AA 274: Principles of Robotic Autonomy Winter 2018

Lecture 18: Decision Making and Dynamic Programming

Scribes: Scott Park, Adrian Piedra, Garrett Scott Taylor, Matthew Wu Tsao and Michal Adamkiewicz

18.1 Introduction

The main objective of this final lecture is to briefly introduce the concept of decision making under uncertainty, which essentially deals with the higher level of the decision making module whereby one is trying to reason about what the other agents in the environment are doing. A useful way of reasoning about the uncertainty of the environment is model the environment as probabilistic. This results in the development of control policies that optimize the dynamical system that evolves through time probabilistically.

18.2 Basic Decision Making Problem

- System: $x_{k+1} = f_k(x_k, u_k, w_k),\ k = 0, \dots, N$
  - Very similar to the dynamics studied in the context of filtering.
- Control constraints: $u_k \in U(x_k)$
- Probability distribution: $P_k(|x_k, u_k)$ of w_k
  - The disturbance the affects the dynamics has a probability distribution that only depends on the current state x_k and current control u_k (Markov assumption).
- Policies: $\pi = \{\mu_0, \dots, \mu_{N-1}\}$, where $u_k = \mu_k(x_k)$
  - Interested in optimizing closed-loop policy in stochastic context for robustness.
- Expected Cost:

$$J_\pi(x_0) = E\Big\{g_N(x_N) + \sum_{k=1}^{N-1} g_k(x_k, \mu_k(x_k), w_k)\Big\}$$

  - g_N is the terminal cost and g_k is the stage-wise cost (note the additive structure of the cost function).
- Decision making problem:

$$J^*(x_0) = \min_\pi J_\pi(x_0)$$

Key points:

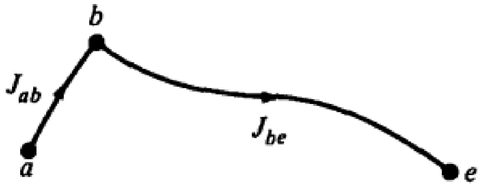
- Discrete-time model
  - Could consider continuous-time model, but in most applications the discrete-time is most natural.

- Markovian model
- Objective: find optimal closed-loop policy
- Additive cost (central assumption to prove the principle of optimality)
- Risk-neutral formulation

18.3 Principle of Optimality

The principle of optimality is a key concept to make stochastic optimal control problem tractable from a computational standpoint.

Consider the simplest case (deterministic, i.e. no stochasticity), and suppose the optimal path for a multi-stage decision-making problem is shown by the figure below



where the first decision yields segment a-b with cost J_{ab}, and remaining decisions yield segments b-e with cost J_{be}. The total optimal cost is then $J^*_{ae} = J_{ab} + J_{be}$.

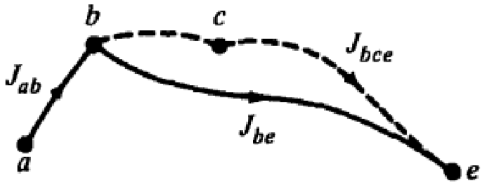
The big claim of the principle of optimality is as follows: if a-b-e is an optimal path from a to e, then b-e is an optimal path from b to e.

Proof by contradiction: Suppose b-c-e is the optimal path from b to e. Then

$$J_{bce} < J_{be}$$

and

$$J_{ab} + J_{bce} < J_{ab} + J_{be} = J^*_{ae}$$



This is a contradiction because J^*_{ae} is already defined as the optimal path from a to e. If J_{bce} was optimal path from b to e instead of J_{be}, then that would imply that J^*_{ae} is not optimal. Therefore, J_{be} must be the optimal path from b to e.

Remark: While the principle of optimality holds for tails of optimal policies, the same is not true for heads of optimal policies. Consider the simple game where there are two strategies π_1, π_2. The costs as a function of time for these strategies are

$$r_1(t) = 10t$$
$$r_2(t) = e^{0.1t}$$

and the goal of the game is to accumulate as much reward as possible for the duration of the game.

$$\pi_T^* = \arg \min_{i \in \{1,2\}} \sum_{t=1}^{T} r_i(t)$$

It is easy to check that for $T = 5$, the optimal policy is π_2, but if $T = 1000$, the optimal policy is π_1. However, the behavior of π_{1000}^* during the first 5 timesteps is not the same as the behavior of π_5^*.

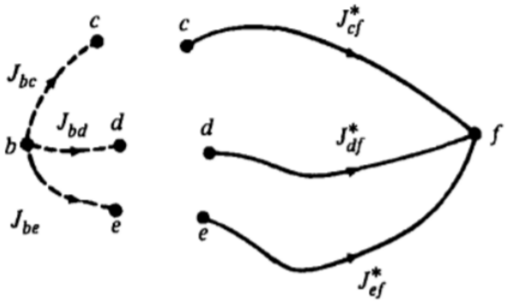
18.3.1 Definition (for discrete-time systems)

Let $\boldsymbol{f}^* := \{\boldsymbol{f}_0^*, \boldsymbol{f}_1^*, \ldots, \boldsymbol{f}_{N-1}^*\}$ be an optimal policy. Assume state $\boldsymbol{x}_k$ is reachable. Consider the subproblem whereby we are at $\boldsymbol{x}_k$ at time k and we wish to minimize the cost-to-go from time k to time N. Then, the truncated policy $\{\boldsymbol{f}_k^*, \boldsymbol{f}_{k+1}^*, \ldots, \boldsymbol{f}_{N-1}^*\}$ is optimal for the subproblem.

Considering this definition, *tail* policies are optimal for *tail* subproblems. Also, mind that the time-dependence is implicit in the notation: $\boldsymbol{f}_k^*(\boldsymbol{x}_k) = \boldsymbol{f}^*(\boldsymbol{x}_k, k)$.

18.3.2 Applying the principle of optimality

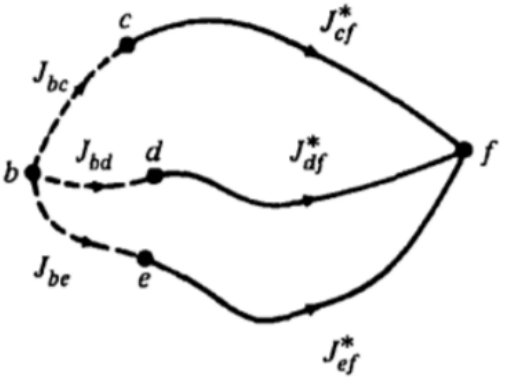
Consider the problem shown in the figure below.



According to the principle of optimality, if $b - c$ is the initial segment of the optimal path from b to f, then $c - f$ is the terminal segment of this path. Hence, the optimal trajectory is found by comparing the following:

$$C_{bcf} = J_{bc} + J_{cf}^*$$
$$C_{bdf} = J_{bd} + J_{df}^*$$
$$C_{bef} = J_{be} + J_{ef}^*$$

The comparison of these three trajectories is shown in the figure below.

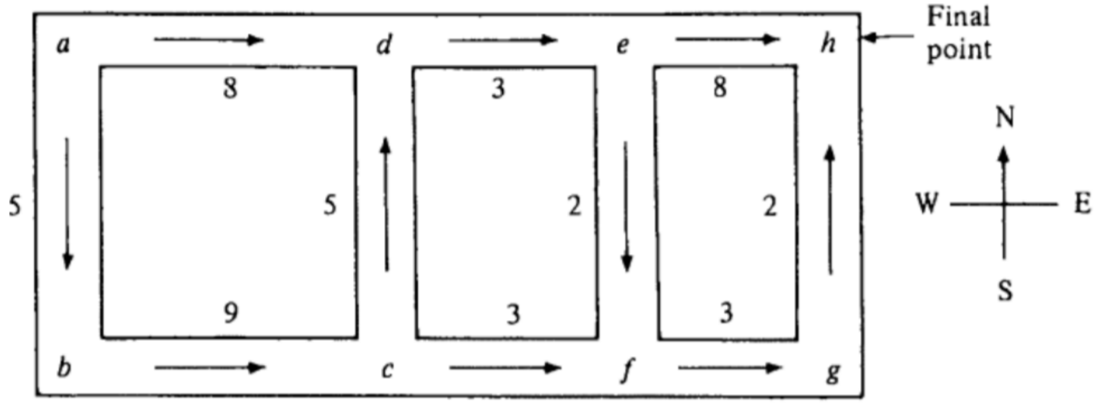


When applying the principle of optimality, one only needs to compare the concatenations of immediate decisions and optimal decisions. This provides a significant decrease in the computation required to solve the problem, and also the amount of possible solutions.

In practice, the principle of optimality is applied *backward* in time. As Soren Kierkegaard said, "Life can only be understood backwards; but it must be lived forwards."

18.3.3 Example problem

As an example of solving a problem with the principle of optimality, consider the figure below. The goal is to start from node a and end at node h while incurring the minimum cost along the path. Consider the following movement map: [North: UP], [South: DOWN], [East: RIGHT], [West: LEFT].



As mentioned earlier, in practice, the principle of optimality is applied backward in time. First, consider the cost-to-go from starting at node h: $J(h) = 0$. Then, consider the cost-to-go and optimal action for node g: $J(g) = 2 + J(h) = 2 + 0 = 2$, $u^*(g) = UP$. Continuing for the rest of the problem, we have the following:

$$J(f) = 3 + J(g) = 3 + 2 = 5$$
$$u^*(f) = RIGHT$$

$$J(f) = 3 + J(g) = 3 + 2 = 5$$
$$u^*(f) = RIGHT$$

$$J(e) = \min(8 + J(h) = 8,\ 2 + J(f) = 7) = 7$$
$$u^*(e) = DOWN$$

$$J(d) = 3 + J(e) = 10$$
$$u^*(d) = RIGHT$$

$$J(c) = \min(5 + J(d) = 15,\ 3 + J(f) = 8) = 8$$
$$u^*(c) = RIGHT$$

$$J(b) = 9 + J(c) = 17$$
$$u^*(b) = RIGHT$$

$$J(a) = \min(5 + J(b) = 22,\ 8 + J(d) = 18) = 18$$
$$u^*(a) = RIGHT$$

Thus, the optimal path for this problem is $a \to d \to e \to f \to g \to h$. The optimal cost associated with this path is $J(a) = 18$.

18.4 Dynamic Programming (DP) Algorithm

Model:

$$\boldsymbol{x}_{k+1} = \boldsymbol{a}(\boldsymbol{x}_k, \boldsymbol{u}_k, k)$$

Cost:

$$J_{\boldsymbol{f}}(\boldsymbol{x}_0) = h_N(\boldsymbol{x}_N) + \sum_{k=0}^{N-1} g(\boldsymbol{x}_k, \boldsymbol{f}_k(\boldsymbol{x}_k), k)$$

Where:

- $\boldsymbol{x}_k$ state at time k
- $\boldsymbol{u}_k$ action at time k
- $\boldsymbol{a}()$ function which tells us what state to go to given where we are and what we did
- $g()$ function which tells us cost of the transition

- $h_N(\boldsymbol{x}_N)$ function which tells us the cost of finishing in state $\boldsymbol{x}_N$
- $\boldsymbol{f}_k(\boldsymbol{x}_k)$ is the policy function which tells us what to do in state $\boldsymbol{x}_N$
- $J_k \boldsymbol{x}_k$ is the cost to go if we are in state $\boldsymbol{x}_k$ at time k

The algorithm starts for the only state of which we can explicitly compute the cost to go (the final state):

$$J_N(\boldsymbol{x}_N) = h_N(\boldsymbol{x}_N)$$

The algorithm then works backwards in time to calculate the cost to go of each state from which we can reach the set of known states. For the cost to go we pick the transition with the lowest transion cost plus future cost to go

$$J_k(\boldsymbol{x}_k) = \min_{\boldsymbol{u}_k \in U(\boldsymbol{x}_k)} g(\boldsymbol{x}_k, \boldsymbol{f}_k(\boldsymbol{x}_k), k) + J_{k+1}(\boldsymbol{a}(\boldsymbol{x}_k, \boldsymbol{u}_k, k))$$

18.4.1 Stochastic case

In the stochastic case we replace the dependence on k with a dependence on w_k - a random variable with a potentially time variable distribution:

Model:

$$\boldsymbol{x}_{k+1} = \boldsymbol{a}(\boldsymbol{x}_k, \boldsymbol{u}_k, w_k)$$

Cost:

$$J_f(\boldsymbol{x}_0) = h_N(\boldsymbol{x}_N) + \sum_{k=0}^{N-1} g(\boldsymbol{x}_k, \boldsymbol{f}_k(\boldsymbol{x}_k), w_k)$$

In the algorithm we replace the calculation of the future cost to go with the calculation with the calculation of the expected cost to go

$$J_k(\boldsymbol{x}_k) = \min_{\boldsymbol{u}_k \in U(\boldsymbol{x}_k)} E_{w_k}\{g(\boldsymbol{x}_k, \boldsymbol{f}_k(\boldsymbol{x}_k), k) + J_{k+1}(\boldsymbol{a}(\boldsymbol{x}_k, \boldsymbol{u}_k, k))\}$$

18.4.2 Example Problem - Inventory control

We have x_k of stock available at time k. We sell w_k (which is random) at time k and we can order u_k to increase our inventory. There is a 10% probability of $w_k = 0$, a 70% probability of $w_k = 1$ and a 20% probability of $w_k = 2$. We can't have more then 2 items on stock at any time and we (obviously) can't sell more then we have. There is no final cost and the incremental cost is defined as

$$g(x_k, f_k(x_k), k) = u_k + (x_k + u_k - w_k)^2$$

We can write the problem in terms of the dynamics:

$$x_{k+1} = a(x_k, u_k, k) = max(0, x_k + u_k - w_k$$

First we set the cost to go for the final state

$$J_3(0) = 0, J_2(0) = 0, J_1(0) = 0$$

We can work backward to find the cost to go at the previous time step:

$$J_2(0) = min_{u_2=0,1,2} E_{w_2}[u_2 + (u_2 - w_2)^2]$$

$$J_2(0) = min_{u_2=0,1,2} E_{w_2}[u_2 + 0.1 * u_2^2 + 0.7 * (u_2 - 1)^2 + 0.2 * (u_2 - 2)^2]$$
$$J_2(0) = 1.3 \text{ when } u_2 = 1$$

Therefore $\mu_2^*(0) = 1$

We go on to the next state. We note that in this calculation we don't consider $u_k = 2$ as that could make us have more then two items in stock

$$J_2(1) = min_{u_2=0,1} E_{w_2}[u_2 + (u_2 - w_2)^2]$$
$$J_2(1) = min_{u_2=0,1} E_{w_2}[u_2 + 0.1 * (u_2 + 1)^2 + 0.7 * (u_2)^2 + 0.2 * (u_2 - 1)^2]$$

$$J_2(1) = 0.3 \text{ when } u_2 = 0$$

Therefore $\mu_2^*(1) = 0$

We can contiue doing this for all the other states

18.4.3 Difficulties of DP

There are essentially three shortcomings associated with Dynamic Programming

- The Curse of Dimensionality
  - Computational and information storage requirements grow exponentially
    - the number of state combinations that must be considered is proportional to the number of possible states raised to the dimension of the problem
  - In the case of imperfect state information, the problem becomes intractable
    - This is often the case for mapping problems, which solves this through the use of partially observable markov decision processes
- The Curse of Modeling
  - When "system stochastics" are complex, it is difficult to obtain transition probabilities
- The Curse of Time
  - Often there is only a short lag time between when enough information is available to compute a solution and when the solution is needed

  - When the system is subjected to control inputs, state information needed to compute subsequent solutions may change
    - On-line replanning is required to mitigate this issue

18.4.4 Solutions to DP Difficulties: Approximate DP (ADP)

There are several ways of dealing with the pitfalls of DP:

- Certainty Equivalent Control
- Cost-to-Go Approximation
- Other various approaches

18.5 Certainty Equivalent Control (CEC)

Key concept is to replace a stochastic problem with a deterministic reformulation

Suppose at each time step, k the future uncertain quantities are fixed for some nominal value of those quantities

we implement the solution on-line in the following way

- 1) $\forall i \geq k$, fix w_i at some nominal value $\bar{w}_i$
  - this leads to a deterministic problem formulation,

$$min \;\; g_N(x_N) + \textstyle\sum_{i=k}^{N-1} g_i(x_i, u_i, \bar{w}_i)$$

$$where \; x_{i+1} = f_i(x_i, u_i, \bar{w}_i)$$

- Subsequently, $\mu_k\bar{(}x_k)$ is used to control the first element in an optimal control sequence and move to time k+1

18.6 Cost to Go Approximation (CGA)

Key concept is to truncate the time horizon a compute an approximate ”cost-to-go” based on said finite time span

The algorithm is referred to an ”n-step look-ahead” policy, and we will discuss the policy in which n=1

”One-Step Look-Ahead” Policy: at each state for k and x_k, use the control input $\mu_k\bar{(}x_k)$, which,

$$\min_{u_k \in U_k(x_k)} \mathbb{E}\left\{g_k(x_k, u_k, \mu_k, w_k) + \tilde{J}_{k+1}(f_k(x_k, u_k, \mu_k, w_k))\right\}$$

$$where \begin{cases} \tilde{J}_N = g_n \\ \tilde{J}_{k+1} \approx J_{k+1}, \; the \; ”true \; cost - to - go” \end{cases}$$

Extending this policy to the ”Two-Step Look-Ahead”: All of the above holds, and $\tilde{J}_k + 1(x_{k+1})$ becomes,

$$\tilde{J}_k + 1(x_{k+1}) = \min_{u_k \in U_k(x_k)} \mathbb{E}\left\{g_{k+1}(x_{k+1}, u_{k+1}, \mu_{k+1}, w_{k+1}) + \tilde{J}_{k+2}(f_{k+1}(x_{k+1}, u_{k+1}, \mu_{k+1}, w_{k+1}))\right\}$$

Ultimately, the $\tilde{J}_{k+n}$ needs to be available to perform this approximation

One possibility is to used the distance squared as an approximation

18.6.1 CGA - Computational Aspects

Some key points

- Assuming $\tilde{J}_{k+1}$ is available and minimization isn't difficult, this approach can be implemented online, and
- Determining an appropriate $\tilde{J}_{k+1}$ is highly critical to the output of the approximation
  - using a simplified surrogate model would allow for a problem approximation
  - using a parametric formulation to compute CGA with a set of tuneable parameters
  - using a rollout approach allows for the use of a suboptimal policy to compute $\tilde{J}_{k+1}$

18.6.2 Problem Approximation

This approach affords us many problem-dependent possibilities. We can,

- assume using nominal values in place of uncertainty quantities is sufficiently accurate
- create a surrogate model of the problem by ignoring some constraints
- assume that subsystem decoupling is non-influential and treat the subsystems independently
- use lower resolution solutions of the system by aggregating states together

18.6.3 Parametric Approximation

This approach allows for the computation of the CGA based on a parameterization of $\tilde{J}_{x,r}$ where x is the current state and $r = (r_1, ..., r_m)$ is a vector of weights that can be tuned

Two key aspects of this approach are as follows,

- 1) it is inherently subjective, because we choose the parameterization
  - for example: (feature extraction)

    $$\tilde{J}_{x,r} = \sum_{i=1}^{m} r_i * y_i(x)$$

    Where, $y_i' s$ are feature vectors
- 2) weight tuning can be accomplished algorithmically
  - probably would want to use a simulation (like Monte Carlo)

18.6.4 Rollout

The take away for this approach is the assumption of some heuristic policy referred to as the ”base policy”

Implementation of the rollout control approach requires a function definition $\forall u_k$

$$Q_k(x_k, u_k) \doteq \mathbb{E}\left\{g_k(x_k, u_k, w_k) + H_{k+1}(x_k, u_k, w_k)\right\}$$

where H_{k+1} is the ”cost-to-go” value of the base policy and Q_k-factors can be

- estimated via Monte Carlo simulation
- approximated through a CEC approach

Remark: Model Predictive Control (MPC) can be viewed as a special case of a rollout algorithm

18.6.5 Other APD Approaches

For those that are interested in additional topics to learn about, the following are also useful APD approaches

- Minimization of the DP equation error
- Direct approximation of the control policies being used
- Approximations of the policy space

18.7 Risk Sensitive Optimization

As discussed earlier, model uncertainty is an issue when trying to solve for an optimal policy in complex or probabilistic environments. The goal of this section is to give a brief introduction in how to formulate cost minimization in a way that is robust to model uncertainty. Recall that we are interested in finding a policy π^* so that

$$\pi^* = \arg\min_{\pi} \mathbb{E}_p[J_\pi(x_0)]$$

where p is a probability distribution that governs the transitions of our system. In many cases, however, we do not know p, so we cannot even evaluate the objective function we wish to minimize. However, depending on domain knowledge, we may know a set Θ for which $p \in \Theta$. Here, Θ is a set of likely candidates of p. Given this knowledge, we can consider the robust formulation

$$\pi^* = \min_{\pi} \max_{q \in \Theta} \mathbb{E}_q[J_\pi(x_0)] \tag{18.1}$$

whereby we aim to minimize our worst case cost over likely transition models in Θ. Note that the risk-neutral situation is a special case of this framework where $\Theta = \{p\}$.

A few remarks are in order. For general Θ, this problem is intractable. Indeed, if Θ is discrete lattice of points, then our robust formulation can be casted as an instance of integer programming which is known to be NP hard in the worst case. However, with some regularity conditions on Θ, minimax problems can be efficiently solved.

Definition: A function $\rho : \mathbb{R}^n \to \mathbb{R}$ is a **Coherent Risk measure** if and only if there exists a set Θ which is a compact, convex subset of all n-dimensional probability vectors such that

$$\rho(v) := \max_{\theta \in \Theta} \theta^T v.$$

While the definition may be abstract, coherent risk measures have several very natural properties. If $\rho : \mathbb{R}^n \to \mathbb{R}$, then the following hold.

- Monotonicity: If $u, v \in \mathbb{R}^n$ are vectors where u is component-wise larger than v, then $\rho(v) \leq \rho(u)$.
- Translation invariance: For $a \in \mathbb{R}$, $\rho(v + a\mathbf{1}) = \rho(v) + a$.
- Positive homogeneity: If $\lambda > 0$ and $v \in \mathbb{R}^n$, then $\rho(\lambda v) = \lambda \rho(v)$.
- Subadditivity: For $u, v \in \mathbb{R}^n$, $\rho(u + v) \leq \rho(u) + \rho(v)$.

The third and fourth properties ensure that ρ is a convex function, meaning that we can minimize coherent risk measures with gradient descent and its variants. If Θ is a polyhedron, then evaluation of its corresponding coherent risk measure can be done via linear programming.

To wrap things up, we present an example whereby coherent risk measures arise as a natural solution to an optimization procedure where robustness is desired.

Example: Empirical risk minimization
Consider the standard inference problem whereby $X, Y \sim P$ where P is unknown, and we wish to predict the value of Y given its corresponding value of X. For simplicity, let's say that X, Y are both discrete random variables that can only take on finitely many values n_x, n_y respectively. We are given pairs $\{x_i, y_i\}_{i=1}^n$ drawn i.i.d. from P as training data. One natural thing to try is to pick a loss function ℓ, and an estimator f_θ parameterized by some weights θ, and choose those weights to achieve low loss on the training data. Specifically, we aim to find

$$\begin{aligned}\theta_{\text{ERM}} &:= \arg\min_\theta \sum_{i=1}^n \ell(f_\theta(x_i), y_i) \\ &= \arg\min_\theta \sum_{(x,y)} \widehat{P}_n(x,y)\ell(f_\theta(x), y) \\ &= \arg\min_\theta \mathbb{E}_{\widehat{P}_n}[\ell(f_\theta(X), Y)]\end{aligned}$$

Here, $\widehat{P}_n$ is the empirical distribution, so that

$$\widehat{P}_n(x,y) = \frac{\text{The number of times } (x,y) \text{ shows up in the traning set}}{\text{number of training pairs}}$$

However, our true goal is not to do well on the training set, but to be able to generalize on unseen samples coming from P. The training set is only useful to us because it gives us some noisy information about the true distribution. Thus what we are really interested in is

$$\begin{aligned}\theta^* &= \arg\min_\theta \sum_{(x,y)} P(x,y)\ell(f_\theta(x), y) \\ &= \arg\min_\theta \mathbb{E}_P[\ell(f_\theta(X), Y)]\end{aligned}$$

However, this is one particular situation where we do not have knowledge of P. We can, however, construct Θ that will contain P with high probability. If the number of training pairs n is large, we expect that $\widehat{P}_n$ will be very similar to P. Intuitively, that means if we set Θ to be the set of probability distributions "close" to $\widehat{P}_n$, then P will be in Θ with very high probability. By Hoeffding's Inequality, for any (x, y) we have:

$$\mathbb{P}(|\widehat{P}_n(x,y) - P(x,y)| > c) \leq \exp\left(-2nc^2\right)$$

Setting $c = \sqrt{\frac{1}{2n} \log \frac{n_x n_y}{\epsilon}}$, we get

$$\mathbb{P}\left(|\widehat{P}_n(x,y) - P(x,y)| > \sqrt{\frac{1}{2n} \log \frac{n_x n_y}{\epsilon}}\right) \leq \frac{\epsilon}{n_x n_y}$$

Now union bounding over all (x, y) we get:

$$||\widehat{P}_n - P||_\infty \leq \sqrt{\frac{1}{2n} \log \frac{n_x n_y}{\epsilon}}$$

with probability at least $1 - \epsilon$. Thus, if we set $\Theta = \{q \in \Delta^{n_x n_y} : ||q - \widehat{P}_n||_\infty \leq C n^{-1/2}\}$ then with high probability, we will have $P \in \Theta$. We can interpret Θ as a confidence region in the sense that it is extremely unlikely that anything outside of Θ could have generated our training data, so we do not consider it. Thus if we want to be robust to the fact that our training set is not exactly the same as P, we can minimize a coherent risk metric induced by our likely candidates in Θ as follows:

$$\theta_{\text{CRM}} = \arg\min_\theta \max_{q \in \Theta} \mathbb{E}_q[\ell(f_\theta(X), Y)]$$

Remark: Note that in this example, the set of likely distributions that generated the training data, Θ, shrinks as the number of training samples n increases. This makes a lot of sense because as you get more data, it is easier to identify the true distribution, thus the need to be robust is lessened. A similar phenomenon can be observed between frequentist and Bayesian estimators. When the number of samples is small, the Bayesian prior has a significant effect on the estimation, but as the number of samples grows, the frequentist and Bayesian estimators converge to one another.