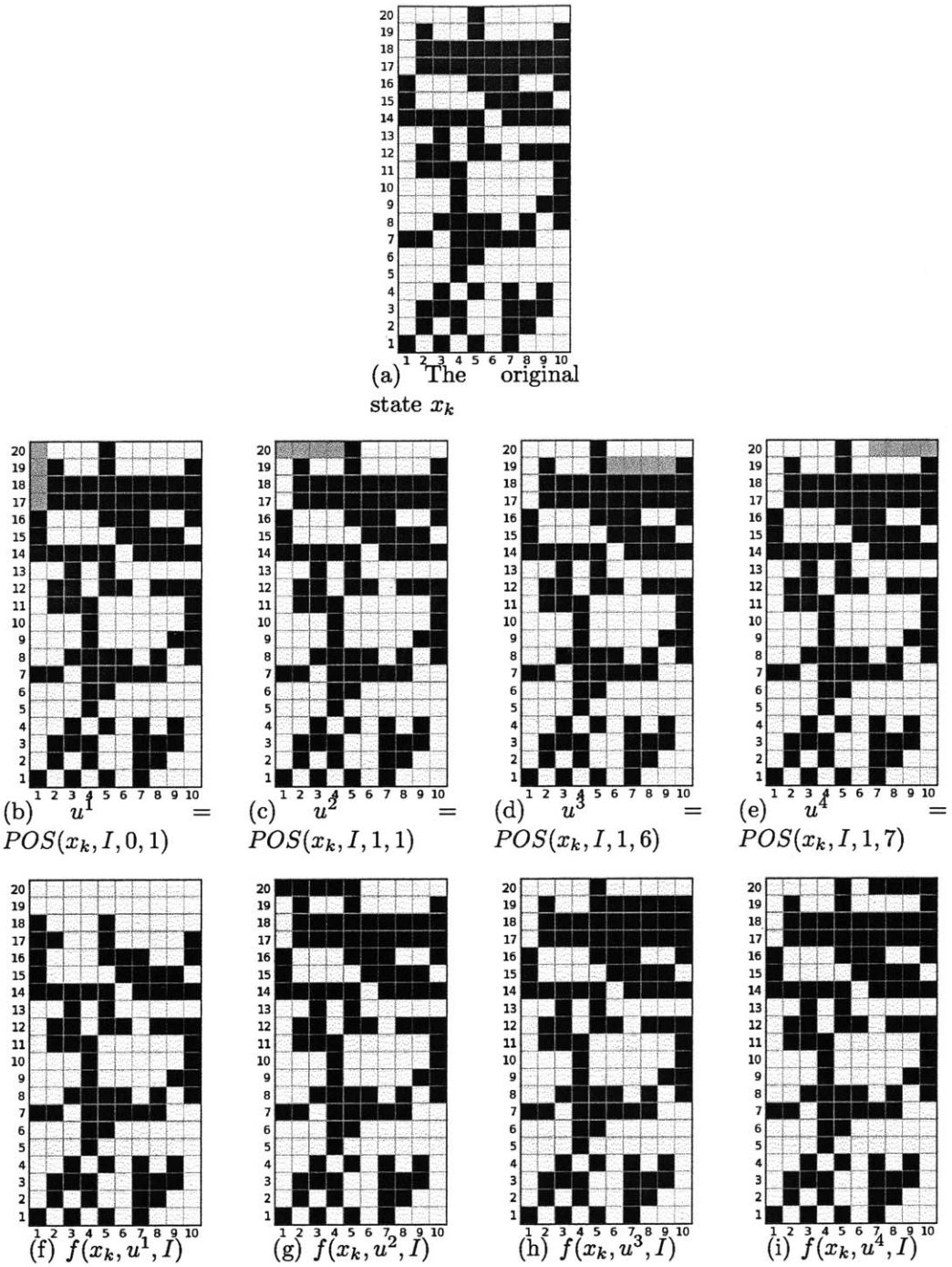


Figure 5-3: This is an example of Tetris. The original state x_k is shown in Figure 5-3(a). The coming tetromino is $y_k = I$. The possible locations to put the tetromino y_k are shown as the light green cells in Figure 5-3(f) to Figure 5-3(i). As a result, the set of admissible controls is $U(x_k, y_k) = \{u^1, u^2, u^3, u^4\}$. The next states with different controls are shown in Figure 5-3(f) to Figure 5-3(i). In Figure 5-3(f), the transition cost g is -2. In all other figures, the transition cost g is 0.

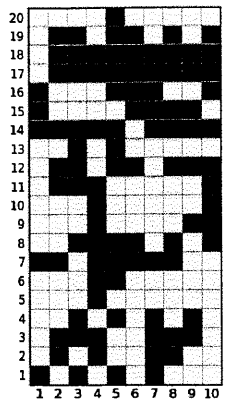
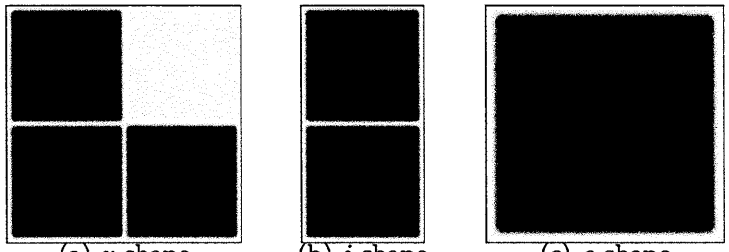
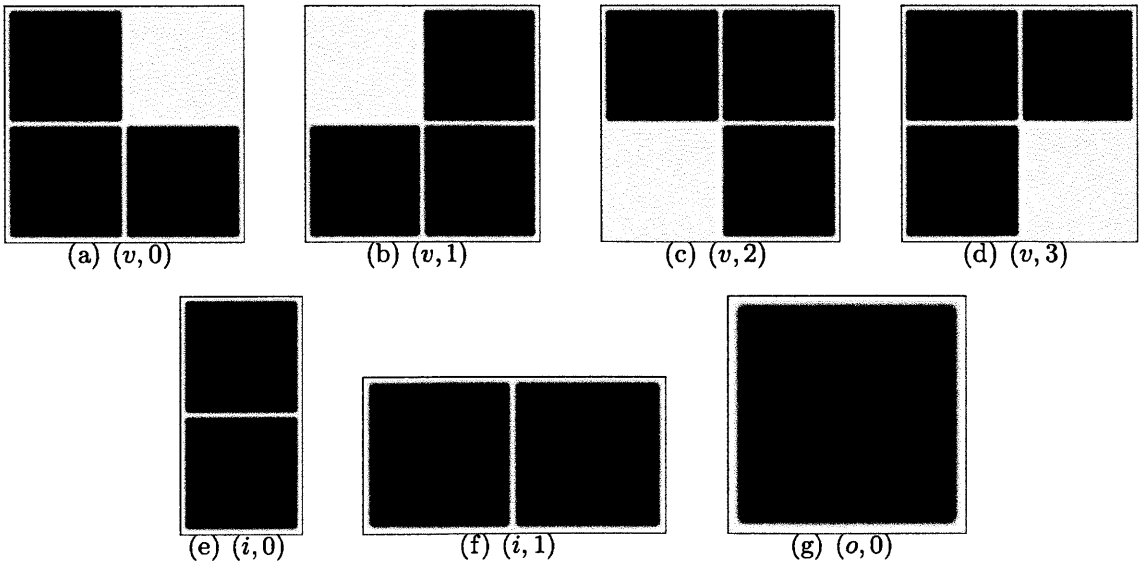


Figure 5-4: When $y_k = O$,there is no location to place the tetromino. As a result, the only admissible control is the termination decision t.



(a) v shape (b) i shape (c) o shape

Figure 5-5: The small tetrominoes



(a) $(v,0)$ (b) $(v,1)$ (c) $(v,2)$ (d) $(v,3)$

(e) $(i,0)$ (f) $(i,1)$ (g) $(o,0)$

Figure 5-6: The configuration for pairs (tetromino, orientation)

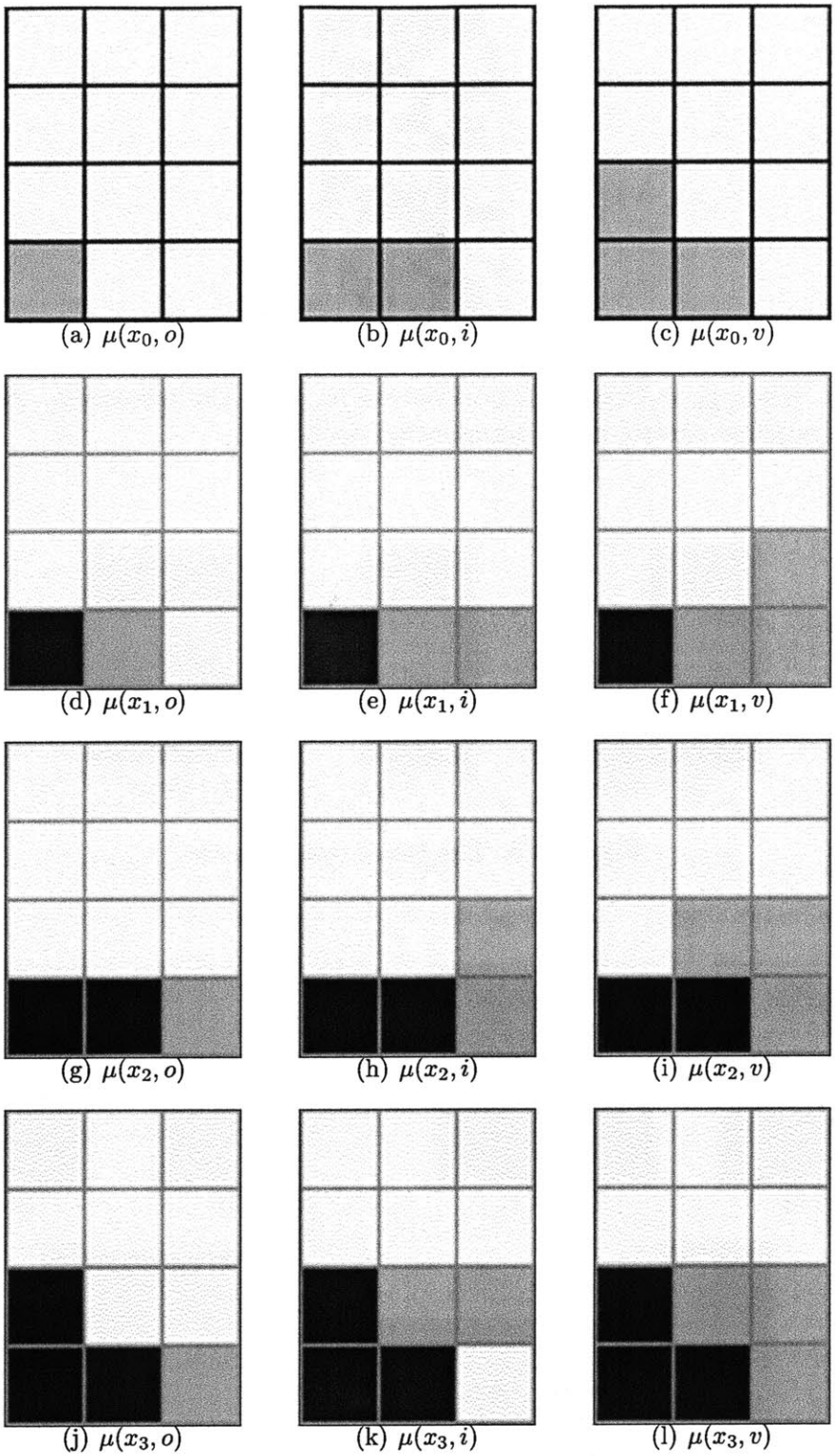


Figure 5-7: The policy μ. The configuration of dark green cells is the state and the set of light green cells is where the coming tetromino is dropped.

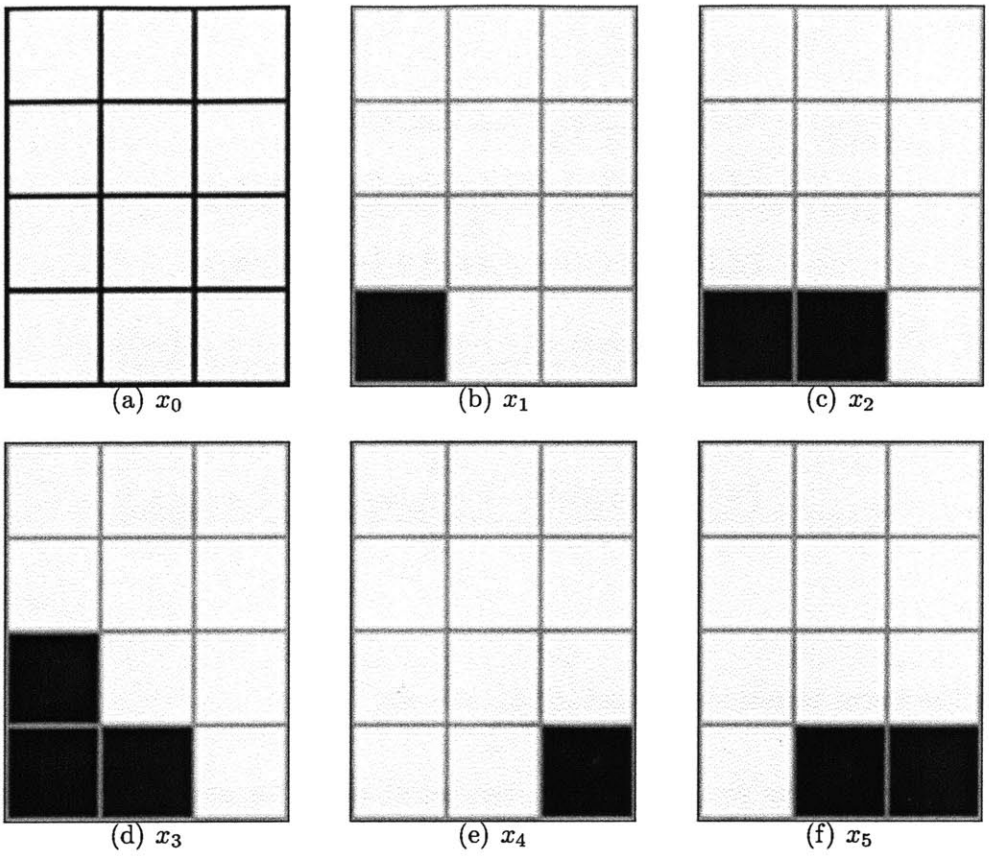


Figure 5-8: Important states for a 3×4 board.

an empty board is $-\infty$ *when* $p_s = 1$.

Proof. Let us first introduce states x_0, x_1, x_2, x_3, x_4, and x_5 as shown in Fig. 5-8 and a *symmetric* controller μ as discribed in Fig. 5-7, where symmetric controller means the places to put the flipped tetrominoes are horizontal flipped for horizontal flipped states. We will show that the policy μ provides $J_\mu(x_0) = -\infty$. We know that the cost-to-go for horizontal flipped states are the same for a symmetric controller. As a result, $J_\mu(x_1) = J_\mu(x_4)$, and $J_\mu(x_2) = J_\mu(x_5)$. Therefore, we can get the following

equations:

$$
\begin{aligned}
J_\mu(x_0) &= \frac{(J_\mu(x_1)) + (J_\mu(x_2)) + (J_\mu(x_3))}{3}. \\
J_\mu(x_1) &= \frac{(J_\mu(x_2)) + (-1 + J_\mu(x_2)) + (-1 + J_\mu(x_4))}{3} \\
&= \frac{(J_\mu(x_2)) + (-1 + J_\mu(x_2)) + (-1 + J_\mu(x_1))}{3}. \\
J_\mu(x_2) &= \frac{(-1 + J_\mu(x_0)) + (-1 + J_\mu(x_4)) + (-1 + J_\mu(x_5))}{3} \\
&= \frac{(-1 + J_\mu(x_0)) + (-1 + J_\mu(x_1)) + (-1 + J_\mu(x_2))}{3}. \\
J_\mu(x_3) &= \frac{(-1 + J_\mu(x_1)) + (-1 + J_\mu(x_2)) + (-2 + J_\mu(x_0))}{3}.
\end{aligned}
$$

Solving the linear equations, we get $J_\mu(x_0) = -\infty$. □

It follows from the same algorithm, the result holds for any $3 \times H$ board with $H \geq 2$. We can also show the same result holds for any $4 \times H$ board with $H \geq 2$. When $p_s < 1$, there is a nonzero probability for each original tetromino to occur, so the modified Tetris game still terminates with probability 1. Based on this fact and this theorem, the optimal cost for the smaller version of Tetris can be as low as possible when we choose p_s arbitrarily close to 1. In our implementation, we set either $p_s = 0.8$ or $p_s = 0$ depending on the board size.

5.3 Pseudometric of Policies

There are several reasons to define a pseudometric over the space of policies. First of all, when we define a policy μ_k by an approximated cost-to-go function $\bar{J}_k = \Phi r_k$, there are infinitely many choices of r_k, but the number of policies is finite. As a result, there must be cases where different values of r specify the same policy. In this case, while the norm of the difference between two approximation vectors is not zero, we should consider other measures of distance where it is zero. Furthermore, we do not know whether good approximation provides a good policy. Instead of having a

good approximation in s weighted norm, we might want to consider how often two controllers are making the same decision.

For the reasons above, we introduce a pseudometric over the space of policies that could measure how similar two policies are. For a pair of state $x \neq 0$ and forecast y, let $S_\mu(x,y)$ denote the set of possible admissible controls attaining the minimum in Eq. 2.3, when μ is optimal, or in Eq. 2.4, when μ is one of the policies suggested by approximator. We define the distance between μ^1 and μ^2 for a state and forecast pair (x,y) to be

$$d(S_{\mu^1}(x,y), S_{\mu^2}(x,y)) \triangleq 1 - \frac{|S_{\mu^1}(x,y) \cap S_{\mu^2}(x,y)|}{|S_{\mu^1}(x,y) \cup S_{\mu^2}(x,y)|}.$$

It is proved that d is a metric space over the space of sets [BKV96]. We define

$$D_\mu(\mu^1, \mu^2) \triangleq \sum_{x,y} q_\mu(x,y) d(S_{\mu^1}(x,y), S_{\mu^2}(x,y)),$$

where $q_\mu(x,y)$ is the occurrence probability for the state and forecast pair (x,y). Then, D_μ is a pseudometric space on policies, as shown in the following theorem:

Theorem 5.3.1. *D_μ is a pseudometric space on policies.*

Proof. We need to verify the following 3 conditions:

1. $\mathbf{D_\mu(\mu^1, \mu^1) = 0}$:

$$D_\mu(\mu^1, \mu^1) \triangleq \sum_{x,y} q_\mu(x,y) d(S_{\mu^1}(x,y), S_{\mu^1}(x,y)) = 0,$$

because $d(S_{\mu^1}(x,y), S_{\mu^1}(x,y)) = 0$, for every pair of (x,y).

2. **Symmetry:**

$$\begin{aligned} D_\mu(\mu^1, \mu^2) &\triangleq \sum_{x,y} q_\mu(x,y) d(S_{\mu^1}(x,y), S_{\mu^2}(x,y)) \\ &= \sum_{x,y} q_\mu(x,y) d(S_{\mu^2}(x,y), S_{\mu^1}(x,y)) \\ &= D_\mu(\mu^2, \mu^1), \text{ from the symmetry of the metric } d. \end{aligned}$$

3. **Triangle inequality:**

$$\begin{aligned} & D_\mu(\mu^1, \mu^2) + D_\mu(\mu^2, \mu^3) \\ \triangleq & \sum_{x,y} q_\mu(x,y) d(S_{\mu^1}(x,y), S_{\mu^2}(x,y)) + \sum_{x,y} q_\mu(x,y) d(S_{\mu^2}(x,y), S_{\mu^3}(x,y)) \\ = & \sum_{x,y} q_\mu(x,y) \left[d(S_{\mu^1}(x,y), S_{\mu^2}(x,y)) + d(S_{\mu^2}(x,y), S_{\mu^3}(x,y)) \right] \\ \geq & \sum_{x,y} q_\mu(x,y) d(S_{\mu^1}(x,y), S_{\mu^3}(x,y)) \\ = & D_\mu(\mu^1, \mu^3), \text{ from the triangle inequality of the metric } d. \end{aligned}$$

□

Furthermore, if $D_{\mu^1}(\mu^1, \mu^2) = 0$, then the two controllers μ^1 and μ^2 are equivalent. In other words, the two policies always choose the same controls $\mu^1(x,y) = \mu^2(x,y)$ for every state and forecast pair (x,y) that they may visit.

Theorem 5.3.2. *If $D_{\mu^1}(\mu^1, \mu^2) = 0$, then the two controllers μ^1 and μ^2 are equivalent.*

Proof. If $D_{\mu^1}(\mu^1, \mu^2) = 0$,

$$\sum_{x,y} q_{\mu^1}(x,y) d(S_{\mu^1}(x,y), S_{\mu^2}(x,y)) \triangleq D_{\mu^1}(\mu^1, \mu^2) = 0.$$

Because $d(S_{\mu^1}(x,y), S_{\mu^2}(x,y))$ is nonnegative, we have $d(S_{\mu^1}(x,y), S_{\mu^2}(x,y)) = 0$ whenever $q_{\mu^1}(x,y) = 0$. Since μ^1 and μ^2 always make the same decision for every

(x, y) that might be visited by controller μ^1, controller μ^2 cannot visit any pair (x, y) which is not visited by μ^1. Therefore, μ^1 and μ^2 are equivalent. □

If the sets $S_{\mu^1}(x, y)$ and $S_{\mu^2}(x, y)$ all have only one element for any (x, y), then this metric could be interpreted as the probability that μ^1 and μ^2 make different decisions following the trajectories under policy μ. As a result, D_μ may be interpreted as the opposite of the degree of similarity between two policies μ^1 and μ^2. As we will see in Chapter 6, the best policies are closer to the optimal policy $\mu^\star$ with the pseudometric space $D_{\mu^\star}$ in our experimentation.

5.4 Feature Functions for Projected Equation Methods

We are interested in using a linear architecture based on the followsing $m = (2W+2)$ features for the cost-to-go function of the board when $x_k \neq 0$ [BI96]:

$$\begin{aligned}\phi^i(x_k) &= \text{ the highest occupied cell of the } i\text{th column, for } i = 1, 2, \cdots, W \\ &= \begin{cases} \max\{j | b_{ij} = 1\}, \text{ if } \{b_{ij} = 1\} \neq \emptyset \\ 0, \text{ otherwise} \end{cases} \text{, for } i = 1, 2, \cdots, W.\end{aligned}$$

$$\phi^{W+i}(x_k) = |\phi^i(x_k) - \phi^{i+1}(x_k)| \text{ , for } i = 1, 2, \cdots, (W-1).$$

$$\phi^{2W}(x_k) = \max\{\phi^i(x_k) | i = 1, 2, \cdots, W\}.$$

$$\phi^{2W+1}(x_k) = \text{the number of holes (empty cells with at least one occupied cell above it.)}$$

$$\phi^{2W+2}(x_k) = 1.$$

We note that the last feature function ϕ^{2W+2} is constant. As a result, the policy does not depend on the corresponding weight. However, the existence of this function provides a different trajectory for the vector r when the projected equation methods are used.

5.5 Aggregation Methods

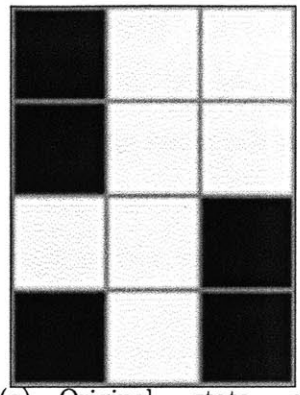
We approximate Tetris into two different *hard aggregation* problems. Hard aggregation means for fixed i, $\phi_{iy} = 1$ for only one aggregate state $y = y_i$, and 0 for all other aggregation states. We define some features so that original states with the same features have the same aggregate state. Here we propose two different ways to formulate aggregate problems. In the first method, the disaggregation probabilities are uniform among some subset of original states depending on the aggregate state. We refer to this method as *aggregation method with uniform disaggregation probabilities.* In the second method, for a fixed aggregate state x, the disaggregation probability d_{xi} is 1 for a certain original state $i = i_x$ and 0 for all other states. We can view the original state i_x with $d_{xi_x} = 1$ as the representative original state of the aggregate state x, and we refer to this method as an *aggregation method with representative states.*

5.5.1 Aggregation Methods with Uniform Disaggregation Probabilities

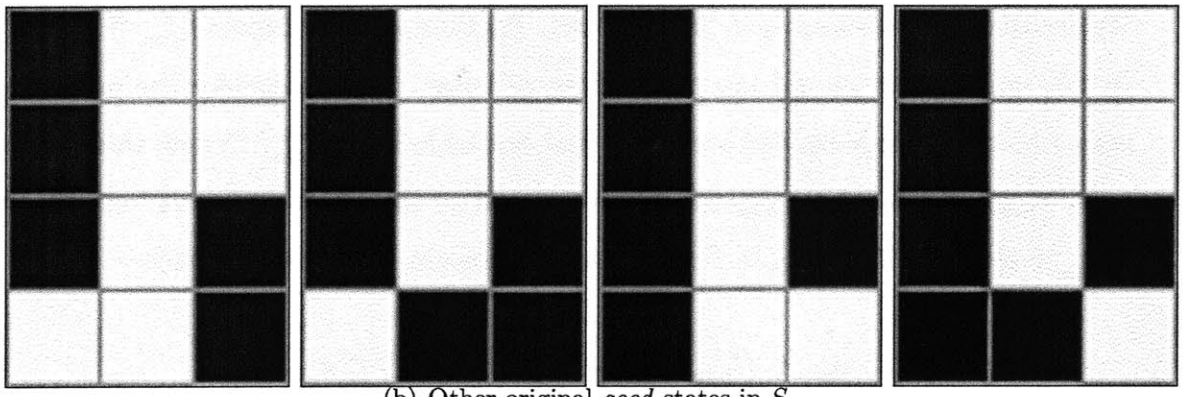
For a $W \times H$ board state x, assume the highest occupied cell in the ith column is $h_i(x)$, for $i = 1, 2, \cdots, W$. The feature $h_i = 0$ if no cell is occupied in the column.[2] Denote $h_{max} = \max_i h_i$, the height of the highest column, and N_{hole} the number of holes in the board. Let s be any positive integer, we define $h_i^+ = \max(h_i, h_{max} - s)$. The aggregate state for x is defined to be $\left(h_1^+, h_2^+, \cdots, h_W^+, N_{hole}\right)$, that is, $\phi_{xi} = 1$ is and only is $i = \left(h_1^+, h_2^+, \cdots, h_W^+, N_{hole}\right)$. For an aggregate state $y = \left(h_1^+, h_2^+, \cdots, h_W^+, N_{hole}\right)$, we denote S_y the set of *good* states i with aggregation probability $\phi_{iy} = 1$. Then the disaggregation probability is defined to be

$$d_{yi} = \begin{cases} \frac{1}{|S_y|}, \text{ for } i \in S_y, \\ 0, \text{ otherwise.} \end{cases}$$

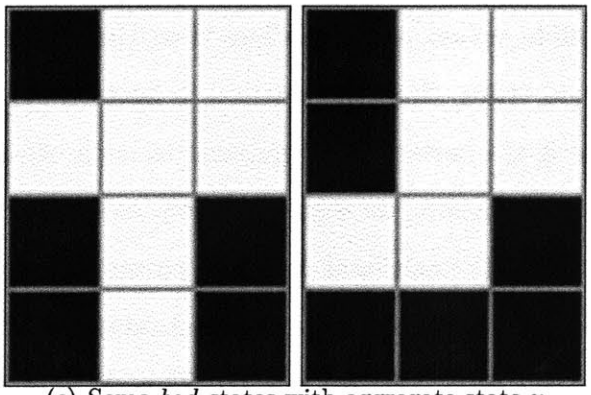
[2]It is true that $h_i(x) = \phi^i(x)$. However, the relation between the matrix Φ and ϕ^i is different in this section. As a result, we use the notation h_i to reduce confusion.



(a) Original state x with aggregate state $y = (4, 1, 2, 1)$.



(b) Other original *good* states in S_y



(c) Some *bad* states with aggregate state y

Figure 5-9: An illustration of aggregation methods with uniform disaggregation probabilities and $s = 3$. In this case, $\phi_{xi} = 1$ if and only if $i = y$. Because $|S_y| = 5$, $d_{yi} = \frac{1}{5}$ for $i \in S_y$ and $d_{yi} = 0$ for $i \notin S_y$.

An illustration is shown in Fig. 5-9 with $s = 3$. Fig. 5-9(a) is the original state x. We see that the height for each column are $(h_1, h_2, h_3) = (4, 0, 2)$. Therefore, $h_{max} = 4$ and $(h_1^+, h_2^+, h_3^+) = (4, 1, 2)$. The number of holes is $N_{hole} = 1$. As a result, the aggregate state is $y = (h_1^+, h_2^+, h_3^+, N_{hole}) = (4, 1, 2, 1)$. Fig. 5-9(b) shows all other original *good* states with the same aggregate state y. Therefore, we know that $|S_y| = 5$, so $d_{yi} = \frac{1}{5}$ for $i \in S_y$. Notice that some original states have aggregate state y but they are *bad* states so they are not in S_y. Examples of those *bad* states are shown in Fig. 5-9(c).

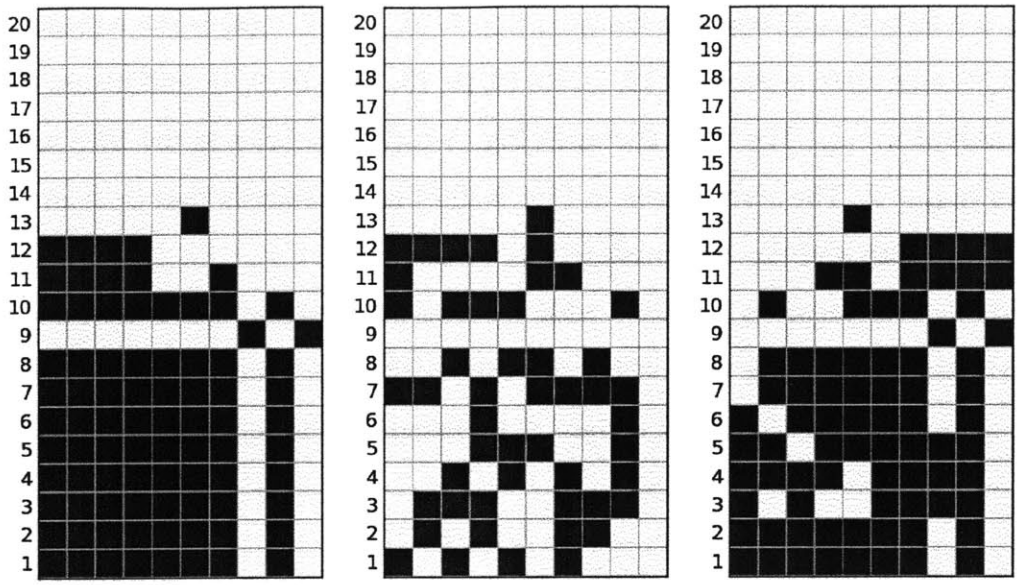
5.5.2 Aggregation Methods with Representative States

We use the same notation as in Section 5.5.1. Furthermore, we denote $h_{min}^+ \triangleq \min_i h_i^+$. In this method, we do not use the number of holes, but we consider the number of rows which contain holes and are above (but not including) h_{min}^+. Denote this number to be r_{hole}. The number r_{hole} can be interpreted as the number of rows between the $(h_{min}^+ + 1)$th and the $(h_{max} - 1)$th row which contain holes, and therefore,

$$r_{hole} \leq (h_{max} - 1) - (h_{min}^+ + 1) + 1 \leq s - 1.$$

Therefore, the possible values of r_{hole} is less then the one of N_{hole}, so this method has fewer aggregation states than the method in Section 5.5.1 with same s. We notice that states which are horizontal symmetric to each other should have the same cost-to-go function. As a result, we will treat them as the same aggregate state. Consider the two base-$(H+1)$ positional notation numbers, $\sigma_1 = \overline{h_1 h_2 \cdots h_W}_{(H+1)}$, and $\sigma_2 = \overline{h_W h_{W-1} \cdots h_1}_{(H+1)}$. If $\sigma_1 > \sigma_2$, the aggregate state $y_x = \left(h_1^+, h_2^+, \cdots, h_W^+, r_{hole}\right)$. Otherwise, $y_x = \left(h_W^+, h_{W-1}^+, \cdots, h_1^+, r_{hole}\right)$. The disaggregation probabilities are $\phi_{xy_x} = 1$ and $\phi_{xy} = 0$ when $y \neq y_x$.

Now, assume we have an aggregate state $y = (y_1, y_2, \cdots, y_W, r_{hole})$. We want to find a representative original state z. We set the height of the ith column to be y_i. Let $y_{max} = \max_i y_i$ and $y_{min} = \min_i y_i$. We know that $y_{min} \neq y_{max}$ unless $y_{min} = y_{max} = 0$. When $y_{min} = y_{max} = 0$, the representative state is an empty



(a) The representative state (b) Possible original state (c) Possible original state

Figure 5-10: The representative state and possible some original states for aggregation state $(12, 12, 12, 12, 10, 13, 11, 9, 10, 9, 2)$ with $s = 4$.

board. Now, we consider the case $y_{min} \neq y_{max}$. Denote $I_{min} = \{i | y_i = y_{min}\}$ and $I_{max} = \{i | y_i = y_{max}\}$. We know that $I_{min} \bigcap I_{max} = \emptyset$. We set the board with the following configuration[3],

$$b_{ij} = \begin{cases} \begin{cases} 1 & \text{, if } i \notin I_{min} \\ 0 & \text{, if } i \in I_{min} \end{cases} & \text{, if } j < y_{min}. \\ \begin{cases} 0 & \text{, if } i \notin I_{min} \\ 1 & \text{, if } i \in I_{min} \end{cases} & \text{, if } j = y_{min}. \\ \begin{cases} 0 & \text{, if } i \in I_{max} \text{ and } y_{max} - r_{hole} \leq j < y_{max} \\ 1 & \text{, otherwise} \end{cases} & \text{, if } y_{min} < j \leq y_i. \\ 0 & \text{, if } y_i < j. \end{cases}$$

The number of rows which contain holes above y_{min} is contributed to the top r_{hole} cells below each of the top occupied holes in the representative state. An example

[3]The representative state does not have to be a *good* state, but it has to be a good representative of the cost-to-go function.

with $s = 4$ for a 10×20 board is shown in Fig. 5-10. Fig. 5-10(a) is the representative state z_y of aggregate state $y = (12, 12, 12, 12, 10, 13, 11, 9, 10, 9, 2)$, and hence, $d_{yz} = 1$ if and only if $z = z_y$. The state shown in Fig. 5-10(b) has holes in the 10th and 11th rows so $r_{hole} = 2$. The state shown in Fig. 5-10(c) is an example that $\sigma_2 > \sigma_1$. It has holes in the 10th and the 12th rows so $r_{hole} = 2$.

Although aggregation methods with representative states seems to be a looser approximation model, we may still prefer this method for some reasons. First of all, in the aggregation methods, we update r_k by following the equation:

$$r_{k+1} = (I - DP_{\mu_k}\Phi)^{-1} D g_{\mu_k}$$

In general, the computational burden for $DP\Phi$ depends on the dimension of P, so it depends on the number of original states. Fortunately, for our problem, the matrix P is sparse, because $\bar{p}_{ij}(\mu) \neq 0$ only when j is the possible next state for i. In the representative states method, because D and Φ contain only one nonzero component at each column and each row respectively, using sparse matrix calculation techniques, the computational time of the production $DP\Phi$ only depends on the the number of aggregate states. The matrix inversion is also significantly reduced because of the sparsity of $I - DP\Phi$ [DLN+94]. On the other hand, for the uniform disaggregation probabilities methods, we need to find the size of S_y for each aggregate state y, which is time-consuming and may need to go through every *good* original states. As a result, the calculation of the representative states methods might be much lighter than the one with uniform disaggregation probabilities.

Chapter 6

Experimental Result and Conclusion

6.1 Results for Projected Equation Methods

We tested the projected equation methods with a smaller Tetris board (3×4) with $p_s = 0.8$. We denote the optimal cost-to-go $J^\star$ and the optimal policy $\mu^\star$. We have obtained $J^\star$ by using (exact) policy iteration. We chose the starting vector $r_0 \triangleq [1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 3,\ 0]'$, as suggested in [BI96]. For the fixed-c methods, we chose c to be 0 and the corresponding term for the projection $\min_r |\Phi r - J^\star|$, denotes the value $c_p \triangleq -144.643$.

We calculate the L_2 distance $L_2(\Phi r_k, J^\star)$, the L_∞ distance $L_\infty(\Phi r_k, J^\star)$, over the *good* states. We also calculate the policy distance $D_{\mu^\star}(\mu^\star, \Phi r_k)$ for the oscillating policies for each method. The results are shown in Table 6.1.

In our experiments, all projected equation methods with exact calculation end up oscillating with more than one policy. For the argument in Section 3.7, it is required that the next vector r_{k+1} depends only on μ_k, not r_k. However, since computers must discretize the vector r, the possible choices of r will be finite. As a result, in our experimental results, vectors r_k end up oscillating among few vectors in $\Re^s$ and so do the policies μ_k.

With $\lambda = 0$ and 0.999, the projected equation methods outperform the methods

Table 6.1: Results for projected equation methods. The second column indicates the number of policies involved in the oscillation. The notation $\approx$ means approximately, and $a \sim b$ means between a and b.

Method		Average cost	$L_\infty(J_k, J^\star)$	$L_2(J_k, J^\star)$	$D_{\mu^\star}(\mu^\star, \mu_k)$
LSTD(0)	2	≈ -70	≈ 52	≈ 810	≈ 0.23
LSPE(0)	5	≈ -70	≈ 55	≈ 920	≈ 0.23
LSTD$(0, c_p)$	4	≈ -70	$\approx 69 \sim 70$	$\approx 1050 \sim 1060$	≈ 0.23
LSPE$(0, c_p)$	4	≈ -70	$\approx 70 \sim 71$	≈ 1070	≈ 0.23
LSTD$(0, 0)$	10	$\approx -1 \sim -9$	≈ 99	$\approx 1810 \sim 1860$	$\approx 0.67 \sim 0.84$
LSPE$(0, 0)$	4	$\approx -9 \sim -10$	≈ 99	≈ 1800	$\approx 0.74 \sim 0.75$
LSTD(0.9)	2	≈ -68	≈ 55	≈ 880	≈ 0.23
LSPE(0.9)	4	$\approx -68 \sim -69$	$\approx 54 \sim 55$	$\approx 860 \sim 870$	≈ 0.23
LSTD$(0.9, c_p)$	4	$\approx -72 \sim -87$	$\approx 51 \sim 63$	$\approx 840 \sim 1180$	$\approx 0.19 \sim 0.31$
LSPE$(0.9, c_p)$	4	$\approx -73 \sim -89$	$\approx 54 \sim 65$	$\approx 900 \sim 1080$	$\approx 0.16 \sim 0.31$
LSTD$(0.9, 0)$	6	$\approx -4 \sim -10$	≈ 99	$\approx 1720 \sim 1830$	$\approx 0.73 \sim 0.77$
LSPE$(0.9, 0)$	31	$\approx -1 \sim -8$	≈ 99	$\approx 1780 \sim 1830$	$\approx 0.72 \sim 0.85$
LSTD(0.999)	4	$\approx -68 \sim -69$	≈ 55	$\approx 870 \sim 880$	≈ 0.23
LSPE(0.999)	4	$\approx -68 \sim -69$	≈ 55	$\approx 870 \sim 880$	≈ 0.23
LSTD$(0.999, c_p)$	2	$\approx -15 \sim -16$	$\approx 289 \sim 299$	$\approx 4060 \sim 4190$	$\approx 0.29 \sim 0.40$
LSPE$(0.999, c_p)$	2	$\approx -15 \sim -16$	$\approx 290 \sim 299$	$\approx 4060 \sim 4190$	$\approx 0.29 \sim 0.40$
LSTD$(0.999, 0)$	8	$\approx -4 \sim -10$	≈ 99	$\approx 1710 \sim 1830$	$\approx 0.73 \sim 0.77$
LSPE$(0.999, 0)$	2	$\approx -4 \sim -10$	≈ 99	$\approx 1720 \sim 1830$	$\approx 0.73 \sim 0.77$

Table 6.2: Best 5 average costs by projected equation methods.

Method	Average cost	$L_\infty(J_k, J^\star)$	$L_2(J_k, J^\star)$	$D_{\mu^\star}(\mu^\star, \mu_k)$
LSPE$(0.9, c_p)$	-88.9909	55.8694	971.636	0.158360
LSTD$(0.9, c_p)$	-86.9333	51.1765	838.332	0.194934
LSTD$(0.9, c_p)$	-86.7808	51.8306	845.953	0.195161
LSPE$(0.9, c_p)$	-79.0661	53.8910	894.058	0.217701
LSPE$(0.9, c_p)$	-77.3788	55.2582	966.091	0.253505

with a fixed weight for the constant feature. However, when $\lambda = 0.9$, the methods with $c = c_p$ perform the best. With $c = 0$, the projected equation methods is as if there are only the other $(s-1)$ features, and 0 is not close to the corresponding weight for any reasonable projected equation methods, so it does not perform well. With the projected value $c = c_p$, the weight is already where a real projection solution should be. As a result, the performance might be comparable with the original projected equation methods. This result also suggests that if we can know a good weight for the constant term c in advance, we may be able to get a better controller. However, in practice, the projection is not calculable and hence c_p is not accessible.

The costs of the best of policies involved in the oscillation for various methods are shown in Table 6.2. The top 5 policies are all from the projected equation methods with $\lambda = 0.9$ and $c = c_p$. We may think with $c = c_p$, the higher λ, the better the performance is. However, this is not so. When $\lambda = 0.999$ and $c = c_p$, the average score is not comparable to good controllers. This may be due to the fact that when λ is close to 1, the projected equation is not stable even without simulation noise but only truncation error. The best score is -88.9909. The L_2 and L_∞ distances for these policies are not the lowest. However, the policy distances $D_{\mu^\star}(\mu^\star, \mu_k)$ are the lowest for the best 4 policies. This makes sense because the lower the policy distance is, the higher the probability this policy makes the same decision as the optimal one.

We also want to compare the results of the exact calculation of projected equation methods and the simulation-based ones. The results of the LSTD(λ) methods are shown in Fig. 6-1. This figure indicates that the larger λ is, the higher the noise is. When $\lambda = 0.999$, the simulation-based result is very different from the exact calcu-

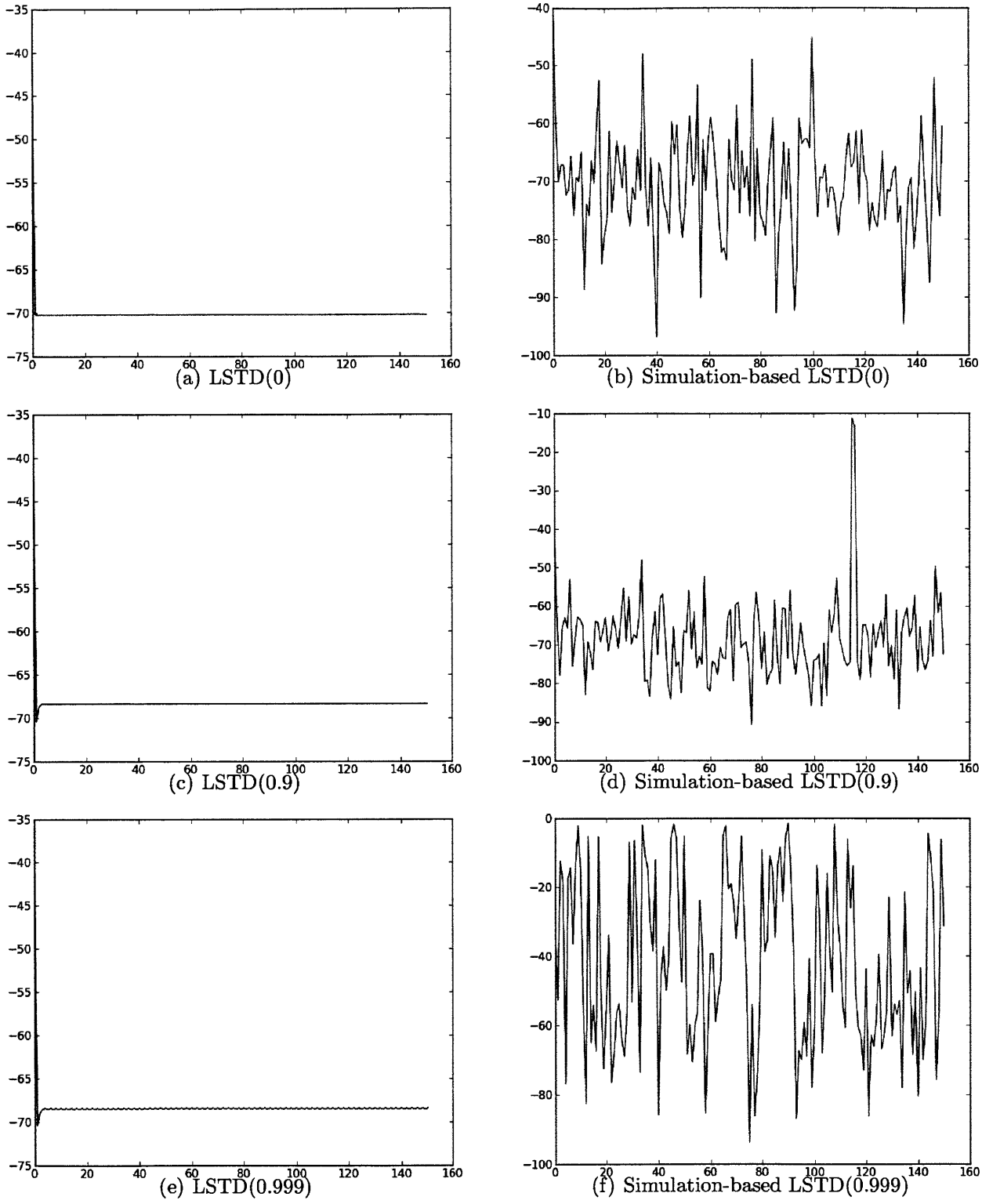


Figure 6-1: A comparison of noise in the LSTD methods with different λ. In the left column, the costs are calculated exactly and the weight vectors r_k are updated by the exact LSTD methods. In the right column, the costs are the average of 100 games and the weight vectors r_k are updated based on a simulation of 100 games.

lation. Many policies updated through simulation-based LSTD(0.999) have average costs more than -20. On the other hand, for both the simulation-based LSTD(0) and LSTD(0.9) methods, most policies have average costs lower than -50. We know that the average costs are the average of 100 games. As a result, the fluctuation of costs in the LSTD(0) method might mainly be due to the variance of the cost of one single game. On the other hand, in the LSTD(0.9) method, two policies have costs about -10 consecutively. These policies are very likely to be obtained because of the simulation error in $\bar{C}$, $\bar{d}$, and $\bar{G}$. From this Figure, we cannot see the policy oscillations for the LSTD(0) and LSTD(0.9) methods, and we can only see the policies are oscillating between two policies for the LSTD(0.999) methods. However, when we look more closely into the vectors r_k, we see that the policies are oscillating between 2, 2, and 4 policies respectively. Therefore, the average cost is not a very good way to discern different policies.

6.2 Results for Aggregation Methods

Some results are shown in Table 6.3. The table shows that the number of aggregate states is much lower than that of the original states in all the cases. The table indicates that the two different types of aggregation state methods perform as well as each other when $s = H$. When $s < H$, the uniform disaggregation probabilities method outperforms the representative states method. A possible reason is that the first method encodes the actual number of holes while the second method only considers the holes of the top few rows. However, when s is greater than or equals 3, both aggregation methods may outperform the projected equation methods.

We also tried to apply the representative states methods to a larger board with original tetrominoes. A comparison between the result for $s = 4$ on a 7×20 board is shown in Table 6.4. The representative states method achieves the average cost about -228. Although not as good as the projected equation methods, it is a promising method. Chances are that when s is getting bigger, the representative states methods may outperform the projected equation methods.

Table 6.3: A comparison between different methods for smaller versions of Tetris games with $p_s = 0.8$. The costs are the average of 100 games. The costs for projected equations are the best from $LSPE$ and $LSTD$ methods with $\lambda = 0$ and 0.9.

Board size ($W \times H$)	3×8	3×10	4×6
Number of *good* states $\sum_{i=0}^{H}(2^W - 2)^i$	2015539	72559411	8108731
Projected equations costs	≈ -5000	≈ -25000	≈ -4000
CE-based random search costs	≈ -5700	≈ -31000	≈ -9291
Uniform disaggregation probabilities costs ($s = H$)	≈ -9079	≈ -84537	
Number of aggregate states	3409	7621	
Uniform disaggregation probabilities costs	≈ -8765 ($s = 4$)	≈ -80218 ($s = 4$)	≈ -25684 ($s = 3$)
Number of aggregate states	1897	3193	5269
Representative states costs ($s = H$)	≈ -9011	≈ -87339	≈ -30010
Number of aggregate states	1741	3851	4985
Representative states costs	≈ -7456 ($s = 4$)	≈ -56052 ($s = 4$)	≈ -216274 ($s = 3$)
Number of aggregate states	571	771	981

Table 6.4: A comparison between different methods for Tetris with original tetrominoes only on a 7×20 board. The costs are the average of 100 games.

Board size ($W \times H$)	7×20
Number of *good* states $\sum_{i=0}^{H}(2^W - 2)^i$	$\approx 1.03 \times 10^{42}$
LSTD(0) costs	≈ -782
CE-based random search costs	≈ -1000
Representative states costs ($s = 4$)	≈ -228
Number of aggregate states	1992847

6.3 Comparisons and Conclusions

In this section we summarize our computational experimentation on Tetris with projected equation and aggregation methods. As expected, the results verify that in the projected equation methods, the policies end up oscillating between a few different policies rather than converging to a single policy, even without simulation error. Consistent with the experiments of [TS09], we may obtain a much better policy by using a random search method such as the CE method.

The experimental results show that different choices of λ provide different results, even without simulation error, consistent with the theory of these methods. However, there are currently no criteria to choose the best λ before running the projected equation methods. As a result, it requires more investigation to find a way to choose λ wisely.

In the aggregation methods, the dimension of r is significantly lower than that of original cost-to-go vectors but larger than that in the projected equation methods. We experimented with two different hard aggregation methods: one is with uniform disaggregation probabilities, and the other is with representative states. The results suggest that the first approach provides slightly better policies. However, this approach requires computational time that depends on the number of original states in order to calculate the disaggregation probabilities. As a result, only the representative states approach can be used for a bigger board. The results show that the performance of aggregation methods is better when s (the number of top rows considered) is bigger.

When the board is small, the aggregation methods provide better policies than the projected equation methods do. However, when the board is big, the number of aggregate states can be very large. As a result, the aggregation methods as implemented in this thesis are not practical for board sizes beyond some threshold. In this case, projected equation methods may be used, but their effectiveness is questionable in view of the much superior results obtained with the CE methods.

In this thesis, we do not aim to build a champion controllers for Tetris. However,

our results indicate that the aggregation methods are applicable for smaller versions of Tetris games. Moreover, we have obtained good results with a number of aggregate states that is much smaller than the number of original system states. With better aggregation and disaggregation probabilities, or with simulation-based calculations which allow more aggregate states, it may be possible to find better aggregation-based controllers for Tetris.

One major issue for ADP algorithms is that, when we replace the optimal cost-to-go vector $J^\star$ in Bellman's Equation with another vector to find a policy, the vectors $\bar{J}$ and $\bar{J} + ce$ yield the same policy for any constant c, where $e = [1, 1, \cdots, 1]'$. This fact is reflected in the projected equation methods, where the best policy is obtained with a fixed weight for the constant feature. This suggests that with projection onto an affine space determined by a heuristically chosen value of c, the projected equation methods may provide better policies. For example, we may choose the affine space based on the knowledge of previous controllers. However, no theory has been developed for choosing this affine space. This may be an interesting topic for future investigation.

The projection Π in the projected equation methods is used to find a good approximation to the policy. We have shown that the pseudometric D_μ may provide a good evaluation of the opposite of the similarity between two policies. The value $D_\mu(\mu^1, \mu^2)$ can also be calculated approximately through a simulation with policy μ. This may help us to understand the policy oscillations for large scale DP problems with the simulation-based projected equation methods. If we have found two policies μ^1 and μ^2 with the approximate distance $\bar{D}_{\mu^1}(\mu^1, \mu^2)$ close to 0 with the projected equation methods. We may infer that the policies roughly start randomly oscillating in some sense. However, more analysis and development of theory are needed to understand the policy oscillation of the projected equation methods.

Bibliography

[BB96] S. J. Bradtke and A. G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22:33–57, 1996.

[Ber05] D. P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, Belmont, MA, 3rd edition, 2005.

[Ber10a] D. P. Bertsekas. Approximate policy iteration: A survey and some new methods. In *Lab. for Information and Decision Systems Report LIDS-P-2833.* MIT, Cambridge, MA, 2010.

[Ber10b] D. P. Bertsekas. Pathologies of temporal difference methods in approximate dynamic programming. *Decision and Control*, 2010.

[Ber11] D. P. Bertsekas. *Dynamic Programming and Optimal Control: Online Chapter Approximate Dynamic Programming.* Athena Scientific, Belmont, MA, May 11 version, 2011.

[BI96] D. P. Bertsekas and S. Ioffe. Temporal differences-based policy iteration and applications in neuro-dynamic programming. In *Lab. for Information and Decision Systems Report LIDS-P-2349.* MIT, Cambridge, MA, 1996.

[BKV96] M. Bezem, M. Keijzer, and C. Volmac. Generalizing Hamming distance to finite sets to the purpose of classifying heterogeneous objects, April 1996.

[Bor08] V. S. Borkar. *Stochastic Approximation: A Dynamical Systems Viewpoint.* Hindustan Book Agency and Cambridge University Press (jointly), Delhi, India and Cambridge, UK, 2008.

[BT96] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, Belmont, MA, 1996.

[Bur96] H. Burgiel. How to lose at tetris. *Mathematical Gazette*, 81:194–200, 1996.

[Cao09] X. R. Cao. Stochastic learning and optimization - a sensitivity-based approach. *Annual Reviews in Control*, 33(1):11–24, 2009.

[CFHM07] H. S. Chang, M. C. Fu, J. Hu, and S. I. Marcus. *Simulation-Based Algorithms for Markov Decision Processes.* Springer-Verlag, London, UK, 2007.

[DHLN03] E. D. Demaine, S. Hohenberger, and D. Liben-Nowell. Tetris is hard, even to approximate. In *Proceedings of the ninth International Computing and Combinatorics Conference*, pp. 351–363, 2003.

[DLN+94] J. Dongarra, A. Lumsdaine, X. Niu, R. Pozo, and K. Remington. A sparse matrix library in c++ for high performance architectures, 1994.

[FVR06] V. F. Farias and B. Van Roy. *Probabilistic and Randomized Methods for Design Under Uncertainty, chapter Tetris: A Study of Randomized Constraint Sampling.* Springer-Verlag, London, UK, 2006.

[Gor95] G. J. Gordan. Stable function approximation in dynamic programming. In *Machine Learning: Proceedings of the Twelfth International Conference*, pp. 351–363. Morgan Kaufmann, San Francisco, CA, 1995.

[Gos03] A. Gosavi. *Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning.* Kluwer Academic, Norwell, MA, 2003.

[Kak02] S. Kakade. A natural policy gradient. In *Advances in Neural Information Processing Systems*, Vol. 14. MIT Press, Cambridge, MA, 2002.

[LKS93] R. P. Lippmann, L. Kukolich, and E. Singer. Lnknet: Neural network, machine-learning, and statistical software for pattern classification. *Lincoln Laboratory Journal*, 6(2):249–268, 1993.

[Mey07] S. Meyn. *Control Techniques for Complex Networks.* Cambridge University Press, New York, NY, 2007.

[Pow07] W. B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality.* Wiley-Interscience, 2007.

[SB98] R. S. Sutton and A. G. Barto. *Reinforcement learning.* MIT Press, Cambridge, MA, 1998.

[SJJ94] S. P. Singh, T. Jaakkola, and M. I. Jordan. Learning without state-estimation in partially observable markovian decision processes. In *Proceedings of the Eleventh International Machine Learning Conference*, pp. 284–292. Morgan Kaufmann, San Francisco, CA, 1994.

[SJJ95] S. P. Singh, T. Jaakkola, and M. I. Jordan. Reinforcement learning with soft state aggregation. *Advances in Neural Information Processing Systems*, 7:361–368, 1995.

[SL06] I. Szita and A. Lorincz. Learning tetris using the noisy cross-entropy method. *Neural Computation*, 18:2936–2941, 2006.

[TS09] C. Thiery and B. Scherrer. Improvements on learning tetris with cross entropy. *International Computer Games Association Journal*, 32:23–33, 2009.

[VR06] B. Van Roy. Performance loss bounds for approximate value iteration with state aggregation. *Mathematics of Operations Research*, 31:234–244, 2006.