CE 273
Markov Decision Processes

Lecture 20

Q-learning and Approximate Linear Programming

Previously on Markov Decision Processes

A very useful value function-like concept in the context of RL is called Q-factors. Recall that the VI step takes the form

$$J_{k+1}(i) = \min_{u \in U(i)} \left\{ \sum_{j=1}^{n} p_{ij}(u)\Big(g(i,u,j) + \alpha J_k(j)\Big) \right\} \forall\, i = 1, \dots, n$$

We define new iterates $Q_{k+1}(i,u) \,\forall\, i = 1, \dots, n, u \in U(i)$ such that

$$Q_{k+1}(i,u) = \sum_{j=1}^{n} p_{ij}(u)\Big(g(i,u,j) + \alpha J_k(j)\Big)$$

Thus, we can rewrite VI algorithm as

$$J_{k+1}(i) = \min_{u \in U(i)} Q_{k+1}(i,u) \,\forall\, i = 1, \dots, n$$

Can the VI algorithm be re-written in terms of the Q values only

$$Q_{k+1}(i,u) = \sum_{j=1}^{n} p_{ij}(u)\Big(g(i,u,j) + \alpha \min_{v \in U(j)} Q_k(j,v)\Big)$$

Previously on Markov Decision Processes

Notice that $Q^*(i,u) = \sum_{j=1}^n p_{ij}(u)\Big(g(i,u,j) + \alpha J^*(j)\Big)$. Thus, one can interpret $Q^*(i,u)$ as the value of taking an action u in state i and behaving optimally thereafter.

$$Q^*(i,u) = \sum_{j=1}^{n} p_{ij}(u)\Big(g(i,u,j) + \alpha \min_{v \in U(j)} Q^*(j,v)\Big)$$

The optimal policy at state i is the control which minimizes the RHS in the following equation.

$$J^*(i) = \min_{u \in U(i)} Q^*(i,u) \,\forall\, i = 1, \ldots, n$$

In general, given the optimal value functions J^*, to recover the optimal policy, one has to find μ^* such that $T_{\mu^*} J^* = TJ^*$.

In that sense, Q-factors are ideal for model-free situations since if an oracle gave you the optimal Q factors, the optimal policies can be derived without the knowledge of one-stage costs and transition probabilities!

Previously on Markov Decision Processes

Suppose for each state i, we extract m features. Let k represent a generic feature. Then, the vector of approximate value functions can be written as

$$\tilde{J} = \Phi r$$

where Φ is

$$\Phi = \begin{bmatrix} \phi_1(1) & \dots & \phi_m(1) \\ \phi_1(2) & \dots & \phi_m(2) \\ \vdots & \vdots & \vdots \\ \phi_1(n) & \dots & \phi_m(n) \end{bmatrix}_{n \times m} \qquad r = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_m \end{bmatrix}_{m \times 1}$$

The rows of the Φ matrix are features and the columns can be interpreted as basis functions/vectors.

Thus, we can think of the subspace $S = \{\Phi r | r \in \mathbb{R}^m\}$ as the subspace spanned by the basis vectors (columns of Φ).

Previously on Markov Decision Processes

From the monotonicity lemma,

$$J \leq TJ \Rightarrow J \leq J^*$$

Also, from the Bellman equations $J^* = TJ^*$. Thus, among all functions J that satisfy $J \leq TJ$, J^* is the "largest".

$$J \leq TJ$$

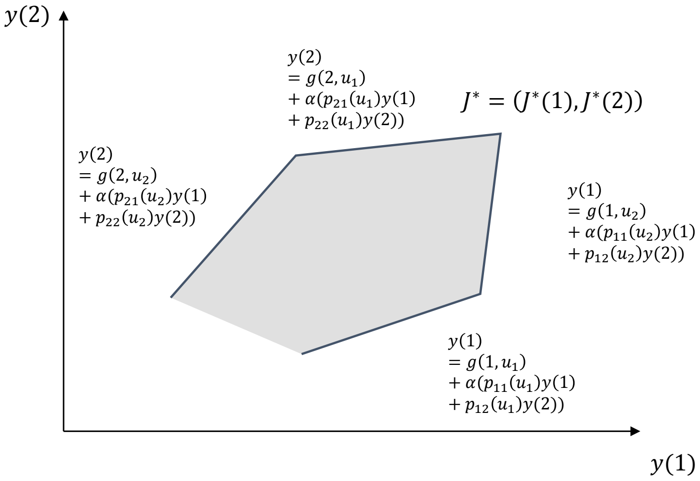
$$\Rightarrow J(i) \leq g(i,u) + \alpha \sum_{j=1}^{n} p_{ij}(u) J(j) \,\forall\, i = 1,\ldots,n, u \in U(i)$$

Thus, if we treat $J(i)$s as the decision variables, these form a linear set of constraints. Since J^* must be "largest" component wise, we can think of J^* as the solution to the linear program,

$$\max_i \sum_{i=1}^{n} a_i y(i)$$

$$\text{s.t. } y(i) - \alpha \sum_{j=1}^{n} p_{ij}(u) y(j) \leq g(i,u) \qquad \forall\, i = 1,\ldots,n, u \in U(i)$$

Previously on Markov Decision Processes



$y(2)$
$y(1)$
$y(2) = g(2,u_1) + \alpha(p_{21}(u_1)y(1) + p_{22}(u_1)y(2))$
$J^* = (J^*(1), J^*(2))$
$y(2) = g(2,u_2) + \alpha(p_{21}(u_2)y(1) + p_{22}(u_2)y(2))$
$y(1) = g(1,u_2) + \alpha(p_{11}(u_2)y(1) + p_{12}(u_2)y(2))$
$y(1) = g(1,u_1) + \alpha(p_{11}(u_1)y(1) + p_{12}(u_1)y(2))$

Lecture Outline

1. Q-learning
2. Approximate Linear Programming

Lecture Outline

Q-learning

Q-learning
Introduction

The methods we saw in the last two classes approximate the value functions associated with a given policy.

One can also directly approximate the true value functions J^* or Q^*!

In today's class we will study two such approaches: Q-learning and approximate linear programming.

In fact, Q-learning is not exactly an approximation method since it converges to the exact optimal value functions Q^*, but in the limit.

Q-learning

Introduction

Q-factors are appealing since if we know $Q*$ then we can construct the optimal policy without the knowledge of the one-step costs and transition probabilities.

However, the VI-like algorithm in terms of the Q-factors is not model free.

$$Q_{k+1}(i,u) = \sum_{j=1}^{n} p_{ij}(u)\Big(g(i,u,j) + \alpha \min_{v \in U(j)} Q_k(j,v)\Big)$$

Q-learning
Introduction

Earlier, we saw that means of samples $x_1, \dots, x_k$ can be constructed recursively as follows.

Find iterates $\mu_k = \frac{1}{k} \sum_{k=1}^{k} x_k$ in terms of previous μ_{k-1} using

$$\mu_k = \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$$

This can be alternately written as

$$\mu_k = \left(1 - \frac{1}{k}\right)\mu_{k-1} + \frac{1}{k} x_k$$

We will use this structure in providing a sketch for a VI-like algorithm with the Q-factors. But first, let's extend this idea to find the fixed point of a mapping that involves an expectation.

Q-learning

Stochastic Approximation

Consider a mapping $F(x) = \mathbb{E}_w[f(x, w)]$, where w is a random variable.

If F is a contraction mapping, then we can find the fixed point using a recursive procedure

$$x_{k+1} = F(x_k) = \mathbb{E}_w[f(x_k, w)]$$

When calculating the above expectation is prohibitive, we sample values from the distribution of the random variable w: $w_1, w_2 \ldots$. One approximate method for estimating x_{k+1} is

$$x_{k+1} = \mathbb{E}_w[f(x_k, w)] \approx \frac{1}{k} \sum_{t=1}^{k} f(x_k, w_t)$$

Notice that we use a fewer samples in the beginning but in the limit, the accuracy of the sample average increases. How many calculations are needed to generate x_{k+1}.

Q-learning

Stochastic Approximation

What if instead, x_k is replaced with with x_t in $1/k \sum_{t=1}^{k} f(x_k, w_t)$? How many calculations do we need to estimate

$$x_{k+1} \approx \frac{1}{k} \sum_{t=1}^{k} f(x_t, w_t)$$

It turns out that the above procedure also converges to the fixed point of the mapping F! Since

$$x_k = \frac{1}{k-1} \sum_{t=1}^{k-1} f(x_t, w_t)$$

the modified iterations can be rewritten as

$$x_{k+1} = \Big(1 - \frac{1}{k}\Big) x_k + \frac{1}{k} f(x_k, w_k)$$

Therefore, denoting $\frac{1}{k}$ using γ_k,

$$x_{k+1} = (1 - \gamma_k) x_k + \gamma_k f(x_k, w_k)$$

Q-learning

Algorithm

Let's apply this idea on the Q-factors. From the Bellman equations

$$Q^*(i,u) = \sum_{j=1}^{n} p_{ij}(u)\left(g(i,u,j) + \alpha \min_{v \in U(j)} Q^*(j,v)\right)$$

This is a fixed point problem $Q^* = FQ^*$ where F is

$$\begin{aligned}(FQ)(i,u) &= \sum_{j=1}^{n} p_{ij}(u)\left(g(i,u,j) + \alpha \min_{v \in U(j)} Q(j,v)\right)\\ &= \mathbb{E}_j[g(i,u,j) + \alpha \min_{v \in U(j)} Q(j,v)]\end{aligned}$$

Comparing this with $x_{k+1} = \mathbb{E}_w[f(x_k,w)]$ and $x_{k+1} = (1-\gamma_k)x_k + \gamma_k f(x_k,w_k)$, fixed point of F can be found using

$$Q_{k+1}(i,u) = (1-\gamma_k)Q_k(i,u) + \gamma_k\left(g(i,u,\xi(i,u)) + \alpha \min_{v \in U(\xi(i,u))} Q_k(\xi(i,u),v)\right)$$

where $\xi(i,u)$ is a sampled future state when we take action u in state i.

Q-learning

Synchronous Version

One can choose γ_k to be any sequence as long as $\sum_k \gamma_k = \infty$ and $\gamma_k^2 < \infty$.

The above algorithms is just like VI where the Q-factors in iteration k are used to update those in iteration $k+1$.

The only difference is that these updates do not require any mathematical model. A simulator of the system is sufficient.

Notice that within each iteration, the updates of the Q-factors must be performed for all state-action pairs. Hence, this approach is also called **synchronous Q-learning**.

Q-learning
Asynchronous Version

On the other hand, in online settings it may not be possible to simulate future states from every state-action pair.

One can still use these type of updates but all state-action pairs must be visited infinitely often for convergence.

In order to ensure this property, we start from a state i_0 and select an action u_0 randomly. The environment takes us to state i_1 and we again pick u_1 randomly and repeat.

This version is called **Asynchronous Q-learning** and can be expressed as

$$Q(i_k, u_k) \leftarrow (1 - \gamma_{\text{numVisits}(i_k, u_k)}) Q(i_k, u_k) + \gamma_{\text{numVisits}(i_k, u_k)} \left(g(i_k, u_k, i_{k+1}) + \alpha \min_{v \in U(i_{k+1})} Q(i_{k+1}, v) \right)$$

where $\gamma_{\text{numVisits}(i_k, u_k)}$ could be $1/(\text{numVisits}(i_k, u_k))$.

Q-learning

Parting Note

The F mapping involving Q-factors is a contraction mapping because one view the problem as a modified MDP in which states are ordered pairs (i, u) actions are singletons $\{u\}$.

$J(i, u)$ for this problem is same as the Q-factors.

The VI version of the Q-factors were modified to suit a model-free setting. Why not do the same with J values and save on memory?

The earlier approximation assumed that $F(x)$ is of the form $\mathbb{E}[f(x, w)]$. The T mapping on the other hand has a minimization outside the expectation (unless all action spaces are singletons).

Lecture Outline

Approximate Linear Programming

Approximate Linear Programming

Let us revisit the LP formulation for discounted cost MDPs.

$$\max_{i} \sum_{i=1}^{n} a_i J(i)$$

$$\text{s.t. } J(i) \leq g(i,u) + \alpha \sum_{j=1}^{n} p_{ij}(u) J(j) \qquad \forall\, i = 1, \ldots, n, u \in U(i)$$

A compact representation of this problem is

$$\max\ a^T J$$
$$\text{s.t. } J \leq TJ$$

How many constraints and variables do we have? What if we combine this formulation with the parametric methods, i.e., replace J with Φr.

$$\max\ a^T \Phi r$$
$$\text{s.t. } \Phi r \leq T \Phi r$$

How many constraints and variables do we have?

Approximate Linear Programming
Introduction

Writing this is an elaborate form

$$\max_{i} \sum_{i=1}^{n} a_i (\Phi r)(i)$$

$$\text{s.t. } (\Phi r)(i) \leq g(i,u) + \alpha \sum_{j=1}^{n} p_{ij}(u)(\Phi r)(y) \qquad \forall\, i = 1, \ldots, n, u \in U(i)$$

The number of variables are reduced but the number of constraints remains the same. Hopefully, not all of them are important.

Solving this model gives us an $\tilde{r}$ using which one can construct a greedy policy

$$\tilde{\mu}(i) \in \arg \min_{u \in U(i)} \left\{ g(i,u) + \alpha \sum_{j=1}^{n} p_{ij}(u)(\Phi \tilde{r})(j) \right\}$$

Notice that we aren't finding the best r for a given policy unlike before but we are trying to approximate the optimal J^* directly. Is this new approximate LP any good?

Approximate Linear Programming

Introduction

Suppose $\|\cdot\|_{1,a}$ denotes the weighted ℓ_1-norm, i.e.,

$$\|J\|_{1,a} = \sum_{i=1}^{n} a_i |J(i)|$$

The following proposition shows that the solution to the approx. LP minimizes a distance measure between the approx. the value function and J^*.

Proposition

A vector $\tilde{r}$ solves

$$\begin{aligned} &\max\ a^T \Phi r \\ &s.t.\ \Phi r \leq T\Phi r \end{aligned}$$

if and only if it solves

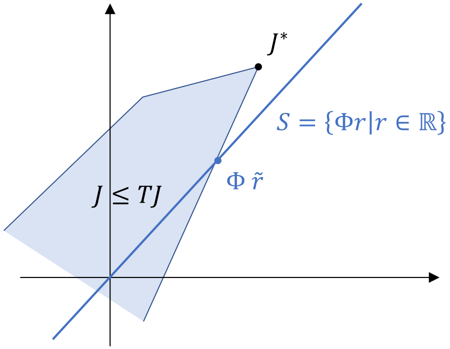
$$\begin{aligned} &\min\ \|J^* - \Phi r\|_{1,a} \\ &s.t.\ \Phi r \leq T\Phi r \end{aligned}$$

However, note that r is not unconstrained, i.e., we cannot choose an approximate optimal value function from any Φr.

Approximate Linear Programming

Geometric Insight

Geometrically, consider a case where the cost functions are 2×1 vectors and r is a scalar.



Approximate Linear Programming

Theoretical Results

Proof.

Any feasible r to the two optimization problems clearly satisfies $\Phi r \leq J^*$.

The objective of the second formulation can be expanded as

$$\begin{aligned}\|J^* - \Phi r\|_{1,a} &= \sum_{i \in S} a_i |J^*(i) - (\Phi r)(i)| \\ &= \sum_{i \in S} a_i \Big(J^*(i) - (\Phi r)(i) \Big) \\ &= a^T J^* - a^T \Phi r\end{aligned}$$

Hence, maximizing $a^T \Phi r$ is equivalent to minimizing $\|J^* - \Phi r\|_{1,a}$. ■

The weight vector a scales the error in the approximation for different states. Therefore, if the weights are higher for some states, one can expect better approximations in such regions.

Approximate Linear Programming

Theoretical Results

The analysis so far tells us that solving the ALP is equivalent to minimizing the distance between the optimal value function and $\Phi\tilde{r}$.

But then, when we use policy $\tilde{\mu}$, we may experience a cost vector $J_{\tilde{\mu}}$ that's different from $\Phi\tilde{r}$. (Why?)

So for the LP approximation to be a good method, we must actually be minimizing the distance between J^* and $J_{\tilde{\mu}}$.

We can be optimistic that the distance between these two vectors is better than that between J^* and $\Phi\tilde{r}$. (Why?) $\tilde{\mu}$ is essentially a rollout policy and empirically such policies have been found to be do well.

Approximate Linear Programming

Theoretical Results

Proposition

Given a probability distribution a, let $(a')^T$ *be another probability distribution defined as* $(1-\alpha)a^T(I-\alpha P_{\tilde{\mu}})^{-1}$. *Then,*

$$\|J_{\tilde{\mu}} - J^*\|_{1,a} \leq \frac{1}{1-\alpha}\|\Phi\tilde{r} - J^*\|_{1,a'}$$

Hence, if the approximate value function $\Phi\tilde{r}$ is close to J^* (which is ensured by the ALP), the performance of the policy generated by it $J_{\tilde{\mu}}$ is also close to the value of the optimal policy.

Proof.

$$\begin{aligned} J_{\tilde{\mu}} - \Phi\tilde{r} &= (I-\alpha P_{\tilde{\mu}})^{-1}g_{\tilde{\mu}} - \Phi\tilde{r} \\ &= (I-\alpha P_{\tilde{\mu}})^{-1}\Big(g_{\tilde{\mu}} - (I-\alpha P_{\tilde{\mu}})\Phi\tilde{r}\Big) \\ &= (I-\alpha P_{\tilde{\mu}})^{-1}\Big(T(\Phi\tilde{r}) - \Phi\tilde{r}\Big) \end{aligned}$$

Approximate Linear Programming

Theoretical Results

Proof.

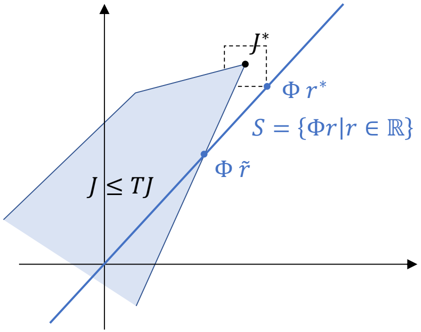
From the constraints of the ALP, $\Phi\tilde{r} \leq T(\Phi\tilde{r}) \leq J^*$. Also, from the optimality of J^*, we have $J^* \leq J_{\tilde{\mu}}$.

$$\begin{aligned}
\|J_{\tilde{\mu}} - J^*\|_{1,a} &= a^T(J_{\tilde{\mu}} - J^*) \\
&\leq a^T(J_{\tilde{\mu}} - \Phi\tilde{r}) \\
&= a^T(I - \alpha P_{\tilde{\mu}})^{-1}\Big(T(\Phi\tilde{r}) - \Phi\tilde{r}\Big) \\
&= \frac{1}{1-\alpha}(1-\alpha)a^T(I - \alpha P_{\tilde{\mu}})^{-1}\Big(T(\Phi\tilde{r}) - \Phi\tilde{r}\Big) \\
&\leq \frac{1}{1-\alpha}(a')^T\Big(J^* - \Phi\tilde{r}\Big) \\
&= \frac{1}{1-\alpha}\|J^* - \Phi\tilde{r}\|_{1,a'}
\end{aligned}$$

■

Approximate Linear Programming

In the following figure, the approximate cost function $\Phi\tilde{r}$ appears to be worse than that obtained by the direct method. (We can't use the direct method on J^* because we cannot simulate its value.)



However if the optimal value function lies in/close to the span of the columns of Φ, then the solution quality of the approximate LP also improves. (Imagine rotating the subspace anticlockwise.)

Approximate Linear Programming

Theoretical Results

The following proposition claims that if Φr^* is the maximum norm projection on S, $\Phi \tilde{r}$ is not very far compared to the distance between J^* and Φr^*. Let $\| \cdot \|_\infty$ be the ℓ_∞ or sup norm.

Proposition

Let e be in the span of the columns of Φ and a be a probability mass function. Then,

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

Proof.

Suppose r^* minimizes the RHS in the above inequality and let $\epsilon = \|J^* - \Phi r^*\|_\infty$. From contraction property of T,

$$\|T(\Phi r^*) - J^*\|_\infty \leq \alpha \|\Phi r^* - J^*\|_\infty = \alpha\epsilon$$

Therefore, $T(\Phi r^*) \geq J^* - \alpha\epsilon e$. One can also show $J^* \geq \Phi r^* - \epsilon e$. Recall that from the constant shift lemma,

$$T(J - \delta e) = TJ - \alpha\delta e$$

Approximate Linear Programming

Theoretical Results

Proof.

Using the above equations,

$$\begin{aligned} T(\Phi r^* - \delta e) &= T(\Phi r^*) - \alpha\delta e \\ &\geq J^* - \alpha\epsilon e - \alpha\delta e \\ &\geq \Phi r^* - (1+\alpha)\epsilon e - \alpha\delta e \\ &= \Phi r^* - (1+\alpha)\epsilon e - \alpha\delta e + \delta e - \delta e \\ &= \Phi r^* + [-(1+\alpha)\epsilon + (1-\alpha)\delta]e - \delta e \end{aligned}$$

Since the above inequality is true for all δ, select

$$\delta = \frac{(1+\alpha)\epsilon}{1-\alpha}$$

For this choice of δ, the above equation simplifies to $T(\Phi r^* - \delta e) \geq \Phi r^* - \delta e$. Hence, $\Phi r^* - \delta e$ is a feasible solution to the LP.

$\because$ e is in the span of the columns of Φ, hence we can always find $\bar{r}$ that satisfies $\Phi\bar{r} = \Phi r^* - \delta e \Rightarrow \|\Phi r^* - \Phi\tilde{r}\|_\infty = \delta$

Approximate Linear Programming

Theoretical Results

Proof.

Using the triangle inequality,

$$\begin{aligned}\|\Phi\bar{r} - J^*\|_\infty \leq& \|J^* - \Phi r^*\|_\infty + \|\Phi r^* - \Phi\bar{r}\|_\infty \\ =& \epsilon + \delta = \frac{2\epsilon}{1-\alpha}\end{aligned}$$

Since $\tilde{r}$ is optimal to the second reformulated ALP,

$$\begin{aligned}\|J^* - \Phi\tilde{r}\|_{1,c} &\leq \|J^* - \Phi\bar{r}\|_{1,c} \\ &\leq \|J^* - \Phi\bar{r}\|_\infty \\ &\leq \frac{2\epsilon}{1-\alpha}\end{aligned}$$

■

The bound we just saw does impose some restrictions by assuming that e is an element of the span. There are improved bounds which do not require this assumption.

Approximate Linear Programming

Example 1

Consider a discrete-time queueing model in which at most one individual may arrive in a period with probability 0.2.

The service probabilities are chosen from $\{0.2, 0.4, 0.6, 0.8\}$ and $g(i, u) = i + 60u^3$, where the state i is the number of customers in the queue and it can vary from 0 to 49,999.

The basis functions are assumed $\phi_1(i) = 1, \phi_2(i) = i, \phi_3(i) = i^2, \phi_4(i) = i^3$ and weights equal $a_i = (1 - \xi)\xi^i$, where ξ is a tuneable parameter.

Approximate Linear Programming
Example 1

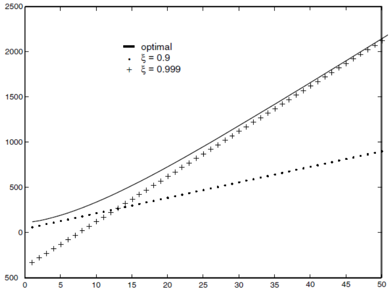


Figure: $\Phi\tilde{r}$ vs. J^*

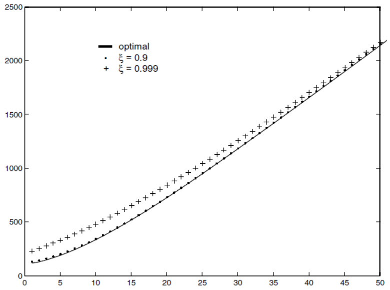


Figure: $J_{\tilde{\mu}}$ vs. J^*

Approximate Linear Programming

Example 1

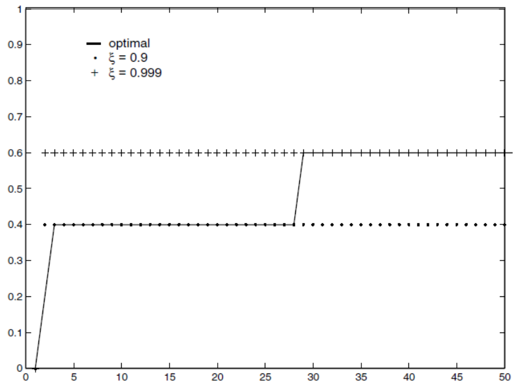
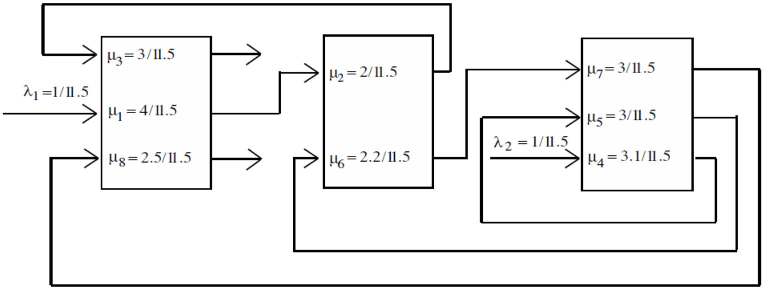


Figure: $\tilde{\mu}$ vs. μ^*

Approximate Linear Programming

Example 2

Here is another example from the paper which deals with a network of queues.



λs represent arrival probabilities and μs denote service probabilities. The decision in this problem is whether each queue must be operated or not.

The state x is a vector in $\mathbb{R}^8$ and the cost-per-stage is assumed $g(x) = |x|$.

Approximate Linear Programming

Example 2

Table: Performance of different policies.

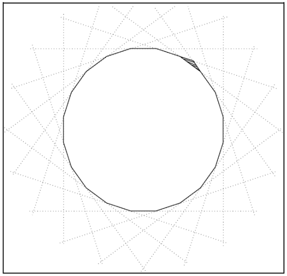
Policy	Average Cost
ADP with $\xi = 0.95$	33.37
LONGEST (Serve longest queue)	45.04
FIFO (First-in-first-out)	45.71
LBFS (Last-buffer-first-served)	144.1

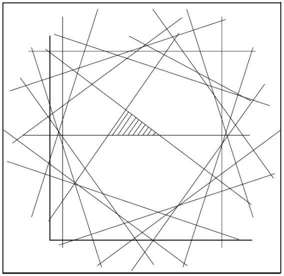
Approximate Linear Programming

Random Sampling

In problems with large state spaces, the number of constraints can make ALP less attractive.

One option to overcome this problem is to randomly sample constraints from some distribution over state-action pairs (i, u).





Approximate Linear Programming

Random Sampling

There are neat theoretical results that show that the scenario on the right is less likely and one need not sample an exponential number of constraints to get good approximations if the distribution for sampling constraints are carefully chosen.

For instance, the following table shows the performance of ALP on Tetris (feature vector comprises of the 22 elements discussed before) with 2 million constraints that are sampled from a distribution.

Table: Performance of different policies.

Policy	Score
TD-learning	3183
Policy Gradient	5500
LP w/ Bootstrap	4274

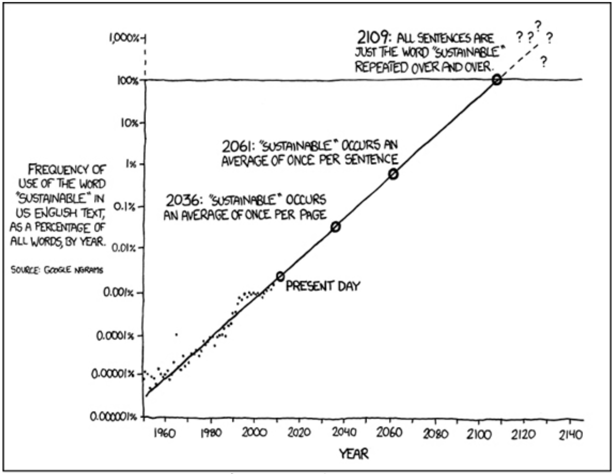
Approximate Linear Programming

Additional Reading

References:

- De Farias, D. P., & Van Roy, B. (2003). The linear programming approach to approximate dynamic programming. Operations research, 51(6), 850-865.
- De Farias, D. P., & Van Roy, B. (2004). On constraint sampling in the linear programming approach to approximate dynamic programming. Mathematics of operations research, 29(3), 462-478.
- Farias, V. F., & Van Roy, B. (2006). Tetris: A study of randomized constraint sampling. In Probabilistic and randomized methods for design under uncertainty (pp. 189-201). Springer, London.

Your Moment of Zen



THE WORD "SUSTAINABLE" IS UNSUSTAINABLE.